Teaching Machine Learning

Student Project Reports

cpsc 599.66 and 601.66 Winter 2007

Department of Computer Science
University of Calgary

Edited by Michael M. Richter

Content

Introduction

Michael M. Richter

Teaching machine learning has two parts. One part is the lectures. These can be found under www.cpsc.ucalgary.ca/~mrichtet/ml. But lecturing is only half of the story. That is, because passive learning by listening does not provide the same expertise compared to active learning by doing. For this purpose a project work was required. Students had the choice to work on their own or to form a group of two.
At the beginning of the course, after some introduction and overview, the projects started. The start had the following steps:

1) Selecting a domain of application as, e.g. spam filters, playing games, cooperative multi-agents etc.
2) Formulating a learning goal in that domain, as improving cooperation.

The choice was completely free.

3) Selecting one or more candidates for learning techniques presented in the course that were focused in the sequel.

These topics were presented first very early and then in some more detail at midterm. In this volume the final reports are listed. Particular emphasis was put on the aspects of the difficulties that occurred during the project and how to overcome them. The difficulties had different sources. The major ones are problems with the tools and getting enough data, or underestimating the complexity.
The free choice of the application domain had the consequence that the authors were quite familiar with it, could use existing environments and use the results for further activities like masters or PhD theses.
Formal projects implementation details are available, write to mrichter@cpsc.ucalgary.ca
The course could be charactized by the key words

Learning systems
Solid foundations
Applications
Interdisciplinary research.

April 2007
Michael M. Richter

Part 1
Class 599.66

Smart Type

By Tyler Bilawchuk

The project I selected is most closely related to a game, in which the player must type the words that appear on the screen. They are given 5 chances, before their "life" runs out, and the game is over. The game keeps track of mistakes and of successes, not only to track the player's progress but also to learn from the player. The more the program is used by the player, the more the program will learn, and will be able to adapt to the player. Each word the player must type has a set of properties attached to it, and these values are automatically adjusted as the game progresses. The words that are not input correctly are given a higher chance to appear, forcing the player to type these words more frequently, with the hope that through repetition, they will learn the correct spelling of these words and will be able to type them with higher precision. Overall, my hope is that by using this application, the player's typing accuracy and speed will increase.

Contents

Domain of Application

- Spoof on "Typing of the Dead"
  - Instead of simply repeating back words and sentences to type, players have "life" and can be killed if they do not meet the requirements
  - It is a Game, or Typing Tutor Application, usually used to improve the player's speed and accuracy, while giving them motivation to continue and improve
- Set of known words
  - Each word has a set of properties attached to it
- A defined rule set will make the program more into a game
  - ie. Life, difficulty settings, etc.

Problem with the Domain

- Typing tutors are quite common and free of charge
  - What differentiates mine from others is the game's ability to *learn* the strengths and weaknesses of the user
- How to learn this information from the player?
  - Learns by tracking the players progress
- How to differentiate between random mistakes (ie. slipping on the keyboard) and actual typing errors
  - Type the word more often, "slipping" errors will eventually be weeded out
- What if the player spends more time correctly typing the difficult words?
  - A time limit will not allow players to spend more time on difficult words, forcing the player to make mistakes when they have trouble

Learning Goal

- Ultimate goal is to make the program respond to the player in the most beneficial way it can
- The program application will learn the mistakes of the player, and the game will change depending on where mistakes are made
  - The program will continue to learn the player's strengths and weaknesses as the player uses the game
- As the program learns from the player's mistakes, it will make the game more difficult as the player progresses
  - A failed word will be noted, and the game will automatically make that word more likely to appear
  - Similarly, each time a word is typed correctly, that word will become less frequent, making all other words slightly more frequent

Starting Data

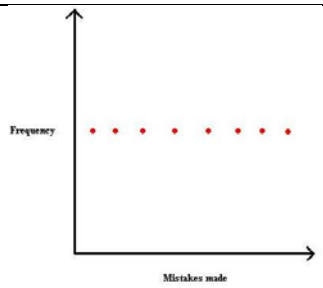
- Words for the list must be chosen, but not at random. Words selected for the list came from researching:
  - The most commonly misspelled words
  - The most frequently used words
  - Dividing up the keyboard (making sure to include all letters, capital letters, double letters (ie. Food, street, etc.)
- What rewards will be provided for the player?
  - A life bar will motivate players to not make mistakes, otherwise they will game over
- Goal: To make zero mistakes, player strives to reach 100% accuracy

What Type of Learning Occurs Here?

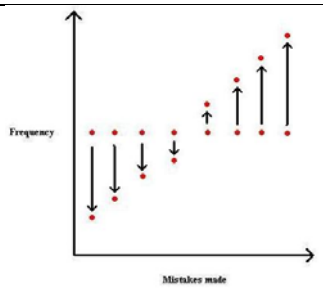
- This is a passive form of unsupervised learning
  - No feedback from the environment
  - Player does not directly tell the program what it did right or wrong, but still gathers information by observing the player
- Symbolic Learning
  - Player attributes are shown as easily understood statistics
  - There is conceptual clustering between "easy" words and "difficult" words
    - Words appear more frequently if player has difficulty with them, less frequently if little or no mistakes are made
- Dynamic
  - Results are constantly changing, the hypothesis for predicting the next word is constantly moving

My Learning Method

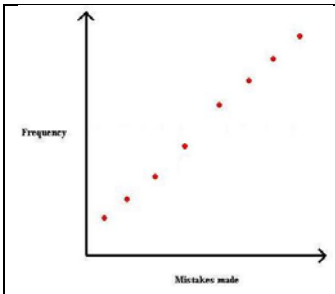
- Unsupervised Behavioural Learning – Dynamic
  - Fitness function
    - The program understands where the player needs most improvement, and this constantly changes as the player continues
  - The program selects it's next "move" (or *word*) by looking at the players past successes and failures
    - By doing so, the program is predicting where the player is most likely to have trouble, and where they need little assistance
- I believe that this form of Dynamic learning is the best fit for the goals I needed to accomplish



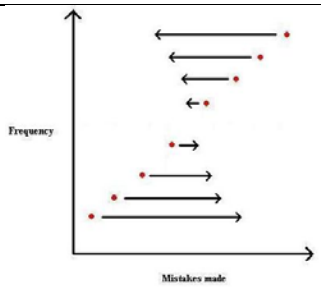
Here we see "Mistakes Made" plotted against the "Frequency" of the word. The red dots indicate different words the player has typed.



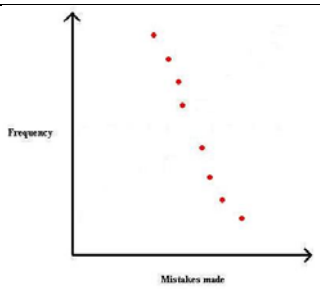
The words that gave the player difficulty have risen in frequency, whereas words that the player has typed correctly become less frequent.



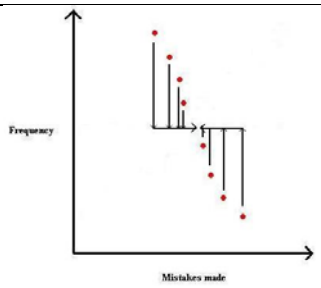
Trouble words are now more frequent. Due to this added frequency, the player should now (in theory) become better at typing these words.



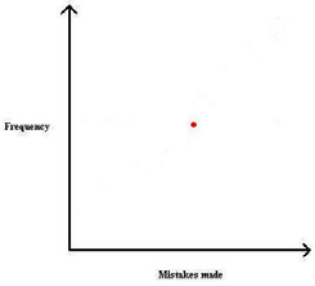
Now the player has improved these "trouble" words, and they now make less mistakes. The high-frequency/high-mistake words now become high-frequency/low-mistake.



Now that the old trouble words no longer provide any difficulty for the player, they will become less frequent.



All the trouble words and the words that the player had few difficulties with now condense to become an equal chance-to-occur frequency.



After much play, the difficult words have become every bit as simple as the easy words. Every word now has roughly an equal chance to occur, and an equal chance that the player will make a mistake on it. No one word stands out as being a trouble word anymore, and the player has overcome their difficulties with their personal trouble words.

Formally Representing Objects

- The changing actions of the program can be represented as a series of percentages and probabilities
- Each word in the program will have a distinct set of *values* or *properties* associated with that word depending on the player's actions, for example:
  - The probability of being the next word to arise
  - How many times the player has typed the word, broken down into "Successes" and "Misses"

Word	Percent chance to occur	Successes	Misses

- At the end of the game, the program will display a set of stats for the player, including suggestions for improvement

Main Interface 1/2

- Here is the main SmartType interface before the game has started
  - Difficulty selection
  - Start and Finish buttons
  - Word Display area
  - User Input area
  - Life bar

Main Interface 2/2

- During gameplay, the interface adjusts itself to the correct state
- Here we can see the player has begun a game, they are currently typing the word "pneumonia", and they have made one mistake, which resulted in 1 life bar being taken away

Example Properties File 1/2

- Properties File for the word “and”
- This is before the player has used the program, the word has a 1% chance of being the next word the player must type
- The “ChanceLow” and “ChanceHigh” values are used by the program to determine what word was chosen when a random number is generated

Example Properties File 2/2

- After the game has seen some use, the word “and” has appeared a number of times and the player has had some difficulty typing the word. The player has had:
  - 5 successes
  - 3 failures
- Now the chance that the word “and” will appear has gone up significantly, from a 1.00% chance to occur to a 2.35% chance to occur. If the player continues to make this mistake, the word will become even more likely to appear, forcing the player to type this word more, in hopes that through repetition they will learn to type it quickly and accurately

Difficulties Encountered

- The database of linked words
  - I found it easier to make a property file for each word, which allows many statistics to be directly associated with the word
- Properly keeping the percentages balanced at all times
  - A rounding function helped to make the percentages evenly distributed
- Range value for determining the next word selected
  - Difficult to properly adjust since one word out of the list would be adjusted by a large number while the rest of the words would be adjusted by a small number

Achieving the Goal

- In order for this project to be a success, one should see an improvement in the players typing ability
- With faster typing abilities and less mistakes being made, the program is a success if the heavily repeated difficult words have become less of a problem for the user
- As seen on the previous graphs, the player should theoretically arrive with all words having the same percentage chance to occur, as no one word is more difficult for the player as the others
- The end result is that the player should get close to 100% accuracy and show signs of improved speed as well

What New Data is Available?

- Statistics gathered from the game, illustrate where the player has had the most problems and most success
  - Directly display what words were typed incorrectly the most and which words did the player have the least amount of trouble with
- This information is useful for both the player and the program, allowing the program to continuously improve the skills of the player, and the player to realize where they need to improve

Future Applications

- Monitoring the player this way can be used in many different software programs
- Games in particular can benefit from smarter Artificial Intelligence when the player is challenged by the computer
  - Fighting, strategy, RPG’s, puzzle, etc.
- Tracking the players successes and failures can be done in most games of today, and making the game react accordingly is a great way to increase the Artificial Intelligence of computer opponents
- Even applications like Microsoft word could use this tactic, keeping track of which words the user most often types, the auto-complete could fill in the rest of the word once it has been tailored to each individuals personal word selection

References

M. M. Richter – Machine Learning
Class notes, CPSC 599.66 Machine Learning, Winter 2007.

Wikipedia – The Most commonly used words in the English Language
http://en.wikipedia.org/wiki/Most_common_words_in_English

Wikipedia – The Most commonly misspelled words in English
http://en.wikipedia.org/wiki/Twenty_most_common_words_in_spoken_English

YourDictionary – 100 Most Often Misspelled Words
http://www.yourdictionary.com/library/misspelled.html

Image Based Classification and Facial Recognition Using Feed forward Neural Networks

Eric Leclerc

Domain of Application

Classification of individuals based on image data of the face.

Problem within the domain

Given a set of k pictures of an individual, can a feed forward neural network with backpropagation correctly identify a person from a previously unseen photograph, provided that the system has been trained with other photographs of the same person?

Learning Goal

The learning goal of the system is to identify a specific person with a high degree of accuracy, given only a finite set of training data. These test images may include variations in pose, facial expression, eye wear, facial hair, and lighting conditions. The system should ultimately be highly accurate in spite of these variations.

Conditions for Achievement of the Learning Goal

In formal terms, the achievement of the learning goal occurs when the total error (i.e. misclassification) of a given validation set of data when tested in parallel with a training set of data. The error is measured as $E_d(w) = \sum_{k \in output} \left(target_k - output_k \right)^2$ where E_d is the error at training example d, summed over the weights of all output neurons in the network. The target of each unit is defined by a supervisory function, and the output is the value represented at the output unit from the feed forward network. The backpropagation algorithm uses a gradient descent search to find a local minimum of error in the output of the system over a finite number of training iterations. In simple terms, the system finds a state where the number of erroneous classifications is minimized.

Data Available

There is a well established research community in the facial recognition field, and a variety of different methods to achieve the desired goal. There are also several different standardized repositories of suitable images for training and testing such a system, with specific lighting, framing, poses, and expressions. Many of these databases are intended for testing commercial level applications, and were therefore needlessly complex for the purposes of this project. There is also a wealth of data regarding artificial neural networks (ANNs) available. My project makes use of a neural network and pgm image package written in C, as well as a simple image database consisting of low resolution 32 * 30 pixel images, to reduce computation time, both from the companion website to Tom Mitchell's Machine Learning textbook, www.cs.cmu.edu/~tom/faces.htm.

Mode of Learning

This system uses supervised learning. For each test, there was a training set consisting of ~6 images of each individual in a forward-facing pose. From the training set, a target value for each output unit is encoded such that a *1 of n* output encoding is realized, using a substring from the filename to determine the proper output for the network to target. This allows the network to adjust the internal weights in a faster timeframe, in theory reaching the stopping condition much quicker than a similar system using unsupervised learning.

Specific Methods

As hinted at above, the specific method for the problem I used is a backpropagating neural network, using a sigmoid "squashing" function, and supervised training to attempt to optimize the hidden layer unit weights in a specified number of "epochs". After each epoch, several metrics are taken, to note the total error, efficiency of classification in the training set, and efficiency in the test set.

The network weights of each of the hidden layer units are updated using the perceptron rule, which revises the weight w_i associated with input x_i where:

$$w_i \leftarrow w_i + \hat{I}\,\ddot{w}_i$$

such that

$$\hat{I}\,\ddot{w} = \hat{I}\,(t - o)\,x_i$$

where η is the learning rate, t is the target value and o is the output value associated with input x_i.

The system also makes use of a term for *momentum* $0 < m < 1$, which helps to keep the system moving past small local minima that often occur with multilayer networks. The momentum term adds a fraction from the weight during the last iteration through the training cycle. After each iteration, the errors are compiled as mentioned above, and when the system reaches a sufficient point, the weight adjustments are changed very little.

The methods in my system which expand on the general method pertain specifically to facial recognition. These methods evaluate the performance of the system, and set up the target (supervised data) for the system.

Training data is obtained from the photo database I used, where filenames are placed in a list for use by the application. Training data consist of 5-7 pictures of 20 subjects, all facing directly into the camera, but having variations in eyewear and facial expression. The test data are 2 random pictures of each of the same 20 subjects, also with the subjects facing straight towards the camera.

When I set out to start the project, I intended on using a more complicated algorithm as suggested by Turk and Pentland ("Eigenfaces for Facial Recognition", also known as the Principal Component Analysis method). I decided on the simpler neural net method under the suggestion of Dr. Richter, as well as to accomodate time restrictions. Furthermore, the eigenface algorithm was not entirely clear to me, another reason to switch implementations.

The implementation of the project is as simple as possible, using a "BPNN" (Back Propagating Neural Network) library and .pgm image utility library. I updated it to the ANSI C99 standard, and implemented methods to encode the outputs of the neural net, as well as evaluating the performance of the system.

Difficulties

The major problem I have had in implementing this problem is a logic problem. In order to classify each output node as a correct or incorrect response, there are four distinct cases for each desired output from the system. I have been unable to think of a clean way to check those four cases for each of up to 20 responses (1 individual per output unit) given the framework I am currently using. I may use a 2d array to store the values for the encodings, which would require reworking most of the performance checking code.

Another issue that came about was a difficulty in understanding the initial algorithm I was going to use for the project. It seemed to be partially because of vague descriptions in the paper, and the description of the mathematics involved. Of course I have changed to a different method for the implementation, so this is irrelevant now.

The data sets I examined were all relatively easy to obtain. I looked at several image databases before settling on the simple image set for the project. The FERET database seems to be the industry standard database, it is quite large and contains quite high resolution photographs of the subjects. The Yale database has less total subjects, but the photos have widely differing lighting conditions and camera positions (some of the faces are quite far away, and some are close up). Again, it seemed to be geared towards an industrial application rather than an educational. The dataset I ended up using has nice small 32 * 30 pixel images, with fairly even lighting and face sizes. I found the set worked quite well, testing for one specific individual only took about 25 iterations to become highly accurate with both the training and test sets. The system seems to be able to cope with the subjects having different facial expressions quite well, and also recognizes subjects with and without sunglasses on.

Sources:

B. Moghaddam, A. Pentland, Probabilistic Visual Learning for Object Representation, IEEE Trans. on Pattern Analysis and Machine Intelligence, Vol. 19, No. 7, July 1997, pp. 696-710.

G. Shakhnarovich, B. Moghaddam, Face Recognition in Subspaces, Handbook of Face Recognition, Eds. Stan Z. Li and Anil K. Jain, Springer-Verlag, December 2004.

M. Turk and A. Pentland, Eigenfaces for Face Detection/Recognition, Journal of Cognitive Neuroscience, vol.3, no. 1, 1991. pp. 71-86

M. Turk and A. Pentland, Face Recognition Using Eigenfaces, MIT Vision and Modelling Group, 1991.

R. Gross, S. Baker, I. Matthews, T. Kanade, Face Recognition Across Pose and Illumination, Handbook of Face Recognition, Stan Z. Li and Anil K. Jain, ed., Springer-Verlag, June, 2004.

R. Gross, J. Shi, J. Cohn, Quo vadis Face Recognition? - The current state of the art in Face Recognition, Technical Report, Robotics Institute, Carnegie Mellon University, Pittsburgh, PA, USA, 2001

T. Mitchell, Machine Learning. McGraw-Hill, 1997. pp.81-126.

W.Y. Zhao, R. Chellappa, Image-based Face Recognition: Issues and Methods, Image Recognition and Classification , Ed. B. Javidi, M. Dekker, 2002, pp. 375-402.

Running the program.

To compile the main facetrain application, navigate to the bpnn directory and type "make". A few warnings will pop up since the libraries were written before the C99 standard came into effect.

To run the facetrain application, the following arguments are needed:

./facetrain -n <networkname> -t <trainingset> -1 <testset1> -2 <testset2>(optional) -e epochs

Where *networkname* is a neural network file of the form "*.net"
Trainingset is a list of photo filenames "*.list"
Testset is a list of test photos for the system *"*.list"*
Epochs is the total number of iterations through the network. Appropriate values range from 50 to 5000

Two other utilities exist in the project directory, *hidtopgm* and *outtopgm*, which map the hidden layer and output layer values into a grayscale .pgm image.

To compile these utilities type “gcc hidtopgm -lm -o hidtopgm” or “gcc outtopgm -lm -o outtopgm”

Both use the same arguments,
command <net-file> <image file> x y n
net-file : a neural network file, saved from running the facetrain application.
image file: filename for the output picture
x: in hidtopgm this is the x pixel value of the input image, and in outtopgm it is 1+ the number of hidden units in the network
y: in hidtopgm this is the y pixel value of the input image, and in outtopgm it is always 1
n: is the target unit to display weightings for.

Lord of the Rings

Landon McClocklin, Stephanie Tkachyk

One aspect of intelligence, be it artificial or natural, is the ability to learn. This concept, once applied, leads to the individual development of intelligence, or the ability to obtain and synthesize data. Learning to play strategy games is one way to further one's intelligence. After multiple repetitions, one adapts to the rules of the game and begins to synthesize the patterns that lead to a successful outcome. Improvement on these patterns increases the abilities within the game.

Strategy games hold several concerns when pertaining to artificial intelligence. Firstly, the strategies required and goals are adjusted continuously. What may be the ideal formula for success for one may provide an instant loss when regarding another instance. This is true not only for varying games, but for separate instances of identical games. Even the opponent must be taken into account to develop a successful strategy. Secondly, in order to survive multiple runs of a strategy game, the player must have the ability to evolve. If the player cannot adapt, eventually, the opponent will and the player will lose consistently. Finally, strategies cannot be imported from other sources. A specific strategy must be created for every instance of every game. Currently, the ability to create a general strategy specific enough to apply to a specific game does not exist. "Win" is not a sufficient formula for success.

A strategy game, a Lord of the Rings battle, has been modified for the purposes of this project. It involves placing good and evil warriors on a grid, and having them move and attack each other until one side is annihilated. It has a simple set of rules, however it requires strategy and understanding of the variability within these rules to win the game. Our project will ultimately begin with a system using random moves, and through measurement of the movement sets, will keep the successful moves and drop the moves which decrease the side's viability. Ideally, our system will develop a player able to develop a strategy which will win consistently.

This game requires two levels of consideration in order to develop a successful strategy. It requires a group mentality, so that the goals of the whole are protected. These goals are the survival of the team and the annihilation of the enemy. For this to be possible however, some semblance of warrior-level survival abilities are required. If only the group is important, the warrior goes in, intent on annihilating the other team, all will perish. If the survival levels for each warrior is more important than the needs of the team, the warriors will run from attack rather than facing the opponents.

To help maintain a balance between the importance of the group and the individual, a Genetic Algorithm was used. The basic algorithm was modified slightly to allow for the complexity of the game. A population of fields, with specific positions and moves for each warrior is maintained. For each move set, any position, move set pairs that match the current field are recorded. The move sets are taken and their fitness measured. The fitness is measured by the ability to succeed in this set of moves. The fittest move sets are kept, while the unfit move sets

are lost. If no fit individual is available, one is created using random moves. The algorithm is repeated until a win or a loss is generated.

Summary:

Learning Goal, Data Creation

- Our goal is to ultimately have the learning side be able to win the game every round, regardless of the positions of the warriors on either side.
- A round consists of one game, from the initial placement of the warriors on the field, to the defeat of one side by annihalation.
- Multiple rounds, with solid wins, would show that the ultimate goal was achieved.
- Our goal is to ultimately have the learning side be able to win the game every round, regardless of the positions of the warriors on either side.
- A round consists of one game, from the initial placement of the warriors on the field, to the defeat of one side by annihalation.
- Multiple rounds, with solid wins, would show that the ultimate goal was achieved.

Fitness Measurement

- When moves were found, the fitness function would measure the overall fitness based on the fitness of each warrior.
  - If the warrior can kill the enemy, fitness increases by 20 points
  - Else if the warrior can injure the enemy, fitness increases by 5 points
  - Else if the warrior moves within hit range of the enemy, fitness decreases by 2 points for each enemy.

Learning Method

- For a game, a moderately supervised learning approach was taken.
  - The results of the game had to be observed and evaluated
  - The specific moves are measured for fitness but are not guaranteed to be ideal.
- We selected genetic algorithms, using a pair of (Field configuration, move set) as our sets to search on.
- We removed crossovers from the algorithm for simplicity purposes.
- We used two types of mutate, mutateMove and mutateAttack
- One Field setup was designed to be able to win or lose based on the randomization.
- Initially, we had thought that a mutatePartial or crossover would not be difficult to implement.

Difficulties

- Since we were unable to implement searches for partial field matches, the data set required for intelligence grew exponentially.
- We had issues storing our data sets between rounds, limiting the amount of learning that could occur.
- Data was easy to obtain, but quality was always questionable.

Source Code

Source code for base project along with example simulation is taken with permission from Marc Schroeder.

Othello

Learning Strategies

Allan Lyons

April 12, 2007

1 Introduction

For this project, I implemented an agent that was intended to learn to figure out strategies on how to play a two person strategic board game. The emphasis was on the learning to play the game as opposed to actually playing it.

2 Learning versus Playing

For the purposes of this project, there is a distinction between learning to play a game and actually playing the game. In this case, learning refers to the situation where only the basic rules of the game, such as what constitutes a valid move, are given to the agent and strategies on how to apply those rules must be learnt. In contrast, if developing an expert player, all available strategy information is also encoded into the client. For example, one strategy that could be encoded for Othello is that there are distinct advantages to occupying certain squares.

My choice for a simple board game is Othello, a two-person strategic board game that only has small rule differences from Reversi. The rules of the game are simple, but the game takes a lifetime to master.

According to the standard rules of Othello, the game is played on an eight-by-eight grid with pieces that have two distinct colours on either side. The object of the game is to have the majority of the pieces on the board at the end of the game flipped to your colour. Valid moves consist of placing a piece on the board next to your opponent's piece such that a line can be drawn from the played piece, across one or more opponent's pieces, to another one of your pieces. When a piece is played in this manner, all of the encompassed opponent pieces are flipped to your colour. This leads to the board being very dynamic since it is possible to flip up to 20 opponent pieces in a single move. If a player has no valid moves, then that player passes. The game continues until neither player has a valid move remaining.

Due to the board being relatively dynamic, Othello has a fairly high branching factor and as a result is difficult to search exhaustively even considering the limited number of turns in a game.

3 Learning Goal and Methods

Learn to recognize board patterns in order to discriminate between "good moves" and "bad moves" in order to win the most number of games.

Since the purpose of this project is to demonstrate learning, no strategy information is encoded into the agent. As such, the learning goal is for the agent to learn to recognize board patterns indicative of good moves and bad moves. In this case, a good move would be one that led to a game being won whereas a bad move is not one that would contribute to a win. Ideally, this learning would be able to be used to rank potential moves so that the agent would be able to choose the move with the best expected outcome.

Since Othello is a two player, turn taking game, a standard mini-max search with alpha-beta pruning was implemented to search the game space a few plies ahead. A neural net was then used to score the board positions at the leaves of this tree. The immediate move that lead to the most advantageous board position while avoiding the opponent's best position would then be chosen.

Our goal was to discover strategies useful for playing Othello. Since the strategies were the goal, no additional information was encoded into the game. The only information that the game agent was provided was a way to determine the available valid moves from a given position as well as a way to determine the final score when the game was completed. Additional information about the game was purposely left out.

Theoretically, there exists an ideal function that would correctly predict the outcome of the game given a board position under the assumption of perfect play by both players. This ideal function would effectively encapsulate all useful strategies for playing the game and thus could be used by a player for picking his/her next move. One property of this ideal function, f, is that $f(s_t) = f(s_{t+1})$ where s is a board state and t is a particular time step in the game sequence. That is, the ideal function would give the same result for any time step in the game sequence.

Since the goal of the project was to see if it was possible for the machine to learn the strategies for playing Othello, it was necessary to use something that could approximate this function without actually knowing the function a priori. A neural network worked adequately as a function approximation for this purpose.

The network was implemented as a fully-connected feed-forward network with 64 inputs, one hidden layer of 55 nodes, one output node and bias nodes in the upper layers. Each square on the board was represented by exactly one input and was directly encoded by a 1 if the square was occupied by a black piece, a -1 if the square was white, and 0 if the square was empty. The activation function for all of the layers in the network was the symmetric sigmoid function $y = \tanh(sx) = \frac{2}{1+e^{-2sx}} - 1$ where x is the weighted sum of the inputs to the node and s is the "steepness" of the function. This results in the output of the net being in the range (-1, 1). The output was interpreted with positive numbers representing a win by the black player and negative numbers as a win by with with larger absolute values indicating more decisive results.

Temporal Difference Learning

In order to train this neural net, the reinforcement learning strategy of Temporal Difference Learning [6] was used to provide training targets for the neural net. Using temporal difference learning to guide the training of a neural net has successfully been used in the context of playing games such as Backgammon. [7]

Temporal difference learning is a reinforcement strategy that can be used for determining the training targets for the neural network. It can be implemented using the TD(λ) algorithm. With a risk of over-simplification, TD(0) can be thought of using the result of the following step as the training target for the neural net whereas TD(1) would use only the final result as the target for every step. An intermediate value of λ gives some result between these two extremes. In my code I implemented TD(0).

4 Tools Used

- Fast Artificial Neural Net Library (FANN) http://leenissen.dk/fann/
- Generic Game Server http://www.cs.ualberta.ca/~mburo/

Difficulties:

- Generic Game Server not well documented
- Training the net by playing games is very slow.

4.1 Generic Game Server

When starting to implement this project, I made the decision to employ the Generic Game Server as a framework for the game playing agent. The advantage of this approach is that the game rules are enforced by the server and a tournament director that can schedule matches between agents is available. The tournament director in particular was useful for training between players. Another advantage is that there are GUI clients available making it easy to play against the agent manually or to watch games in progress.

Also included with GGS is a game development kit written as a C++ library that is intended to ease the development of clients that connect to the GGS. This library handled all of the network communication with and parsing messages from the server. Also, in the case of Othello, it was able to generate all possible moves from a given board state and could properly handle updating the game board.

The main problem that I had with GGS is the lack of documentation that matched the current code. Setting up and configuring the server took consider-ably longer than it should have. Also, I didn't find any documentation that clearly explained the protocol and and command set that was to be used to communicate with the server.

Lack of updated documentation was also an issue with the game development kit. The documentation that was there was from an earlier version that only worked with Othello. Since then, this code has been modified and abstracted away from Othello and thus class names, etc. were different than suggested by the documentation. However, all of this code is freely available and it does seem that there are plans for it to be developed further. With a bit of documentation, it would be useful for turn-based game AI research.

One side effect of choosing to use this toolkit, was that it pretty much dictated that I would be developing in C++. If the code had been easier to follow, I may have been able to deduce the communications protocol and may have implemented with a different language.

4.2 Fast Artificial Neural Network

This actively developed C library seems to have a community using and supporting the code base. It is reasonably cross-platform and has bindings for most major languages. It can also be compiled on systems that lack a floating point unit in a form that only uses integer calculations.

There are a few things that this library is missing that would have been nice for this project. Currently, there is no provision for manually constructing a network and specifying all of the connections. Being able to manually construct all of the connections in the net would have been helpful in this instance due to the high degree of symmetry in Othello. This would have led to a network that more easily captured the information contained in the layout of the board. Although not part of the library now, this feature should be relatively easy to add.

The other feature that would have been nice to have would be to share weights between some other connections in the net. This may have been helpful due to the high degree of symmetry. Since the Othello board is symmetric with respect to both reflection and rotation, it would be good for the neural net to report the same result without regard to its orientation.

5 Implementation and Results

The intention was to implement a client that would learn by playing games against itself and other automated agents. Theoretically, this could lead to an unending series of games if at least one of the

players introduced some level of randomness in its move choices. The randomness in the move choices is necessary to cover the game space since Othello is otherwise completely deterministic. The GGS system facilitates this type of approach since it includes a tournament director for tracking and starting matches between players. Also, it provided for random starts to games where the server would randomly play the opening moves of the game before turning the game over to the players.

However, due to the overhead imposed by the GGS infrastructure as well as the increased computation for the searching during the games, a few changes were necessary. Rather than training solely on the choices made by the client, I found that I could train on the actual moves of the game. This is subtly different. Since I trained on the actual moves in the game, I was able to use recorded games I found on the Internet for training and so I was able to train on much larger samples several orders of magnitude faster. However, this was a compromise since the agent then ended up training on different data than it would have had it trained on games that it was actually playing.

Although it could play Othello, there are some changes that I would have liked to make to see how much they improved the play. My agents did learn to play competitively with another automated player that could search several moves ahead. My agent isn't a very accomplished player, and there are several changes I would have liked to try in order to improve its learning results.

Due to the limitations of the neural network library described above, I was unable to capture the board symmetry in the neural net as well as I would have like to. If you examine an Othello board, you will notice that the board is symmetric with respect to both rotation and reflection. That is, if the board is rotated or reflected on any axis, there is no impact on the game. Furthermore, the 64 squares on the board can be grouped into just 10 equivalence classes that are static with respect to the board symmetry. For example, the four corner squares form one equivalence class, the 8 squares that border the corners horizontally and vertically form a second equivalence class, and so on. What I would have liked to try would have been to divide the board into eight triangles of ten squares and then used these feature maps as input to the rest of the network. Note that the boundary squares are included in more than one triangle. Since each of these eight triangles are equivalent, they would be candidates for weight sharing on the connections. By sharing weights, a board and its rotation or reflection would be scored the same since they really are equivalent in the game of Othello.

The other feature that I would have liked to try is to recognize the different phases of the game. An Othello game can be divided into three phases of roughly equivalent length and each phase has a different strategy. I would have tried implementing these three phases by using three different neural nets with the output being selected from each net depending on the phase of the game. This would enable each net to learn the playing strategy appropriate for its particular stage of the game. A smoothing function could be used to gradually switch from one net to the other rather than having a sharp cut-off. Other than the change in the network, the rest of the program would be the same.

The problem of using a single neural net the way that I did was that it required the neural net to behave in three different and possibly conflicting ways depending on the phase of the game. It may be possible that a more complex neural net with additional hidden layers may be able to learn to detect the transition between the phases of the game. However, this may have the result of increasing the required training time.

6 *Bibliography*

[1] Jonathan Baxter, Andrew Tridgell, and Lex Weaver. TD-leaf(λ): Combining temporal difference learning with game-tree search.

[2] Michael Buro. Generic game server. http://www.cs.ucalberta.ca/~mburo.

[3] A. V. Leouski and P. E. Utgoff. What a neural network can learn about Othello. Technical report, Department of Computer Science, University of Massachusetts, Amherst, MA, USA, 1996.

[4] Steffen Nissen. Fast artificial neural network library. http://leenissen.dk/fann/

[5] Raul Rojas. Neural Networks: A Systematic Introduction. Springer-Verlag, 1996.

[6] Richard S. Sutton and Andrew G. Barto. Reinforcement Learning: An Introduction. MIT Press, Cambridge, MA, USA, 1998.

[7] Gerald Tesauro. Temporal difference learning and td-gammon. Commun. ACM, 38(3):58-68, 1995.

Part 2
Class 601.66

Report on Cooperation via Learning

CPSC 601.66
For Professor Richter
Tyson Kendon

Abstract

Cooperation via Learning is a project to illustrate how an agent that is not able to communicate directly with another agent can cooperate by watching the other agents' output.

