ICS5110

APPLIED MACHINE LEARNING

January 13, 2016

Matthew Zammit
University Of Malta
M.Sc Artificial Intelligence

Contents

0.1 INTRODUCTION

In this project the issue being tackled is that of predicting the winner of a tennis match by using supervised Machine Learning techniques on labelled data of previously played matches. In this Section, a brief description of the game of Tennis and an explanation of the dataset is given.

0.1.1 The Game Of Tennis

In this section a brief explanation on how the game of tennis is played and some basic rules of the game are given. In this explanation only the game type known as 'Singles' is discussed. 'Singles' is a game played between only two opposing players. Other types of tennis matches may consist of teams of players, known as 'Doubles'. A tennis match is made up sets, games and points. The player who wins the most of a predefined number of sets wins the match. Usually these are best of 3 for women and best of 5 sets for men. The first player to win 6 Games wins a set. There are situations where a tie-breaker is needed for a player to win the set. However, this scenario is not dealt with in this explanation. The player would need to score points to win a game. Points in tennis are made up of a set of values $\{0, 15, 30, 40\}$. In a new game both players start from 0. The player who wins the next point after 40 wins the game. If both players have 40 points each, one player has to win by two clear points to take the game.
One important thing about the game of tennis is that the player who is serving for the game has the advantage for that game and therefor should win it. If the player does not win the game he is serving for he loses the advantage he had. Thus the opposing player now would have a game in hand and the next service. This scenario is called a Break. Players alternate between receivers and servers throughout the match, so that each have the service advantage. When serving, the ball has to land in the correct service box of the opposing player to continue playing. The server has two chances(1^{st} and 2^{nd} serve) to play a correct serve. If he fails to do so, a point goes to the opposing player. This is known as Double Fault.

Table 1: A Typical Tennis Score Board

	SETS	GAMES	POINTS
Player A	2	5	40
Player B	1	3	15

In Table 1, it could be seen that player A is playing for what is called the match point. If he wins the next point, he will win the game, the set and the match. It is a common saying for tennis commentators to say "Game. Set. And Match." when a player wins a match.

0.1.2 The Dataset

The dataset was retrieved from the UCI[1] Machine Learning Repository. The dataset consists of 8 CSV files containing the statistics of all the final rounds matches of the major tournaments for both men and women of the year 2013. Each file represents a particular tournament with the statistics of each match in

[1]https://archive.ics.uci.edu/ml/datasets/Tennis+Major+Tournament+Match+Statistics

that tournament and with target value $t \in \{1,0\}$. $t = 1$ means that player 1 won the match, whilst 0 means player 2 won the match(or Player 1 did not win the match). Table 2 displays the attributes recorded for player 1 for each match in the dataset. This is same attributes are present for player 2.

Table 2: Attributes for Player 1 with Target Value

Attribute	Attribute Description	Attribute Type
Player 1	Name of Player 1	String
FSP.1	First Serve Percentage for player 1	Real Number
FSW.1	First Serve Won by player 1	Real Number
SSP.1	Second Serve Percentage for player 1	Real Number
SSW.1	Second Serve Won by player 1	Real Number
ACE.1	Aces won by player 1	Numeric-Integer
DBF.1	Double Faults committed by player 1	Numeric-Integer
WNR.1	Winners earned by player 1	Numeric
UFE.1	Unforced Errors committed by player 1	Numeric
BPC.1	Break Points Created by player 1	Numeric
BPW.1	Break Points Won by player 1	Numeric
NPA.1	Net Points Attempted by player 1	Numeric
NPW.1	Net Points Won by player 1	Numeric
TPW.1	Total Points Won by player 1	Numeric
ST1.1	Set 1 result for Player 1	Numeric-Integer
ST2.1	Set 2 Result for Player 1	Numeric-Integer
ST3.1	Set 3 Result for Player 1	Numeric-Integer
ST4.1	Set 4 Result for Player 1	Numeric-Integer
ST5.1	Set 5 Result for Player 1	Numeric-Integer
Result	Result of the match	Binary(0/1)

Upon seeing the data set two issues were found. First issue was that some attributes were not needed for this exercise. For example the games won in each Set by a player(ST1.1,ST2.1,...,ST5.1) same for player 2. These could be considered if values were to be normalised per game instead per match. However, for this project they were ignored. Another attribute that was not used was the player name. This could be used to train the models on the players' past performances. However, the data set would need further pre-processing as in some files the player's first names were recorded with initials only and there was no unique id to distinguish the players. The second issue with the data set was that there were some missing values. For some tournaments whole columns of attributes were missing. These could not be used for this project and so those attributes were not considered.