Introduction

A group of agents cooperating as part of a distributed multi-agent problem optimization would normally exchange information to allow them to cooperate in an intelligent fashion, if this communication is not permitted the agents would normally have no way to communicate. Using clustering an agent should be able to inspect the solutions produced by the other participating agents and adapt its search control in the way it would have had that information been shared directly from the other agents.

Background

The Stakeholder Search Framework was developed to investigate semi-cooperative search where agents participating in a search are attempting to simultaneously maximize both a global goal (as in cooperative search) and a local goal (as in competitive search). Cooperation via Learning limits the information exchange between the participating agents to only complete solutions. Because no information is conveyed about what parts of the search space will be investigated is shared; an agent hoping to cooperate will have to observe the changes in solutions from other agents and learn what which parts of the search space are being investigated. An agent would then alter its search control to focus on areas being ignored by other agents and ignore areas where other agents are focusing.

Cooperating agents should go through less search states that are common among them. The quality of the learning can be evaluated by measuring the number of unique solutions seen by all three agents.

The Data

While the Stakeholder system is versatile in which types of problems it can be used to investigate, recent work has been with the Package Delivery Problem (PDP). In the Package Delivery Problem there is a set of packages which must be delivered to different locations. A set of drivers has to be assigned to deliver all of these packages such that every package has exactly 1 driver and the order in which that driver will deliver it. The system seeks to optimize a delivery plan that has the lowest total distance traveled by all drivers. (Since Stakeholder is being used in its cooperative mode, all the agents are trying to optimize this measure.)

Solutions to the PDP are formally described as a set of triples (package, driver, order) with package representing the identity of the package to be delivered, driver representing which of the drivers is to deliver it and order representing what order the specified driver will deliver the package in.

In clustering it would be ideal to measure the change in the triples over time to allow the learning agent to determine areas of the domain that are ignored by the other agents. For simplicity and clarity within clusters the clustering was reduced to measure only whether or not an agent changes the assignment for a particular package. For the data to cluster; the package was described as a set of attributes representing each possible change and each attributed was described over a domain of {0, 1}. This was represented in Attribute Relation Format File (ARFF) used by WEKA for clustering.

The Implementation and Construction

Currently there are three types of agents that have been written to work within the Stakeholder framework: a Genetic Algorithm, and Branch and Bound Tree and a Tabu Search. The learning method was tested using only the Genetic Algorithm because it was simpler to implement the behavioral change.

The clustering algorithm was implemented using the Weka data mining tool. Weka was selected because it had a Java API that allowed it to be used from within the Stakeholder system without reliance on external tools. Weka was also able to provide several clustering alternatives for evaluation. Unfortunately the Weka tool only handles input and output from file and as such the benefit of integrating it directly into code was somewhat lessened. Development was also slowed by spending time changing data to conform to Weka's ARFF file definition.

Hoping that the data about the packages would allow it to naturally fall into two clusters of changing and unchanging the data was originally clustered using a density based clustering. It was not possible to divide the data into clusters using density based clustering. The system would usually find one large cluster with possibly one data item outside that cluster. The lack of clusterability resulted from two main things, firstly while the designation in the change file implied that the data was binary, the Weka clusterer took it to be a real number. Secondly the size of the data was very small; normally the problems were designed with ten packages to limit the possible scope of the search space (with 3 drivers and 10 packages there are 30! possible solutions to the problem) and this results in there being very few data items to put in any

particular cluster. That is to say that given a large number of data items 10% might be an interesting cluster, but with a small data set 10% does not give us enough data change to use.

To improve the split provided by clustering the clustering method was changed to K-means clustering with K set to two. Forcing all data items into two clusters (generally with 3 – 5 data items) was able to provide sufficient data for the agent to adapt its pattern to.

Given the two clusters the agent would select the cluster with the lower normalized number of changes. The genetic algorithm agent, when not given a cluster to focus on, preformed mutation and crossover with an equal probability for each package. When given a focus cluster the agent changes its behavior: the mutation was changed to reduce the probability of mutation of packages outside the focus cluster and to increase the probability of mutation of the packages in the focus cluster. Crossover was adapted to select a cut point that divided the focus cluster in half each time.

Evaluation

To measure the benefit of cooperation via learning the calculation of search space coverage was used. For each group of participating agents the set of unique solutions seen by all agents is collected. The coverage is then calculated as the size of that set. The improvement to the search using cooperation via learning can be seen by an increase in the coverage count, because each agent will repeat less of the search states seen by other agents.

Unfortunately due to time constraints and an implementation error it was not possible to get results measuring the effectiveness of cooperation via learning for.

Problems and Remarks

The predominant problem developing cooperation via learning was the difficultly of adapting a pre-existing system to use a machine learning system. As is often the case in adapting two pieces of software together a lot of time was spent arranging information for the Weka clusterer to use and then reintegrating that information into the Stakeholder system. It may have proved more convenient had Weka allowed for an online information exchange; however exchanging information via files is acceptable if somewhat more cumbersome.

Originally the measure of evaluation was to look at the quality of the solutions compared to some optimal solution, however with the three driver, ten package problem sets it is not possible to determine that optimal set. Therefore the measure of evaluation had to be adapted to the coverage scheme.

In the future it is hoped that the cooperation via learning strategy can be expanded to a system to model agents in a semi-cooperative environment.

References

Denzinger, J.: Knowledge-Based Distributed Search Using Teamwork Proc. ICMAS-85, San Francisco, 1995, pp. 81-88.

Denzinger, J.: Distributed Knowledge-based Search 1999.

Denzinger, J. ; Kronenburg, M.: Planning for distributed theorem proving: The teamwork approach, Proc. KI-96, Dresden, Springer LNAI 1137, 1996, pp. 43-56.

Denzinger, J.; Offermann, T.: On Cooperation between Evolutionary Algorithms and other Search Paradigms Proc. CEC-99, Washington, IEEE Press, 1999, pp. 2317-2324.

Fedoruk, A. ; Denzinger, J.: *A General Framework for Multi-agent Search with Individual and Global Goals: Stakeholder Search*, International Transactions on Systems Science and Applications (ITSSA), Vol 1(4), 2006, pp. 357-362.

Ian H. Witten and Eibe Frank (2005) "Data Mining: Practical machine learning tools and techniques", 2nd Edition, Morgan Kaufmann, San Francisco, 2005.

Discovering Communities of Mobile Users on Call Detail Records

© Keivan Kianmehr
Department of Computer Science
University of Calgary
Calgary, Alberta, CANADA
kiamehr@cpsc.ucalgary.ca

1. Introduction

In telecommunication, a Call Detail Record (CDR) is a record containing information about recent system usage, such as the identities of sources (points of origin), the identities of destinations (endpoints), the duration of each call, the amount billed for each call, the total usage time in the billing period, the total free time remaining in the billing period, and the running total charged during the billing period. The format of the CDR varies among telecom providers and call-logging software [1].

In recent years, we witness dramatic increase in the competition among telecommunication companies in order to detain their current customers and acquire new ones. For this reason, the ability to dynamically classify and predict customers' behaviors according to their calling patterns obtained from CDR data has attracted considerable in the research community; it is beneficial for [2]:

- Churn Prediction: The goal is to understand when and why company's customers are likely to leave so that appropriate action can be planned. Customers become "churners" when they discontinue their subscription and move their business to a competitor company. This has been developed in the telecommunication industry using data mining techniques. Data mining is applied in this area to perform two major tasks: a) predict whether a particular customer will churn and when it will happen; and b) understand why particular customers churn. By predicting which customers are likely to churn, the telecommunication company can reduce the rate of churn by offering the customers new incentives to stay.

- Identifying Calling Communities: Identifying possible calling communities can be used for determining a particular customer's value according to the general pattern behavior of the community that the particular customer belongs to. This helps the effective targeted marketing design which is significantly important for increasing profitability in the telecommunication industry.

The main focus of this project is to use an unsupervised machine learning technique, namely clustering to classify customers of a mobile service provider into appropriate calling communities according to the statistics extracted from the CDR data. Clustering technique is one

of the most prominent approaches for identifying unknown classes amongst a group of objects, and has been used as a tool in many fields such as biology, image analysis, and finance. The classification algorithms evaluated in this project use an unsupervised learning mechanism, wherein unlabeled training data is grouped based on similarity. Once an acceptable clustering has been found using the similarities and dissimilarities in the training data set, the clustering is transformed into a classifier by using a classification technique. In this approach, the clusters are labelled and a new object is classified with the label of the cluster which it is most similar to. However, the type of the classifier that is to be used to identify calling communities and customers' value is very crucial. For instance, in a marketing campaign, it is very costly to select non-appropriate customers to verify the impact of the marketing campaign in order to modify the strategy toward the target group. The fuzzy classification is one possible approach that offers more convenience for selecting customer subgroups and for measuring the efficiency and validity of the communities regarding the classification goal [3]; for example, the marketing campaign design. The flexibility of fuzzy approach featured by the application of membership functions provides the ability to increase or decrease the homogeneity between the targeted customers depending on whether the proposed products are very specific or intended for a large community.

In this project, the agglomerative hierarchical clustering approach has been applied for clustering task. It can produce an ordering of the objects (cluster tree), which may be more informative for the nature of the data being used here. For classification task, two different techniques have been selected. The support vector machine (SVM) as a statistical-based learning approach has been used to build the classifier model. A fuzzy-genetic algorithm has been also applied for the classification task. The SVM algorithm has been used for identifying crisp user communities and fuzzy-genetic algorithm has been adapted for assigning a particular customer to possibly more than one community with different degree of memberships.

2. CDR Data and Calling Neighbors

Because of the confidential nature of the customers' information and also privacy preserving, the information provided by telecommunication companies is very limited for modelling except the CDR data. From the CDR data, it is possible to extract customer's calling destination numbers, duration and frequency for each destination number for a particular period of time. Customer's calling neighbors can be also identified by using the links of who calls whom [2]. There are two types of calling neighbors:

1. Direct calling neighbour: A person who calls the customer or whom the customer calls. The majority of these neighbors of a customer may be outside of the service provider's own network, and no information about them is available. An example of direct calling neighbour can be described as follows: members of a family calling plan call each other heavily.

2. Indirect calling neighbor: A person who calls the same numbers as another customer does. For example, employees of a large organization such as a bank who work in different local branches may call their headquarter branch frequently, and each employee is possibly a calling neighbor of other employees. Even though employees of different

branches may not know each other, they can be classified into the community of colleagues.

To measure the closeness of the first type of calling neighbors, the calling frequency or duration has been used. For the second type of neighbors, a distance measure, which will be presented in more detail in section 3.2, has been applied to quantify the closeness of a neighbour. Discovering calling neighbors will result in forming calling communities. However, there are two major challenges in using the CDR data for finding the calling communities:

- The majority of the destination phone numbers are outside of the service provider's network. Therefore, information about these customers such as their calling records and links between them is not available.

- The other types of phone numbers are those in the service provider's network, and each of them corresponds to a customer of the service provider. The connectivity between these customers is very sparse because customers more likely call numbers outside their home network.

After identifying the customers' communities, the information derived from the calling communities can be used for building a classifier model. This classifier model is able to assign a new customer to one or possibly more than one existing community according to his/her general calling pattern extracted from the CDR data and closeness of his/her direct and indirect calling neighbors. In terms of market campaign design, the characteristics of communities can be used for increasing the profitability of the service provider company by targeting appropriate communities and customers. Intuitively, people in the same community have more likely similar behaviours because of their influences on the community. For instance, if a customer more frequently calls member of a community in which members generally accept service promotion offers, that particular customer can be considered for targeted service promotions.

3. Model Details

In this section, the classifier model that has been developed for the course of this project will be described in more detail. The system consists of three major phases: 1) Data Pre-processing, 2) Clustering, and 3) Classification.

3.1. Data Preprocessing

The CDR data used in this project is from an Iranian wireless service provider. Because of the confidential nature of the data and business rules, the CDR data for a short period of time is given. However, the results show that the classifier model built based on the information extracted from calling links within the given CDR data perform classification at a reasonable accuracy.

At the first step of the data preprocessing, all the phone numbers (customers) within the service provider's network have been identified. Since the CDR data includes information about the customers, all the source phone numbers have been considered as the subscribers of the service

provider. The given data set consists of 55000 calling records of 2000 subscribers (distinct phone numbers within the service provider's network). Calls with very low duration
(less than 5 seconds) are assumed to have no effect on identifying the subscriber's neighbors and are ignored. At the second step, for each subscriber, his/her own phone number, called destination numbers (within and outside of the home network), and each destination number's calling duration and frequency in that particular period of time are extracted from the CDR data. During pre-processing of the CDR data, inactive customers have been excluded from the data since these numbers greatly skewed the distance distribution. Inactive customer is referred to a customer who barely makes a phone call within a particular time period.

3.2. Clustering

3.2.1. Distance Measures

In order to identify the closeness of a particular customer to his/her direct calling neighbors, a similarity measure weighted by call duration or frequency between two phone numbers has been used. The weighted similarity measure is a first order distance [2] defined as follows:

$$D_1(i,j) = \frac{1}{w_{i,j}} \tag{1}$$

, where $w_{i,j}$ is the call duration or frequency between customers (or phone numbers) *i* and *j*. As described earlier, the direct calling neighbors of customers within the service provider's network are very sparse because most of the phone numbers are outside the service provider's network. Therefore, it is needed to include another distance measure, namely the second order distance [2] which reveals relationships between customers according to their indirect calling patterns where two customers have common direct calling neighbors. In order to define the second order distance between a pair of customers *i* and *j*, the set of phone numbers customers *i* and *j* called, respectively, in the time period, are weighted by calling duration or frequency:

$$N_w(i) = \{w_i(1), w_i(2), \ldots, w_i(n_i)\} \tag{2}$$

, where n_i is the total number of distinct phone numbers customer *i* called during the time period. Here $\sum_{k=1}^{n_i} w_i(k) = 1$, where $w_i(k)$ is the normalized value of calling duration or frequency. Finally, the second order distance between two customers *i* and *j* is defined as follows:

$$D_2(i,j) = 1 - \frac{1}{2} \sum_{x \in N(i) \cap N(j)} w_i\left(k_i(x)\right) + w_j\left(k_j(x)\right) \tag{3}$$

, where *x* is a common phone number which both customers *i* and *j* called in the time period, and $k_i(x)$ and $k_j(x)$ are the corresponding weights of common called phone number *x* in $N_w(i)$ and $N_w(j)$, respectively. According to Equation 3, two customers are considered very close when they call some common numbers frequently (or heavily) regardless of how many other numbers they regularly call. The distance between a pair of customers is 1 (maximum possible distance) if they do not call any common phone number.

3.2.2. Clustering Technique

Using the first and second order distance measures, a hierarchical clustering approach which works based on links between customers, can be applied to identify communities. In order to incorporate both the first and second order distance measures into the similarity measure of the clustering algorithm, the following distance measure has been defined:

$$D(i,j) = (1-\alpha)D_1(i,j) + \alpha D_2(i,j) \tag{4}$$

The term α controls the degree of importance of the first and second order distance measures and typically depends on the network characteristics of the mobile service provider. As $\alpha \rightarrow 0$, the similarity measure approaches the first order distance measure. Intuitively, α captures the customers indirect calling neighbors. Equation 4 makes it possible for clustering algorithm to merge clusters with the most number of links, which are defined as the common neighbors of the customers based on both direct and indirect calling patterns. That is, the clustering algorithm identifies communities the members of which have short first order distance but large second order distance between each other, as well as communities whose members have large first order distance but short second order distance between each other.

For discovering calling communities in this project, the agglomerative hierarchical clustering algorithm has been used. It is a bottom up clustering approach that investigates grouping in the given data by creating a cluster tree according to a particular distance measure [4]. The tree is a multi-level hierarchy, where clusters at one level are grouped together as clusters at the next higher level. This allows the user of the system to decide what level or scale of clustering is most appropriate for customers of a mobile service provider. The MATLAB Statistics Toolbox has been used for conducting the hierarchical clustering in this project. The basic procedure to perform hierarchical clustering is as follows:

1. Find the similarity or dissimilarity between every pair of objects in the data set.
2. Group the objects into a binary, hierarchical cluster tree.
3. Determine where to divide the hierarchical tree into clusters.

3.3. Classification

3.3.1. Distance Features

In this project, the information extracted from calling links and clusters have been used to build a classifier model to predict communities that a particular customer belongs to. The following input features have been constructed for each customer from his/her calling neighbors based on the first order and second order distances. The features are defined as follows:

- Total number of customer's direct calling neighbors.
- Percentage of a customer's calls made to her/his closest direct calling neighbor.
- Percentage of a customer's direct calling neighbors which are inside of the service provider's network.

- Percentage of a customer's direct calling neighbors which are outside of the service provider's network.
- The shortest distances of a customer to all existing classes.
- Percentage of direct calls to neighbors (within the network) belonging to all existing classes.
- Percentage of indirect call neighbors (within the network) belonging to all existing classes.

For every customer within the service provider network, a feature vector based on the above feature definitions is built. This feature vector represents relationships between this particular customer and all other customers within the existing communities. Then a set of feature vectors, each of which corresponds to a specific customer, is used as the training set for the classification algorithm.

3.3.2. Classification Technique

For building a classifier model in this project, two different approaches have been used. The first approach is Support Vector Machine (SVM) from the family of statistical-based learning algorithms. The main motivation of choosing SVM is that for many real world classification problems, it has been found to be more effective and faster than other machine learning methods such as neural networks. However, SVM suffers from understandability and interpretability. The second approach that has been applied for classification task is a fuzzy genetic rule-based algorithm. In contrast to SVM, this rule-based approach is understandable but suffers from efficiency issues. In this project, the characteristics of the fuzzy-rule based approach has been used for the possibly of doing fuzzy classification.

For conducting the classification based on SVM, a MATLAB interface of LIBSVM [5] has been used. LIBSVM is a free library for SVM classification and regression. The fuzzy-genetic rule-based algorithm has been implemented in MATLAB. The MATLAB genetic algorithm built-in functions have been integrated into the implementation. The rest of this section presents the fuzzy-genetic rule-based approach in more details.

Fuzzy-Genetic Rule-Based Systems [6]: In order to build a fuzzy rule-based system, the major task is to find an appropriate fuzzy rule set which represents the problem. Genetic algorithms have shown to be a powerful tool for performing generation and optimization of fuzzy rule-base and generation and tuning of membership functions [7]. In a fuzzy rule-based system, fuzzy if-then rules for an n-dimensional pattern classification problem are defined as follows:

$$\text{Rule } R_j : \text{If, } x_1 \text{ is } A_{j1} \text{ and } \ldots \text{ and } x_n \text{ is } A_{jn} \text{ then Class } C_j \text{ with } CF_j \tag{5}$$

, where R_j is the label of the j-th fuzzy if-then rule, j indexes the number of rules, $x = (x_1, x_2, \ldots, x_n)$ is an n-dimensional pattern vector, A_{ij} is an antecedent fuzzy set with linguistic label (i.e., a linguistic value such as small or large) on the i-th axis, C_j is a consequent class, and CF_j is a certainty grade. In this project, all attribute values are normalized into real numbers in the unit interval [0, 1]. As the antecedent fuzzy sets A_{ij}'s, five linguistic values shown in Figure 1 and

"don't care" have been used. Therefore, the number of combinations of the antecedent fuzzy sets is 6^n which is very large in the case of high-dimensional problems.

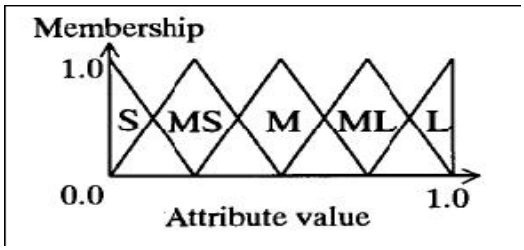


Figure 1: Membership functions of five linguistic values (S: small, MS: medium small, M: medium, ML: medium large, and L: large).

As shown in figure 1, the meaning of each linguistic value is specified by a triangular membership function on the unit interval [0, 1]. *"don't care"* has been handled by a special linguistic value with the following membership function:

$$\mu_{don't\ care}(x) = \begin{cases} 1 & 0 \le x \le 1, \\ 0 & \text{otherwise.} \end{cases} \tag{6}$$

In this project, a small number of fuzzy if-then rules has been randomly generated. When antecedent fuzzy sets of a fuzzy if-then rule are specified, its consequent class and certainty grade are determined by applying a heuristic [4]. After generating a small initial fuzzy if-then rules, the genetic algorithm has been applied to optimize the initial rule set so that it will able to classify the test set with a reasonable classification accuracy. This project does not involve the adjustment of membership functions or certainty grade.

Suppose the fuzzy if-then rule R_j in Equation 5 is denoted by its n antecedent fuzzy sets as $R_j = A_{j1} \ldots A_{jn}$. That is, R_j is coded as a string (chromosome) of length n. Let S be a set of N fuzzy if-then rules (i.e., $S = \{R_1, \ldots, R_N\}$). S is denoted by a concatenated string of the length $n \times N$, where each substring of length n corresponds to a single fuzzy if-then rule. That is, the rule set S is formulated as:

$$R_j = A_{11} \ldots A_{1n} A_{21} \ldots A_{2n} \ldots A_{N1} \ldots A_{Nn}. \tag{7}$$

Figure 2 shows a chromosome representing a fuzzy rule-based set containing k rules related to a problem with four features, X_1, X_2, X_3, and X_4. Each rule is represented by four genes indicating the index of the fuzzy set associated to the four features, respectively. The *"don't care"* value is represented by 0 within the chromosome.

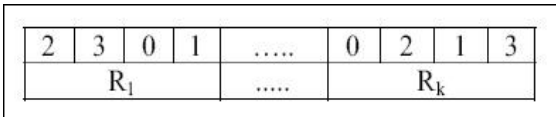


Figure 2: Chromosome representing a fuzzy rule base with k rules.

The fitness of the rule set *S* is measured as:

$$fitness(S) = NCP(S) \tag{8}$$

, where *NCP(S)* is the number of correctly classified training patterns by *S*.

The genetic algorithm has been set to use the uniform crossover, where each substring is handled as a block. That is, some rules are exchanged between the two parents by the crossover. A mutation operation has been set to randomly replace an antecedent fuzzy set of a rule with another one.

4. Evaluation Criteria

The following sections discuss the methods that have been applied to verify the cluster formation and to evaluate the classifier model.

4.1. Evaluating Cluster Formation

After linking the customers of the service provider network into a cluster tree, it has to be verified that the tree represents significant similarity groupings. The MATLAB Statistics Toolbox provides functions to perform cluster evaluation task [4] as described in the rest of this section.

In order to measure the validity of the cluster information generated by the linkage function, it is compared with the original proximity data generated by the distance function. If the clustering is valid, the linking of objects in the cluster tree should have a strong correlation with the distances between objects in the distance vector. The MATLAB built-in *"cophenet"* function compares these two sets of values and computes their correlation, returning a value called the *"cophenetic"* correlation coefficient. The closer the value of the *"cophenetic"* correlation coefficient to 1, the better is the clustering solution.

The approach that has been used to determine the natural cluster divisions is to compare the height of each link in the generated cluster tree with the heights of neighboring links below it in the tree. If a link has approximately the same height as neighboring links, it indicates that there are similarities between the objects joined at this level of the hierarchy. These links are said to exhibit a high level of consistency. If the height of a link differs from neighboring links, it indicates that there are dissimilarities between the objects at this level in the cluster tree. This link is considered to be inconsistent with the links around it. In cluster analysis, inconsistent links can indicate the border of a natural division in a data set. The MATLAB built-in cluster function uses a measure of inconsistency to determine where to divide a data set into clusters. This measure can be set by the user.

4.2. Evaluating Classifier Model

In this project, the cross validation method [9] has been used for evaluating the accuracy of the classifier model. Basically, the data is randomly divided into 5 disjoint groups. The first group is set aside for testing and the other four are put together for model building. The model built on the 80% group is then used to predict the group that was set aside. This process is repeated a total of five times as each group in turn is set aside. Finally, a model is built using all the data. The mean of the five independent error rate predictions is used as the error rate for the final model. The cross validation has been implemented in MATLAB for the course of this project. In the context of time complexity, the running time of both SVM and fuzzy-genetic classification algorithms have been measured and compared.

5. Experimental Results

Using the CDR data, a cluster tree has been built. The validity of the cluster tree has been evaluated using MATLAB built-in cophenet function. For the customer classification problem, the number of clusters produced by the clustering algorithm is an important consideration because minimizing the number of clusters is cost effective during the classification stage. However, the overall accuracy of the classification is also important. Therefore, the overall effectiveness of the clustering algorithm is calculated using overall accuracy of the classifier model. This overall accuracy measurement determines how well the clustering algorithm is able to create communities that contain customers with similar behaviors. The number of correctly classified customers in a cluster is referred to as the True Positives (TP). Customers that are not correctly classified are considered False Positives (FP). The overall accuracy is thus calculated as follows:

$$overall\ accuracy = \frac{\sum TP \text{ for all classes}}{\text{total number of customers}} \tag{9}$$

Figures 3 and 4 show how the clustering algorithm was evaluated with K being the number of clusters created from the cluster tree. The minimum, maximum, and average results for the clustering algorithm are shown as well. The classification algorithm used for evaluating clusters is SVM. The reason that fuzzy-genetic classifier has not been used is because of its running time. Considering the above analysis it has been decided to set the number of clusters to 10 for building the classification model. That is, every cluster has been labeled as a community, and customers within all the 10 communities have been used as the training set for the classification algorithm.

The examination of the overall accuracy between the SVM classifier and the fuzzy-genetic approach using 5-fold cross validation can be seen in Table 1. The LIBSVM default parameter settings have been used for running the SVM algorithm. Based on some initial test runs, the following setting have been applied for running the fuzzy-genetic classifier:

1. The number of fuzzy rules: 40, 60 or 80,
2. The number of rule sets: 20,
3. Crossover probabilities: 0.9,
4. Mutation probabilities: 0.1,

5. Stopping condition: 500 generations.

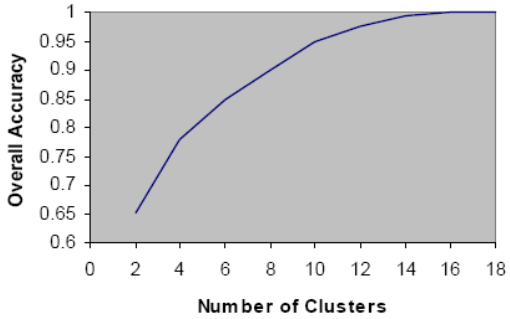


Figure 3: Accurcay using hierarchical clustering with SVM.

Data Set	Average	Minimum	Maximum
Accuracy	89.97%	65%	100%
Number of Clusters	9	2	18

Figure 4: Accurcay using SVM.

For the given CDR data, SVM has an average overall accuracy of 98.5% whereas in comparison, the fuzzy-genetic classifier has an overall accuracy of 82.5%. Thus, we find that SVM outperforms the fuzzy-genetic classifier by 13%. This shows that using genetic algorithm for tuning the fuzzy rules of the fuzzy-genetic classifier results in a reasonable accuracy but not as high as the accuracy obtained by SVM. However, the rules in the fuzzy-genetic classifier are easily understandable and interpretable.

Algorithm	**Average**	**Minimum**	**Maximum**
SVM	**99.75%**	**97.95%**	**100%**
Fuzzy-Genetic	**86.4%**	**75.25%**	**94%**

Table 1: Overall accuracy of each algorithm

Data set	**Number of rules**	**Accuracy**	**CPU time**
CDR Data	**40**	**86.4%**	**448 min**
	60	**80.80%**	**365 min**
	80	**73.15%**	**378 min**

Table 2: Running time of fuzzy-genetic classification algorithm

The runtime of both approaches is an important consideration because the model building phase is computationally time consuming. For the analysis, all operations are performed on a Dell Optiplex 745 with an Intel Core2 Duo 6600 @ 2.4 GHz processor and 3 GB of RAM. The number of data objects in the training set is 2000. In general, the runtime for the SVM classifier was significantly less than the fuzzy-genetic algorithm when building the classification model. For example, with 2000 objects running 5-fold classification took less than a second whereas fuzzy-genetic took much longer to build the classification model. The running time of the fuzzy-

genetic approach while varying the number of fuzzy rules is shown in table 2. Although the SVM classifier was faster, the size of the training set is ultimately limited by the amount of memory because both approaches must load the entire training set into memory before building the model.

Recall that, the fuzzy classification is one possible approach that offers more convenience for selecting customer subgroups and for measuring the efficiency and validity of the communities regarding the classification goal, like the marketing campaign design. The classes identified by SVM are crisp and the classifier model does not provide the ability for assigning a particular customer to more than one class. However, the characteristics of the fuzzy-genetic classifier model which contains a set of fuzzy if-then rules can be used for possibly doing fuzzy classification. The procedure applied for assigning a customer to more than one class with different degree of membership (i.e., fuzzy classification) is as follows:

The fuzzy if-then rules in the final classifier model are divided into n subsets, where n is the number of distinct available classes (communities), for instance if the clustering has identified 5 communities, then there exist 5 subsets. For a particular customer, every subset is examined to see whether there is any compatible rule with the feature vector representing the customer calling pattern. If exists, such a rule is able to assign the particular customer to the class of the subset under test. The certainty factor of the compatible rule will then represent the degree of membership of this customer to its identified class (community). If the algorithm finds more than one compatible rule with the customer feature vector from the same subset, then the average certainty factor of all compatible rules is considered to be the customer's degree of membership to this class. The same procedure will be repeated for a particular customer using all the subsets and a user may be assigned to different classes with different degrees of membership. The class in which the customer has the highest degree of membership will be identified as the main class. Meanwhile, this particular customer may belong to other existing classes with lower degree of membership. A customer does not belong to a particular class if its degree of membership to that class is zero.

6. Conclusion

In conclusion, this project demonstrated that information derived from customer's calling neighbors can be used to identify mobile users' communities. This is especially important for targeted marketing campaigns since the CDR data is often the only primary data source available for the customers. Based on the assumption that customers in the same calling community might behave similarly, targeted efforts can be focused on certain communities. Further, by identifying the community leader, marketing efforts can be directed to the community leaders since they are believed to have influence on other community members' behavior. The fuzzy classification proposed in this project provides more convenience for selecting customer subgroups and for measuring the efficiency and validity of the communities regarding the marketing campaign design. The flexibility of fuzzy approach featured by the application of membership functions provides the ability to increase or decrease the homogeneity between the targeted customers depending on whether the proposed products are very specific or intended for a large community.

References

[1] URL: [http://en.wikipedia.org/wiki/Call detail record], Last Accessed on 9/4/2007.

[2] L. Yan, M. Fassino, and P. Baldasare: "Predicting customer behavior via calling links", *Proceedings. 2005 IEEE International Joint Conference on Neural Networks*, IJCNN '05, Page(s): 2555- 2560 vol. 4, 2005.

[3] N. Werro, H. Stormer, and Andreas Meier: "A Hierarchical Fuzzy Classification of Online Customers", *Proceedings of the IEEE International Conference on e-Business Engineering*, ICEBE'06, Shanghai, China, October 2006.

[4] URL: [http://http://www.mathworks.com/], Last Accessed on 9/4/2007.

[5] C.C. Chang and C.J. Lin: "LIBSVM: A Library for Support Vector Machines", URL: [http://www.csie.ntu.edu.tw/_cjlin/libsvm], 2001, Last Accessed on 16/7/2006.

[6] H. Ishibuchi, T. Nakashima, and T. Kuroda, "A hybrid fuzzy genetics-based machine learning algorithm: Hybridization of Michigan approach and Pittsburgh approach", *Proceedings of IEEE International Conference on Systems, Man and Cybernetics*, vol. 1, pp. 296-301, Tokyo, Japan, October 12-15, 1999.

[7] O. Cordn, F. Herrera, F. Gomide, F. Hoffmann, and L. Magdalena: "Ten Years of Genetic Fuzzy Systems: Current Framework and New Trends" , *Proceedings of Joint 9th IFSA World Congress and 20th NAFIPS International Conference*, pp.1241-1246, Vancouver, Canada, 2001.

[8] H. Ishibuchi, K. Nozaki, and H. Tanaka: "Distributed Representation of Fuzzy Rules and Its Application to Pattern Classification", Fuzzy Sets and Systems, Vol.52, No.1, pp.21-32, 1992.

[9] H. A. Edelstein: "Introduction to Data Mining and Knowledge Discovery", (3rd ed.), Potomac, MD: Two Crows Corp, 1999.

Learning coordination of a multi-agent system For competition in the ARES system

By Jordan Kidney

Table of Contents

Table of Figures

List of Data Tables

Introduction

The main goal of this project is the creation of a system that learns how to coordinate a team of agents to compete against another team of agents in the ARES system. The ARES system, or Agent Rescue Emergency Simulator, simulates a basic rescue scenario in an urban disaster zone. The goal of a team of agents is to save as many survivors within the simulation as then can within a specific number of rounds. When multiple different teams of agents are in the simulation there is the potential for the teams to work together or to compete against each other. Competition is done by trying to either gain more points then other teams or to prevent the other teams form getting points in some fashion. The learning component was realized with the use of a genetic algorithm that produces different strategies to be used by the team of agents. The following sections will go over motivations and descriptions of key components to the project.

The problem domain

The problem domain, or learning task, is to find a strategy that can be used by a group of agents to compete against another team of agents within the ARES system. This is realized by having the learning component find a strategy that minimizes the total number of points gained by the other team in the simulation. The task of manually creating such a strategy would be difficult or very time consuming, just imaging having to think and order all actions over a sequence of time such that it minimizes the total number of points another tem gets. The application of machine learning to this problem creates a situation where a series of steps can be taken to learn and develop a complex strategy that would be difficult to do manually otherwise.

The targeted team in this case is is a project submitted by students for an introductory class to multi-agents systems at the University of Calgary. Their project team produced the best results in the end test simulations for the class, thus becoming a good candidate to test for the reaction they took to a highly comparative team of agents, as described above.

The ARES system

The Agent Rescue Emergency Simulator (ARES) system (see [1],[2],[3],[4],[5]) is a simulation program that can be used as a teaching tool for demonstrating concepts from multi-agent systems. The environment simulated within the ARES system is based upon a city that has been

hit by an earthquake. This is very similar to Robocup Rescue as already stated, but the objects within the world and the actions that can be taken have been simplified down to what we consider to be the basic features relevant for multi-agent systems. This simplification centers a round the interactions of the agents. An example of this is a rubble object; rubble represents material that must be moved out of the way. To remove rubble, a predetermined amount of agents must work together to remove the object during the same round in the simulation. This forces agents to coordinate and come together to accomplish the task.