In the following section, the Machine Learning Algorithms used for this problem are discussed in detail. These algorithms are Decision Trees(DT), Random Forest(RF) and Artificial Neural Networks(NN). The Decision Tree algorithm was chosen so that rules could be extracted from the model generated so that one could see which attributes are best to look at for predicting matches. Random Forest were chosen because they work well with large data, they add a some randomness and overcome the overfitting problem that DT tend to fall for. Neural Networks were considered to have a different approach at solving the problem. Other methods used to help these algorithms with problems revolving around the raw data are also discussed in this section. These consist of techniques such as Vector Normalisation and Feature Re-Scaling, Principal Component Analysis and Feature Selection with Genetic Algorithms. Also techniques on how the models are to be evaluated are discussed.

0.2 BACKGROUND

In this section a variety of machine learning techniques are discussed in detail. These range from techniques used for supervised learning classifiers, advantages and disadvantages of such techniques; problems and techniques dealing with the size of the feature space and the ranges of their values. Also methods on quantifying how good the classifier perform are discussed in terms of evaluation metrics.

0.2.1 Machine Learning Algorithms

In this section the supervised machine learning algorithms used in the project are discussed. K-Nearest Neighbours(KNN) algorithm was used early on in the project as a quick means to check if the problem could be further investigated by more advanced machine learning techniques. Thus it is vital to mention briefly what the intuition behind KNN is and describe also how and why the algorithm works. After this, several other techniques were used such as Decision Trees, Artificial Neural Networks and as well ensemble techniques such as random forest are discussed in detail.

0.2.1.1 KNN

The intuition behind KNN is that similar points(of the same target) should be located close(near) to each other in the space they occupy. KNN is a very simple technique that allows fast implementation and quick intuition on the complexity of the dataset. The training dataset in KNN, are the vectors themselves with their corresponding target value. To classify a new test vector, the euclidean(or other) distance between the test vector and the vectors in the training dataset is computed. The targets $t_1, t_2, ..., t_k$ of the least k distances between $\vec{q}$ and $\vec{x}_1, \vec{x}_2, ..., \vec{x}_k$ are taken and the mode is computed on the list to get the target with the highest count and assign it to the new data point. Although the mode is mentioned as the function that classifies the vector; other methods can be used such as weighting the distance between the $\vec{q}$ and the top k vectors. Algorithm 1 shows briefly the inner workings of KNN.

Algorithm 1 KNN algorithm

```
1: procedure KNN($k, \vec{q}, X$)                  ▷ k, $\vec{q}$ query vector, X is the training data
2:     dsts ← dist($\vec{q}, X$)  ▷ matrix of $(d_i, t_i)$; distances between $\vec{q}$ and X, with corresponding targets
3:     sort(dsts[ :  ,0],ASCENDING)                ▷ Sort the list from least to greatest distances
4:     return mode(dsts[1:k,1])                    ▷ Return the mode of first k rows of the target column
5: end procedure
```

Some disadvantages of KNN are that this technique does no actual learning, in the sense that the original dataset is kept, and is not replaced with some prototypes or some weights of a function. Thus for every new query vector, the distances have to be recomputed so that it could be classified. So while training is fast, testing on the other hand is slow $O(n^2)$ and increases rapidly with the growth of the data. In KNN, k is a hyper-parameter which the user has to define and thus is not be learnt by the algorithm. The best k, for the testing set can be found by the use of methods such as k-fold cross validation.