Depending upon the configuration of the system, the agents will either have the ability to work with all the agents currently connected to the system or only with agents that have been declared as part of their group. This allows for the creation of environmental rules that can push the interaction between teams of agents from a cooperative scenario to a more competitive environment. Agents within the system interact within a two-dimensional grid environment built up of squares (See Figure 1).

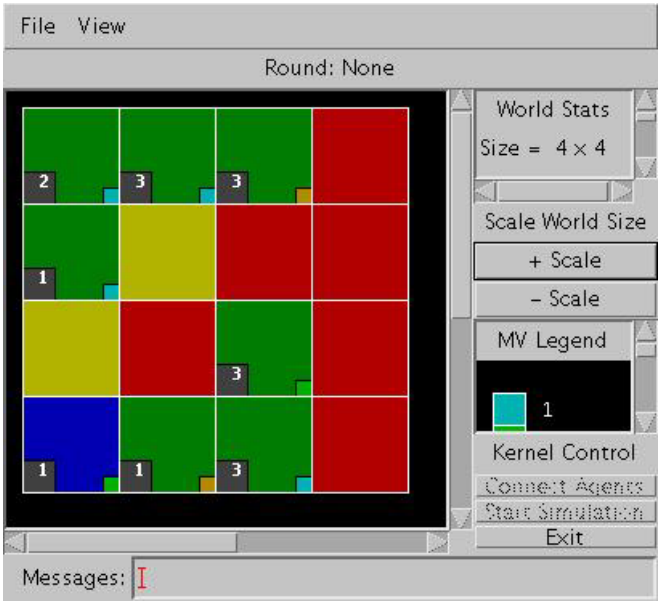


Figure 1 Basic screen shot of the environment in ARES

Each square in the grid represents a single point where an agent can influence the environment directly (See Figure 2). The agents jump from square to square as they move throughout the world. Each square in the world has the following properties:

1. A square can be classified as one of the following types (1) Normal - agents can safely move onto the square. (2) Fire - the entire square is on fire. (3) Killer – represents a zone where agents will die (like holes) and finally (4) Charging - a zone that can be used to regain lost energy.
2. A square consists of a stack of layers (built up of material), where each layer holds only a single object. Currently there are only three types of objects in the ARES system: (1) Rubble - represents material that has to be removed by the agents. Each rubble object has an associated value that indicates the number of agents that are needed to remove this

object (this is represented by the number inside the grey box). This value forces agents to coordinate and come together upon the same square in the world at the same time to remove the object. (2) Survivor - a single person that can be saved in the world. When they are saved, the survivors are simply beamed to safety. (3) Survivor Group - this is the same as a survivor; it just allows for multiple survivors to be located in a single layer on the stack.

3. Each square also has an associated move cost value; this value indicates the cost in energy for moving onto the square from any direction.

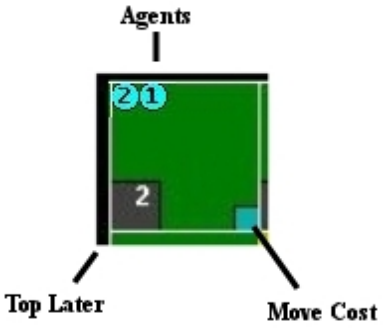


Figure 2 Basic Square in ARES world explained

The Learning System

System overview

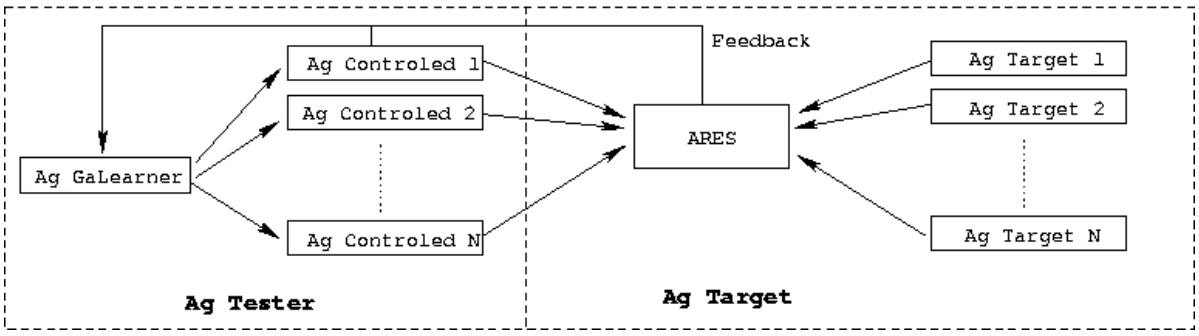


Figure 3 Learning system model breakdown

Within the learning system (Ag Tester) there are two types of agents: **Ag GaLearner** and **Ag Controlled**. **Ag GaLearner** represents the genetic algorithm that is used to develop the strategies used by the agents. **Ag Controlled** are the agents that execute the strategy produced against the ARES system and the targeted team. The **Ag Target** component of the system

represents the ARES system itself and the team of agents that the learner team is competing against.

Basic representation of a strategy

A strategy for a single agent that is generated by the learner is a series of actions to be executed over time. So it can be thought of as a sequence of actions where the first action represents time slice 1 and the next action represents time slice 2 and so forth for all the actions that are to be executed during one simulation in the ARES system. A team strategy is created by making a sequence of actions for each of the controlled agents and gathering them into a set.

Instantiation of the Genetic algorithm

The following section describes the specific instantiation of the genetic algorithm used in this project:

- **Representation of an individual:** an individual is represented by a single strategy that can be used by the controlled team of agents, as described above.
- **Genetic Operators:**
    - ***Elitism -*** This operator copies a top percentage of the individuals from the current population into the next population. The individuals that are copied are the ones who produce a good fitness with respect to the current competitive environment.
    - ***Crossover -*** This operator takes two parents provided by the selection mechanism and recombines them to create a new child. The recombination occurs by first generating a random crossover point and then taking the pair of action sequences from each parent for the same tester agent and splitting them at the crossover point. This process if continued with the same crossover point for each sequence of actions in the strategy.
    - ***Mutation -*** This operator picks a sequence of actions from the individual at random and then picks a random time slice to change an action at.
- **Fitness function:** the fitness of an individual is calculated by executing the strategy using the ARES system and counting the number of points gained by the targeted team (**NumSave**). The fitness then becomes **–1*NumSave**.

Experimentation

The experimentation phase used to evaluate the learning system followed the basic steps listed below:

1. The creation of the worlds used for the testing phase.

2. The creation of base tests for comparison with the results returned from our model. These tests are based upon collecting observations made about the behavior of the targeted team of agents in the test worlds without the interaction with our teams. The base tests include the following two steps:
    - Simulations were done with the test worlds with only the targeted team of agents. No other teams of agents were allowed in during these simulations.
    - The targeted team was run against the test worlds again. This time the team was duplicated and run as the second team during all the simulations.
3. Execution of the tests with our system in place as the second team of agents in the test worlds.

The targeted team of agents used for these evaluations came from a team of undergraduate students in the Winter 2004 semester class for the multi-agent systems course. This student team had produced an outstanding project that showed the best qualities for any team of agents since the first time the course had been taught. The following subsections will describe the specific setup for the experiments run for this project.

Worlds used for the simulations in the ARES system

For the experiments six different worlds were created, the six worlds can be broken down into two different categories:

1. **Special (3 worlds) :** these worlds are sparse and create a specific situation for the agents to deal with (see Figure 4 for an example).
2. **Random (3 worlds) :** these are randomly generated worlds for the agents to interact in. (see Figure 5 for an example)

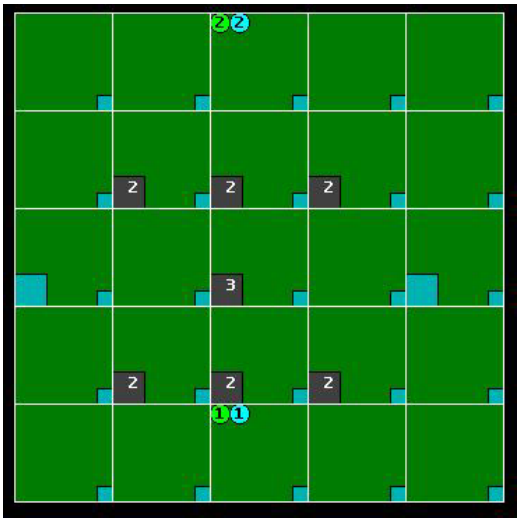


Figure 4 Example Special case experiment world

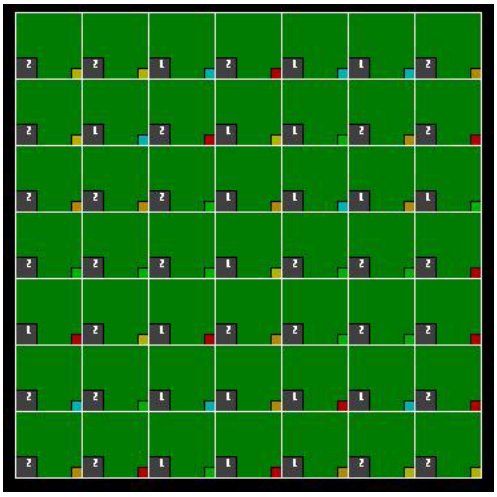


Figure 5 Example Random experiment world

Configuration of ARES

Below is a list of the specific settings for the configuration of ARES that were used for every simulation.

- Each team in the world was allowed two agents.
- Each team that has at least one agent participating in the save gains a single point.
- Fire does not spread in the world.
- Each simulation consists of 50 rounds.
- Agents from different teams were not allowed to communicate with each other through the channels set up in ARES.

It should be noted that in the original design of the system each agent is given one second to compute its action for a round during the simulation. Due to this and how it was originally implemented in the system, a single round with four agents connected to the system (two on each team) will take 4 seconds to execute. Because of this, a single simulation takes roughly 200 seconds (approx 4 minutes).

Configuration of the genetic algorithm

Below is a list of the specific settings for the genetic algorithm used as the learner during the experiments.

- **[Population size:]** 7
- [Individual size and length:] Since two agents are allowed for each team, an individual in the population contains two corresponding sequences for each of the tester agents controlled by the learner agent. Following upon this, since each simulation has fifty rounds it follows that each sequence is composed of fifty actions, one for each round.

- **[The selection mechanism used:]** 3-tournament selection. In three-way tournament selection, three individuals are chosen at random. From these three individuals, the one with the highest fitness becomes one of the parents used for breeding.
- **[The end condition:]** The genetic algorithm is run for ten generations for each simulation.
- **[The pseudo random number generator (PRNG):]** The Mersenne Twister algorithm is used as the PRNG for each simulation. A seed of zero was used and the state of the PRNG is saved between executions of experiments. (Specifically, the Mersenne twister implementation from the GNU scientific library was used (see [6]).

It should be noted that since a single simulation/test strategy with ARES takes about four minutes, it follows that a single step in the genetic algorithm to find the fitness of each individual will take about twenty-eight minutes. A complete execution of one experiment takes roughly two hundred and eighty minutes (5 hours). Due to the type of computer the experiments were run on, this execution time can extend anywhere from 7 to 15 hours.

Experiment Results

Statistics about the experiment test worlds

This section will cover the basic statistics about the number of survivors/points, and the total saves that could be achieved for an environment/world by the targeted team of agents. This number relates to the targeted team working on their own or with other teams of agents. In the case of the random worlds, these numbers are just the potential maximum values based upon the total number of survivors in the world. The second column in the table gives the total number of survivors that can be found in each world. The table below shows the total number or survivors that can be potentially saved in each world.

World	Total Survivors in the world
Special Case 1	8
Special Case 2	11
Special Case 3	9
Random 1	99
Random 2	98
Random 3	99

Table 1 Number of survivors in each experiment world

Base Experiments Results

Table 2 shows the results from the base experiments where only the targeted team was allowed in each world. The second column gives the total number of survivors that were saved by the targeted team. The final column shows the percentage value of saves based upon the total number of survivors in the world.

World	Number of Saves	Total percentage saved
Special Case 1	6	75%
Special Case 2	4	36.66%
Special Case 3	8	88.88%
Random 1	11	11.12%
Random 2	12	12.25%
Random 3	14	14.15%

Table 2 Base experiment results with only one team

It should be noted that only for the special case worlds 1 and 3 were the targeted team of agents able to save all of the possible survivors, without the help of more agents, when there is only one team of agents in the world.

Table 3 shows the base results when the targeted team was duplicated and used as teams one and two for each simulation with the test worlds. The second column shows the total number of points gained by team one, and the third column shows the total number of points gained by the second team. The fourth column shows the total number of actual survivors saved in the world. The final column shows the percentage of saves based upon the total number of possible saves that could have been made by the combined cooperation of both teams.

World	Team 1	Team 2	Total Saved	Total %
Special Case 1	5	4	6	75%
Special Case 2	8	8	8	72.73%
Special Case 3	6	6	6	66.67%
Random 1	5	8	11	11.12%
Random 2	9	9	17	17.35%
Random 3	11	9	20	20.21%

Table 3 Base experiment results with two duplicate teams

It can be seen by looking at the last three columns from table 3, that the point values and the actual number of survivors saved does not match up. This is due to the scoring mechanism used in ARES when teams work together. For these simulations each participating team in a save gains one point. By looking at these numbers, we can see a beginning point for why just using the point scores for each team is an inappropriate measure of success when looking for high level behavior. For example, when just looking at the point scores for special case 1 we could get the impression that all the survivors were saved in the world, since the combined score of the team is nine. This would then indicate that at, one point, the teams worked together and one team gained an extra point. In reality, when we include the actual number of survivors saved, we can see that

these scores indicate that the teams worked together many times, but did not save all possible survivors in the world. Based upon this, all comparisons made during the experimentation phase are based upon the actual number of saves made by each team. Figure 6 depicts the comparison between the number of survivors saved from the first and second base case experiments.

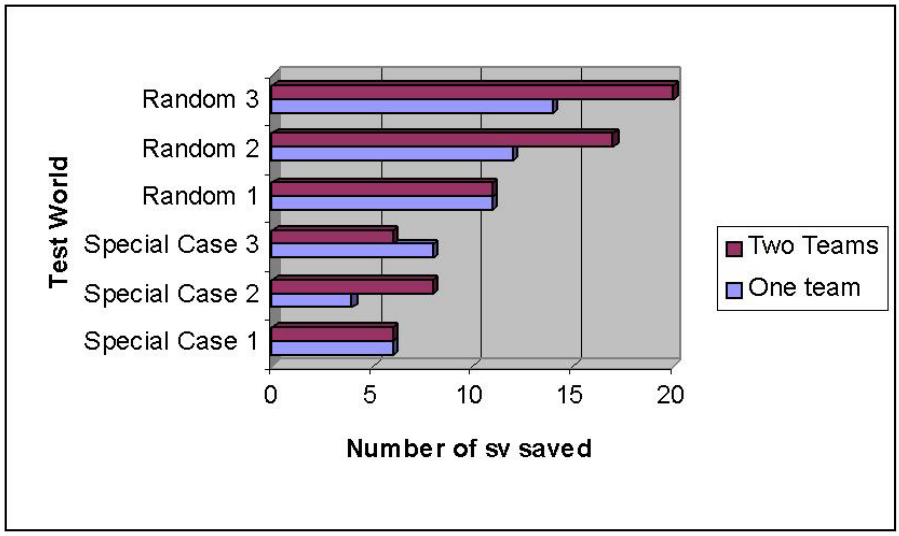


Figure 6 Comparison of base test results

By quickly scanning Figure 6 it can be seen that the results vary between one team in the world and two teams. For some of the worlds such as Special case 2, Random 2 and Random 3 there is an improvement that could indicate cooperation between the team's agents. But in Special Case 3 there is a decrease that could indicate competition between the teams.

Results with the Learning system in place

In Table 4 and Figure 7, we can see a consistent and large decrease between the number of saves when compared to the base test done with a single team of the targeted agents. This shows that the behavior of the targeted agents in this situation does not take into account trying to defend itself against the aggressive agents produced by our testing system. It should be noted that the experiment for Random 1 demonstrates that our model
is not infallible, but it is not far off from what was produced in the base results.

World	Num they saved with out model	Base (Two teams)	Base (one team)	Percentage
Special Case 1	2	5	6	25%
Special Case 2	4	8	4	36.37%
Special Case 3	1	6	8	11.12%
Random 1	6	5	11	6.07%

Random 2	8	9	12	8.17%
Random 3	10	11	14	10.12%

Table 4 Results with learning system in place

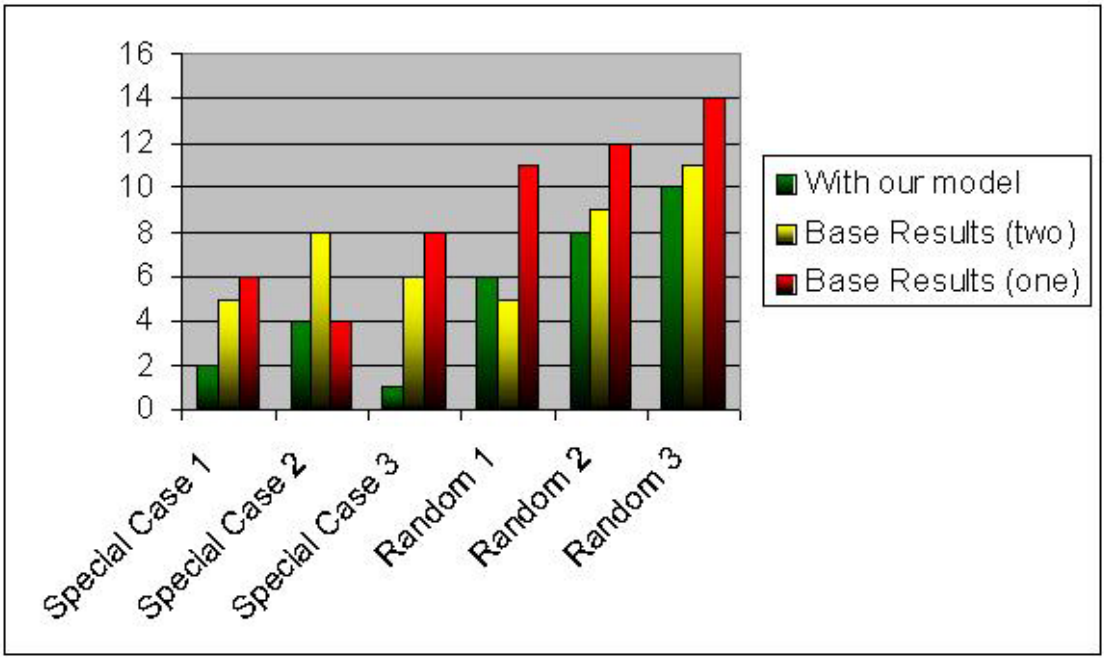


Figure 7 Comparison or results with learning system in place

The real benefits of these results can been seen when comparing the results returned by the base tests that involved two teams of agents, and the results returned when our model was used. In the case of the base tests, the resulting values did not give a clear way to describe the behavior of the agents when dealing with different teams. These values indicated that certain situations may have occurred, but it does not give a clear-cut view of what specifically happened. This fact becomes even clearer when viewing the differences in results returned for the competitive experiments. In this case, the results given by our model show clearly that the team does not deal well with an aggressive group of agents. Based upon the results returned by the base test, we would not have been able to see this form of behavior in the targeted team of agents. Thus the system was able to find a strategy that helped to minimize the number of saves made by the targeted team.

Conclusion

The results from the experiments clearly showed that the goal of the project has been met. The learning system was able to find a way to aggressively compete with the targeted system by minimizing the possible points gained.

Basic difficulties with the project

The following is a list of some of the basic difficulties found during the research done for my project, and the implementation of the system for the experimentation phase:

- Basic improvements could have been made to the ARES system to help increase the execution time of a single simulation.
- Additional information could have been added into the genetic operators. This information could have helped in the selection of points to change that in the end could help to increase the quality of the results.
- The basic structure of the tester agents can be improved to help create more complex cooperation situations.
- Experimentation could be done into the application of different learning algorithms or models for the learning agent. In addition to this, different algorithms could be combined together and evaluation made on the resulting quality of the results could be made.

Future Work

In terms of modifications that could be made to my model, there are many different directions that could be taken. A possible beginning point would deal with the group of tester agents. For this project the tester agents were fed basic sequences of actions. The resulting coordination found by the learner agent then would only be applicable for the current environment and situation used for a simulation. The format and structure of the information could be modified to include a more complex model that would be applicable for a range of different uses in different variations of the environment as situation at hand.

In the case of sequences of actions, the intelligence of the tester agents was limited to stepping through each action. The result of this is the fact that these learned sequences are only applicable to the current situation. In the case of the ARES system, a change in the world would nullify the previous sequences. The end result of this is the fact that the tester agents would not be able to adapt to these changes.

A possible beginning point of this would be to build more intelligence into the models used by the tester agents. Each tester agent could be given specific roles and predefined reactions. This setup could allow for the tester agents to become more aware of the current situation they are in. Based upon this, the tester agents then could possibly react even when they are given conflicting information. The learner agent then could supply a plan of attack to each tester agents and whom they should work with. The learner agents, who still handle the coordination that occurs between the agents, would hold the global view of events. Each agent then would implement specific coordination between the tester agents following the plan they were provided with from the learner agent.

In addition to this, the tester agents could be organized to be aware of the other tester agents within the environment. This would then allow for the possibility direct coordination among the tester agents during the execution of a testing strategy. With this direct communication, the learner agents could create more complicated plans. These plans could include points where one tester agent must signal another tester agent to go ahead with their next actions. With all of this in mind, a tester agent could become more reactive to the situation at hand and take more intelligent actions as the current testing strategy is executed.

References

[1] J¨org Denzinger and Jordan Kidney. ARES 2: A tool for evaluating cooperative and competitive multi-agent systems. In Proc. of 18th Conference
of the Canadian Society for Computational Studies of Intelligence, Victoria, pages 38–42, 2005.

[2] J¨org Denzinger and Jordan Kidney. Teaching multi-agent systems using the ARES simulator. ITALICS online journal http://www.ics.ltsn.ac.uk/pub/italics/, 2005.

[3] J¨org Denzinger, Jordan Kidney, and Melissa Bergen. Teaching cooperation
in multi-agent systems with the help of the ARES system. In Proc. of WCCCE-03, 2003.

[4] Jordan Kidney. Adding competition into ARES: providing a more versatile platform for teaching multi-agent systems. In Proc. of AAMAS 2004 workshop on Teaching Multi-Agent systems, New York, pages 30–36, 2004.

[5] ARES Website. http://www.cpsc.ucalgary.ca/~kidney/ARES/, (April 2007)

[6] Erick Cant´u-Paz. On random numbers and the performance of genetic algorithms. In Proc. of the Genetic and Evolutioary Computation conference (GECCO), pages 1019–1026, 2002.

Forest Cover Type Classification

Hong Xu & Lizhe Ma

Outline

1. Problem analysis and data description
2. Data collection and pre-process
3. Methods used for classification
4. Conclusion and experiments
5. Appendix
6. Reference

1. Problem analysis and data description

The task of this project is to classify the forest cover type for a given observation which means it is a classification problem. There are many algorithms we can use for classification. We can use decision tree, clustering, support vector machine and neural network. For this project, we chose k-means clustering, support vector machine and backpropagation neural network for classification.

We have 581, 012 observations (instances) totally and 54 attributes. There are 12 measures including 10 quantitative variables, 4 binary wildness areas and 40 binary soil type variables. And we have 7 forest data types which is used as the target as the classification algorithms. Appendix B provides more detailed data description. In order to get accurate result, it is necessary to preprocess the data before they are applied to the data mining algorithm. The following section discusses how to preprocess the data and how to use data mining algorithms to mine the data.

2. Data collection and preprocess

1) Data collection and visualization

Collecting data is the firs step for the data mining task, improved data quality can improve any analysis on it. Since the privacy issue, some data can not be collected. However, in our project, we got the data from the [1].

Data visualization can give analyzers the general idea of data distribution and help them to choose better data preprocess approaches to clean the data. In our project, we randomly select part of the source data to do the analysis since the original source data is huge and use WEKA to visualize the data. The following figures visualize the first attributes of the source data and the target.

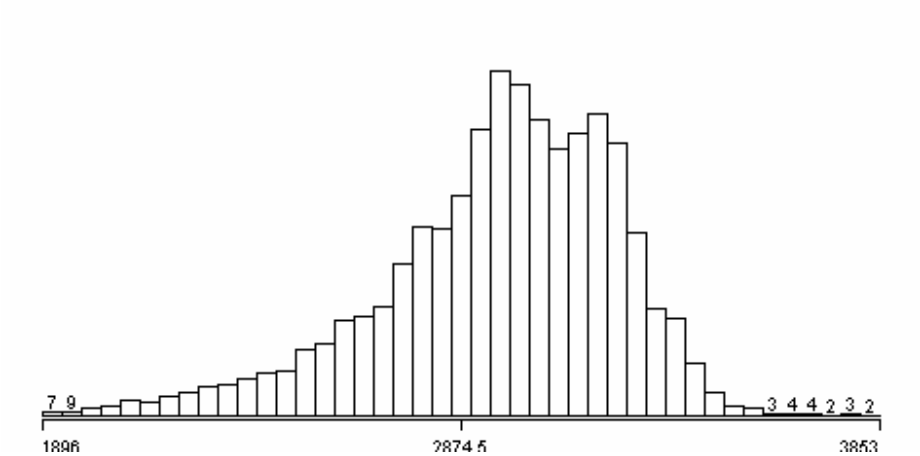


Figure 1, data distribution for the first attributes.

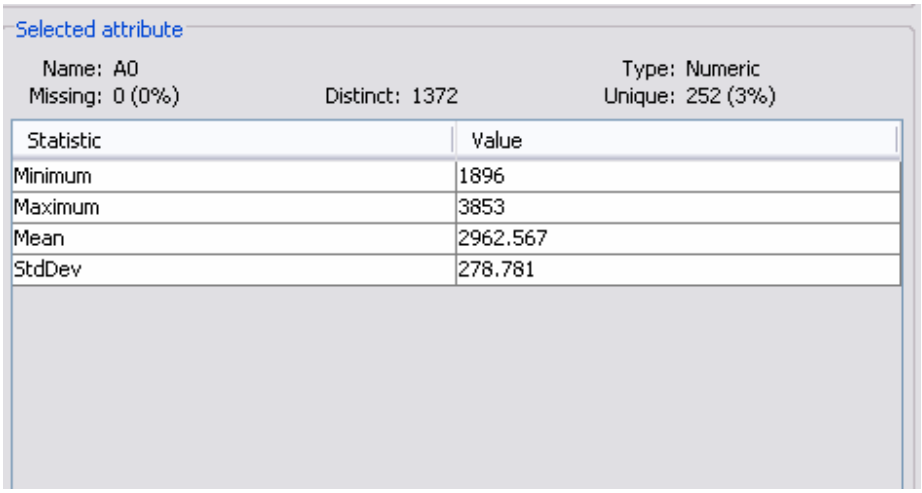


Figure 2, statistic analysis for the first distribute

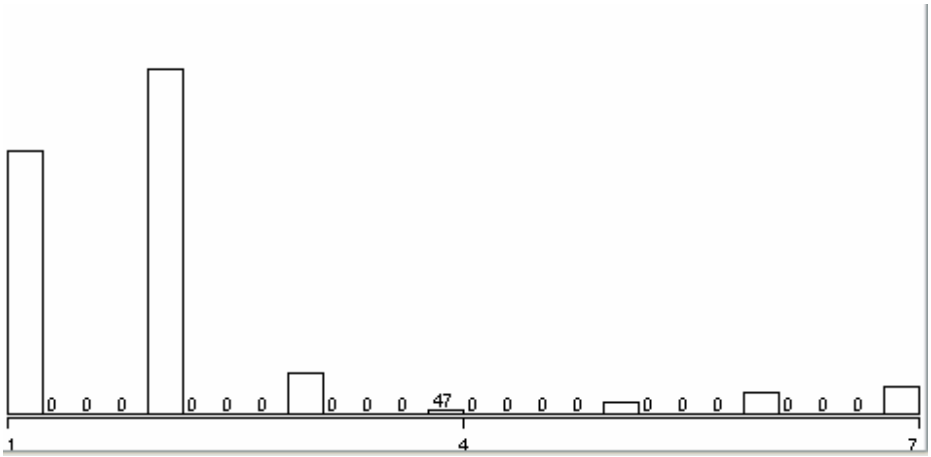


Figure 3, data distribution for the target

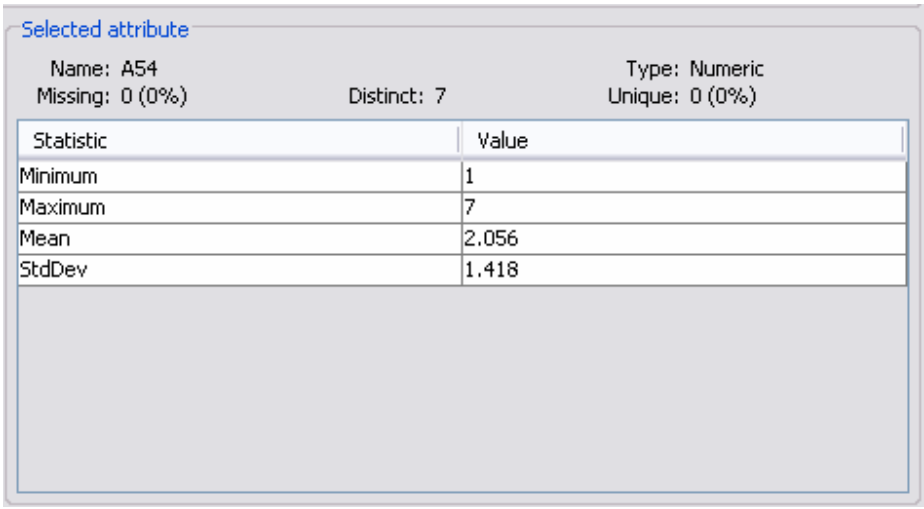


Figure 4, statistic analysis for the target

The full data visualization is in appendix A, from appendix A, we can see the first nine attributes are outweighing features, and we have to normalize them to prevent them from overpowering.

2) Data preprocessing

Even the needed data is collected, the preprocessingfg procedure is still have to applied to the collected data to improve its quality which is the key issue in data mining. Data preprocess include data cleaning, data integration, data normalization, and data reduction.

Data cleaning is used to deal incomplete data which means there are missing values or missing attributes in the data and noisy data which are errors or outlier. Attribute mean of all samples belonging to the same class can be used to handle missing values, but it is just an estimate, it can leads to invalid results. Cluster analysis and regression can be used to deal noisy data, the noisy data are the data which are outside any cluster in cluster analysis and a regression function is used to detect noisy data in regression analysis.

Inconsistency and redundancy issues exist when data from different source are integrated together, such as schema heterogeneity, repeated tuples in different source databases. Data integration is used to deal these issues, and metadata is used in this approach.

The amount of collected data may be huge which makes the analysis very difficult, data reduction can be used to reduce the size of data but still show the same analysis. There are four methods can be used for data reduction which are data aggregation, dimension/attribute reduction, data compression and discretization. Data compression means reduce the redundancy and discretization means transferring numeric data to categorical values.

And one feature might overpower other features, normalization is used to prevents outweighing features with large range. Min-Max is a common approach for data normalization. It is a linear transformation of the original input range into a newly specified range. [2]

3) Choosing proper data preprocessing techniques

In our project, the data is from US Geological Survey and US Forest Service, the number of instance is 581012, the number of attribute is 54, there is no miss attribute values, and there is few noisy data after we apply cluster method to the source data (the detail of approach will discuss in later in this report), so it is not necessary to apply data cleaning technique to the source

data. The data is very collected and there are no inconsistency and redundancy issues. However, the amount of the source data in this project is huge, and we got the out of virtual memory error when we applied the matlab methods to the source data, so we randomly selected part of the data for the analysis. There are a few features overpower other features, so data normalization was used to avoid this. We used the premnmx function in matlab to normalize the data, it can preprocess the data so that the minimum is -1 and maximum is 1. This algorithm is

```
pn = 2*(p-minp)/(maxp-minp) - 1;
```

Where p is the R*Q matrix of input vector, minp is the R*1 vector containing minimums for each p, maxp is the R*1 vector containing maximum for each p, and pn is the R*Q matrix of normalized input vectors. [3]

There are 54 attributes, and we can use the feature selection methods to reduce the dimension to get better results, and we used WEKA to get the selected the features. We chose CfsSubsetEval and ClassifierSubsetEval as the attribute evaluators, and BestFirst, GreedyStepwise and GeneticSearch as the search methods.

In CfsSubsetEval evaluation, a subset of attributes in which each individual has high predictive ability and has low intercorrelation between each other is preferred. In ClassifierSubsetEval evaluation, attributes subsets on training data are evaluated.

In BestFirst search method, the space of attribute subsets are searched by greedy hillclimbing augmented with a backtracking facility. The number of consecutive non-improving nodes can be used to control the level of backtracking done. In GeneticSearch search method, a simple genetic algorithm is used to perform the search. In GreedyStepwise search method, a greedy forward or backward search is performed through the space of attribute subsets.

The following tables show the selected features by using CfsSubsetEval and ClassifierSubsetEval evaluation methods with different search methods.

Feature Selection	Evaluation method	Search method	Selected features
1	CfsSubsetEval	GreedyStepwise	3,5,7,8
2	CfsSubsetEval	BestFirst	3,5,7,8
3	CfsSubsetEval	GeneticSearch	1,2,3,5,6,7,8,9,10,12,15,19,21,23,27,28,33,46,47,48,53
4	ClassifierSubsetEval	GreedyStepwise	None
5	ClassifierSubsetEval	BestFirst	None
6	ClassifierSubsetEval	GeneticSearch	14, 16

Table 1, selected features by combining CfsSubsetEval and ClassifierSubsetEval evaluation methods with different search methods

We construct the following table for summarizing the selected features.