0.2.1.2 Decision Trees

A Decision Tree(DT) is a white-box classifier. Meaning that the model produced by the classifier can be easily interpreted and understood. Thus a person would be able to know why certain samples are classified into a particular class. This is a desired property of models in a lot of areas. A DT creates axis-parallel splits in the feature space according to how an attribute discriminates between the classes. This decision splits the space with a decision boundary orthogonal to that axis. Many of these decisions at different axis would create different decision boundaries intersecting each other at right angles. Thus, forming decision boundaries of the form of rectangles in that space.
The order of the attributes chosen by the DT depends on how well that attribute is able to discriminate between the classes. An attribute which is able to split the space in half for 2 classes or k for k classes is a much better attribute than an attribute with less discriminative ability. This would produce the purest children nodes. For this the entropy(impurity) produced by that attribute and its values needs to be measured. Entropy is defined as follows;

$$S = \sum_i -p_i \log p_i$$

Where, p_i is the probability of class i and S is the entropy.
One way of constructing trees by using entropy is by the use of Information Gain,

$$IG_i = S(i) - \sum_i^v p(v_i) * S(v_i)$$

DT however tend to overfit and create complicated trees that do not generalise the data well. Pruning is a method used to tackle this problem where the tree is cutoff at the point where the score on the validation set starts to decrease. Overfitting can especially happen when there is large feature space and the number of samples is small. Dimensionality reduction and/or feature selection techniques might be used to help DT perform better. The way DT works is by using heuristic learning mechanisms that find the local optimum solution at each node in a greedy fashion. Finding a DT that is globally optimum across the data is an NP-Hard problem[1]. Some of the problems mentioned above can be tackled by using a ensemble technique that creates many Decision Trees. This particular type of ensemble technique is called a Random Forest.

0.2.1.3 Random Forest

Random Forest(RF) is an ensemble machine learning technique that can be used for both classification and regression problems. RF constructs multiple and independent DT on a subset of samples, by using a bootstrapping technique. Bootstrapping takes n' random number of samples from a dataset of n instances with replacement. Meaning that the same sample may be used twice or more in the training phase. The rest of the training samples are unique for that training set. A DT is then trained on this subspace of the sample and added to the RF. This procedure is done several times to build a k number of estimators. This approach add randomness to the construction of the tree. For classifying vectors, RF takes the vote of each classifier built and the vector is then classified to the class predicted by the majority of the estimators. The parameters for RF are the number of trees to construct(k) and the maximum number of features(f') to use. The more DT present in a RF model is the better. However, after a number of estimators the results would stop improving.

0.2.1.4 Artificial Neural Network

Artificial Neural Networks(NN) is a machine learning technique inspired by how the biological and neurological brain works. The technique makes use of synapses, perceptrons and activation functions such as found in a brain. The NN is based on the logistic regression and tries to solve the problems that logistic regression alone could not, for example XOR can not be solved by logistic regression alone. NN solves the problem of logistic regression by adding a hidden layer with multiple units in between the input and the output layers. In this layer higher order terms are learnt by the network on their own. So the user does not have to specify the features before hand for the algorithm to work. The formula below shows the logistic function(sigmoid) also known as a squashing function. The output of the sigmoid is between 0 and 1, for $z \geq 0$ the function outputs

$$h_\theta(\vec{x}) = \frac{1}{1 + e^{-\sum_{i=1}^{n} \theta_i x_i}}$$

The NN is basically a sum of weighted logistic functions that are able to generate high order terms that are able to create complex decision boundaries that are then squashed again by a logistic function to output the prediction for the input given. NN make use of the Back Propagation method to update the weights of each layer that that target value depends upon. NN Can also you a gradient descent or stochastic method to update the weights. With gradient descent the update is done in batches. So after all the training examples have been seen it update the new weights according the average sum error of all the training examples. With the stochastic approach, after every each sample seen it will adjust the weights. The stochastic is faster and one stop it at any point during the learning. With gradient descent one has to wait.
NN are also known to overfit the data, and they generate black box models.

0.2.2 Feature Scaling

Feature Scaling is a set of techniques used to standardise the data in the preprocessing stage of the procedure. Data is usually recorded as it is observed in the form of statistics. So the ranges of each attribute recorded may vary a lot from the other attributes' ranges as can be seen in dataset used for this project. Thus attributes with large values or variation may end up contributing more than attributes with smaller ranges. This may effect the performance of some machine learning techniques whilst others are not effected by this problem. It is dependent on the nature of how the machine learning technique used 'learns' the data. For example, the gradient descent algorithm converges faster by using a normalised dataset, whilst for Decision Trees the data does not need to be normalised before being used. Feature scaling techniques are used to tackle this problem. In this section we will discuss two feature scaling techniques used in this project in the form of vector normalisation and feature rescaling.

0.2.2.1 Vector Normalisation

Vector normalisation is a technique used to scale down vectors to unit length, meaning of length 1. This technique ensures that each component of the vector contributes equally. The normalised vector is constructed in the following way,

$$\vec{x'} = \frac{\vec{x}}{||x||}$$

Where $\vec{x} = x_1, x_2, ..., x_p$ are the vector components and $||x|| = \sqrt{x_1^2 + x_2^2 + ... + x_p^2} = \sqrt{\sum_{i=1}^{p} x_i^2}$ is the length of the vector. All the components of each vector is divided by the length of that vector so that the norm of that vector becomes 1.
Vector normalisation benefits machine learning techniques that use euclidean distance between the vectors to calculate similarities between the vectors. For example, KNN is one algorithm which would benefit from this preprocessing step.

0.2.2.2 Feature Rescaling

Another technique used to standardise data is feature rescaling. Feature rescaling scales down the attributes of the vector to the range between [0,1]. The normalised component is constructed as follows;

$$x'_{ij} = \frac{x_{ij} - min(x_j)}{max(x_j) - min(x_j)}$$

where x_{ij} is the current component of the matrix of row i, column j and $min(x_j)$, $max(x_j)$ are the minimum and maximum values of the column vector j, containing all the values of that attribute for each vector.
The formula to set the range of each attribute between 0 and 1 has been described above. However, this is no the only range to use. Other ranges such as [-1,1], [-0.5,0.5] and others can be used depending on the nature of the dataset and the problem to be learnt.

0.2.3 Feature Selection and Dimensionality Reduction

Recorded features in the dataset may not be useful to discriminate enough between the different classes, and their inclusion in feature selection may effect the performance of the learning algorithm. Certain attributes might work well together whilst adding others might have a negative effect on the performance of the classifier. In this section the applications of genetic algorithms for feature selection and PCA for dimensionality reduction will be discussed on how these methods tackle the mentioned problems of feature space.

0.2.3.1 Principal Component Analysis

As already described above observed data is usually collected in the form of statistics. So, the number of dimensions of the dataset depend on the number of attributes used for recording the instances being observed. This is called the observed dimensionality. However, it may be the case that the pattern to be learnt lies in lower dimensions than that of the observed dimensions. This is called the true dimensionality. If this is so, the machine learning technique would thus be trying to learn events which will not happen, instead on focusing on what actually needs to be learnt. Thus wasting resources, such as time and space because the machine learning technique has to take into account for each observed event that has been recorded. Some attributes of the collected data may be highly correlated together, such that the movement of one attribute explains the rise or fall of another attribute. These attributes may rise or fall together or as one rises the other one falls. Principal Component Analysis(PCA) is a technique used to tackle this issue of high dimensionality.
Principal Component Analysis(PCA) works by investigating the relationships between the attributes of the dataset(i.e the covariance between each attribute) and the high dimensional vectors are then projected onto a lower dimension depending on how much information is to be kept from the higher dimensions.
The procedure of PCA starts by computing the covariances between each and every attribute. The covariance of two attributes is computed by the following;

$$cov(x_1, x_2) = \frac{1}{n}\sum_{i=1}^{n}(x_{i1} - \mu_{x1})(x_{i2} - \mu_{x2})$$

Where μ_{x1} is the mean of x_1 and μ_{x2} is the mean of x_2.
What is important to note from the value of the covariance is the sign and whether it is close to 0. A positive number shows that the attributes move together and a negative value shows that the attributes move against each other. A value close to 0 from either side of the axis shows that there is little relationship between the attributes.
Instead of using the formula above directly, the data is first centred at 0 by computing $(x_{i1} - \mu_{x1}$ first so that a covariance matrix is constructed as follows;

$$\sum = M.M^T = \begin{bmatrix} \sigma(x_1,x_1), \sigma(x_1,x_2), ..., \sigma(x_1,x_n) \\ \sigma(x_2,x_1), \sigma(x_2,x_2), ..., \sigma(x_2,x_n) \\ \dots\dots\dots\dots\dots\dots\dots\dots \\ \sigma(x_n,x_1), \sigma(x_n,x_2), ..., \sigma(x_n,x_n) \end{bmatrix}$$