Feature selection	Selected features
FS1	14, 16
FS2	3, 5, 7, 8
FS3	1,2,3,5,6,7,8,9,10,12,15,19,21,23,27,28,33,46,47,48,53

Table 2, selected features

From table 1, the combination of CfsSubsetEval with different search methods gives us better results than the combination of ClassifierSubsetEval with different search methods, and this is improved by the later experiments.

3. Methods used for classification

In data collection and preprocess section, we discuss the how to collect the data, how to visualize the data and how to clean data for this project. In this section, we discuss applying matlab clustering and classification methods to mine the data.

1) Clustering

We use k-means clustering from matlab to cluster the data, and use silhouette to evaluate the results. Two-phase iterative algorithm is used in kmans to minimize the sum of point-to-centroid distances. In the first phase, points are assigned to their nearest cluster centroids and the cluster centroids are recalculated iteratively; in the second phase, points are individually reassigned if it could reduce the sum of point-to-centroid distances, and the cluster centroids are recalculated iteratively. We use four distance measure to calculate the point-to-centroid distances. These four distance measures are sqEuclidean in which the mean of the points in each cluster is treated as the centroid, cityblock which sums the absolute differences, cosine in which distance is one minus the cosine of the included angle between points and correlation in which distance is one minus the sample correlation between points.

The silhouette is chosen to evaluate the results, it measures how close each point in one cluster to points in the neighboring clusters. The measure ranges from -1 to 1, 1 means the points are very distant from neighboring clusters, 0 means the points can be assigned to one cluster or another, and -1 means that the points are assigned to a wrong cluster. The distance measure can be Euclidean, sqEuclidean, cityblock, cosine, correlation. The following figure was used to calculate the silhouette values.

```
[s, h] = silhouette(X, clust, distance)
```

X is the n-by-p matrix which is plotted by silhouette, rows of X correspond to points, and columns correspond to coordinate. Clusters are defined by clust, it can be a numeric vector containing a cluster index for each point, or a character matrix containing a name for each point. S is an n-by-1 matrix and contains the returned silhouette values for each point. H is a figure handle.

The following figures show the plotted silhouettes using different distance measure.

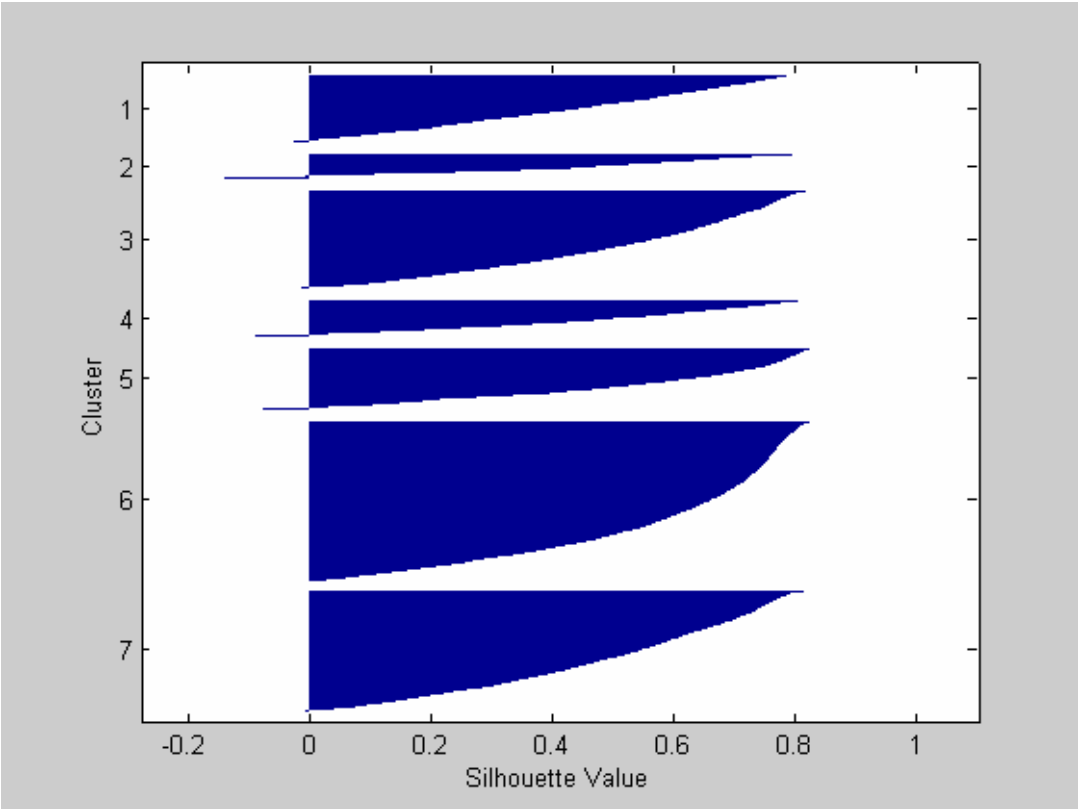


Figure 5, plotted silhouette value using sqEuclidean as the distance measure

From figure 5, we can see that most points in cluster 6 have large silhouette values which mean they are somewhat well separated from neighboring clusters. However, cluster 1, 2, 3, 4 and 5 has a few points having negative values , and cluster 7 has many points with low silhouette values, indicating cluster 1, 2, 3, 4, 5, 7 are not well separated.

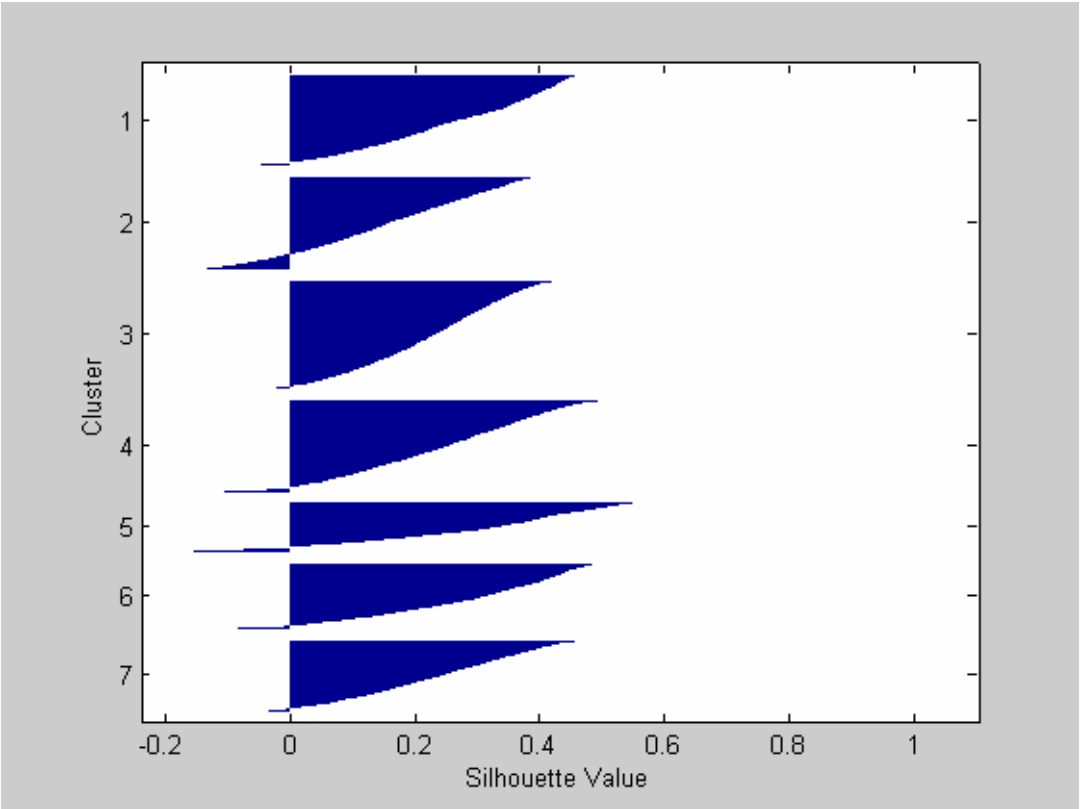


Figure 6, s

From figure 6, we can see that all the clusters have low silhouette values and have some points with negative silhouette values indicating those clusters are not well separated.

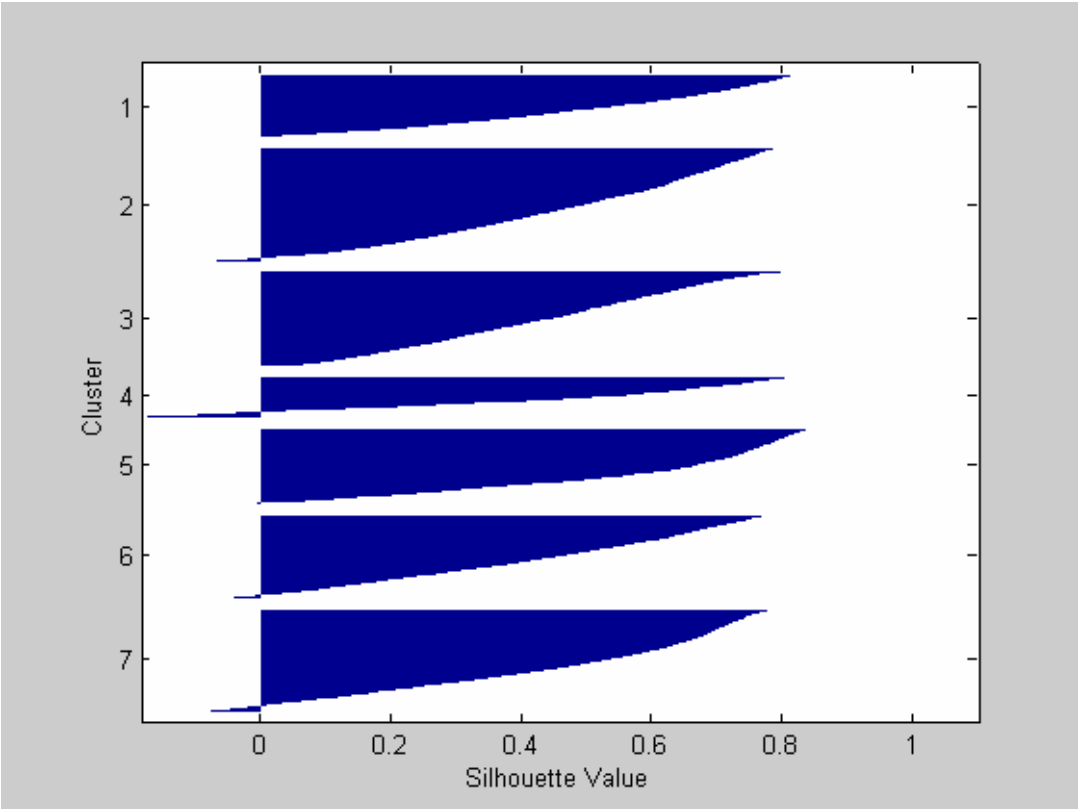


Figure 7, plotted silhouette value using cosine as the distance measure

Cluster 2, 4, 6 and 7 have points with negative silhouette values, and cluster 1, 3, 5 have many points with low silhouette values, indicating all clusters are not well separated.

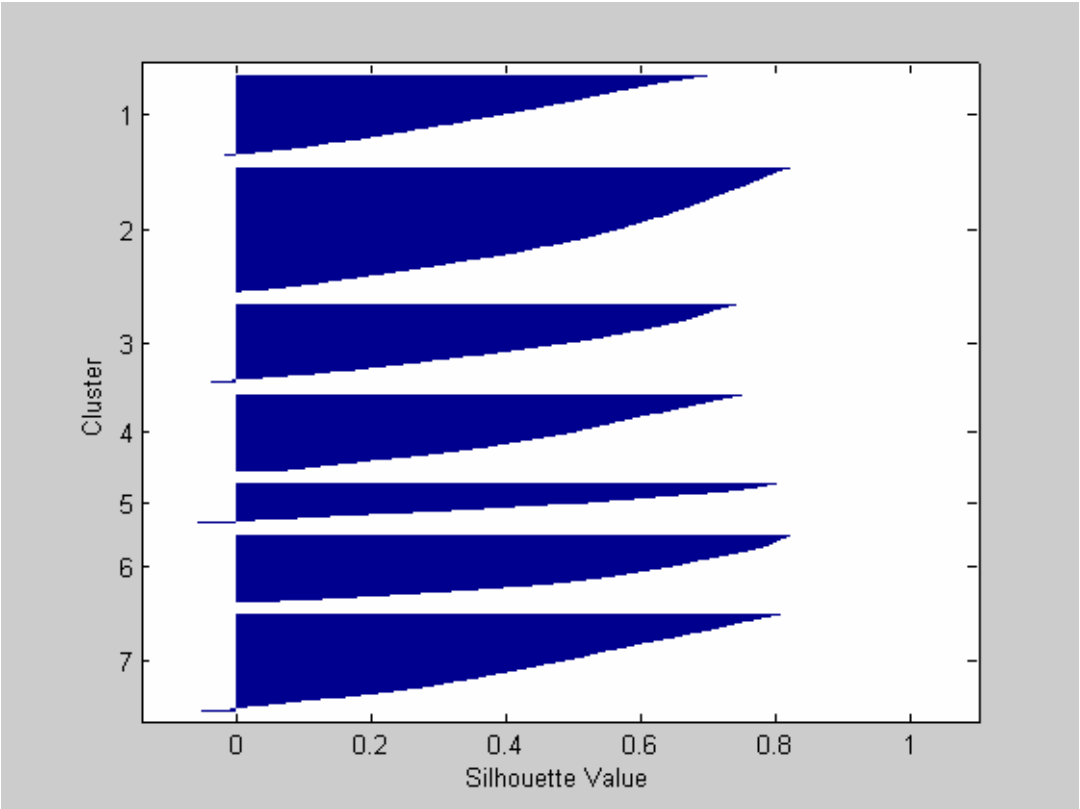


Figure 8, plotted silhouette value using correlation as the distance measure

Cluster 1, 3, 5 and 7 have a few points with negative silhouette values, cluster 2, 4, 6 have many points with low silhouette values, indicating all clusters are not well separated.

From previous analysis, we can not use k-means clustering to do classification for this specific application.

2) Support vector machine (SVM) for classification

Support vector machine uses supervised learning method for classification or regression; it can minimize the empirical classification errors and maximize the geometric margin at the same time.

Support vector machine is based on the concept of decision planes which define decision boundaries. A decision plane separates between a set of objects having different class memberships [4]. The following figure shows a classification support vector machine's error function.

Classification SVM Type 1

For this type of SVM, training involves the minimization of the error function:

$$\frac{1}{2}w^T w + C\sum_{i=1}^{N}\xi_i$$

subject to the constraints:

$$y_i\left(w^T\phi(x_i)+b\right)\geq 1-\xi_i \text{ and } \xi_i \geq 0,\ i=1,\ldots,N$$

Figure 9, classification SVM error function.

In Figure 9, C is the capacity constant, w is the vector coefficients, b is a constant, and ξ_i are parameters for handling nonseparable data. The kernel ϕ is used to transfer data from input to feature space.

We downloaded a matlab support vector machine toolbox which is called libsvm to do the classification. The function svmtrain is used to train the network, and function svmpredict is used to classify the testing data. In function svmtrain, it has an option called kernel type which has 4 values that are linear, polynomial, radial basis, and sigmoid. We tried all of them and Radial basis kernel type gave us the best results [5, 6].

i. Random selection test

For random selection testing, we randomly selected 10000 instances from source data, and then randomly selected 50% instances for training and left for testing every time. We tried to use SVM to classify these instances without normalization and feature selection, with normalization only, with normalization and feature selection1, with normalization and feature selection2, and with normalization and feature selection3. We tried 10 times for each testing, and used the average MSE value and accuracy value as the final values. The following figures show detailed results of each test.

```
mseValList =

    1.9558    2.0000    1.9720    2.0064    1.9414    1.9438    1.9848    1.9908    1.9786    1.9608


accuValList =

   49.2200   47.9400   48.2600   48.5000   48.7800   48.4800   49.6200   48.5800   48.6000   48.7600
```

Figure 10, SVM classification using random selected dataset without data normalization and feature selection

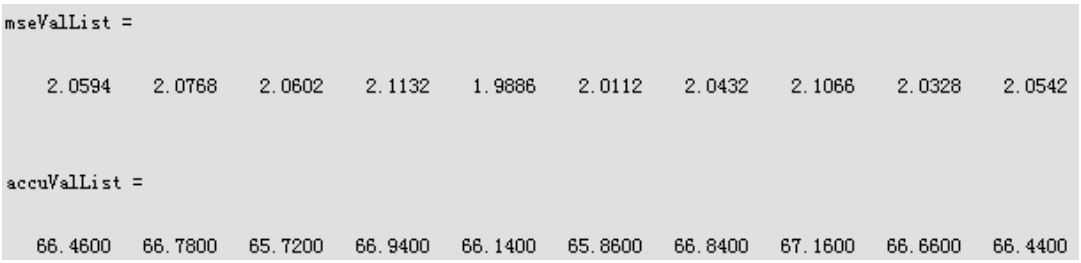


Figure 11, SVM classification using random selected dataset with data normalization only

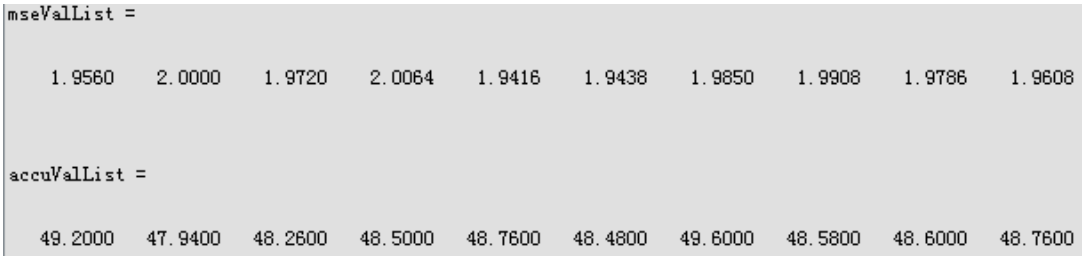


Figure 12, SVM classification using random selected dataset with data normalization and FS1

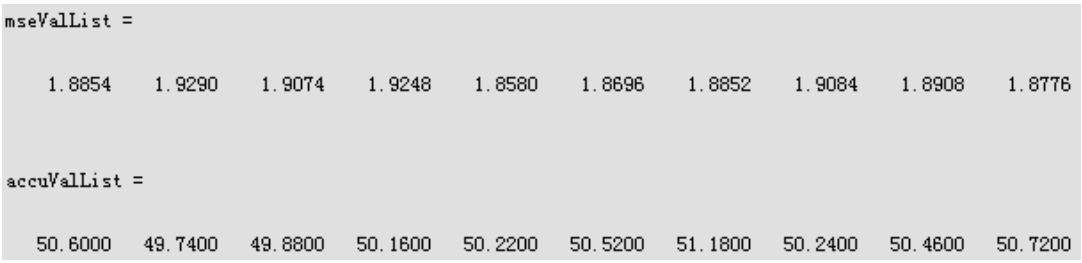


Figure 13, SVM classification using random selected dataset with data normalization and FS2

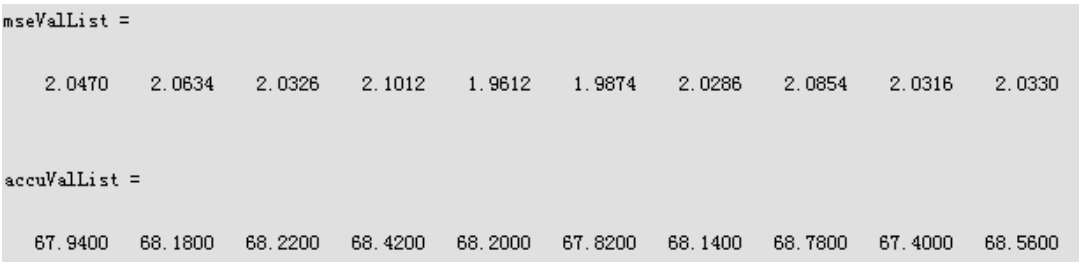


Figure 14, SVM classification using random selected dataset with data normalization and FS3

The following table calculates the average MSE and accuracy values showed in Figure10, Figure11, Figure12, Figure13 and Figure14.

	Without normalization	normalization only	Normalization + FS1	Normalization + FS2	Normalization + FS3

	and FS				
Average MSE	1.9734	2.0546	1.9735	1.8936	2.0371
Average Accuracy (%)	48.6740	66.5000	48.6680	50.3720	68.1660

Table 3, average MSE and accuracy values using random data selection

From table 3, we can see that combination of data normalization and feature selection3 gave us the best average accuracy.

ii. Cross validation test

For cross validation testing, we randomly selected 10000 instances from source data, and used 5 folds cross validation every time. We tried to classify these instances without normalization and feature selection, with normalization only, with normalization and feature selection1, with normalization and feature selection2, and with normalization and feature selection3. We tried 10 times for each testing, and used the average MSE value and accuracy value as the final values. The following figures show detailed results of each test.

```
mseValList =

    2.1110    2.0470    1.9715    1.9025    2.0360


accuracyValList =

   48.3000   48.9500   48.1500   48.1500   48.8500
```

```
averageMSEVal =

    2.0136


averageAccuracyVal =

   48.4800
```

Figure 15, SVM classification using 5 folds cross validation without data normalization and feature selection

```
mseValList =

    2.2100    2.2180    2.0575    1.8915    2.1005


accuracyValList =

   67.9000   67.6500   67.0500   68.7000   66.3000
```

```
averageMSEVal =

    2.0955


averageAccuracyVal =

   67.5200
```

Figure 15, SVM classification using 5 folds cross validation with data normalization only

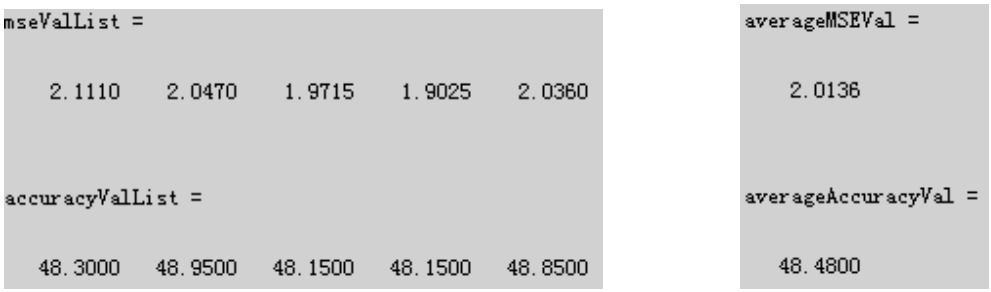


Figure 16, SVM classification using 5 folds cross validation with data normalization and FS1

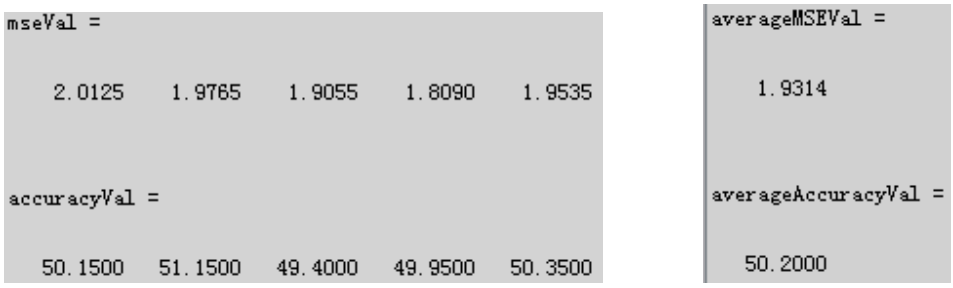


Figure 16, SVM classification using 5 folds cross validation with data normalization and FS2

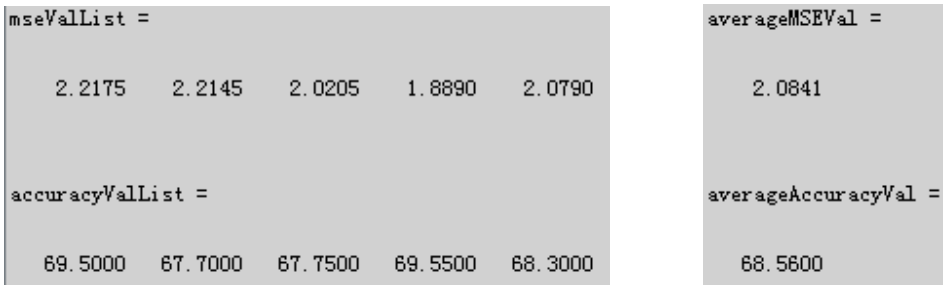


Figure 17, SVM classification using 5 folds cross validation with data normalization and FS3

	Without normalization and FS	normalization only	Normalization + FS1	Normalization + FS2	Normalization + FS3
Average MSE	2.0136	2.0955	2.0136	1.9314	2.0841
Average Accuracy (%)	48.4800	67.5200	48.4800	50.2000	68.5600

Table 4, average MSE and accuracy values using 5 folds cross validation

From table 4, we can see that combination of data normalization and feature selection3 gives use the best average accuracy.

Comparing with table 3 and table 4, we find that these two approaches give us almost the same results, and the combination of data normalization and feature selection3 always give us the best average accuracy. And when we compare the results with normalization only with the results

with normalization and FS3, we can find that they give us very close result which means that data normalization plays a very important role in improving the average accuracy in this project.

3) Neural network

We used backpropagation neural network with an adaptive learning rate for faster training. Backpropagation was created to generalize the Widrow-hoff learning rule for multiple-layer network and nonlinear differentiable transfer functions. The input vector and target vector inputted by the user are used to train the network. Standard backpropagation is a gradient descent algorithm, and the weights are moved along the negative of the gradient of the performance function. The learning rate is constant in the standard backpropagation algorithm, and the performance is very sensitive to the setting of this constant learning rate. However, in variable learning rate faster training, the learning rate can keeps the learning steps as large as possible and keep the learning stable. Since it takes time to train the neural network, we used faster training approach for our project.

i. Random selection test

For random selection testing, we randomly select 10000 instances from source data, and then randomly selected 50% instances for training and left for testing. Since we do not know how many layers we need and how many neurons each layer needs, so we start from two layers. We used the neural network to classify the data without the data normalization and feature selection, with data normalization, and with data normalization and feature selection 3. For the two layers neural network, we used 'tansig' as the first layer, 'purelin' as the second layer and 'traingda' as the training function. For the three layers neural network, we used 'tansig' as the first and second layer, 'purelin' as the third layer and 'traingda' as the training function.

For the first testing, we use the neural network to classify the data without the data normalization and feature selection using two layer neural networks. The following table shows the detailed results.

The following table shows the detailed results without data normalization and feature selection.

Num of Neurons of Firs Layer	Num of Neurons of Second Layer	MSE Value	Accuracy Value
5	1	1.9549	0.4920
10	1	1.7818	0.5084
15	1	1.9695	0.4902
20	1	1.7854	0.4792
25	1	1.7365	0.4396

Table 5, MSE and accuracy values using two layers neural network without data normalization and feature selection

From table 5, we can find that it gives us the best accuracy value when the number of neurons of first layer is 10.

We increased the number of layers into 3 in the second testing, and set the number of neurons of second layer to 10 which we got from the first testing. The following table shows the detailed results for classifying source data using three layers neural network without data normalization and feature selection.

Num of Neurons of First Layer	Num of Neurons of Second Layer	Num of Neurons of Third Layer	MSE Value	Accuracy Value
5	10	1	1.9726	0.4920
10	10	1	1.8579	0.4426
15	10	1	1.7406	0.4294
20	10	1	1.9549	0.4920
25	10	1	1.7758	0.5280

Table 6, MSE and accuracy values using three layers neural network without data normalization and feature selection

All the accuracy values are not good in table 5 and table 6 when we used neural network to classify the source data without data normalization and feature selection.

The following tests try to use neural network to classify the source data with data normalization only. For the first testing, we use the neural network to classify the data with the data normalization only using two layers neural networks. The following table shows the detailed results.

Num of Neurons of Firs Layer	Num of Neurons of Second Layer	MSE Value	Accuracy Value
5	1	1.8708	0.4722
10	1	1.8573	0.4758
15	1	1.8864	0.4928
20	1	1.8784	0.4648
25	1	1.8792	0.4926

Table 7, MSE and accuracy values using two layers neural network with data normalization only

From table 7, we can find that it gives us the best accuracy value when the number of neurons of first layer is 15.
We increased the number of layers into 3 in the next testing, and set the number of neurons of second layer to 15. The following table shows the detailed results classifying source data using three layers neural network with data normalization only.

Num of Neurons of First Layer	Num of Neurons of Second Layer	Num of Neurons of Third Layer	MSE Value	Accuracy Value
5	15	1	1.9996	0.4920
10	15	1	2.0611	0.4920
15	15	1	2.1222	0.4354
20	15	1	1.9968	0.4192
25	15	1	1.9877	0.4082

Table 8, MSE and accuracy values using three layers neural network with data normalization only

All the accuracy values are not good in table 7 and table 8 when we use neural network to classify the source data with data normalization only.

The following tests try to use neural network to classify the source data with data normalization and feature selection3 together. For the first testing, we used the neural network to classify the data with the data normalization and feature selection using two layers neural networks. The following table shows the detailed results.

Num of Neurons of Firs Layer	Num of Neurons of Second Layer	MSE Value	Accuracy Value
5	1	1.9227	0.4258
10	1	1.9051	0.4836
15	1	1.8027	0.5006
20	1	2.181	0.4838
25	1	1.8342	0.4722

Table 9, MSE and accuracy values using two layers neural network with data normalization and feature selection3

From table 9, we find that it gives us the best accuracy value when the number of neurons of first layer is 15.
We increased the number of layers into 3 in the next testing, and set the number of neurons of second layer to 15. The following table shows the detailed results classifying source data using three layers neural network with data normalization and feature selection.

Num of Neurons of First Layer	Num of Neurons of Second Layer	Num of Neurons of Third Layer	MSE Value	Accuracy Value
5	15	1	1.8479	0.5000
10	15	1	1.8381	0.4532
15	15	1	1.7346	0.5136
20	15	1	1.7404	0.4972
25	15	1	1.7588	0.4496

Table 10, MSE and accuracy values using three layers neural network with data normalization and feature selection3

All the accuracy values are not good in table 9 and table 10 when we use neural network to classify the source data with data normalization and feature selection.

From table 5 to table 10, we can see that we did not get good accuracy using neural network with random selected data.

ii. 5 folds cross validation

Three approaches give us almost the same MSE value and accuracy value when we compare table5, 6, 7, 8, 9, and 10. We chose the last approach, which classifies the source data with data normalization and feature selection 3, for 5 folds cross validation, since it does not only show the

same analysis with the approach which classifies the source data with data normalization only, but also improve the analysis efficiency with less features.
For the two layers neural network, we use 'tansig' as the first layer, 'purelin' as the second layer and 'traingda' as the training function. For the three layers neural network, we use 'tansig' as the first and second layer, 'purelin' as the third layer and 'traingda' as the training function.

Num of Neurons of Firs Layer	Num of Neurons of Second Layer	MSE Value	Accuracy Value
5	1	2.0107	0.4848
10	1	1.9201	0.4721
15	1	1.8536	0.4693
20	1	1.8013	0.5076
25	1	1.7909	0.4584

Table 11, MSE and accuracy values using 5 folds cross validation with two layers neural network with data normalization and feature selection3
From table 11, we can find that it gives us the best accuracy value when the number of neurons of first layer is 20. We increased the number of layers into 3 in the next testing, and set the number of neurons of second layer to 20.

Num of Neurons of First Layer	Num of Neurons of Second Layer	Num of Neurons of Third Layer	MSE Value	Accuracy Value
5	20	1	2.1519	0.3118
10	20	1	2.0884	0.4848
15	20	1	2.1014	0.4623
20	20	1	2.0040	0.4038
25	20	1	1.8905	0.4033

Table 12, MSE and accuracy values using 5 folds cross validation with three layers neural network with data normalization and feature selection3

All the accuracy values are not good in table11 and table 12 when we used neural network to classify the source data using 5 folds cross validation with data normalization and feature selection.

From table 5 to table 12, we can see we can not get good accuracy values no matter which approach we chose, however, the SVM can give us good accuracy values when we apply data normalization, so SVM can used for this project.

4. Conclusion and Experiments

We tried three data mining classification methods for this problem, one of them is clustering which is an unsupervised learning, and the left two are classifications which are supervised learning.
We used k-means for clustering, and used silhouette to measure the clustering result. We tried four distance measures for k-means clustering and silhouette, but none of them can give us a good result. We randomly selected 10,000 instances for clustering, and we also tried to use 20,000 instances for clustering, however, the later case took too long to give us the results. Since

there are 54 attributes and also some attributes are overpowering others, it is hard to separate them very well in 7 clusters.