Where $\sigma(x,y) = (x'_{ij})(y'_{ij})$ and $M = [x_{ij} - \mu_j]$.
Now that the covariance matrix is found, the next step in PCA is to project the vectors to dimensions always taking into consideration the next highest variance with respect to the latest projection such that it is orthogonal to that axis. This is done because PCA tries to preserve as much variance as possible, so that the least information is lost with each projection to the next Principal Component(dimension). PCA achieves this step by finding the Eigen values and Eigen Vectors of the covariance matrix. These are special vectors that do not change their direction under a linear transformation. This is exactly what is needed for this application. The eigen values of the covariance matrix are found by the following equation,

$$A\vec{v} = \lambda\vec{v}$$

$$\det(A - \lambda I) = 0$$

Once $\lambda_1, \lambda_2, ...\lambda_n$ are found by the above equation, the values obtained can be substituted back into the equation to solve for the Eigen vectors $A\vec{v_1} = \lambda_1\vec{v_1}$and $A\vec{v_2} = \lambda_2\vec{v_2}$ up to $A\vec{v_n} = \lambda_n\vec{v_n}$. Where A is the variance-covariance matrix and $v_1, v_2, ..., v_n$ are the eigen vectors.
The sum of the eigen values $\lambda_1, \lambda_2, ..., \lambda_n$ explain the variance of all the principal components which is equal to the total variance of the original matrix.
As shown above PCA assumes that the data is linear in nature as both covariance and Eigen vectors work with linear relationships.

0.2.3.2 Genetic Algorithm

Genetic Algorithm(GA) is an optimisation technique inspired by biological concepts of evolution. GAs are used to find global minimum to an optimisation problem. The algorithm makes use of biological concepts such as natural selection, mutation and reproduction to find solutions. GAs use an objective function to find the best fit also known as the fitness function. Interestingly Genetic Algorithms can be used for the problem of feature selection. Feature Selection is also an optimisation problem(finding the best set of features).

Algorithm 2 Genetic Algorithm

```
    procedure GA(epochs, n_pop, n_par, mut)        ▷ #epochs, #population, %parents , %mutation
2:      for e = 0; e < epochs; e++ do
            if $e = 0$ then
4:              pop ← initPopulation(n_pop)
            else
6:              for i = 0; i < n_pop; i++ do
                    scores ← fitness(decode(pop[i]))            ▷ calculate the fitness function
8:              end for
                scores ← sort(scores)                           ▷ Sort in descending order
10:             parents ← scores[:n_par]                        ▷ Set parents from highest scoring chroms
                pop ← zeros(pop)                                ▷ Reset population
12:             pop[:n_par] ← parents                           ▷ Include parents in population
                for i = n_par; i < n_pop; i++ do
14:                 par1, par2 ← selectRandom(parents)
                    pop[i] ← crossOverAndMutate(par1,par2,mut)
16:             end for
            end if
18:     end for
    end procedure
```

GAs start with an initial population of random chromosomes. A chromosome is an encoded string of symbols (usually $s \in 0,1$); representing a solution in the search space. In the feature selection case the solution is the set of features that yield the best results for a specific classifier and the chromosome represents the set of features to be used. Every chromosome is passed to a fitness function that returns a score.The results of each chromosome are sorted in a descending order so that the best chromosomes are located at the top. p number of chromosomes are selected from the whole population, where p is a predefined percentage. These selected chromosomes will be the parents for the next generation. The process of creating new chromosome is called cross-over. First, two parents(x_i, x_j) are selected at random. An index $k(k < n)$ is found at random so that the child is constructed from $(x_{i1}, x_{i2}, .., x_{ik})$ (the first parent) and $(x_{jk}, x_{jk+1}, ..., x_j n)$ of the second one. After the child has been constructed, it is passed through a mutation procedure. In this procedure some of the chromosomes' values might get flipped $(0 \leftarrow 1$ OR $1 \leftarrow 0)$ if the random number assigned to a component's index during this step is exceeds a threshold. After mutation, the child created is added to the next generation. The cross-over-mutation procedure is repeated until the maximum number of the population is reached. The parents themselves are also included in the next generation. The same process is repeated for each generation created. The scores of the 2^{nd} generation should be greater than those of the first, since it is filled with chromosomes made by the best chromosomes from the first generation.

0.2.4 Model Selection

Datasets are usually split in Three ways. One split is called the training set, this is used to train the model on. Another split is called the validation set. This set is used to get results from the model learnt so that it could be tweaked and tuned on. A lot of learning models have hyper-parameters that are defined by the user. The last split is the test set. This set is used to test the final model chosen after it has been tuned. Problems arise when one comes to split the data. By having a fixed split, the model might get tuned on only that part of the data, can achieve high scores on that set but is not able to generalise enough to unseen data. The splits maybe fortunate or unfortunate. Meaning that it may be the case that a certain split might contain instances which do not show the true picture of the data set. There are several elaborate techniques used to tackle these problems. With such techniques the true error rate of the model can be computed on the whole dataset. One of these techniques is called the k-fold cross validation method which will be discussed in this section.

0.2.4.1 k-Fold Cross Validation

k-Fold Cross Validation is a method that ensures that all the instances in the data set are considered both for training and testing. In k-Fold Cross Validation, the dataset(i.e test set removed) is split into k partitions. For each experiment a partition is used as the testing set and the rest is used for training. So for a dataset of n number of examples and k partitions, for the first experiment the testing set would be $x_0..x_k$, while $x_{k+1}..x_n$ are used for training. For the second experiment the testing partition moves k examples to the right, such that $x_k..x_{2k}$ is the new test set and the training set is now composed of the left partition $x_0..x_k$(was the testing set of the last experiment) and the right partition $x_{2k+1}..x_n$. This keeps going until the testing parting moves to the end of the dataset. Thus ensuring that all the examples have been considered for both testing and training the model. Sometimes, the examples are shuffled before they are passed through this method. The number of folds used (i.e k) depends on the number of examples in the dataset. For large datasets, as a rule of thumb $k = 3$ or $k = 10$ should be enough. For smaller datasets, it might be considered that k be larger, such that only one example is left for testing for each experiment. This method of choosing $k = n$ for k-Fold Cross Validation is often called the Leave-One Out method. When having a large k, such as is the Leave-One-Out method the computation is much larger than having a smaller one since there would be more experiments to be carried out.

0.2.5 Evaluation Metrics

Once trained the model has to be tested to quantify the quality of the output of the classifier is. This process is known as the evaluation of the classifier. In this section Accuracy and the Area Under the Curve(AUC) of the Receiver Operating Characteristic(ROC) are discussed.

0.2.5.1 Accuracy

Accuracy is the ratio of correctly classified examples over all the examples. When a classifier classifies an example it can classify a positive class as a positive (True Positive) and a negative as a negative(True Negative). In either case the algorithm classified the examples correctly. However, it can also classify a positive as a negative (False Negative) and a negative as a positive (False Positive). With these terms one can build formulas that have a meaning with respect to how good the classifier is able to perform something. One of these is Accuracy and is defined as follows;

$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN}$$

This formula is able to show how many correct classifications the algorithm made from all the possible classifications.

0.2.5.2 ROC Curve and AUC

Other metrics such as True Positive Rate(TPR) can be computed from the mentioned cases. TPR is the ratio of the number of positive examples classified correctly over the all the positive examples and is defined as follows;

$$TPR = \frac{TP}{TP + FN}$$

Another metric is the False Positive Rate(FPR)

$$FPR = \frac{FP}{FP + TN}$$

These two metrics can then be plotted against each other to come up with the ROC curve. This curve shows how good a classifier is. The closer the ROC curve is to the (0,1) point the better classifier. The lower it is to the middle means that it has the same probability of getting a correct classification as a coin toss. The Area Under Curve, comes intuitively from what has just been explained. The larger the area under ROC is the better. For a number of thresholds the ROC points are the (TPR,FPR) at that time.