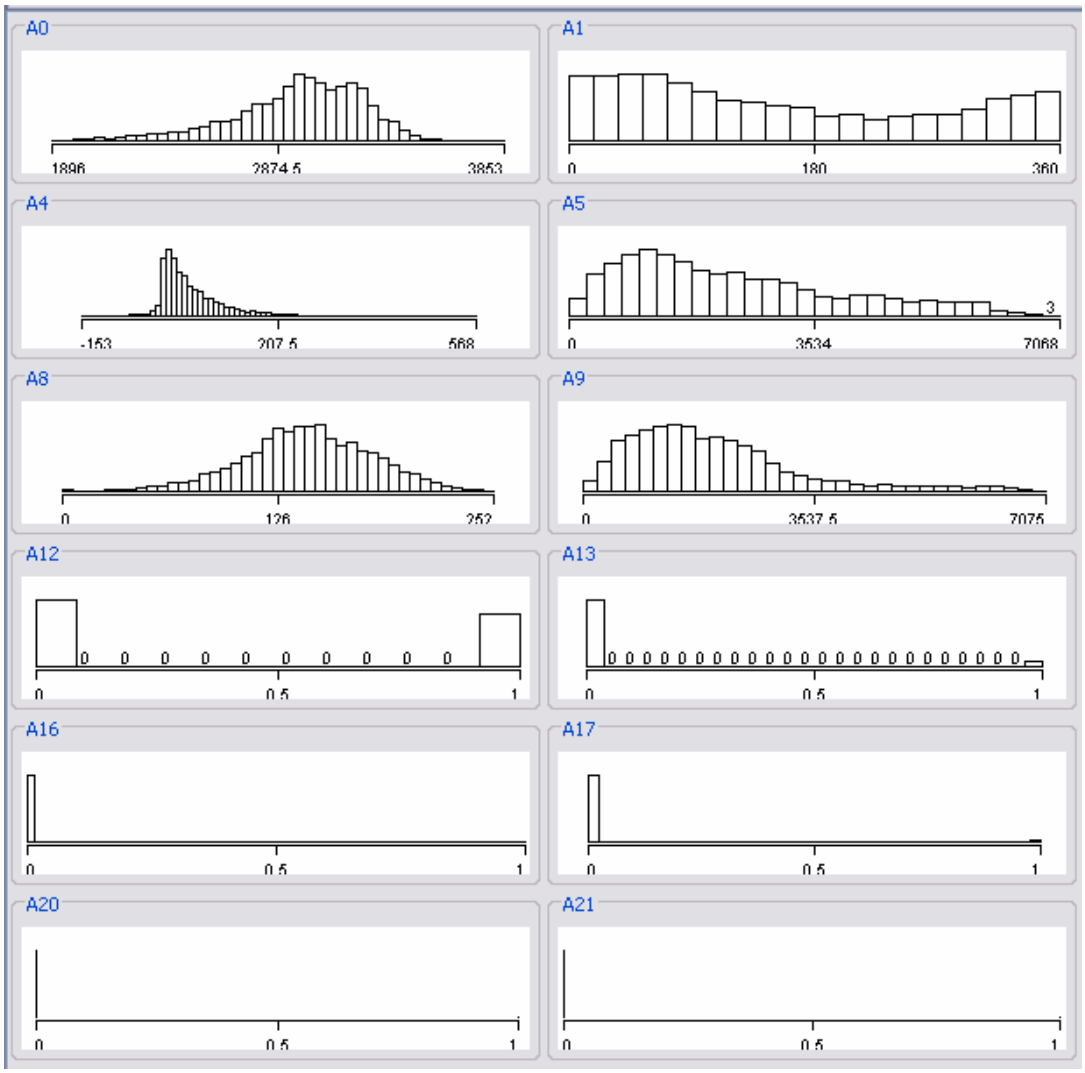
For the classification, we used support vector machine and neural network to classify the source data. We downloaded a support vector machine toolbox for matlab [5]. We chose the nu-SVC as the SVM type, there are four kernel types which are linear, polynomial, radial basis and sigmoid however, only radial basis kernel type gave us the best results. From the tables 3 and 4, we can see that data normalization plays an important role in the SVM classification, and it prevents outweighing features with large range. With the help of data normalization, the accuracy increased into an acceptable range.

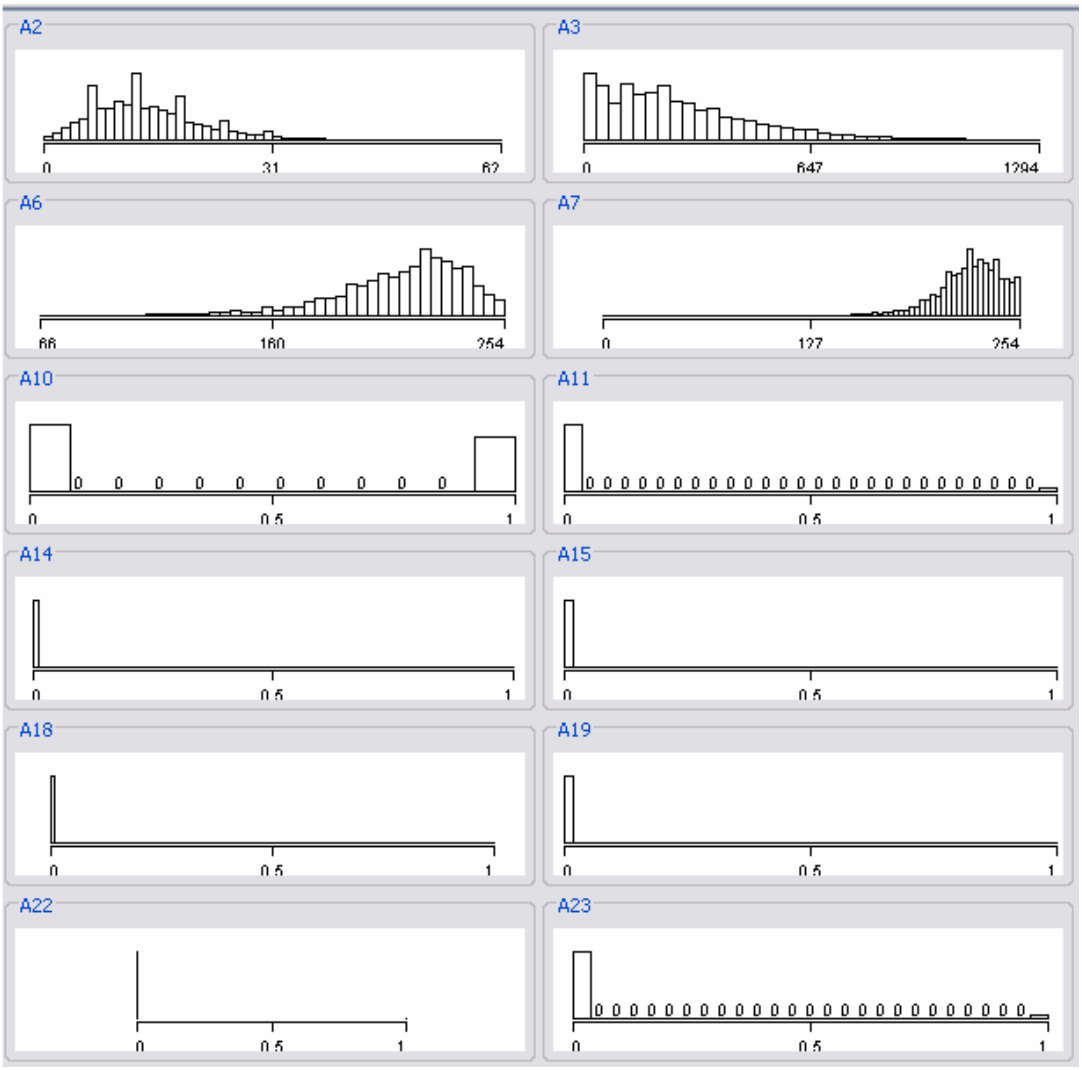
We used the built in neural network toolbox to classify the source data, we tried two layers neural network first, we used 'purelin' as the transfer function for the second layer, and used 'tansig' as the transfer function for the first layer, since this combination can approximate any function with a finite number of discontinuities, arbitrarily well, given sufficient neurons in the hidden layer for the two layers neural network [7]. We also tried three layers neural network by adding one more hidden layer to the two layers neural network and using 'tansig' as the transfer function for this hidden layer. We tried data normalization and feature selection, however, the best accuracy is around 50% which is not acceptable. We used 10,000 instances to train the neural network, maybe we need more instances for the training, however, it was too slow to get the results even we used faster training when we tried to use 20,000 instances for the testing. I think the reason that we could not get acceptable results is we did not have proper values to initialize the weights before we did the training and testing.

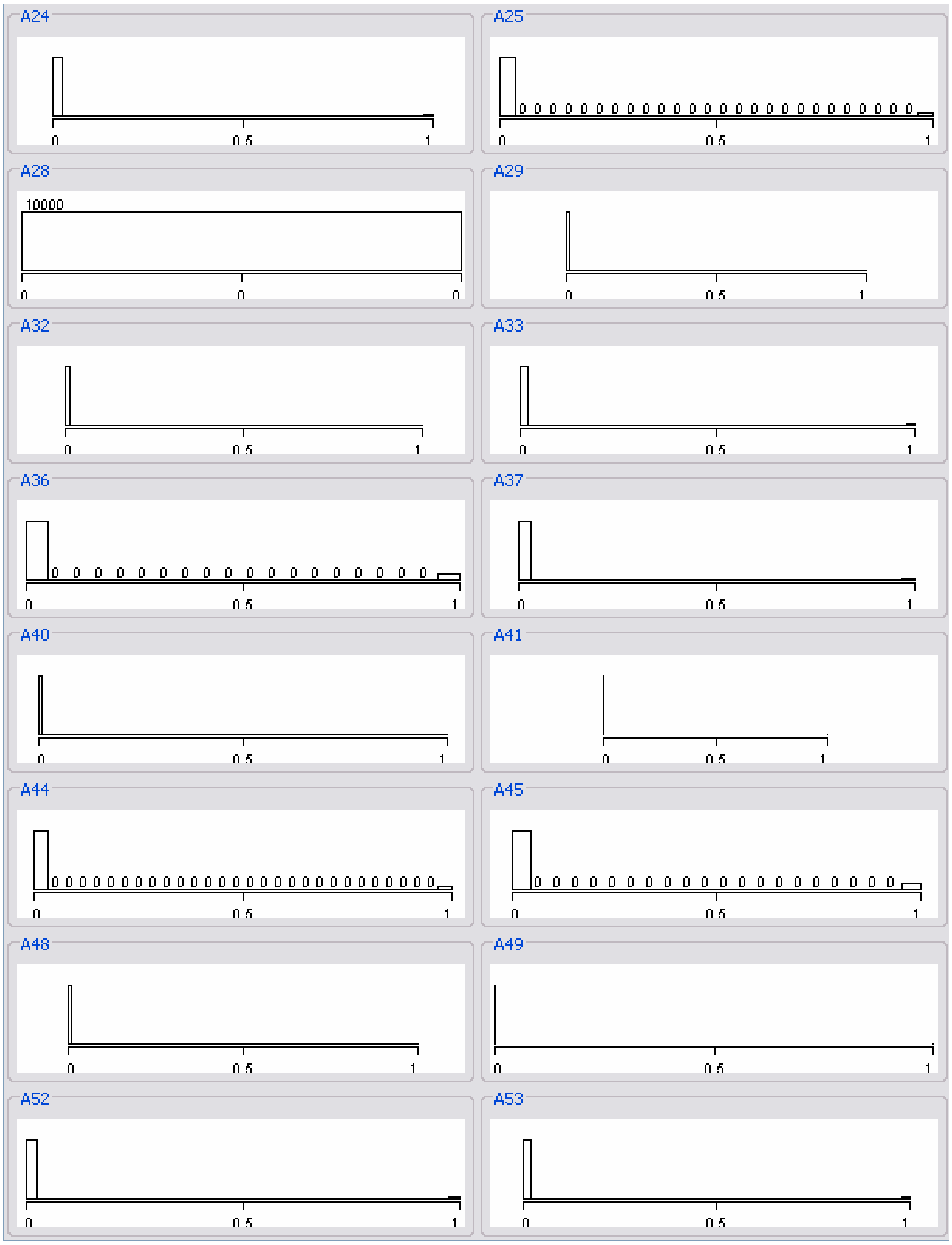
From previous analysis, we can see that support vector machine can be used for this project since they can give us the acceptable results.

We reduced the attributes from 54 to 21 which can help people to plant the trees depending on these fewer attributes to green the earth.

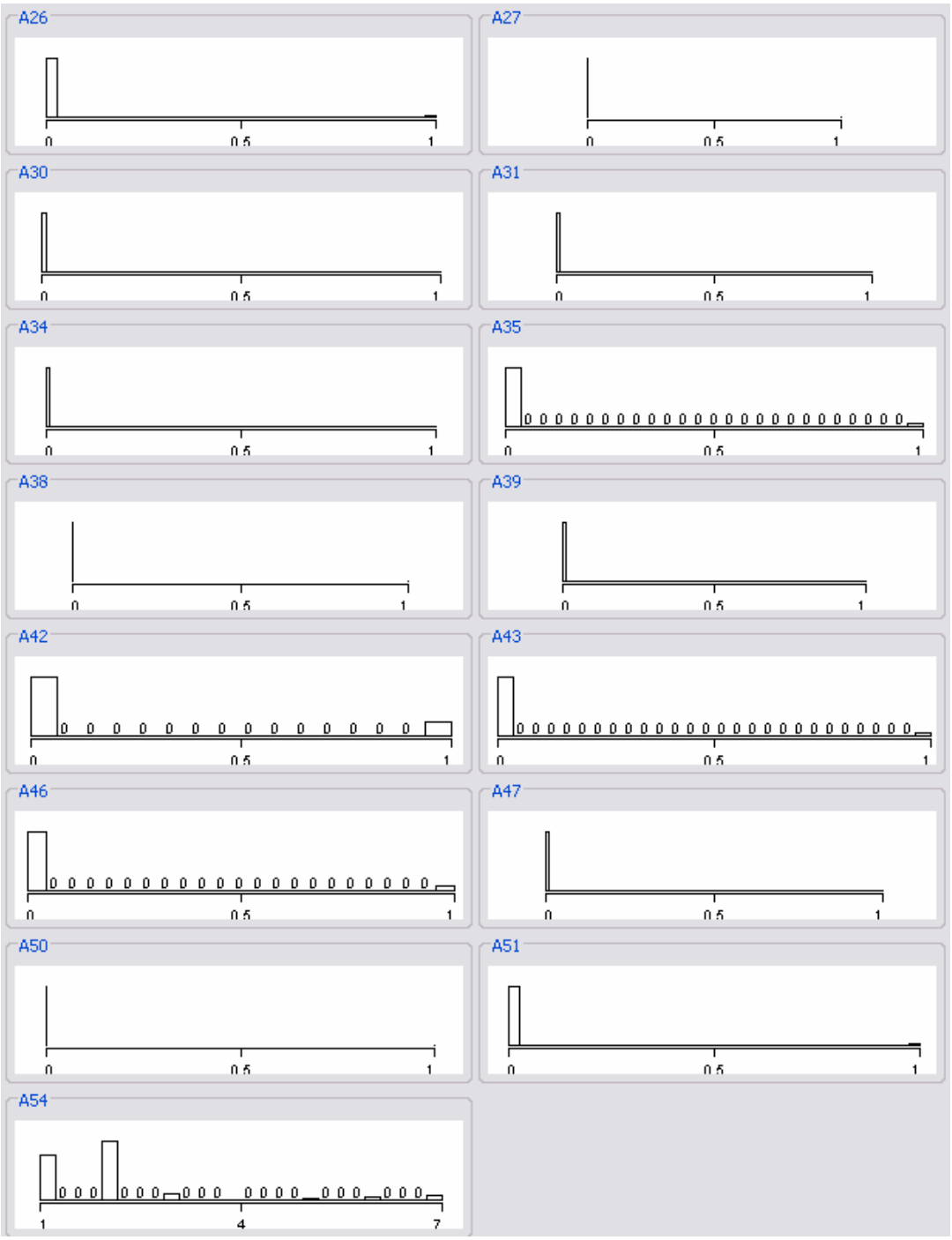
Appendix A – Data visualization




A2
0
31
62
A3
0
647
1294
A6
66
160
254
A7
0
127
254
A10
0 0 0 0 0 0 0 0 0 0
0
0.5
1
A11
0
0.5
1
A14
0
0.5
1
A15
0
0.5
1
A18
0
0.5
1
A19
0
0.5
1
A22
0
0.5
1
A23
0
0.5
1



A24
A25
A28
10000
A29
A32
A33
A36
A37
A40
A41
A44
A45
A48
A49
A52
A53


A26
A27
A30
A31
A34
A35
A38
A39
A42
A43
A46
A47
A50
A51
A54

Name	Data Type	Measurement	Description
Elevation	quantitative	meters	Elevation in meters
Aspect	quantitative	azimuth	Aspect in degrees azimuth
Slope	quantitative	degrees	Slope in degrees
Horizontal_Distance_To_Hydrology	quantitative	meters	Horz Dist to nearest surface water features
Vertical_Distance_To_Hydrology	quantitative	meters	Vert Dist to nearest surface water features
Horizontal_Distance_To_Roadways	quantitative	meters	Horz Dist to nearest roadway
Hillshade_9am	quantitative	0 to 255 index	Hillshade index at 9am, summer solstice
Hillshade_Noon	quantitative	0 to 255 index	Hillshade index at noon, summer soltice
Hillshade_3pm	quantitative	0 to 255 index	Hillshade index at 3pm, summer solstice
Horizontal_Distance_To_Fire_Points	quantitative	meters	Horz Dist to nearest wildfire ignition points
Wilderness_Area (4 binary columns)	qualitative	0 (absence) or 1 (presence)	Wilderness area designation
Soil_Type (40 binary columns)	qualitative	0 (absence) or 1 (presence)	Soil Type designation
Cover_Type (7 types)	integer	1 to 7	Forest Cover Type designation

Code Designations

Wilderness Areas:

1 -- Rawah Wilderness Area
2 -- Neota Wilderness Area
3 -- Comanche Peak Wilderness Area
4 -- Cache la Poudre Wilderness Area

Soil Types:

1 to 40 : based on the USFS Ecological Landtype Units for this study area.

Forest Cover Types:

1 -- Spruce/Fir
2 -- Lodgepole Pine
3 -- Ponderosa Pine
4 -- Cottonwood/Willow
5 -- Aspen
6 -- Douglas-fir
7 -- Krummholz

References

[1] http://kdd.ics.uci.edu/databases/covertype/covertype.html
[2] Coharian & Grossman, Data Preprocessing, Illinois Institute of Technology, 2003
[3] http://www-ccs.ucsd.edu/matlab/toolbox/nnet/premnmx.html
[4] http://www.statsoft.com/textbook/stsvm.html
[5] http://www.csie.ntu.edu.tw/~cjlin/libsvm/
[6] CHil-Wei Hsu, Chih-Chung Chang, and Chih-Jen Lin, *A Practical Guide to Support Vector Classification*
[7] Matlab neural network help document
[8] http://kdd.ics.uci.edu/databases/covertype/covertype.data.html

CPSC 661
Machine Learning

Prediction Accuracy Validation

Brenan Mackas

Table of Contents

Astract

This paper discusses a method for identifying under what criteria a prediction result is more or less accurate. The approach taken was to train various neural networks on the data used to in the prediction as well as the accuracy statistics for each of these predictions. Analysis of the ability for a neural network to correctly or incorrectly predict the accuracy of the prediction method being evaluated provides insight into the validity of the evaluated as well as an opportunity to determine under what particular conditions the prediction method has optimal and/or worst case results.

Introduction

Whenever one develops a system for making predictions, the accuracy of these predictions is the primary concern. Accuracy is generally given as a simple percent accuracy, and while this statistic can give a rough guess as to how accurate any given prediction is, it does nothing to identify under what criteria the prediction is especially good or bad. It also does not verify that the high expected accuracy rate of the predictions is in fact correct, rather than simply caused by a number of lucky guesses.

The original goal of this method (here on referred to as the Neural Network Verification Method or NNVM) was going to be to establish under what circumstances a weather prediction for a given area is most likely to be correct. However, when archival weather forecasts were un-attainable the application of NNVM for this project moved on to the verification of a software project time estimation system called AQUA [aqua]. AQUA takes an existing base of software projects with known completion times, and based on similarities to these existing projects predicts the effort required to complete a new project. For AQUA, in addition to attempting to identify which attributes most effect the accuracy of the result, analysis of the results will provide insight into whether the success of AQUA is legitimate or somewhat attributed to luck.

The application of the NNVM to AQUA required three main steps: Extraction and preprocessing of data; creation, training, and testing of neural networks, and finally the extraction, processing and analysis of the results. All data processing was done with ruby scripts and Java Neural Network Simulator (JNNS) was used for all the neural network tasks.

AQUA System and Datasets

AQUA is an analogy based effort estimation method that uses both case-based reasoning and collaborative filtering in an attempt to accurately estimate the effort required to complete a

software development effort. When supplied with the development project attributes, AQUA searches its base of software development projects for the most similar projects and gives an estimation of completion time based on the completion time of these similar projects.

Two tested datasets that AQUA had been tested on were supplied; the first USP05 (University Students Projects 2005) was an internal dataset containing 76 different projects, each with 15 different attribute fields. These attribute fields varied from abstract concepts such as the internal complexity of the project to more specific attributes like development tools. The second dataset, Mends03 was an external dataset and contained 34 software projects, each with 8 different attribute fields.

Implementation Process

Data Pre-Processing

Throughout this project, XML was used as the data storage format of choice, the reasons for choosing XML include both the widely available support for processing XML (a concern for anyone else who might like to attempt some data mining on the results data), and because of the fact it allows the use of either xpath or xquery to aid in processing the data.

Both datasets were obtained in Excel ".xls" format, after converting these files to an intermediate XML format, the data needed to be converted to JNNS's ".pat" data format. Details about the actual conversion process from XML to the pattern files can be found within the conversion scripts and will be omitted in this paper; however the normalization techniques used to obtain inputs appropriate for a neural network are explained below:

Null Values

Null values had two different occurrence types within the datasets: The first was within attributes representing a numeric quantifying value. In this case the null value was replaced with a 0. The second type of null value was as within attributes most concisely described as presence declaration attributes, that is the attributes specified the inclusion of a given property within that project (an example of this type of null entry would be the DBMS field of USP05, where if no database was used null could be listed as the DBMS used). These presence declaration attributes are discussed more in depth in the text normalization subsection.

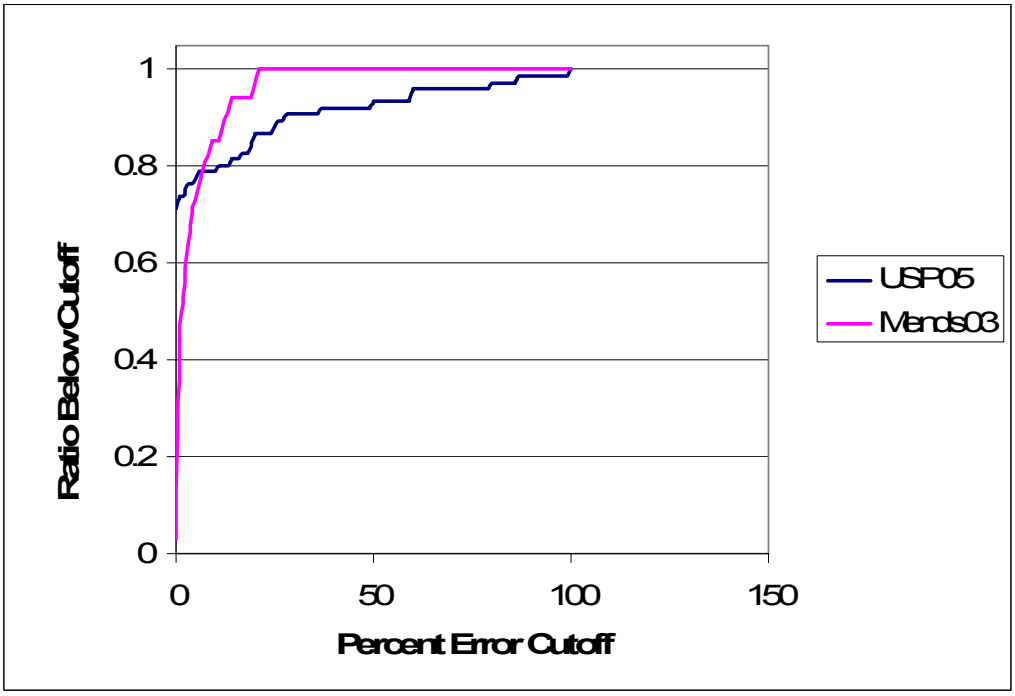
Missing Values

Some of the numeric quantifying attributes contained an inconsistent number of values throughout the dataset. In order to create a uniform number of inputs for the neural networks, attributes that contained less than the maximum number of values has 0 inserted for each missing value (an example of where this occurred was within the team size attribute in which if there were multiple teams of people working on a project, multiple team sizes were listed).

Neural Network Output Targets

Two different ideas for representing the correctness of a project effort estimation during the training of the neural networks. The first was to simple set a single output as the percent error of

the effort estimation. The second method was to use two outputs, one for correct and one for incorrect. Correct is defined by the percent error of the effort estimation falling below a certain cutoff point, all values less than or equal to the cutoff are assigned the output "*1 0*" and all projects with percent error greater than the cutoff are assigned an output of "*0 1*". See figure 1 for a graph showing the ratio of projects below the cutoff percentage versus the cutoff percentage; as can be seen both datasets had a large majority of their projects with a percent error of less than 5%. All networks trained using the cutoff dual output method were trained on percent error cutoffs of 1%, 3%, 5%, 10%, and 15%. These cutoff points were chosen due to the bias towards low percent error within the datasets.



Ratio of Projects Below Cutoff vs. Cutoff Percentage
Figure 1

Text Normalization

All string values were first down cased and then checked for accidental misspellings. As neural networks only accept numeric inputs, all attributes containing string values need to be changed to some numeric input format. For a given attribute with string values, each unique normalized value was assigned a unique number. Once all unique values had been assigned an id number, a neural network compatible input format for that attribute was created by creating a binary on/off input for each of the possible attribute values. For example if a field has possible values of *"a","b","c"* and a given project only contained "a" and "c", the input for this attribute for this project would be "*1 0 1*". Using this type of input for presence declaration values also takes care of the neural network requirement that all input sequences be the same length.

With the data normalized and the JNNS pattern files generated, the next step is to train and test the neural networks.

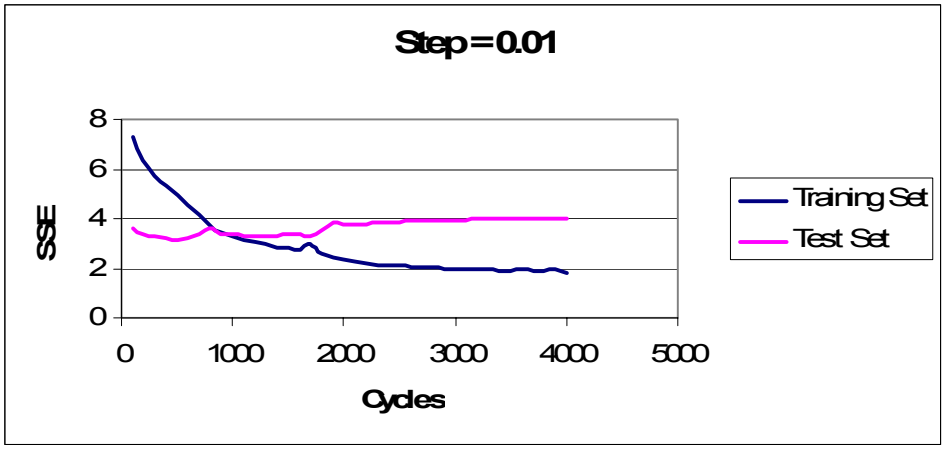
Training and Testing Networks

JNNS requires the user to manually create and train each network individually, as such it was decided that each dataset would be trained on three different neural network configurations all of which were standard feed-forward back-propagation networks. After normalization the Mends03 dataset resulted in eight inputs. The three different neural network configurations trained and tested for this dataset. The first has a single layer of 15 hidden nodes; the second had a single hidden layer of 50 nodes, and the third two hidden layers with 20 nodes each.

The USP05 projects resulted in 66 inputs each. Again three different network configurations were used: A single hidden layer of 75 nodes, a single hidden layer of 150 nodes, and two hidden layers each with 70 nodes.

For both project sets each all three networks configurations was trained twice (with different random initializations) for each of the different cutoff values as well as another pair for the percent error output format. A step width of 0.2 was used in both the dual output and single output case, and a maximum non-propagated error of 0.1 was used in the case of the dual outputs, and 0.02 in the case of the single percent error output (the reduction of max. non-propagated error was needed in this case as due to the very low variation in output values, the network converged to zero sum square error too quickly otherwise, and hence did not properly train).

There were some problems with the neural networks not converging well during training, especially with the Mends03 dataset. This is likely due to how general the input data is for this dataset. Attempts to tune the training process to obtain better convergence on the training data resulted in over fitting the data to the training set, and actually hurting the performance of the test/verification set. A graph showing this over fitting is given in Figure 2.



Overtraining of the Training Set Decreasing Accuracy of the Test Set
Figure 2

Once each network was trained it was tested both on its training set and testing/verification set; results for both of these were saved along with the network and a log containing the Standard Square Error progression of both data sets during training.

Results and Analysis

Results data is contained in Appendix A for the Mends03 project set and Appendix B for the USP05 project set. A more complete analysis including the correct positive, false positive, correct negative, and false negative for each value of each attribute was done, however the resulting was excluded from the paper for space reasons. That data can be found within the spreadsheets included with the source code of this project. In the data files there are four types of classification: Correct Positive, Incorrect Positive, Correct Negative, and Incorrect Negative. For further clarification a Correct classification under the NNVM is one in which the experimental result agrees with the result in the data file (i.e. a successful prediction of accuracy), an incorrect classification is one in which the experimental result disagrees with the result in the data file (i.e. neural network guessed AQUA would be able to correctly estimate the effort, but AQUA did not, this would be an incorrect positive).

For the most part the neural networks were unable to predict the accuracy of AQUA. In most situations the networks were simply guessing that the estimated result would be positive every time (this can be seen through the majority of correct positives and incorrect negatives). This is especially true of the Mends03 project set in which the networks were quite consistently outputting the same thing for every project in the test/verification set. The lack of the success in the Mends03 project set is likely due to the very general attributes, as well as the small training dataset size (lack of conversion was also a consequence of this).

With the USP05 project set much of the same behavior is observed, however there are several stand out cases in which the networks were able to correctly guess the success of the effort estimation in not only the positive cases but also a number of the negative cases as well. This hints that the NNVM was able to learn some of the estimation behavior of AQUA, and as such that in the case of USP05 there may be more than luck involved. The belief that more than luck is involved for USP05 is further enhanced by analyzing the individual attribute value scores. Most of the attribute values with high occurrence count attributes are very heavily sided to either correct or incorrect. A more in-depth analysis of the attribute value statistics would provide more insight into this.

Problems Encountered

JNNS

JNNS seems to be better suited to training a small number neural networks with a large dataset. The lack of any scripting functionality combined with the fact you may only have one network open at a time equates to a large amount of time spent wasted sitting and clicking when training a large number of networks. The other functionality complaint I have is that by default it tests the network with the training set when saving the results, requiring the user to constantly change back and forth between the two datasets. The second issue also would be substantially mitigated when working in less of a batch training environment.

A better choice for this project would likely have been something like Weka that allows for full automation through the API. This functionality would greatly increase one's ability to test many, many more different neural network configurations.

Datasets

It would be very interesting to see the results of this experiment run again with the same datasets if they had many more entries. Because of the generally of much of the dataset attributes, I expect that part of the reason for the often seen lack of convergence was due to the fact that given that high level of generality there simply was not enough examples for the neural network to properly classify the data (this is particularly evident in the Mends03 dataset).

Conclusion and Future Work

This paper presented a method for both evaluating the legitimacy of any prediction method as well as help to identify what attributes and attribute values have the most adverse effects on the accuracy of the prediction method. The Neural Network Validation Method is completely generically extensible to evaluate any classification/prediction system with the small exception of that the neural networks need to be created to fit the dataset, however this could easily be scripted in the future.

The NNVM was used to evaluate the results of AQUA, an analogy based effort estimation system, achieving mixed results. In the case of the USP05 project set while a large portion of the results appear to be arbitrary, there are some trends that lend credit to AQUA's evaluation method.

Due to time constrains there was not the opportunity of run a clustering algorithm or in depth data mining techniques on the statistical data for each individual field value, and to this end extensive statistic gathering has already been performed and is more or less ready waiting for the next step. Doing this would provide a much more accurate view of which fields/groups adversely affect the accuracy of the estimated effort.