0.3 EXPERIMENTS

In this Section the experiments carried out and the results obtained are discussed in detail.

0.3.1 Dataset Processing

The dataset is made up of 8 CSV files, each file containing the statistics of all the matches of a particular Major tournament of the year 2013. Several issues had to be resolved before the dataset could be used. Some issues had to be solved manually and others programmatically. Some of the CSV files had the values of some columns under the wrong header cell. So the column headers had to be matched manually so that the dataset would reflect the correct information. For example, for some files, the columns for Break Points Won(BPW) and Break Points Created(BPC) were switched. Thus, this would mean that for each match a player would have won more break points than those created, resulting in a Break Point percentage higher than 100 or 1. Same issue was also present for Net Points Won and Net Points Attempted. The statistics of the dataset were checked with other online statistics and it was noted that these were mistakes and that a simple switch of the header names would fix these issues. Another issue was missing values. For missing values, no method was attempted to enter an estimated value such as for example the mean of that attribute. So any missing value was set to -1. This number was chosen because 0 could not be used since it would mean that the attribute was observed. This would result in inaccuracies in the dataset, which would in turn effect the learning of the models. Thus, for any attribute that was being considered and its value was missing the sample it was in would be ignored.
Because of the number of issues present and the steps required to preprocess the dataset, it was decided to store the contents into an SQL database to have the data organised better and for better querying abilities. An SQLite database system was used. SQLite is a database library that does not make use of servers or any configurations and can be implemented quickly. Thus it was a perfect choice for this project. Figure shows the pipeline used to build the datastore as was required.

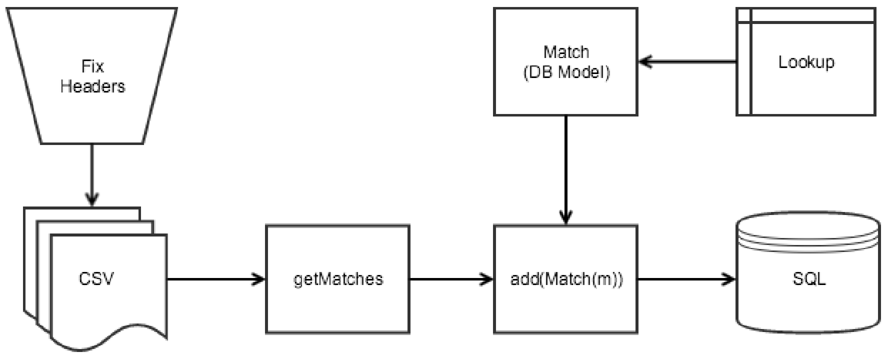

Figure 1: Pipeline. Retrieving, processing and storing the dataset

Additional information could be retrieved from the files and stored in the database. This part involved appending other information to the samples that was not available as part of the statistics gathered but could be taken from meta-information of the file(such as the filename). The information added was the tournament name, gender of the competitors, and year. For example, the filename 'AusOpen-men-2013'.csv was parsed and stored as `match_tour` = 'AusOpen', `match_gender` = True and `match_year` = 2013 so that each sample in that file would have these three new columns added to the match statistics. The dataset was split into two sets. One set was used to train and validate the models. The other set was used as the testing set. The divide was done in such a way that the models would be trained on the first 3 major tournaments of the year and tested on classifying the matches of the last one as shown in table.

Table 3: Order of Major Tournaments in A Year

Order of Play	Major Tournament Name	Month of Play	Purpose
1	Australian Open	January	Train/validation set
2	French Open	May/June	Train/validation set
3	Wimbledon	June/July	Train/validation set
4	US Open	August/September	Test set

0.3.2 Feature Set

Feature Set was the first set used to train the models. WSP and BWP are features that were constructed by the use of other features. WSP(Win Serve percentage) calculates the percentage of the player won on his serve, be it the first or the second serve.

$$WSP._i = (FSP._i * FSW._i) + (1 - FSP._i) * SSW._i$$

Where, FSP = First Serve Percentage,

FSW = First Serve Won Percentage,

SSW = Second Serve Won Percentage

BWP(Break Points Won Percentage) is the percentage the player was able to break his opponent. This means that if in a game there were several opportunities in a match for the player to win a game against his serve but failed to do so, BWP would equal to 0. On the other it would be 1 if the player managed to win all his break points. BWP is calculated as follows;

$$BWP._i = \frac{BPW_i}{BPC_i}$$

Both these formulas were constructed in this work [2].