Bibliography

[aqua] Jingzhou Li, Günther Ruhe, Ahmed Al-Emran, Michael M. Richter: A flexible method for software effort estimation by analogy. Empirical Software Engineering, Volume 12, Number 1, February 2007, 65-106

Michael M. Richter, Neural Networks 1
http://pages.cpsc.ucalgary.ca/~mrichter/ML/neural%20nets%201.pdf

Michael M. Richter, Neural Networks 2
http://pages.cpsc.ucalgary.ca/~mrichter/ML/neural%20nets%202.pdf

Baldi, P. and Hornik, K. 1989. Neural networks and principal component analysis: learning from examples without local minima. Neural Netw. 2, 1 (Jan. 1989), 53-58. DOI= http://dx.doi.org/10.1016/0893-6080(89)90014-2

SNNS user manual
http://www-ra.informatik.uni-tuebingen.de/SNNS/UserManual/UserManual.html

Appendix A

Mends03 Results

Cutoff = Cutoff Percentage
CP = Correct Positive
IP = Incorrect Positive
CN = Correct Negative
IN = Incorrect Negative

Cutoff	Network	Set	CP	IP	CN	IN
1	mends_8_15_2a	Training	0	12	14	0
1	mends_8_15_2a	Verification	0	3	5	0
1	mends_8_15_2b	Training	0	12	14	0
1	mends_8_15_2b	Verification	0	3	5	0
1	mends_8_20_20_2a	Training	0	12	14	0
1	mends_8_20_20_2a	Verification	0	3	5	0
1	mends_8_20_20_2b	Training	0	12	14	0
1	mends_8_20_20_2b	Verification	0	3	5	0
1	mends_8_50_2a	Training	12	0	0	14
1	mends_8_50_2a	Verification	3	0	0	5
1	mends_8_50_2b	Training	2	10	13	1
1	mends_8_50_2b	Verification	0	3	5	0
3	mends_8_15_2a	Training	17	0	0	9
3	mends_8_15_2a	Verification	5	0	0	3
3	mends_8_15_2b	Training	17	0	0	9
3	mends_8_15_2b	Verification	5	0	0	3
3	mends_8_20_20_2a	Training	17	0	3	6
3	mends_8_20_20_2a	Verification	5	0	0	3
3	mends_8_20_20_2b	Training	17	0	0	9
3	mends_8_20_20_2b	Verification	5	0	0	3
3	mends_8_50_2a	Training	17	0	1	8
3	mends_8_50_2a	Verification	5	0	0	3
3	mends_8_50_2b	Training	17	0	1	8
3	mends_8_50_2b	Verification	5	0	0	3
5	mends_8_15_2a	Training	20	0	1	5
5	mends_8_15_2a	Verification	5	0	0	3
5	mends_8_15_2b	Training	20	0	1	5
5	mends_8_15_2b	Verification	5	0	0	3
5	mends_8_20_20_2a	Training	20	0	2	4
5	mends_8_20_20_2a	Verification	5	0	0	3
5	mends_8_20_20_2b	Training	20	0	1	5
5	mends_8_20_20_2b	Verification	5	0	0	3

Cutoff	Network	Set	CP	IP	CN	IN
5	mends_8_50_2a	Training	20	0	0	6
5	mends_8_50_2a	Verification	5	0	0	3
5	mends_8_50_2b	Training	20	0	1	5
5	mends_8_50_2b	Verification	5	0	0	3
10	mends_8_15_2a	Training	22	0	0	4
10	mends_8_15_2a	Verification	7	0	0	1
10	mends_8_15_2b	Training	22	0	0	4
10	mends_8_15_2b	Verification	7	0	0	1
10	mends_8_20_20_2a	Training	22	0	0	4
10	mends_8_20_20_2a	Verification	7	0	0	1
10	mends_8_20_20_2b	Training	22	0	1	3
10	mends_8_20_20_2b	Verification	7	0	0	1
10	mends_8_50_2a	Training	22	0	1	3
10	mends_8_50_2a	Verification	7	0	0	1
10	mends_8_50_2b	Training	22	0	1	3
10	mends_8_50_2b	Verification	7	0	0	1
15	mends_8_15_2a	Training	25	0	0	1
15	mends_8_15_2a	Verification	7	0	0	1
15	mends_8_15_2b	Training	25	0	0	1
15	mends_8_15_2b	Verification	7	0	0	1
15	mends_8_20_20_2a	Training	25	0	0	1
15	mends_8_20_20_2a	Verification	7	0	0	1
15	mends_8_20_20_2b	Training	25	0	0	1
15	mends_8_20_20_2b	Verification	7	0	0	1
15	mends_8_50_2a	Training	25	0	0	1
15	mends_8_50_2a	Verification	7	0	0	1
15	mends_8_50_2b	Training	25	0	0	1
15	mends_8_50_2b	Verification	7	0	0	1

Appendix B

USP05 Results

Cutoff = Cutoff Percentage
CP = Correct Positive
IP = Incorrect Positive
CN = Correct Negative
IN = Incorrect Negative

Cutoff	Network	Set	CP	IP	CN	IN
1	USP05_66_150_2a	Training	41	1	12	0
1	USP05_66_150_2a	Verification	11	1	2	4
1	USP05_66_150_2b	Training	41	1	12	0
1	USP05_66_150_2b	Verification	12	0	2	4
1	USP05_66_70_70_2a	Training	41	1	12	0
1	USP05_66_70_70_2a	Verification	12	0	1	5
1	USP05_66_70_70_2b	Training	41	1	12	0
1	USP05_66_70_70_2b	Verification	12	0	1	5
1	USP05_66_75_2a	Training	41	1	12	0
1	USP05_66_75_2a	Verification	12	0	0	6
1	USP05_66_75_2b	Training	42	0	12	0
1	USP05_66_75_2b	Verification	12	0	0	6
3	USP05_66_150_2a	Training	41	1	12	0
3	USP05_66_150_2a	Verification	14	0	0	4
3	USP05_66_150_2b	Training	41	1	12	0
3	USP05_66_150_2b	Verification	14	0	0	4
3	USP05_66_70_70_2a	Training	41	1	12	0
3	USP05_66_70_70_2a	Verification	12	2	0	4
3	USP05_66_70_70_2b	Training	41	1	12	0
3	USP05_66_70_70_2b	Verification	13	1	0	4
3	USP05_66_75_2a	Training	41	1	12	0
3	USP05_66_75_2a	Verification	14	0	0	4
3	USP05_66_75_2b	Training	41	1	12	0
3	USP05_66_75_2b	Verification	13	1	0	4
5	USP05_66_150_2a	Training	43	0	11	0
5	USP05_66_150_2a	Verification	14	0	1	3
5	USP05_66_150_2b	Training	42	1	11	0
5	USP05_66_150_2b	Verification	12	2	0	4
5	USP05_66_70_70_2a	Training	42	1	11	0

Cutoff	Network	Set	CP	IP	CN	IN
5	USP05_66_70_70_2a	Verification	13	1	0	4
5	USP05_66_70_70_2b	Training	42	1	11	0
5	USP05_66_70_70_2b	Verification	13	1	1	3
5	USP05_66_75_2a	Training	42	1	11	0
5	USP05_66_75_2a	Verification	12	2	0	4
5	USP05_66_75_2b	Training	42	1	11	0
5	USP05_66_75_2b	Verification	14	0	0	4
10	USP05_66_150_2a	Training	43	1	10	0
10	USP05_66_150_2a	Verification	13	1	1	3
10	USP05_66_150_2b	Training	43	1	10	0
10	USP05_66_150_2b	Verification	13	1	0	4
10	USP05_66_70_70_2a	Training	43	1	10	0
10	USP05_66_70_70_2a	Verification	13	1	1	3
10	USP05_66_70_70_2b	Training	43	1	10	0
10	USP05_66_70_70_2b	Verification	13	1	0	4
10	USP05_66_75_2a	Training	43	1	10	0
10	USP05_66_75_2a	Verification	14	0	0	4
10	USP05_66_75_2b	Training	44	0	10	0
10	USP05_66_75_2b	Verification	14	0	1	3
15	USP05_66_150_2a	Training	46	0	8	0
15	USP05_66_150_2a	Verification	13	1	1	3
15	USP05_66_150_2b	Training	46	0	8	0
15	USP05_66_150_2b	Verification	14	0	1	3
15	USP05_66_70_70_2a	Training	46	0	8	0
15	USP05_66_70_70_2a	Verification	14	0	1	3
15	USP05_66_70_70_2b	Training	40	6	8	0
15	USP05_66_70_70_2b	Verification	13	1	2	2
15	USP05_66_75_2a	Training	41	5	8	0
15	USP05_66_75_2a	Verification	12	2	2	2
15	USP05_66_75_2b	Training	46	0	8	0
15	USP05_66_75_2b	Verification	14	0	0	4

Bayesian Spam Filtering

Ahmed Obied
Department of Computer Science
University of Calgary
amaobied@ucalgary.ca
http://www.cpsc.ucalgary.ca/~amaobied

Abstract. With the enormous amount of spam messages propagating on the Internet these days, anti-spam researchers and developers are trying to build spam filters to get rid of such unsolicited messages as accurately as possible. In this paper, I describe a machine learning approach based on Bayesian analysis to filter spam. The filter learns how spam and non-spam messages look like, and is capable of making a binary classification decision (spam or non-spam) whenever a new email message is presented to it. The evaluation of the filter shows its ability to make decisions with high accuracy (96.24% in the worst case and 99.66% in the best case).

Keywords: Machine Learning, Text Classification, Spam

1 Introduction

In today's highly technical world and our computer-connected society, email has become the fastest and most economical form of communication available. The widespread use of email enticed direct marketers to bombard unsuspecting email inboxes with unsolicited messages regarding everything from items for sale and get-rich-quick schemes to information about accessing pornographic Web sites [8]. The volume of these unsolicited messages (referred to as spam messages) has grown tremendously in the past few years. Many anti-spam researchers believe that spam is responsible for anywhere from 35% to 65% of all email traffic on the Internet today, with a whopping annual growth rate of 15% to 20% [11]. Many are concerned that spam could be the end of email [11].

A precise definition for spam does not exist since spam is capable of evolving. To many, spam is defined as unwanted email messages. To others, spam is defined as unsolicited commercial email (UCE) or unsolicited bulk email (UBE). In the past few years, a number of anti-spam techniques have been proposed and used to combat spam such as whitelisting, blacklising, and greylisting of domain names, keyword-based filtering, heuristic-based filtering, etc. All of these techniques, however, require heavy maintenance and cannot achieve very high overall accuracy. The best heuristic-based spam filter available on the Internet today cannot achieve more than a 95% accuracy. Moreover, such accuracy drops substantially when spam evolves. To overcome the disadvantages of the anti-spam techniques described above and achieve very high accuracy, one can use machine learning techniques. Spam filters that are based on machine learning techniques are capable of learning, let the user define what spam is and generate few false positives. Moreover, such filters adapt easily with the way spam evolves and are capable of achieving an accuracy rate higher than 95% [11] if the filter is well trained.

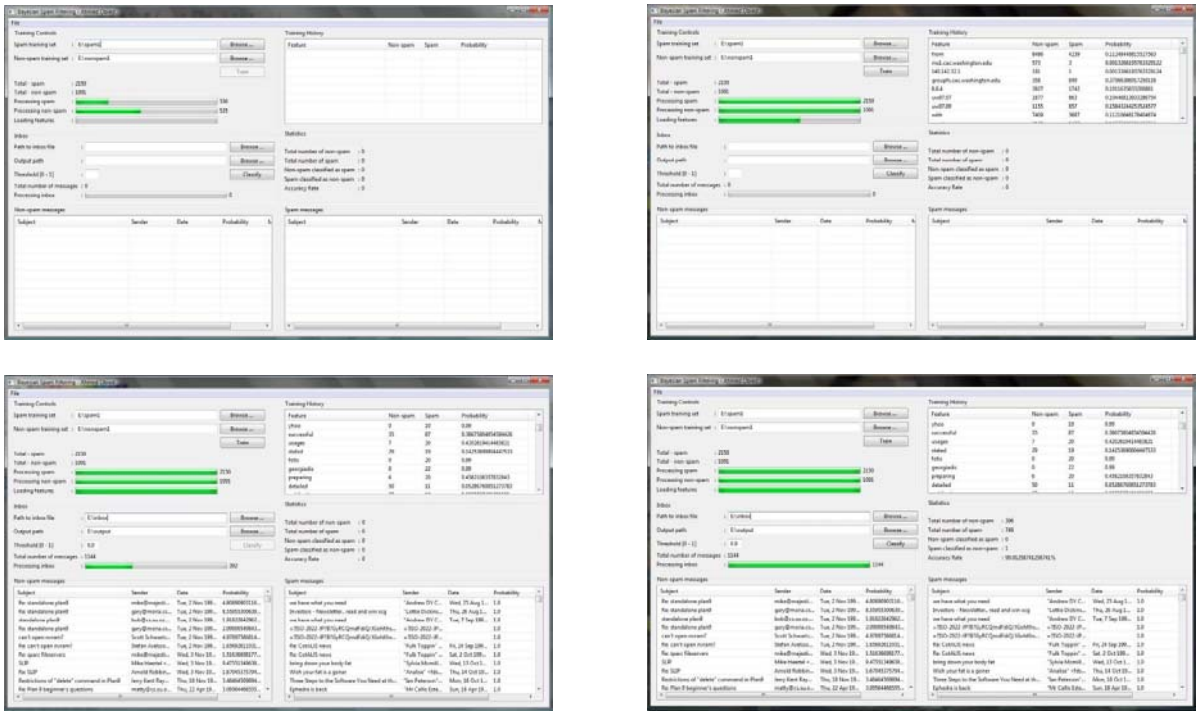


Fig. 1. Spam Filter

2 Approach

The process of identifying the disposition of a presented text into a particular category is known as text classification [11]. I implemented a statistical text classifier (filter) in Java that is based on Bayesian analysis. The filter classifies an email message into one of two categories: spam or non-spam. I used a supervised learning approach to enable the filter to build a history of what spam and non-spam messages look like. Figures 1 shows screen shots of the filter that was implemented. Give two classes $C = \{c_1 = spam, c_2 = non - spam\}$ and features $f_1, f_2, \ldots, f_n$, the probability that these features belong to a certain class using naive Bayesian can be expressed as follows:

$$P(C|x_1, \ldots, x_n) = \frac{P(x_1, \ldots\_x\_n|C)P(C)}{P(x_1, \ldots, x_n)}$$

Assuming conditional independence, one can compute $P(x_1, \ldots, x_n|C)$ as follows:

$$P(x_1, \ldots, x_n|C) = \prod_{i=1}^{n} P(x_i|C)$$

To classify an email message as spam, one can check if it exceeds a specific threshold:

$$\frac{P(c_1 = spam \mid x_1, \ldots, x_n)}{P(c_2 = non - spam \mid x_1, \ldots, x_n)} \geq \beta$$

2.1 Dataset

I found a number of public spam corpora that can be used to train and test a spam filter. The corpus I used to train the filter and test it is the TREC 2006 Spam Track Public Corpora [1] since it contains a large number of email messages. The corpus contains 37,822 email messages; 12,910 are non-spam messages and 24,912 are spam messages.

The problem with the TREC corpus is that the non-spam and spam messages are not separated into two different directories. Both types of messages are placed in the same directory. The only way to know if a message is spam or non-spam is by looking at an index file which has the message number and its type. Since having the spam and non-spam messages in two separate directories can make the training and testing easier, I decided to write a program in C to move the spam messages into a spam directory and the non-spam messages into a non-spam directory. Every time a message is moved to a specific directory, the program re-writes the file name of the message to include a letter in it (n for non-spam messages and s for spam messages). These letters in the file names are used later in the testing phase to identify the messages that have been classified correctly and incorrectly. The program also checks if a message is encoded in base-64 and drops it if it is indeed encoded in base-64. Base-64 is used to encode email attachments. Since such encoding does not increase the effectiveness of the filter during the training phase but can decrease it, I decided to drop such messages.

After processing the corpus, I ended up with a total of 36,391 email messages; 12,594 non-spam messages and 23,979 spam messages. I then divided the corpus into two corpora: a training corpus and a testing corpus. The training corpus contains 70% of the original corpus: 25,474 email messages; 8,816 non-spam messages and 16,658 spam messages. The testing corpus contains 30% of the original corpus: 10,917 email messages; 3,778 non-spam messages and 7,139 spam messages.

2.2 Feature Extraction

The feature extractor in the filter is used to extract features $vj = f_1, f_2, f_3, \ldots, f_n$ from an email message and construct feature vectors $V = < v_1, v_2, v_3, \ldots, v_j >$ that can be used in the classification phase. A feature can be anything in an email message. It can be a word, a phrase, a number, an HTML tag, etc. The way the features are represented can affect the filter's accuracy to some extent.

An email message contains two parts: a header and a body. The header consists of fields usually including at least the following:

– From: This field indicates the sender's email address.
– To: This field indicates the receiver's email address.
– Date: This field indicates the date and time of when the message was sent.
– Subject: This field indicates a brief summary of the message's content.
– Received: This field indicates the route the message took to go from the sender to the receiver.
– Message-ID: This field indicates the ID of the message. Every email message has a unique ID at a given domain name: id@senders-domain-name.

On the other hand, the body of an email message contains the actual content of the message. The header and the body of an email message is separated by an empty line. To extract features from an email message, I do the following:

1. Parse an email message and extract the header and the body parts.
2. Remove the fields from the header of the message since they appear in every email message.
3. Extract features from the header by tokenizing the header using the following delimiter:
   \n\f\r\t\ /&%# {}[]! +=-() ‘”*?:;<>
4. Extract features from the body by tokenizing the body using the following delimiter:
   \n\f\r\t\ ./&%# {}[]! +=-() ‘”*?:;<>
5. Ignore features of size strictly less than 3 and digits.

To store the features in the training phase, I use a hash table of nodes. Each node stores the following information:

– Number of occurrences in spam messages (initialized to 0).
– Number of occurrences in non-spam messages (initialized to 0).
– Probability (initialized to 0).

Every time a feature is extracted, I check if the email is spam or non-spam. If it is spam then the number of spam occurrences is incremented by one. Otherwise, if it is non-spam then the number of non-spam occurrences is incremented by one. After extracting all the features from the training set and filling the hash table with such features, I start enumerating through the elements of the hash table to compute the probability for each feature. To compute the probability P for a feature f, I use the following formula:

$$\frac{\frac{s}{t_s}}{\frac{s}{t_s}+\frac{kn}{t_n}}$$

s is the number of occurrences of feature f in spam messages, t_s is the total number of spam messages in the training set, n is the number of occurrences of feature f in non-spam messages, and t_n is the total number of non-spam messages in the training set. k is a number that can be tuned to reduce false positives by giving a higher weight to number of occurrences of non-spam features. In my implementations, I set k to 2. If a feature f only appears in a single corpus (spam or non-spam) then I assign it a specific value. If it appears only in spam messages then I assign it 0.99. On the other hand, if it appears only in non-spam messages then I assign it 0.01. All the features have probabilities between 0 and 1 exclusively. The closer the probability of a given feature is to 1, the more spammy such feature is and vice versa.

2.3 Classification

The analysis engine is the core of the filter. It is the part that applies Bayes theorem based on Graham’s assumption [4] that there is no prior knowledge of spam and non-spam messages, and

computes the probability distribution of features in a given email message. After training the filter and having all the necessary information stored in the hash table, the filter becomes capable of making decisions based on what it has seen in the training set. Every time an email message is presented to the filter in the classification phase, I do the following:

1. Parse an email message and extract the header and the body parts.
2. Extract features from the header and body using the feature extraction method described above.
3. Look up the features in the hash table and retrieve their corresponding probabilities. If a feature is not in the hash table then this means that it is a feature that the filter has not seen in the training set. In this case, I assign it a probability of 0.5. After looking up the features and retrieving their corresponding probabilities, I compute the interestingness of the features. The interestingness I_f for a feature f can be computed as follows:

$$I_f = |0.5 - P_f|$$

4. Extract the most interesting features from the list of features. To do that, I sort the list by interestingness in descending order and take the first N features. The value of N that I used is 15 since it has been shown to be the best choice [11].
5. Combine the probabilities of the N most interesting features using Bayes theorem and based on Graham's assumptions [4]:

$$P = \frac{P_1 P_2 \dots P_N}{P_1 P_2 \dots P_N + (1 - P_1)(1 - P_2) \dots (1 - P_N)}$$

At the end, I get P which is a value between 0 and 1. The closer P is to 0, the more likely the message is non-spam and the closer P is to 1, the more likely the message is spam. The threshold I used to determine whether a given email message is spam is 0.9, that is, if $P \geq 0.9$ then I classify the email message as spam otherwise I classify it as non-spam.

2.4 Testing

To make the testing of the classification easier, I included a built-in testing functionality in the filter. The filter takes as input a directory specified by the user that contains email messages (spam and non-spam). The output of the classification is a directory that is also specified by the user. When the filter starts the classification, two directories are created in the output directory: a spam directory and a non-spam directory. If the filter classifies a given email message as spam then the filter copies it from the input directory to the spam directory. Similarly, if the filter classifies a given email message as non-spam then the filter copies it from the input directory to the non-spam directory.

I mentioned earlier in the dataset section that every message has a letter in its file name: s if the email message is spam and n if the email message is non-spam. When the filter completes the classification process, it traverses the spam and non-spam directories in the output directory to count the total number of spam messages s_m that have been classified as non-spam and the total number of non-spam messages n_m that have been classified as spam. To calculate the total number of non-spam messages that have been classified as spam, the filter traverses the spam directory and increments a counter n_m (initialized to 0) by 1 every time it encounters an email

message that has the letter n in its file name. To calculate the total number of spam messages that have been classified as non-spam, the filter traverses the non-spam directory and increments a counter s_m (initialized to 0) by 1 every time it encounters an email message that has the letter s in its file name.

After calculating the values of n_m, s_m, and knowing the total number of email messages in the input directory t, I calculate the accuracy A of the filter in percentage using the following formula:

$$A = \frac{t - (n_m + s_m)}{t} * 100$$

2.5 Evaluation

To evaluate the filter's performance, I performed the following test cases:

Test Case 1 The inbox contains 5,000 email messages: 2,500 non-spam messages randomly chosen from the training set and 2,500 spam messages randomly chosen from the training set.

- Total number of email messages: 5,000
- Total number of non-spam messages: 2,500
- Total number of spam messages: 2,500
- Total number of email messages classified as non-spam: 2,517
- Total number of email messages classified as spam: 2,483
- Total number of non-spam messages classified as spam messages: 0
- Total number of spam messages classified as non-spam messages: 17
- *Accuracy: 99.66%*

Test Case 2 The inbox contains 5,000 messages: 1,250 non-spam messages randomly chosen from the training set, 1,250 non-spam messages randomly chosen from the testing set, 1,250 spam messages randomly chosen from the training set, and 1,250 spam messages randomly chosen from the testing set.

- Total number of email messages: 5,000
- Total number of non-spam messages: 2,500
- Total number of spam messages: 2,500
- Total number of email messages classified as non-spam: 2,549
- Total number of email messages classified as spam: 2,451
- Total number of non-spam messages classified as spam messages: 27
- Total number of spam messages classified as non-spam messages: 76
- *Accuracy: 97.94%*

Test Case 3 The inbox contains 5,000 messages: 2,500 non-spam messages randomly chosen from the testing set and 2,500 spam messages randomly chosen from the testing set.

- Total number of email messages: 5,000
- Total number of non-spam messages: 2,500
- Total number of spam messages: 2,500
- Total number of email messages classified as non-spam: 2,606

- Total number of email messages classified as spam: 2,394
- Total number of non-spam messages classified as spam messages: 41
- Total number of spam messages classified as non-spam messages: 147
- *Accuracy: 96.24%*
-

Test Case 4 The inbox contains 5,000 messages: 2,500 non-spam messages randomly chosen from the testing set, 2,500 spam messages randomly chosen from the testing set.

- Total number of email messages: 5,000
- Total number of non-spam messages: 2,500
- Total number of spam messages: 2,500
- Total number of email messages classified as non-spam: 2,590
- Total number of email messages classified as spam: 2,410
- Total number of non-spam messages classified as spam messages: 20
- Total number of spam messages classified as non-spam messages: 110
- *Accuracy: 97.39%*

2.6 Challenges

The main challenge that I found while working on the project was dealing with memory. When you train the filter with enormous amount of email messages, storing all the features in memory might not be the best solution. At first, I was running out of heap space. However, I tried increasing the heap size (which can be done in Java at run time using the -Xms and -Xmx flags) and everything worked fine. Although I tried using a database (MySQL) to store the features at the beginning of my implementation, the access time required to store or retrieve a feature was taking a long time which made me end up storing everything in main memory. In the future, if one wants to deal with large training datasets then the best solution would be to store everything on disk and use a technique similar to paging. A hash table of size n can be stored in main memory. If the number of features in the hash table reaches n then the data in the table can be swapped to disk. The data on disk can be loaded to main memory in a FIFO fashion any time the filter tries to classify a given email message. Although loading the data from disk to main memory and then accessing main memory is slower than accessing main memory directly, it is definitely better and faster than storing the data in a database and querying it.

2.7 Conclusion

More than half of all email traffic we see on the Internet these days is spam. Although a number of anti-spam techniques have been used to counter spam, none of these techniques have the potential to deal with the way spam evolves over time except the anti-spam techniques that are based on machine learning approaches. These techniques are essentially text classifiers but instead of classifying a given text into different categories, they classify a given text (email message) into two categories (spam or non-spam).

In this paper, I described a machine learning approach based on Bayesian analysis to filter spam. The filter learns of what spam and non-spam messages look like and can make binary classification decisions (spam or non-spam) based on what it has learned. The filter does not require any heavy maintenance. All what you need to do is train it once and you are done. After

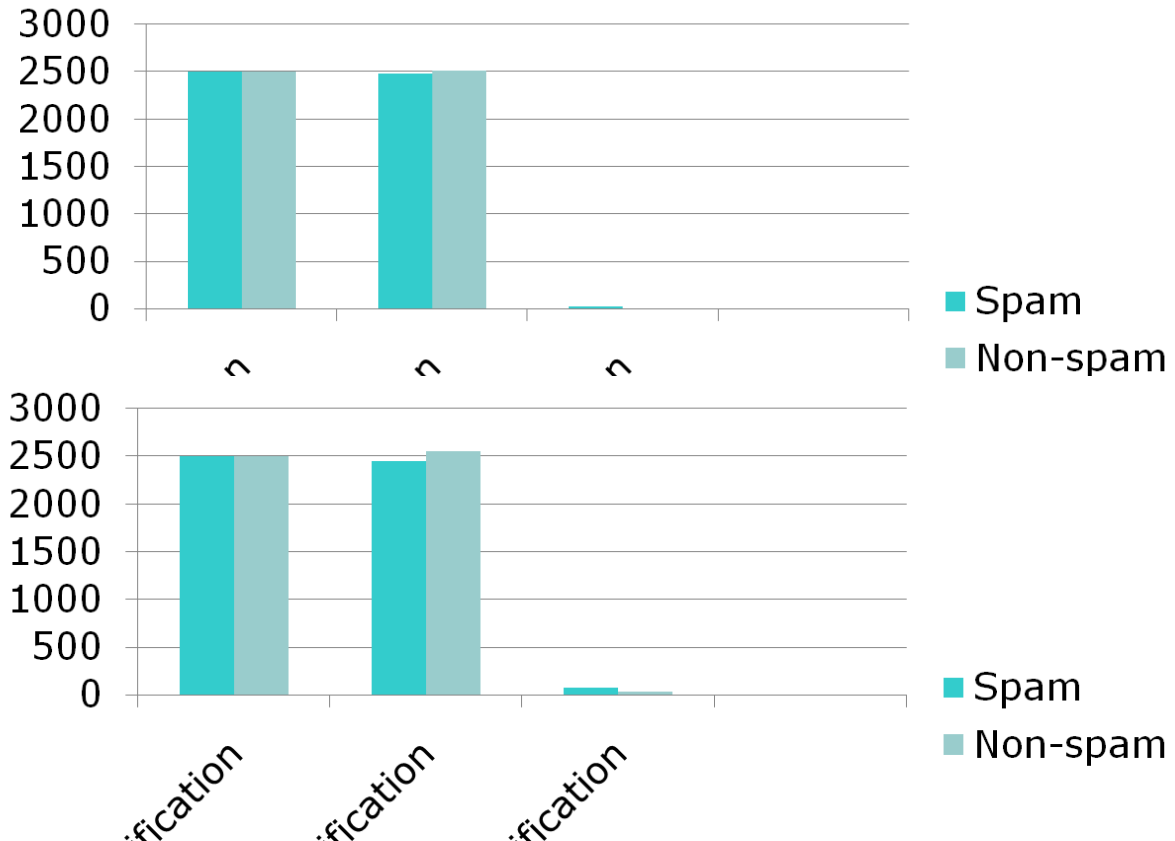
training the filter, it becomes capable of filtering spam with high accuracy as shown in the evaluation section.

References

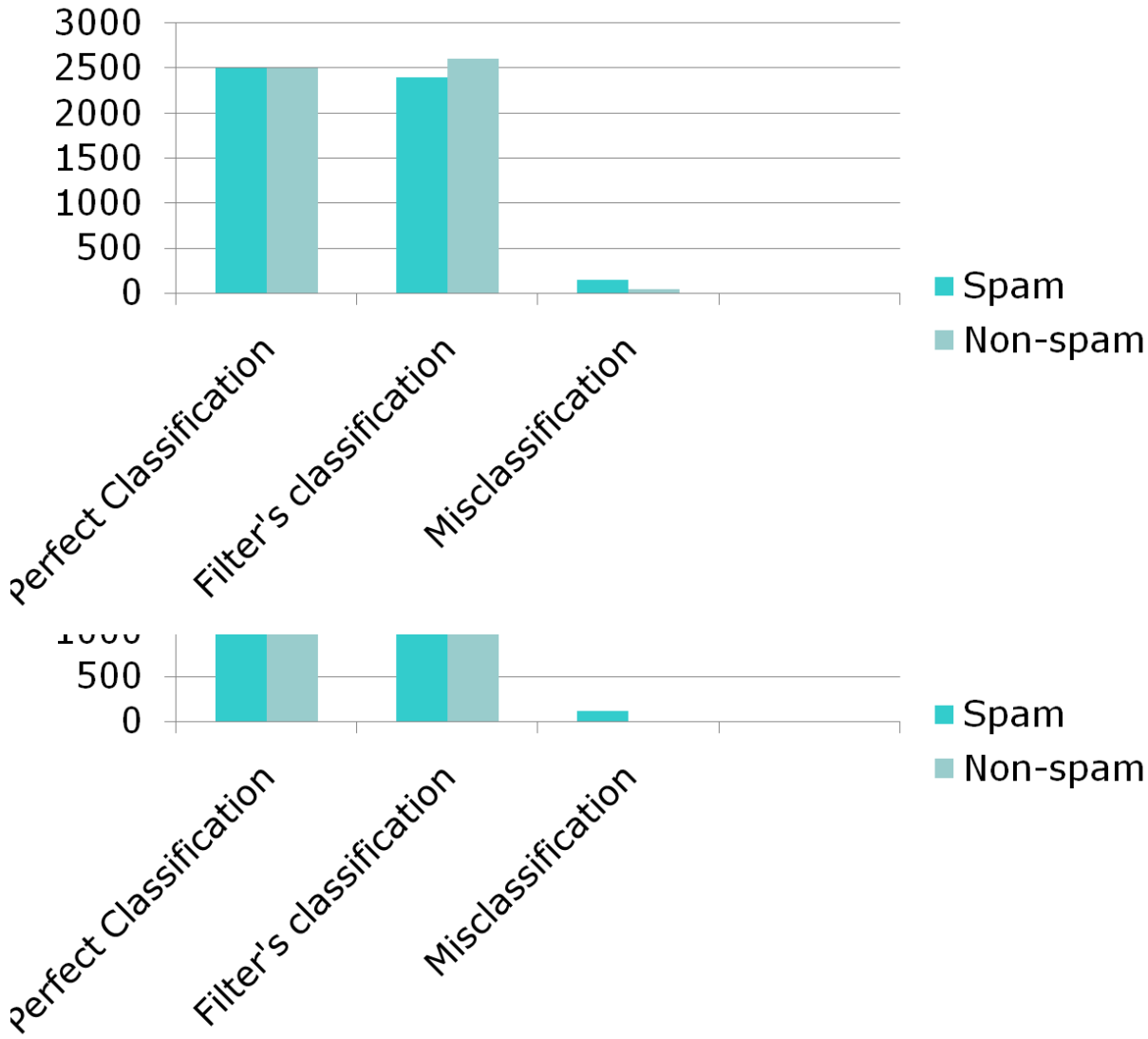
1. TREC 2006 Spam Track Public Corpora, http://plg.uwaterloo.ca/~gvcormac/treccorpus06/about.html.
2. W. Cohen. Learning Rules that Classify Email. *In Proc. of the AAAI Spring Symposium on Machine Learning in Information Access*, pages 124–143. 1996.
3. P. Graham. A Plan for Spam. 2002.
4. P. Graham. Better Bayesian Filtering. 2003.
5. P. Graham. Filters that Fight Back. 2003.
6. E. Michelakis, I. Androutsopoulos, G. Paliouras, G. Sakkis, and P. Stamatopoulous. Filtron: A Learning-Based Anti-Spam Filter. *In Proc. of the 1st Conference on Email and Anti-Spam (CEAS 2004), Mountain View, CA, USA, 2004.*
7. C. Pu, S. Webb, O. Kolesnikov, W. Lee, and R. Lipton. Towards the Integration of Diverse Spam Filtering Techniques. *In Proc. of IEEE International Conference on Granular Computing, pages 7 – 10, 2006.*
8. M. Sahami, S. Dumais, D. Heckerman, and E. Horvitz. A Bayesian Approach to Filtering Junk E-Mail. *In AAAI-98 Workshop on Learning for Text Categorization, Madison, Wisconsin, USA, pages 55–62, 1998.*
9. S. Webb, J. Caverlee, and C. Pu. Introducing the Webb Spam Corpus: Using Email Spam to Identify Web Spam Automatically. *In Proc. of the Third Conference on Email and Anti-Spam (CEAS 2006), 2006.*
10. J. Zdziarski. Ending Spam. No Starch Press, 2005.

Appendix: Charts

Test case 1:



Test case 3


3000
2500
2000
1500
1000
500
0
Spam
Non-spam
Perfect Classification
Filter's classification
Misclassification
500
0
Spam
Non-spam
Perfect Classification
Filter's classification
Misclassification

Stroke Similarity Metrics

Luke Olsen
University of Calgary

Abstract

In this project, we investigate the utility of several stroke similarity metrics for learning in sketch-based modeling applications. Two of the methods are based on previous work in established fields: the Loci method is a standard technique in optical character recognition, while a method based on Fourier analysis comes from motion analysis. Two original methods are investigated, one of which is a modification of the Loci method. The final method is based on quantizing the directions of a vector-based stroke representation. These four methods are evaluated for their ability to produce meaningful clusters in the absence of any additional information, as well as their ability to produce matching templates in a supervised setting. The angle quantization method is shown to have the best matching performance while also offering an efficient implementation.

1 Introduction

Time-series data appears in many diverse domains, such as trajectory and motion analysis, character recognition, robot motion planning, and sketch-based interfaces. Time-series data is simply an ordered sequence of point locations, $S = \{p_1, p_2, ..., p_n\}$, where p_i is a point in some coordinate system. For instance, a tracking system may use frames from a video camera to detect moving objects and build trajectories in image space.

The domain of interest for this project is sketch-based interfaces, so we will often refer to time-series data as a *stroke*. A stroke is simply a time-series sequence of points acquired from an input device, usually a mouse or tablet device.

A common query in these applications is: how similar are strokes S_1 and S_2? If S_1 and S_2 have the same length, then a straightforward comparison such as Euclidean distance would likely give a useful result. In general, however, we would like to compare sequences of different lengths while still getting intuitive results.

Most similarity measures transform the input data to reduce the complexity, produce a more abstract representation, or both. In this paper, we consider four similarity metrics for time-series data and investigate their suitability for use in learning applications.

In Section 2, a brief discussion of relevant earlier works in similarity metrics is presented. The four similarity metrics considered in this project are described in detail in Section 3, while Section 4 discusses the results of the evaluation of these metrics. Finally, Section 5 presents some conclusions and possible directions for future work.

2 Previous Work

Time-series data is used in a number of domains in computer science. For example, in robot motion planning [8], the task is to find an unobstructed path through a region filled with obstacles. The path found through this region is represented as an ordered series of points $P = \{p_{start}, p_1, p_2, ..., p_n, p_{end}\}$, meaning the robot should move from p_{start} to p_1 in a straight line, then from p_1 to p_2 in a straight line, and so on until the goal p_{end} is reached. The ordering of the points is critical to the correctness of the path.

Robot motion is an example of the more general problem of time-series data. There are several approaches to matching time-series data, such as Fourier transforms [1,10], wavelet transforms [3], and others. Each method shares the common goal of approximating the series with a fixed number of coefficients, because the body of work on matching and clustering fixed-

length feature vectors is far greater than for variable-length features.

Another domain that deals with time-series data is character recognition (OCR). Strictly speaking, most OCR work deals with characters represented not as strokes but as pixel arrays, but a stroke can be converted to a pixel array easily. Mori et al. [9] provide a survey of classical OCR techniques, among them the Loci method of Glucksman, which provides an inspiration for this project.

Sketch-based interfaces are an increasingly important paradigm in human-computer interaction, promising more intuitive and discoverable ways of using computer applications. Consider a 3D modeling package that offers a mesh-cutting operation; for instance, a user may want to cut a sphere in half to create a bowl. Traditional modeling packages might offer a menu-based tool for this operation, while in a sketch-based system the user could just draw a line through the sphere where they wish to cut it.

Igarashi is a leading researcher in sketch-based interfaces; his Teddy system [6] remains a seminal work in the area, demonstrating the power of such interfaces while also offering many directions for future researchers to follow.

The stroke recognition methods used in sketch-based modeling systems are typically borrowed from the other domains mentioned above, primarily trajectory analysis. In particular, the Fourier method is often used to construct stroke features. In [11], Shin uses Fourier-space features to match input sketches to a database of objects, and uses the results to replace the sketch with a 3D object. In this way, a fully 3D scene can be created from sketch input.

3 Similarity Metrics

A similarity metric provides a way to compare two objects S_1 and S_2. There are two components that make up a similarity metric: first, a method for converting the objects to a common representation (the *feature space*); second, a measurement of how far apart the objects are in this space (the *distance function*).

For each of the methods described in this section, the second component is the same. Namely, because all the feature spaces consist of n-tuples of scalars, we can use a Euclidean distance measure to compare features. The methods differ in how the strokes are mapped into their feature space.

3.1 Characteristic Loci (CL)

In 1968, Glucksman [4] developed a method for character recognition based on the idea of "characteristic loci" (CL). For a binary image I, each pixel is assigned a code based on how many lines are intersected by rays cast from the pixel in four directions: up, down, left, and right (Figure 8).

Let a, b, l, and r represent the number of lines intersected by the ray cast above, below, left, and right from pixel $I(i,j)$, respectively. The loci-code image L consists of a code constructed from a, b, l, and r as follows:

$$L(i,j) = (a << 3k) \wedge (b << 2k) \wedge (l << k) \wedge r,$$

where << denotes a binary shift operation, ^ denotes a binary AND operation, and k is the number of bits allotted to each intersection count.

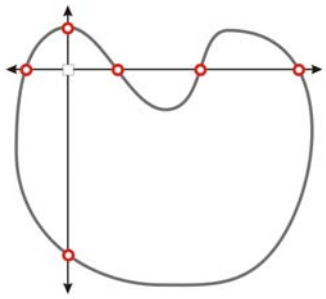


Figure 8: Loci codes are based on ray-line intersections in four directions.

To use the loci method for stroke comparisons, we need a way to convert a sequence of points S to a binary image I. The rasterization of a sequence of points is a fundamental operation in computer graphics; Bresenham's line algorithm [2] provides a method for determining which pixels between p_1 and p_2 fall under the line segment p_1p_2. By applying this algorithm to all adjacent pairs of points p_ip_{i+1} in S, we can acquire an image representation of S. This conversion is not computationally insignificant, however.

To construct a feature vector from L, we sum up the instances of each possible code. Consider an image with 2 bits allotted for each direction (a maximum of 3 intersections per direction). In this case, there are $2^{4\cdot2} = 256$ possible codes. So, the Loci image would produce a 256-dimensional feature vector in this case. For k bits per direction, a 16^k-dimensional feature space results.

3.2 Modified Loci (ML)

One drawback of the Loci method is that is insensitive to local variations. That is, only the overall shape is significant in terms of the codes. For example, the method has difficulty distinguishing between circles and squares because all pixels in the interior of these objects – indeed, any convex object – will have the same code, $\langle a, b, l, r\rangle = \langle 1,1,1,1\rangle$.

In an attempt to address this issue, a variation of the Loci method was investigated. The Modified Loci

(ML) method is based on the idea of identifying locally similar regions, based on the width (distance between nearest strokes to the left and right) and height (distance between nearest strokes above and below) at each pixel (Figure 9).

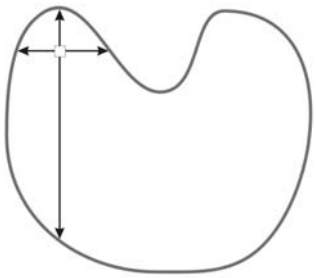


Figure 9: Modified Loci codes are based on the local width and height at each pixel.

Let w and h represent the pixel width and height at $I(i,j)$. To have some level of scale invariance, we should use a normalized representation for w and h. If I is an $m \times n$ image, then we can represent the local width and height as a percentage of the maximum value, that is:

$$W = w/n \times 100, \text{ and}$$

$$H = h/m \times 100.$$

As in the original Loci method, we can construct a code from these values by storing each in the low and high bits of a binary value. If L_M represents the Modified Loci image, then

$$L_M(i,j) = (W << k) \wedge H,$$

where k is the number of bits allotted to W and H. Typically, k ≈ 5 bits provides enough resolution for reasonable results.

To construct a feature vector, we can again sum up the instances of each unique code in L_M. Consider $k = 4$; then there are $2^{2 \cdot 4} = 256$ possible codes, and the corresponding feature vector would have 256 dimensions in this case. In general, for k bits the feature space has 4^k dimensions.

3.3 Fourier Coefficients (FC)

Fourier Analysis is a well-established technique for signal processing. Given a discrete signal $f(x)$ with n samples, the Fourier transform $F(f(x))$ of $f(x)$ consists of n complex values that represent the magnitude and phase components of sines and cosines of various frequencies. Because the "power" of a typical signal is concentrated in lower frequencies, the Fourier transform can be used to reduce the complexity of a signal by retaining only the first $m < n$ coefficients.

Naftel and Khalid [10] propose a method for learning trajectory behaviour from features in Fourier space. To do this, they must convert a trajectory $T = \{p_1, p_2, \ldots, p_n\}$ to Fourier space. The Fourier transform operates on 1D signals, whereas T is a 2D signal. The trajectory is therefore split into two signals, T_x and T_y, and the Fourier transform is applied to them independently: $X = F(T_x)$ and $Y = F(T_y)$.

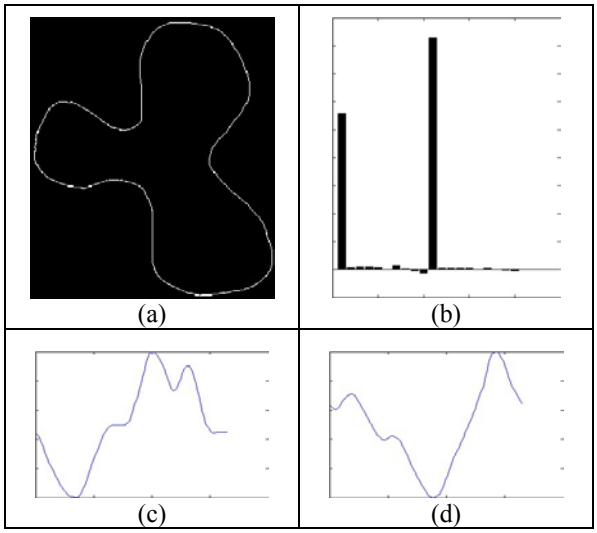
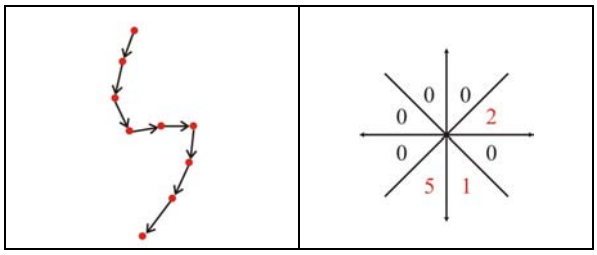


Figure 10: Constructing a feature with the FC method: (a) a stroke; (b) the corresponding feature; (c) T_x; (d) T_y.

To construct a feature vector from X and Y, we take the first m components of each, which become $2m$ feature elements (real and imaginary components). Thus the entire feature requires $4m$ elements in total (Figure 10b). However, because the higher components of X and Y are typically insignificant, m can be chosen to be quite small ($m \approx 5$).

3.4 Angle Quantization (AQ)

The final method considered in this project is called Angle Quantization (AQ). The idea is simple: for every pair of points p_i, p_{i+1} in S, we construct a corresponding vector $v_i = p_{i+1} - p_i$ (Figure 11a). We then quantize the unit circle $[0, 2\pi]$ into k equal-size bins, and count the number of vectors in each bin (Figure 11b). The corresponding feature vector is then just the sequence of these sums (Figure 11c).



(a)	(b)
<2, 0, 0, 0, 0, 5, 1, 0> (c)	

Figure 11: Angle quantization of a stroke (a): for each angle range, the number of stroke vectors in that range are summed (b); the corresponding vector (c).

In practice, the feature vector should be normalized to unit length, to temper the strong dependence on the sampling rate and/or scale of the stroke.

The optimal number of bins, k, is application-dependent, but in general $k < 30$ works well. Larger values can make the quantization too sensitive to noise and user error.

Unlike the three methods previously discussed, the AQ method is sensitive to the stroke direction. Depending on the application, this may or may not be desirable. For instance, in character recognition we may want to recognize a single vertical stroke as a '1', regardless of whether the stroke was drawn from top-to-bottom or vice versa.

On the other hand, in sketch-based systems, the directional dependence can be very valuable. Consider a sketch-based system in which the stroke of Figure 11a represents a gestural input stroke, such as a mesh cutting operation. The opposite stroke may well represent a different gesture entirely, so we would prefer to match only strokes drawn in the same direction.

If k is evenly divisible by 4, then directional independence can be realized with a slight modification. By examining Figure 11, we can notice that if the stroke direction was reversed, then the sums in (b) would be in the opposite bin (shifted by π). The corresponding feature vector would be the feature in (c) shifted by 4 elements, i.e. <0, 5, 1, 0, 2, 0, 0, 0>. This idea can be extended easily to larger k.

4 Evaluation

To evaluate the descriptive power of each similarity metric described above, we consider some typical uses of similarity metrics. In Section 4.1, the ability of each metric to cluster a data set in an unsupervised setting is investigated, while Section 4.2 considers the more common task of template matching.

For evaluation, we acquired two sets of strokes: a training set R with 105 strokes, and a query set Q with 16 strokes. All strokes S in R were representatives from one of 10 possible object classes, (Figure 12).

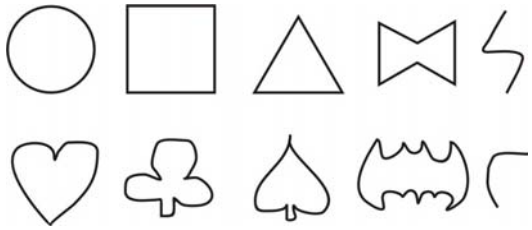


Figure 12: The object classes used in evaluation. (*Top*) circle, square, triangle, bowtie, bolt. (*Bottom*) heart, club, spade, bat, hook.

4.1 Clustering

Unsupervised clustering is a common method of analyzing a data set based on how similar the objects in the set are, which can be used to discover or 'learn' classes of objects. For instance, given a set of strokes that contains some circles and some squares, a clustering of the data might reasonably be expected to build one cluster containing circles and another containing squares.

Each of the methods considered here performs a considerable reduction in the dimensionality of a stroke. A stroke that may contain several hundred or even thousands of points is reduced to a feature with only 20 or 100 elements. This reduction necessarily loses some information about the character of the stroke, but how much?

Fehler! Verweisquelle konnte nicht gefunden werden.—Fehler! Verweisquelle konnte nicht gefunden werden. Show the results of clustering the training set R with each of the four methods. It is difficult to quantify the quality of the clustering, but we can visually evaluate the results and make some observations.

The CL method is able to cluster some objects better than others. The two gesture-like objects, the hook and the bolt, are both clustered quite successfully. The squares and, to an extent, the triangles, are also clustered decently, although the triangles tend to be mixed in with bowties and hearts. The remaining objects – the club, spade, and bat – all seem to be lumped into one cluster.

The ML method performs similarly. The bolt, hook, and square cluster nicely, while the remaining objects are a bit scattered.

The FC method also clusters the gestures well, although the bolt and hook each take up two clusters each. This leaves the rest of the objects to be lumped into too few clusters, although the squares and bowties cluster well, and the triangle and spade are reasonably clustered together.

The AQ method performs as well as can be expected, given the nature of the method. Strokes with similar angles, such as the bowtie and triangle, and the heart and spade, are clustered together. The square, bolt, and hook are again clustered nicely, while the circle, club, and bat also cluster together.

Overall, the clustering results are difficult to interpret. No method clustered the objects in a perfect manner, but when one considers the way that the features are constructed, the results make sense.

4.2 Template Matching

In a template matching scenario, we have a training set containing strokes for which we know what object class they correspond to. That is, we know that stroke S_i represents a circle; S_j represents a square, and so forth.

For each object class, we construct a template by averaging the features of all strokes from that class. This is conceptually equivalent to supervised *k*-means clustering: each cluster contains objects of a single type, and the mean of the cluster becomes the template.

Once we have a set of templates, we can use our similarity metric to find the nearest template to a query stroke *q*. Presumably, the user knows what they intend the stroke to represent, so the matching success can be assessed with a simple binary classification: correct or incorrect.

Figure 5 presents the results of matching our query set Q to templates constructed from the training set R, for each of the four methods.

The performance of the FC method is quite poor. Perhaps it can be attributed to an implementation error, but when one looks at the features produced by this method, the poor results are not necessarily surprising.

Because the strokes in the query and training set are generally smooth, the zero-th Fourier components completely dominate the features; the high-frequency activity is almost negligible. Thus the features for each object class are too close to reliably distinguish them.

The CL method performs reasonably well, but it has difficulty distinguishing between certain types of objects. Convex objects all generate the same interior codes, which tend to dominate the feature vectors. Thus circles, squares, and triangles have similar features. As well, the Loci features are invariant to mirroring about a primary axis, so, for instance, the heart and the spade are often mistaken for one another.

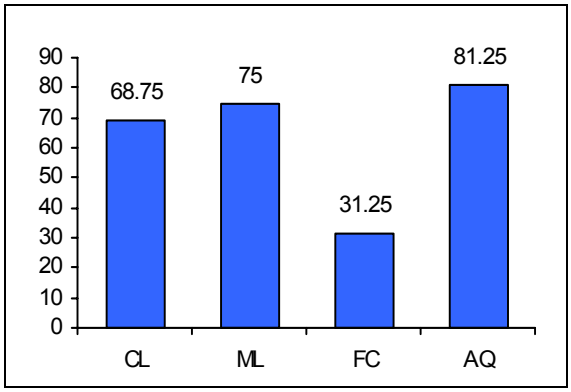


Figure 13: Template matching success rate for each method. The success rate is the number of query strokes S in Q that were matched to the correct template.

The ML method outperformed the original CL method, which is encouraging. This is primarily because convex objects can be distinguished more easily, as they produce very different features.

Finally, the AQ method provided the best results. This result is encouraging, and indicates that this method may be useful in a sketch-based system. However, the error rate is still almost 20%, so there is certainly room for improvement.

One clear source of error in the AQ method is the loss of locality (this applies to the FC method also). Consider the two strokes in Figure 14. The vectors in each stroke are identical – and therefore the AQ features are identical also – yet the strokes are clearly different. Once the vectors are put into a certain bin, the relationships between adjacent vectors is completely lost.

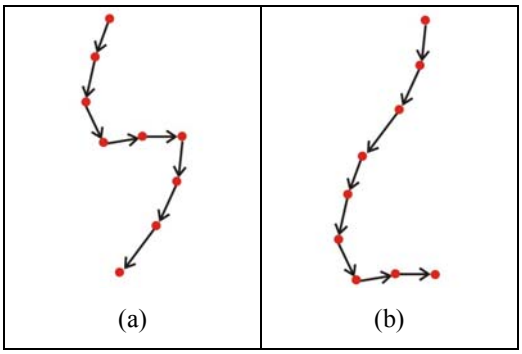


Figure 14: Loss of locality in the AQ method; the feature vectors of (a) and (b) are the same.

4.3 Implementation Considerations

This project was implemented in Matlab. Matlab is a scientific computing environment that offers a large variety of toolboxes for different domains, such as image analysis, and is optimized for matrix and vector

operations. When implementing algorithms in Matlab, one must try to vectorize operations and avoid looping constructions whenever possible, which is a different environment than most programming languages. As such, the ability of an algorithm to be vectorized will dramatically impact its performance in Matlab, but not necessarily in other languages.

Of the methods considered in this project, three of the algorithms vectorize well: CL, FC, and AQ. The ML method unfortunately does not, and so its performance suffered greatly.

For the template-matching experiments in Section 4.2, each method was used to construct features for the 16 query strokes and then find the best matching template. The running times for each method are summarized in Figure 15.

Because the ML method did not vectorize well, its running time is far higher than the others. The CL method was also higher than FC and AQ, because the size of a CL feature is an order of magnitude larger. Overall, AQ offered the best performance, while also providing the highest matching success.

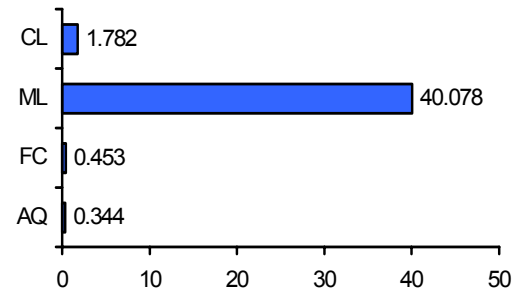


Figure 15: Time required for each method to match 16 query strokes.

6 Conclusion & Future Work

In this project, four similarity metrics were investigated to evaluate their utility in a sketch-based interface. Two of the methods were based on existing work: the CL method is used in character recognition, while the FC method has been used in trajectory analysis. The other two methods are new methods: ML is an extension of the CL method, and the AQ method is based on a vector representation.

The methods were first evaluated on their ability to cluster a set of strokes without supervision. None of the methods performed a perfect clustering, but this can be attributed to the types of objects being considered as well as the nature of each method. The results were not particularly informative.

The second evaluation provided more insight. From a training set, templates were constructed for each object class, which were then used to classify query strokes. The two novel methods, ML and AQ, were found to provide the best matching success.

In the future, we intend to incorporate a stroke similarity metric into a sketch-based modeling system to provide an gesture-based interface. By having a template-based method, the system will be able to learn from the user and reliably interpret the input.

References

1. Agrawal, R., Faloutsos, C., and Swami, A., *Efficient Similarity Search in Sequence Databases*, Proc. of the 4th International Conference of Foundations of Data Organization and Algorithms, 1993.
2. Bresenham, J., *Algorithm for Computer Control of a Digital Plotter*, IBM Systems Journal, 4(1):25-30, 1965.
3. Chan, K-P and Fu, A., *Efficient Time Series Matching by Wavelets*, Proc. of 15th International Conference on Data Engineering, 1999.
4. Glucksman, H.A., *Classification of Mixed-Font Alphabetics by Characteristic Loci*, 1967 Digest of 1st Annual IEEE Computer Conference, 1967.
5. Hu, W., Xie, D., Tan, T., and Maybank, S., *Learning activity patterns using fuzzy self-organizing neural network*, IEEE Transactions on Systems, Man and Cybernetics, Part B, Volume 34, Issue 3, June 2004.
6. Igarashi, T., Matsuoka, S., and Tanaka, H., "*Teddy: A Sketching Interface for 3D Freeform Design*" ACM SIGGRAPH'99, Los Angels, 1999, pp.409-416.
7. Knoll, A., "*Experiments with Characteristic Loci for Recognition of Handprinted Characters*", IEEE Transactions on Computers, Vol. C-18, Issue 4, 1969.
8. Latombe, J.C., *Robot Motion Planning*, Springer Publishing, 1991.
9. Mori, S. and Suen, C.Y., Yamamoto, K., *Historical review of OCR research and development,* Proceedings of the IEEE, Volume 80, Issue 7, July 1992.
10. Naftel, A. and Khalid, S., *Motion Trajectory Learning in the DFT-Coefficient Feature Space,* IEEE International Conference on Computer Vision Systems, 2006 ICVS '06., Vol., Iss., 04-07 Jan. 2006.
11. Shin, H., *Magic Canvas: Interactive Design of 3D Scenes from Freehand Sketches*, Dept. of Infor-mation Science, University of Tokyo, 2007, *http://www-ui.is.s.u-tokyo.ac.jp/~shin/research.htm*

Web Data Extraction Using Clustering

Justin Park

1. Introduction

World Wide Web is increasingly becoming a gigantic heterogeneous knowledge base. However the information available on the web is meant to be viewed by humans, not by computers. The information is visually structured such that it is easy for humans to recognize data records and presentation patterns. Current search engines' keyword search paradigm is not sufficient enough to capture some of the fine grain information that humans can understand. For instance, keyword search blurs the boundaries between data records on a page. It also does not allow comparisons or correlation between several pages. Furthermore, most of the contents on the web are in the "hidden web", which is not indexed by search engines. These pages are usually generated by scripts that present the data from their databases, and often contain more valuable information than the pages indexed by search engines. If computers can recognize the patterns in which the data are represented, the applications are endless, including product comparison, automatically synchronizing calendars according to events, and question answering system. The common way to interface web application with heterogeneous data sources on the web is through a wrapper. A wrapper translates data source to structured data that is understood by web applications. Due to the virtually infinite number of data schema on the web, custom fitting a wrapper for each data source on the web is impractical. It is equally inconceivable to enforce every websites to publish their web data schema.

In this work, an automatic wrapper generator WDE(Web Data Extractor) is built. The only input to the tool is an URL pointing to an HTML document. The aim of the tool is to be able to extract data from any kind of repeating data structure, including product listings, readers' comments, sports scoreboard and forums. The tools analyzes an HTML document by assigning match score to each node and identify the repeating pattern by clustering the match score using k-means clustering algorithm.
The empirical study of the tool shows that it can extract data from product listings slightly better than the leading tool MDR[1].
However, for forums and reader commented pages, WDE outperformed MDR by several folds.

2. Design

WDE is aimed at being an automated wrapper generator that requires only one URL as input. The document is parsed and some pre-processing is done in step 1. The modified document is then passed to step 2 where it calculates match scores for all pairs of nodes. A clustering algorithm is used to cluster the pairs into two clusters – one cluster for high similarity and the second cluster for low similarity. In the 4^{th} step, the similar cluster from the previous step is separated into groups such that the members of each group have high similarity with each other. In the 5^{th} step, all possible compositions of records are proposed. In the 6^{th} step, the record instances are compared against each other to calculate the match score. The match scores for the all pairs of proposed records are fed to the clustering algorithm again to identify the record instances that are different from the rest. In the 7^{th} step, the records are refined. The final output is all the records the tool proposed. The tool isn't completely automatic as of the moment. User must manually select

the record group. For example, from a blog website, one record group may be the navigation menu items, and another record group contains the blog entries. Then, it is up to the user to select the appropriate record group. Figure 1 describes the design and the flow of data.

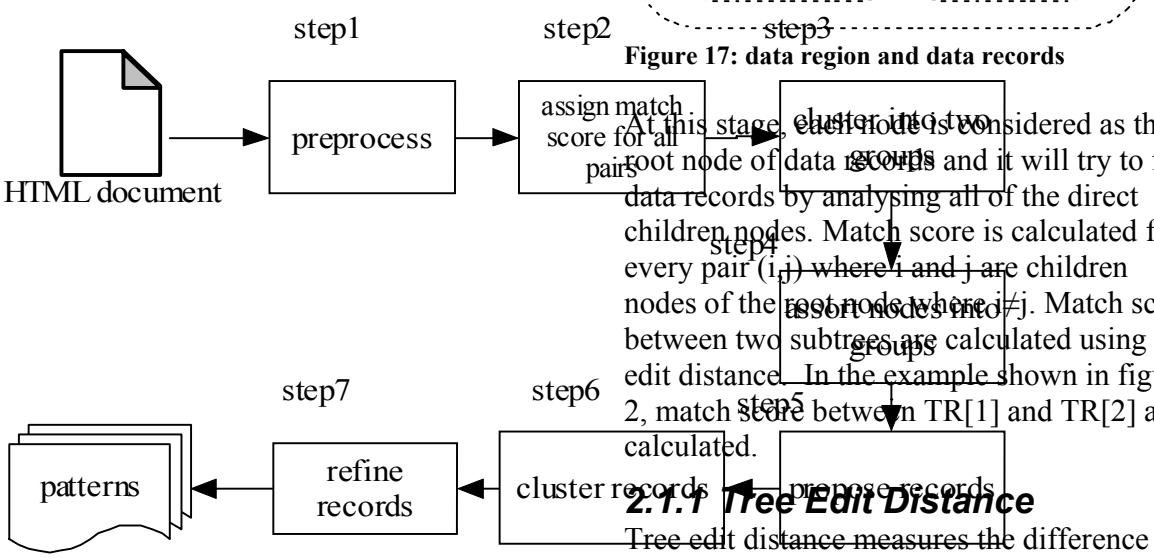


Figure 16. Design of WDE

2.1 Finding Similar Nodes

Every node in the parse tree is considered as a potential data region. Data region is defined similarly as in the Depta paper[2]. Data region is a node and its descendants that contain data records. In the example shown in figure 2, the table node and its descendants is a data region because the root node, table, contains data record 1 and data record 2.

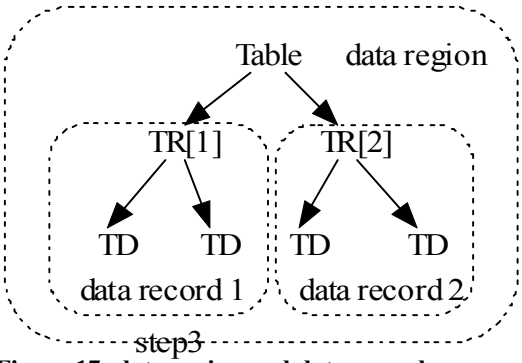


Figure 17: data region and data records

At this stage, each node is considered as the root node of data records and it will try to find data records by analysing all of the direct children nodes. Match score is calculated for every pair (i,j) where i and j are children nodes of the root node where i≠j. Match score between two subtrees are calculated using tree edit distance. In the example shown in figure 2, match score between TR[1] and TR[2] are calculated.

2.1.1 Tree Edit Distance

Tree edit distance measures the difference between two trees by the number of insertions, deletions, and substitutions required to modify one tree into the other tree. Using tree edit distance, mapping M can be formulated as set of ordered pairs (i,j) where i denotes the ith child of tree A and jth child of tree B such that for all (i_1,j_1), (i_2,j_2) in M:

1. $i_1 = i_2$ *iff* $j_1 = j_2$;
2. $A[i_1]$ is on the left of $A[i_2]$ *iff* $B[j_1]$ is on the left $B[j_2]$;
3. $A[i_1]$ is an ancestor of $A[i_2]$ *iff* $B[j_1]$ is an ancestor of $B[j_2]$.

Intuitively, the mapping preserves the order of the siblings. In the figure 3, the dotted line shows the matching relationship between two nodes. In tree A, node b is left of node c, and in tree B, b node is left of node c, therefore the mapping (A_b,B_b) and (B_c,B_c) satisfies the second condition, where X_i is the node i of the tree X. The mapping (A_d,B_d) and (A_c,B_c)

satisfies the third condition because A_c is ancestor of A_d and B_c is ancestor of B_d.

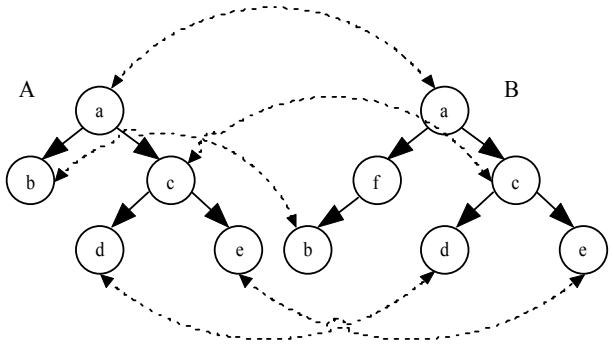


Figure 18: matching nodes for tree edit distance

2.1.2 Simple Tree Matching

The Depta paper[2] proposed SimpleTreeMatching algorithm and it is the basis of the record matching and partial tree alignment algorithms. Simple tree mapping is similar to the mapping formulated from tree edit distance, except matching nodes are not allowed to cross levels. If a node A_i of tree A matches with B_j of tree B, then the parent of A_i must match with parent of B_i. In Simple Tree Matching, match score between two nodes is 0 if the node symbols do not match. If they match, the match score is calculated by the total number of descendants that matches while preserving the sibling order and the parent-child relationship.

2.2 Weighted Simple Tree Matching

In weighted tree matching, the parent node weighs differently than the children nodes. The algorithm is similar to SimpleTreeMatching proposed in Depta paper [2] with two major differences. First, in weighted tree matching, the match score is normalized from 0 to 1, where 1 is perfect match. Second, a matching parent node contributes different weight than the children nodes. Let α be a real number between 0 and 1. Let two trees A and B have the identical symbol at the root node. The match score of the two trees is $\alpha*1 + Ms_{A,B} * (1- \alpha)$, where $Ms_{A,B}$ is the match score between the children nodes of tree A and B. The constant α determines how the weight should be spitted between the parent node and the children.

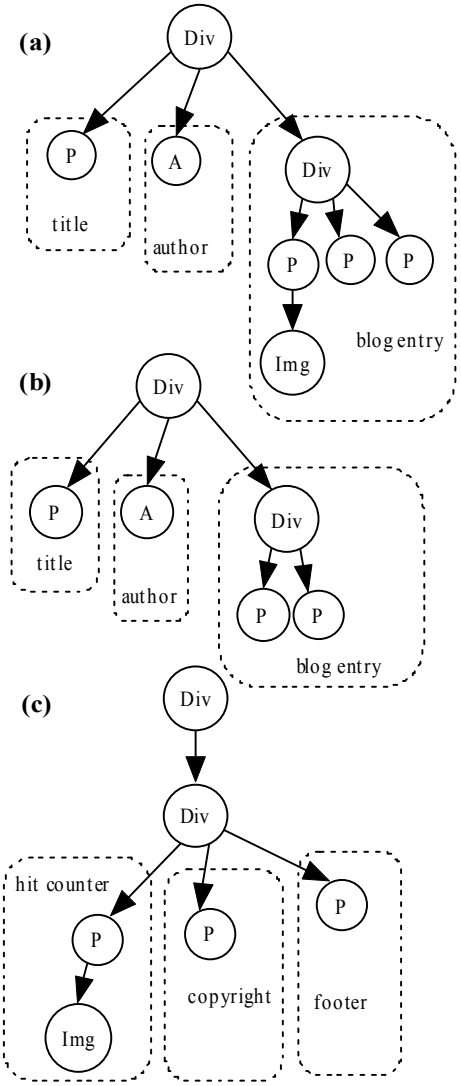


Figure 19. An example of matching trees. 5a and 5b are parse trees of a blog entry, and 5c is a parse tree of a footer found in the same web page

The table 1 shows a walk-through of how weighted simple tree matching works on comparison between trees 4a and 4b. The constant α is set to 0.5 in this example. Since the algorithm is recursive, the table starts from the inner most invocation of recursion, i.e. calling the function on leaf nodes. The

columns 4a and 4b represent the nodes that are passed into WeightedSimpelTreeMatching as arguments. The values in those two columns are represented using XPath notation. The M column shows the matrix M. The first column and first row of M are omitted in the view because they are always 0's. If the M column contains only one value, it means that the matrix is 2x2 with 0's in the first row and first column, and the indicated value in the (2,2) position of the matrix. The Ms column shows the match score after M[n,m] has been normalized and multiplied by 1- α weight and added to the root node's weight. Specifically, the Ms column is (M[n,m]/n+M[n,m]/m)/2 * (1- α) + α. The rows 1 and 2 of table 1 have 1.0 for M because the nodes are perfect match. The Ms column for those rows are also 1.0 because (1.0/1+1.0/1)/2 * (1-0.5) + 0.5 = 1.0. The rest of the tree is processed in the similar manner until we get to comparison of div/div, which is shown in row 9. In row 9 of table 1, the matrix M has the children of 4a's div/div as the rows, and the children of 4b's div/div as the columns. The (1,1) position of M is 0.5 because the Ms field of row 3 is 0.5. The (1,2) value is 0.5 because the row 4's Ms is 0.5. The value of (2,2) is max(M[1,1]+1.0, M[2,1], M[1,2]) = 1.5. Similarly, the value in position (3,2) is 2.0 because M[2,1]+1.0 is greater than both M[2,2] and M[3,1]. The Ms of row 9 is (2.0/3+2.0/2) * (1-0.5) + 0.5=0.92. In row 10, the matrix M and match score Ms is calculated in similar manner as in row 9. The final match score between the trees 4a and 4b is 0.98, as shown in Ms column of row 10. The match score between these two trees are very high compare to the match score between 4a and 4c, which is 0.83. Contrast to the match score calculated using SimpleTreeMatch, WeightedSimpleTreeMatch assigns higher score between 4a and 4b than between 4a and 4c.

Table 5: Weighted Simple Tree example

	4a	4b	M	Ms
1	div/div/p	div/div/p	1.0	1.0
2	div/div/a	div/div/a	1.0	1.0
3	div/div/p[1]	div/div/p[1]	0.5	0.5
4	div/div/p[1]	div/div/p[2]	0.5	0.5
5	div/div/p[2]	div/div/p[1]	1.0	1.0
6	div/div/p[2]	div/div/p[2]	1.0	1.0
7	div/div/p[3]	div/div/p[1]	1.0	1.0
8	div/div/p[3]	div/div/p[2]	1.0	1.0
9	div/div	div/div	<table><tr><td></td><td>p[1]</td><td>p[2]</td></tr><tr><td>p[1]</td><td>0.5</td><td>0.5</td></tr><tr><td>p[2]</td><td>1.0</td><td>1.5</td></tr><tr><td>p[3]</td><td>1.0</td><td>2.0</td></tr></table>	0.92
10	div	div	<table><tr><td></td><td>p</td><td>a</td><td>div</td></tr><tr><td>p</td><td>1</td><td>1</td><td>1</td></tr><tr><td>a</td><td>1</td><td>2</td><td>2</td></tr><tr><td>div</td><td>1</td><td>2</td><td>2.92</td></tr></table>	0.98

2.5 Clustering

The tool should eventually be completely automatic. Thus, any kind of machine learning should be unsupervised. There are two places where clustering is performed: step 3 and 6 of figure 1. The first clustering is aimed at finding the similar nodes. The second clustering is aimed at finding records that are different from the rest.

2.5.1 Clustering similar nodes

In step 2, match scores are assigned to each pair of nodes in the HTML parse tree. To identify repeating patterns in the HTML structure, the nodes must be grouped by the similarity such that for each node in a group has high cohesiveness with other nodes in the same group, and low cohesiveness with the nodes in other groups. Since the tree edit distance can only measure relative similarity between two nodes, there is no global absolute reference. In this work, two approaches were considered. First, for each node, group the children nodes in isolation from other nodes in the HTML document. This approach is considered because of the observation that elements that belong to one record are normally organized such that the nodes are either siblings or children. However, some nodes contain too few nodes for the clustering algorithm to be effective. Also, some HTML

documents have instances of records appearing under different parents.
The second approach was to feed to the clustering algorithm all pairs in the entire document. In this case, the clustering algorithm will cluster into two groups. In one group, the members are all pairs of nodes that have high similarity. In the second group, the members are all pairs of nodes that have low similarity. The drawback of this approach is that both highly structured and loosely structured records are treated the same. However, it does not restrict on the minimum number of nodes required to run the clustering algorithm. The second approach is chosen because it can handle any number of nodes.

From the Java code, it makes calls to Weka's K-means clustering component. The input to Weka has the format:
node1, node2, match-score, descendant-ratio.

The first two fields are not used by k-means algorithm. The domain of the second field, match-score, is between 0 and 1.0 where 1.0 indicates identical structure between the two nodes. The last field is the number of descendants in the smaller subtree divided by the number of descendants in the larger subtree. For example, in the figure 4, node1 has 5 descendants and node2 has 8 descendants. The value in the 4^{th} field is 5/8. This field is introduced because the match score gives higher weight to the upper nodes and less weight on the lower nodes. Consider two trees that have the identical structure for upper 3 levels. One tree has only three levels but the second tree has 50 levels. Since the simple tree edit distance uses geometric weighting system, the two trees in the above example will have very high similarity. In practice, the two trees should be categorized into different groups. The children-ratio field's function is to distinguish such cases. Also note that inclusion of this attribute does not equate to using a constant weights for evaluating the match-score.

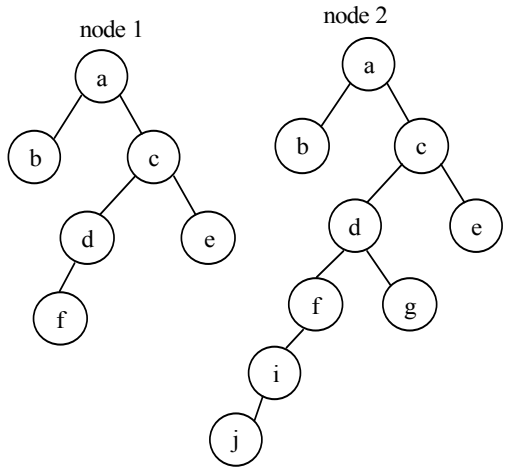


Figure 20. Descendant ratio

After clustering the pairs into two groups, the nodes are separated into further groups. If a pair A,B and B,C were grouped into the same cluster, the three nodes are grouped into one group because they have common node B, which means that A,B, and C are similar. The purpose of this grouping is to identify components of records. For example, in a blog, one component could be title, and another component is the blog entry. After these groups are identified, all possible composition of records is proposed. For every set of proposed records, second clustering is performed to identify malformed records.

2.5.2 Clustering similar records

In this phase, proposed records are examined to look for records that are different from the rest of the records created using the same heuristic. For every record instance, a dummy is rooted at the top and the match score is computed for every pair. The input for the clustering algorithm is as follows:

record1,record2,match-score

In this clustering phase, only one attribute is used. The number of descendants in each

record should be roughly equal because it was taken care of in the first clustering phase. The records are grouped into two clusters. The records that fall into the dissimilar records are either adjusted or removed. For each malformed record, if the number of top-level subtrees is more than the average number of top-level subtrees, the least matched subtrees are deleted from the record. If the top-level subtree is less than the average, the record is expunged. After making modifications, match score is computed again. This process of finding dissimilar records and adjusting it is repeated until the every record has similarity above the threshold r.

3. Experiments

Twenty websites were used to test the two phases of clustering. The websites had visually easy to identify record structures and the websites were previously not exposed to the tool during the development phase. After the first clustering phase, the tool creates several groups of similar nodes. Since many of the clusters are not "interesting" nodes, such as navigational menu items, the numbers recorded in the tables below concerns only with the "interesting" clusters.

Phase 1.

In the first clustering phase, similar nodes in a HTML document are grouped together. The match scores for all pairs of the node are fed to the clustering algorithm to split the pairs into two groups. After the clustering, groups of similar nodes are formed by unioning the pairs that intersect. The tool is evaluated on the precision and recall of nodes belonging to the correct cluster. Precision measures the number of correct nodes in the group over the number of nodes in the group. Recall measures the number of correct nodes in the group over the number of correct nodes that should be in the group. The sense of "correct nodes" is arrived at by visual inspection of the HTML document. Table 2 shows the result of the experiment. The first column is the url of websites, the second column is the number of correct nodes in the group, the third column is the number of nodes missed in the group, and the third column is the number of incorrect nodes in the group.

Table 6. Clustering Phase 1

url	corr	miss	incorr
pricerunner.com	25	0	0
download.com	10	0	0
operawatch.com	15	0	1
shopping.yahoo.com	15	0	0
nexttag.com	15	0	0
chapters.ca	10	0	1
dealtime.com	21	0	25
messages.yahoo.com	19	0	0
dailykos.com	15	0	0
gizmodo.com	20	0	0
shoutwire.com	15	0	0
mls.ca	0	6	0
amazon.com	24	0	0
radioshack.com	4	11	0
allrecipes.com	20	0	0
discussion.forum.nokia.com	15	0	0
youtube.com	10	0	0
forum.java.sun.com	9	2	0
weblog.xanga.com	10	0	0
forums.gentoo.org	25	0	0
Precision	**0.91**		
Recall	**0.93**		

Phase 2

After records are proposed, match score is computed for every pairs of records. The clustering component clusters the records into two groups. If the clustering made no mistakes in the phase 1, then the second phase clusters every record into one cluster. Only the websites that incorrectly grouped nodes in the first phase had an effect in the second phase because the incorrect grouping of nodes will lead to proposing a record that is incorrect.

The table below shows the number of malformed records identified in the second column and the number of malformed records that were not identified.

url	identified	missed
pricerunner.com	-	-
download.com	-	-
operawatch.com	1	0
shopping.yahoo.com	-	-
nexttag.com	-	-
chapters.ca	1	0
dealtime.com	4	21
messages.yahoo.com	-	-
dailykos.com	-	-
gizmodo.com	-	-
shoutwire.com	-	-
mls.ca	-	-
amazon.com	-	-
radioshack.com	-	-
allrecipes.com	-	-
discussion.forum.nokia.com	-	-
youtube.com	-	-
forum.java.sun.com	-	-
weblog.xanga.com	-	-
forums.gentoo.org	-	-

After the second phase of clustering and adjusting the records, the final records are outputted to files. The precision and recall of the final version is same as the results in phase 2 because if malformed are identified, in all four cases, the heuristic of the tool were able to arrive at the correct records.

4. Analysis

In the test data set, a few websites did not group the nodes correctly. Some websites intentionally generate HTML pages that are different for each record in order to debunk automatic wrapper generators. To combat these debunk attempts, a better match scores and heuristic must be used. In other cases, the incorrect group was resulted due to too much diversity in the input to the clustering algorithm. For example, if some pairs have match score of 0.99, and others have match score less than 0.6, then it is easy to separate the pairs into two groups. However, if the match scores range from 1.0 down to 0.1 with even distribution within the range, then the separation becomes very difficult. There are three ways to combat this challenge.

First way is to run the clustering algorithm in smaller segments of input data. Isolating every subtree may cause problems on the subtree that has too few children to run clustering algorithm. To properly segment the input data, it requires some heuristic to maximize the effectiveness of the clustering algorithm.

Second way is to find a better match score function. The match score function could be improved by including the PCDATA types. For example, text can be categorized into several groups, such as titles (first letter of every words capitalized), numbers, and long passages of text. Another way to evaluate match score is to match the two trees bottom up starting from leaf nodes instead of from the root nodes.

The third way is to cluster the pairs into more than 2 clusters. The pairs of nodes could be clustered into three groups: one for high similarity, one for low similarity, and one for the rest of the nodes. The third could try to merge with the high similarity group and also with the low similarity group and choose the best alternative.

The tool took in the upper bounds of 5 minutes to complete executing for some websites. The majority of the time was taken in the second clustering phase. The second clustering has very little impact in terms of improving the precision and recall. A possible way to reduce the execution time is to be more selective in proposing records.

5. Conclusion

The k-means clustering algorithm is very effective in identifying the similar nodes in

HTML documents. However the clustering algorithm is only as good as the similarity function. Clustering the HTML nodes had much larger impact on extracting the correct records than the clustering the proposed records. In the experiment with 20 websites, the tool's precision and recall were both at 93%. The test data set was very small, but it appears that the tool is working well on both rigidly structured and loosely structure websites. The tool could be improved to run faster and be fully automatic.

References

[1] B. Liu , R. Grossman , Y. Zhai, Mining data records in Web pages, Proceedings of the ninth ACM SIGKDD international conference on Knowledge discovery and data mining, August 24-27, 2003, Washington, D.C.

[2] Zhai, Y. and Liu, B. Web Data Extraction Based on Partial Tree Alignment. Proceedings of the 14th international conference on World Wide Web (Chiba, Japan, 2005). p. 76-85.

Machine Learning: CPSC 601.66
Term Project Report: 2007.04.19
University of Calgary

"Discovering Natural Language Grammar Using Unsupervised Learning: Fuzzy Clustering And Hierarchical Density Clustering"

Investigator: Scott S. Walker
Professor: Dr. Michael M. Richter

a. Application Domain

This project concerns itself with the domain of Natural Language Processing (NLP); specifically, the task of discovering grammatical structure from unformatted text.

Grammatical structure can mean structure at any level: characters, groups-of-characters, words, groups-of-words, phrases, groups-of-phrases, sentences, groups-of-sentences, and so on. In order to limit the project scope appropriately for this term project, only grammatical structure at the word level was investigated, i.e. "parts of speech".

The application data used is preprocessed appropriately for this word level investigation, as described below. If this project were to be extended to include the character and group-of-characters structural levels, then presumably no preprocessing would be required for this word level of investigation.

1. Application Data

The source data is an open source novel from Project Gutenberg included in the Natural Language Tool Kit (NLTK), chosen at random, in this case *Emma*, by Jane Austin. A brief investigation of other NLTK Project Gutenberg books, including the *King James version of the Bible*, a collection of Shakespeare's plays (*Hamlet*, *Macbeth*, and *Julius Caesar*), and a collection of Walt Whitman's poetry in his book *Leaves of Grass*, all showed similar results, but these results are not presented here.

Because character and group-of-characters levels were skipped in this investigation, preprocessing was required to provide appropriate data to the word level analysis, including:

- Setting all characters to lower-case;
- Removing underscores from around words, which are used to indicate emphasis in some Project Gutenberg corpora;
- Removing periods which are part of the tokens "Mr.", "Mrs.", "Dr.", "St.", etcetera, to facilitate the tokenizing process, below;

- Replacing periods at the end of a sentence with two tokens, ".*", so that the star indicates breaks between sentences;
- Splitting the document string into a list of tokens (either a word, a piece of punctuation, the sentence splitting "*" token, or the special case of the group-of-characters "'s" which indicate possession).

2. Problem Description

In NLP, the problem of determining parts of speech from language structure is normally avoided by considering it to be a solved problem. There are electronic forms of dictionaries readily available from which a part of speech can be looked up; likewise, there are language grammars which linguists have developed which describe the allowed grammars of a language at a higher level, and which can be used to disambiguate a word's part of speech designation from the dictionary lookup.

There are two problems with this approach. First, it is a form of bootstrapping past the problem, allowing human intelligence to solve the difficult portion of the problem, rather than having an artificially intelligent system do the solving. In other words, there is certainly much more that can be learned there. Second, the human coding of idealized grammars from linguistics are exactly that—idealized—and cannot approach human level accuracy in processing the entire range of actual language structure employed by people. The first is primarily an issue only if one's goal is to eventually build a human-level artificial intelligence in general; however, the second is required for enhanced robustness in the specific problem domain of NLP.

Also note that some systems use some understanding of the content and meaning of the text to assist with disambiguation, however the intent of this project is to discover grammar without any assistance from the semantic meaning of the text.

3. Learning Goal

The investigator's goal is for the system to automatically find word level grammatical patterns (i.e. parts of speech) in the raw structure of the text, without any knowledge of the semantic meaning of the text and, indeed, without any a priori knowledge at all, except assumptions for which there is evidence of hard-wiring into the human brain (such as natural language having a hierarchical structure, and the relative positions and relative frequencies of words as the patterns that the human brain recognizes).

In principle, the goal would be achieved completely if:

- Word tokens are each placed in an appropriate group along with every other word of a particular part of speech;
- Word tokens which can be multiple parts of speech are placed into every appropriate group above;
- Punctuation tokens are also each placed in appropriate token group(s) according to their usage.

In practice, this would also require the hierarchical nature of part of speech membership to be recognized; for example, verbs can have multiple tenses, such as past, present, and future, to name just a few. This investigation's limited scope will not require such a membership hierarchy, so the learning will be considered to be completely successful if all tokens in a group can be shown to be related together at some level of the part of speech hierarchy. Since applying an appropriate label to each group, e.g. "verb", requires knowledge of language meaning, which is outside the scope of the project, this labeling task is external to the system.

4. General Learning Method

To detect the patterns of tokens in the text which describe the grammatical structure of that text, this project employs unsupervised learning, specifically clustering.

Given the second part of the learning goal described above, it was expected that the chosen clustering method would take care of allowing a token to belong to multiple clusters. The best method would also determine the number of clusters automatically. Finally, the chosen clustering method should be easily extensible to allow hierarchical clusters of clusters. These possibilities are summarized below, without concern for processing power required:

Table 1 – Comparison Of Clustering Methods For This Application

Clustering method	Intrinsically allows membership in multiple clusters?	Automatically determines qty. of clusters?	Easily extended to hierarchical clustering?
k-Mean/Medoid	N	N	YES
Hierarchical	-	-	YES
Density-based	YES	YES	YES
Fuzzy	YES	N	YES

Work was begun without taking into account the need to automatically determine number of clusters. In that context, it was felt that fuzzy clustering was the best match to the problem domain, conceptually, and so fuzzy clustering was initially chosen for the clustering method. The primary goal was to prove the concept of using clustering to group words into their parts of speech, and discover whether any level of success at all could be achieved.

Afterwards, knowing that using clustering to learn parts of speech is in principle achievable, density-based clustering was attempted because it meets all three desired criteria as seen above. Also, density-based clustering is far less computationally expensive than fuzzy clustering, and these computation requirements were found to be a limiting factor as described later. In fact, density clustering is so much faster that iterative and hierarchical methods could also be applied within reasonable processing time frames, and so this was also done.

5. Specific Learning Method

The specific learning method applied to this problem domain is fuzzy clustering, a type of unsupervised learning. Each token is a node to be clustered. The attribute vectors for each node include the set of normalized frequencies of occurrence of particular tokens found both one position and two positions before and after the token represented by that node.

Formally: Each token q in $\mathbf{T}$, is described by the scalar frequency, F(q), and the 4*n-dimensional attribute array, $\mathbf{A}(q)$, which is a concatenation of the n-dimensional normalized frequency arrays, $\mathbf{F}_{-2}(q)$, $\mathbf{F}_{-1}(q)$, $\mathbf{F}_{+1}(q)$, and $\mathbf{F}_{+2}(q)$, where:

- q indicates the token being queried
- $\mathbf{T}$ is the set of tokens in the dataset, ordered by descending frequency
- $\mathbf{F}$ is the set of normalized frequencies corresponding to $\mathbf{T}$

  e.g. $\mathbf{F} = \{0.01, \ldots\}$ indicates that $\mathbf{T}(0)$ accounts for 1% of all tokens
- $\mathbf{F}_p$ is the set of normalized frequencies of occurrence of a token in position p relative

to the token position

e.g. $\mathbf{F}_{-2}(7) = \{0.03, \ldots\}$ indicates that $\mathbf{T}(7)$ occurs in the position two spots before this token 3% of the time

- **A** is the set of attribute vectors of **T**, each formed from all normalized frequency vectors $\mathbf{F}_p$ for that token (called the query token, numbered q)

  e.g. $\mathbf{A}(m) = \{\mathbf{F}_{-2}(0), \ldots, \mathbf{F}_{-2}(n-1), \mathbf{F}_{-1}(0), \ldots, \mathbf{F}_{-1}(n-1), \mathbf{F}_{+1}(0), \ldots, \mathbf{F}_{+1}(n-1), \mathbf{F}_{+2}(0), \ldots, \mathbf{F}_{+2}(n-1)\}$ is the attribute array for token *m*.

  - Only the relative frequencies of tokens in one or two spots before and after the query token are considered in this analysis, i.e. $\mathbf{F}_{-2}$, $\mathbf{F}_{-1}$, $\mathbf{F}_{+1}$, and $\mathbf{F}_{+2}$
  - Relative frequencies of tokens immediately before and after the query token, i.e. $\mathbf{F}_{-1}(q)$ and $\mathbf{F}_{+1}(q)$ are normalized such that each sums to 1.0; the positions two before and two after the query token, i.e. $\mathbf{F}_{-2}(q)$ and $\mathbf{F}_{+2}(q)$, are considered to be less important, and so are normalized such that each sums to 0.5

Note that each of these sets above could properly also have a superscript '1', indicating tokens at level one of the grammatical structure hierarchy—i.e. word level; however, the superscript has been omitted here since only the word level grammatical structure is being considered at this time.

This system is implemented in the python programming language due to its ease of use for prototyping, its powerful string manipulation capabilities, and the existence of the NLTK external library and tutorial. Fuzzy clustering and density clustering were both implemented directly by the investigator, to ensure full control if necessary.

6. Difficulties

The first major difficulty encountered involved the degree of fuzziness allowed in the clusters—regulated by the "fuzzifier" variable $m > 1$ (typically between 1 and 2) in the fuzzy clustering algorithm, with 1 the set crispness increasing as *m* decreases towards 1. Allowing too much fuzziness caused two major problems:

- Significant increase on the number of steps—and therefore the time—required for convergence
- Tendency to converge to "mush", i.e. all nodes belonging to all clusters with equal degrees, especially for larger numbers of clusters

Tuning the fuzzifier variable to 1.1, i.e. fairly crisp sets, allowed convergence within a reasonable time and completely eliminated any convergence to mush in these experiments, while still giving some degree of indication that certain nodes should properly belong to more than one cluster (if the results were interpreted with a sufficiently low membership cutoff level, for example 10%). This was not an ideal solution, however, because less fuzziness meant nodes could not belong as obviously to more than one cluster.

The second major difficulty encountered involved scalability relating to the limited processing time available. Fuzzy clustering does not scale well to large numbers of nodes, high dimensionality attribute arrays, or large numbers of clusters, all of which are present in this data. To facilitate preliminary investigations, node quantities were limited to 20 or 40, attribute quantities were limited to 200 (times 4), and number of clusters were limited to around one dozen, which kept processing time down at 1 to 15 minutes on the development machine. In a final trial, these limits were raised to 200 nodes and 24 clusters, which required just over 12 hours of processing time.

The third major difficulty encountered involved scalability relating to the limited memory available. For an ideal representation of the data involved, it is desirable to include all tokens in the clustering, and for each token to have attribute arrays which include relative frequencies of all tokens in the relevant positions one and two spots before and after. For the chosen novel, this would require an approximately 7000 * 28000 array of floating point values, or ~750MB for 4-byte floating point variables. For the number of words in a typical college dictionary, i.e. ~200K words, this would increase to over 600GB. No memory limitation was actually reached because only the most frequent 200 words were processed in order to limit processing time; however, had this limitation been reached, it could have been resolved primarily by using a "sparse array" type of data structure where zero-valued entries are not actually stored, since most word pairs never occur in practice, especially for low frequency words.

It should also be possible to improve upon each of these three problem situations through "token merging". This would involve performing clustering on the tokens in stages and then, at the end of each stage, combining tokens in the attribute arrays which belong strongly to each particular cluster. If we assume that there are perhaps 50 parts of speech clusters to be found, then the attribute array would need only to contain 4 * 50 = 200 clustered entries (plus any words which do not yet belong strongly to a cluster), thereby keeping the attribute array length reasonable. The smaller attribute arrays would also reduce processing time, and could be expected to decrease or even eliminate the possibility of converging to mush. Regarding the attribute arrays, it may also improve results to include the relative frequency of the token itself as an attribute, since extremely common words are more likely to be similar with each other than with an extremely uncommon word.

A fourth difficulty involved words not falling into multiple parts of speech clusters with fuzzy clustering, due partly to low fuzziness employed as needed for both convergence and speed. Density clustering was then investigated, as it also allows tokens to belong to multiple clusters; however, there were extremely few occurrences of tokens belonging to more than one cluster. Part this can no doubt be attributed to the tendency of a particular author to use a word predominantly as one particular part of speech, rather than all possible parts of speech. However, it appears that density clustering might simply be unable to perform the highly sensitive task of grouping appropriate words together while still allowing enough overlap of groups to allow membership in multiple groups as appropriate, at least under the experimental scenarios investigated so far. Perhaps the contributions of different attributes to membership in different parts of speech must be teased apart from each other, by subtracting out cluster averages for example, and the token split or cloned. This would be an additional process, rather than expecting the clustering process to take care of membership in multiple clusters by itself.

7. Results and Evaluation

Table 2, below, compares the clusters generated by simple fuzzy clustering and by simple density clustering for the top 20 most frequent tokens encountered. In both cases, algorithm parameters have been tuned to give the best results possible. The results of density clustering are deemed superior due to better coherence within a cluster and the additional upside of no false positives in its clustering. Some of the tokens do not have a pair token to be clustered with in the top 20 most frequent tokens, and so it is correct that they not be clustered under density clustering; others would have been clustered under a hierarchical scheme, as seen later in Table 4, but were not under simple clustering.

Table 2 – Initial Comparison of Simple Fuzzy Clustering To Simple Density Clustering

Simple Fuzzy Clusters (ordered by degree of membership)	**Cluster Type**	**Simple Density Clusters**
, ; --	Phrase Separators	, ; --
the a her	Determiners & Possessive Pronouns	the a her
he she i you it	Pronouns	he she i you it
of in is was to for had	Verbs (n-person singular)	is was had
	Prepositions	of in to for
have be not had	Verbs (miscellaneous) + 'not'	
that but and as	Simple Conjunctions	but and
	Other Conjunctions	that as
*	Sentence Splitter	
.	End of sentence	
“	Quotation mark	
	Correctly Not Clustered under the two members requirement	* . “ not
	Not Clustered	have be

The results of the simple fuzzy clustering (10% membership cutoff) of the top 208 most frequent tokens are given below in Table 3. Note the intriguing result, highlighted in green, that parentheses appear not to act like other punctuation, but act instead like words—specifically, the conjunction part of speech. Also note that cluster cohesion is significantly less than in density clustering in Table 4. Since fuzzy clustering was not optimized, due to processing time constraints, it is not a fair comparison to indicate errors as in density clustering in Table 4; cluster were not labeled as carefully (possible errors).

Table 3 – Simple Fuzzy Clustering

#	**Cluster Type**	**Cluster Members** (ordered by degree of membership)
1	Sentence Splitter	*
2	End of Sentence	! . ?
3	Phrase Separator	; , : --
4	Quotation Mark	“
5	Conjunction	that and but when though which if while as ( ) or than even sure
6	Interjection; Adverb	oh yes
7	Prop Noun; Comm Noun	weston elton woodhouse bates churchill knightley fairfax father body
8	Noun	day letter moment time way man morning friend woman room thing mind pleasure body sort other father
9	Title & First Name	mr mrs miss frank jane
10	Pronoun + First Name	he she they we i there you it emma who harriet
11	*?*	him them himself herself away me us highbury again hartfield home out love up done over enough pleasure mind more sure body

12	(Poss) Adj + Adv + Det	the his your a some their every an such any its one my another her this many something no - nothing so all too two 's own
13	Verb (Auxiliary)	should must can will may shall cannot would might could am
14	Verb (Person Singular past/present)	was is had has were does seemed came are did saw felt heard am
15	Verb (Present Trans. A.)	give make be take see go say come have look heard
16	Verb (Present Trans. B.)	think hope wish know do thought sure
17	Verb (Present Part.)	being having going half
18	Adjective	good little first young most great last better very other two sort 's own more too - look
19	Adverb	always not never really only ever rather just still quite been heard no so nothing felt too
20	Adjective	much long happy well soon sure enough
21	Preposition + Adverbs	in for from on with by of at to about into upon without after made like before even all heard than too so or over
22	Misc Conj. (Interjections, Adverb+Adjective types)	now indeed then here yet however
23	Interrogative Pronouns	how what poor
24	*?*	dear said

The results of the hierarchical density clustering of the top 210 most frequent tokens are given below in Table 4, with errors or poor quality memberships highlighted in yellow (22 / 210 = 10%), including two entire clusters. Note that 64 tokens (30%) did not join any cluster.

Table 4 – Hierarchical Density Clustering

#	Cluster Type	Density Cluster Members
1	End of sentence	? . !
2	Phrase Separator	{; -- ,} :
3	Title (noun)	mrs mr
4	Last name (proper noun)	{{weston knightley elton} woodhouse fairfax bates}
5	Noun	woman man
6	Noun	room morning moment day
7	Noun	thing body
8	Noun, Adv	something nothing
9	Adj, Adv, Noun	last first
10	Adj & Pron	two other
11	Noun/Pron, Verb	own father
12	Verb (Auxiliary)	have be
13	Verb (Present Participle)	having being
14	Verb (Present Transitive)	take see make give
15	Verb (Per. sing. past/pres) Verb (Auxiliary) Verb	{{were was is had are} has} {{would will must could} should might may did} {shall seemed does cannot can}

16	Adj, Noun, Adv	much long
17	Adj, Noun, Adv	more better
18	Adverb (incl. emphatic)	really never ever always
19	Adv, Inter; Pron Pronoun (sing; plural) First Name (proper noun) Adv, Adj; Pron, Adj, Det; Adj, Adv, Pron/Noun Relative Pronoun Conjunction, Noun Conjunction Adj, Noun, Adv Preposition - Adverb	{there it} {{they she i he} we} {{harriet emma} jane} {{very so} {this them the such my me his him her every any an a} your too their some most herself} {which that as} {when if} {or but and} {little great} {{with to on of in from for by at} without into about} {you while what us upon up though over out one no myself its how home himself highbury hartfield good few} {before away another all again after -}

The results were evaluated according to how cohesive the clustered groups were, what proportion of tokens were clustered, and what proportion of tokens were clustered poorly or in error. The correctness of cluster membership was determined by comparing against the parts of speech tags for the words in the Canadian Oxford Dictionary. Words may be used as multiple parts of speech, but often a particular author will have a bias in how they use the word; so, if there was a commonality or commonalities to the dictionary derived parts of speech possibilities for all or most tokens in a cluster, then it was assumed that this was the way the word was being used. For example, all words in a cluster can be an adverb, among other things which do no match cohesively; in this case, it is assumed that the words are being used primarily as adverbs. The same could hold true for multiple parts of speech at once, in which case the cluster name orders those possibilities identically to the dictionary order (which indicates the prevalence of that word's usage as that part of speech) or with an ampersand between them as the dictionary does (which indicates acting as both parts of speech simultaneously).

Note that, due to the preliminary and exploratory nature of this investigation, the evaluation was quite subjective and was done by hand. For a more thorough and robust investigation, a more sophisticated method of evaluation will be needed, and it should be automated (e.g. using electronic dictionaries).

The fuzzy clustering clustered all tokens as it is intrinsically required to, although the groups were not perfectly cohesive, and 10% of tokens were clustered poorly or in error. No hierarchical version of this fuzzy clustering was attempted, due to limitations on processing times. Fuzzy clustering worked well whether the distance measure was a euclidean or city-block metric.

The density clustering clustered only 70% of tokens, but those clusters had better cohesion and virtually no errors. Density clustering worked noticeably better with a city-block distance metric than with a euclidean distance metric, and so that was used for all experiments. The radius, e, was adjusted until the optimum (i.e. maximum) number of clusters was found as a primary consideration, and the optimum (i.e. maximum) total number of tokens were clustered as a secondary consideration. Note that a form of hierarchical clustering was performed, by looking at three different quantities of input data:

- 76 tokens: minpts = 2, e=1.925, 13 clusters found directly, 72% of tokens clustered
- 152 tokens: minpts = 2, e = 1.975, 15 clusters found directly, 52% of tokens

clustered

- 210 tokens: minpts = 2, e = 2.2225, 19 clusters found directly, 70% of tokens clustered
- Hierarchy: 42 clusters found in total (counting all sub-clusters in the hierarchy), as reported in table above.

This hierarchical addition to the density clustering method was performed by hand for simplicity, though it could easily be automated.

As an initial probe into clustering and grammar, the investigator felt that the learning was extremely successful, especially since there are obvious major improvements which would almost certainly improve the results and the performance (these were discussed under the Difficulties section), and also considering that there was no guarantee for these methods to work *at all* on this unsupervised learning problem.

Fuzzy clustering is a slightly more flexible method, but it is slow, results may vary from one trial to the next, and it has the drawback of false positives. Density clustering is slightly less flexible, but is orders of magnitude faster, has highly repeatable results, and does not have the downside of false positives. Hierarchical density clustering allows the most features to be extracted (i.e. the most representative clusters to be found), while still maintaining the benefits of density clustering in this domain, and so is currently the preferred clustering choice for these investigations.

There were two sub-optimal aspects to the specific experiments run which were not found until afterwards:

- The two spaces after any end of sentence punctuation—i.e. period, question mark, and explanation point—were replaced with a single star acting as a sentence splitting token. Properly, *two* stars should have been used, so that the system is not able to see past the beginning or end of a sentence with its word pair frequency analysis (which can see two tokens ahead or behind). However, this appears to affect neither the fuzzy clustering nor the density clustering results, because end-of-sentence punctuation tokens are still grouped together appropriately in both cases.
- Hyphenated words were split into three tokens—e.g. "twenty-one" became {"twenty", "-", "one"}. Properly, these should have remained a single word because the hyphen is a part of the word and not a punctuation token in the sentence. This had a minor negative effect on the density clustering, because the mis-clustering of the hyphen is the only error made by the system using density clustering. One expects this to have a negligible affect on fuzzy clustering, because removing that token from its cluster would likely not shift its center any significant amount.

Finally, over the course of this investigation, it became increasingly clear that the statistical relationships within a text reflect both patterns of structure *and patterns of meaning* because the discovered clusters do not simply reflect groupings by grammatical structure but rather also reflect groupings by semantic meaning. Examples of this include the pairings: {man, woman}, {first, last}, {something, nothing}. Other examples involve many many other clusters which divide apart words which are properly the same part of speech, but whose semantic meaning causes them to be used quite differently; for example, the Adverb clusters in general. Language structure and semantic meaning appear to be much more strongly coupled than one might naively assume, and there appears to be much to learn here from a purely linguistic perspective.

8. Future Work

As described in previous sections, significant improvements could likely be made by implementing the following:

- "Token merging" within attribute array to reduce the effects of noise and reduce processing time
- Include a token's own relative frequency as part of its attribute array to improve accuracy
- Token cloning/splitting (including properly allocating the appropriate portion of the attributes to the appropriate cloned/split token) to allow membership in multiple clusters, while decoupling that capability from the type of clustering used

As well, this investigation should be extended to include include multiple levels of the natural language grammatical structure hierarchy, and not just the word-level. This involves more than simply extending the clustering methods to different levels in the hierarchy. Rather, it requires finding suitable methods for tokenizing text at each level (which was skipped in this investigation by the preprocessing employed which split words at the space character, and added sentence splitter characters). This investigator expects that statistical/frequency methods will also prove to be appropriate for this type of task, but that iteratively performing tokenizing and clustering activities while sliding up and down the structural hierarchy levels will also be required.

9. Future Applications

A full investigation would extend from the character-level up to multiple sentences or paragraphs in the natural language grammatical structure hierarchy, as discussed in the Future Work section. Results from such an investigation could have many applications:

- One more tool to help human linguists study language structure, and perhaps even assisting in the translation of heretofore untranslated languages
- Automated grammar checkers with rules generated automatically, potentially for any language, and without requiring human domain experts to hand-build the rules
- Automatically calculating a "grade level" for the text, potentially in any language, and without requiring human domain experts to hand-build the rules
- One more tool in the spam filtering toolbox
- One more tool in the search and retrieval toolbox
- One more step towards a "strong" artificial intelligence which can learn just by reading text

Even at only the word level analysis, this investigation has already shown some worth for the first application above—helping linguists study language—by discovering the potentially novel result that parentheses are more like a word (i.e. the conjunction part of speech) than they are like any other punctuation.

This research direction appears to hold much promise.

10. References

- Natural Language Tool Kit and Tutorial, http://nltk.sourceforge.net/.
- Project Gutenberg, http://www.gutenberg.org/.
- University of Calgary CPSC 601.66 Machine Learning Course Notes, Dr. Michael M.

Richter, http://pages.cpsc.ucalgary.ca/~mrichter/ML/.