Where, BPW = Break Points Won

BPC = Break Points Created

Table 4: Feature Set

Feature	Feature Description
FSP.1	First Serve Percentage - Player 1
FSP.2	First Serve Percentage - Player 2
FSW.1	First Serve Won - Player 1
FSW.2	First Serve Won - Player 2
SSW.1	Second Serve Won - Player 1
SSW.2	Second Serve Won - Player 2
WSP.1	Win Serve Percentage - Player 1
WSP.2	Win Serve Percentage - Player 2
DBF.1	Number of Double Faults - Player 1
DBF.2	Number of Double Faults - Player 2
ACC.1	Number of Aces - Player 1
ACC.2	Number of Aces - Player 2
BWP.1	Break Points Won Percentage - Player 1
BWP.2	Break Points Won Percentage - Player 2

Other constructed and observed features could have been included in the feature set such as Unforced Errors, Total Points Won Percentage, Winners and Net Points Won Percentage. However, by including these features the sample space got reduced to 261 samples from a total of 943 samples. This is because some Tournaments have a whole attribute of those mentioned omitted from the observations. By using Feature Set 1, only around 30 instances are removed from the sample set because of missing values. So, in total 711 samples are used for training and validating and 202 samples are used for testing.

0.3.3 Initial Tests with KNN

Initial tests were carried out using the KNN algorithm to check the validity of the problem, to see if the problem could be tackled with machine learning techniques. For the purpose of this experiment both the train and test set were used. k-fold cross validation was used with k=8. The reason behind this number was that since there are a total of 8 tournaments, every test partition would emulate the testing of a whole tournament during the cross validation. The experiments with KNN showed promising results with the best accuracy found with k=13 achieving an accuracy of 68.9% across the whole dataset. The feature vectors were then normalised to unit length and the tests were re-run again. The accuracy improved with the normalised dataset and the best accuracy was of 70.51% with k=16. The accuracy of KNN with k=13 also improved to 69.10% with the normalised dataset. KNN was then not considered anymore for other experiments and more tuning as it had served its purpose to show that other advanced ML techniques could be used.

0.3.4 Decision Tree

After the KNN experiments it was decided to test a Decision Tree Classifier on the dataset. DT have some nice properties as a model; one of these properties is that DT do not need to go through a normalisation or standardisation process. So for this experiment the data was left in its raw state. Another good property of DT that was sought after was that they produce a white-box model. Thus, one could analyse the tree after it has been built and extract rules from the tree and see which features are best at discriminating the classes. The same 8-fold Cross Validation method that was used in KNN, was also used for DT and an accuracy rate of 79.54% was achieved from the model. This method was run once as there was no hyper-parameters defined by the user that needed to be found. Although there are parameters such as `max_depth` of trees and `feature_selection`, it was decided that for this experiment these parameters would be left as default values. Meaning no `max_depth`(i.e DT keeps recursively going until the leaf nodes are found) and `feature_selection` set to full(meaning all of the features would be considered for every node split). The `max_depth` of the tree was dealt with in a separate experiment.

0.3.4.1 Decision Tree with PCA

Since it is known that DT might suffer from high dimensional feature space, dimensionality reduction was applied on the DT using the PCA method. For PCA, the feature values were scaled using the min max rescaling method. This was done because PCA internally works with covariances and scaling every component would help PCA not getting biased by components of large values and ranges. However, one might also mention that even though this was implemented there were no features with exaggerated values that might have hindered other components. The threshold for information kept was set at 0.95 of the original data. With this threshold a total of 8 PCA components of the original 14 were kept. However, the performance of the DT with PCA decreased to a mere 64.34%. Another experiment was done to check which percentage of information kept would yield the highest accuracy. The best percentage to keep was found to be at 0.92 using 7 of the PCA components. However, the accuracy achieved was 66.19%, 13.35% less than that achieved by the DT without PCA.

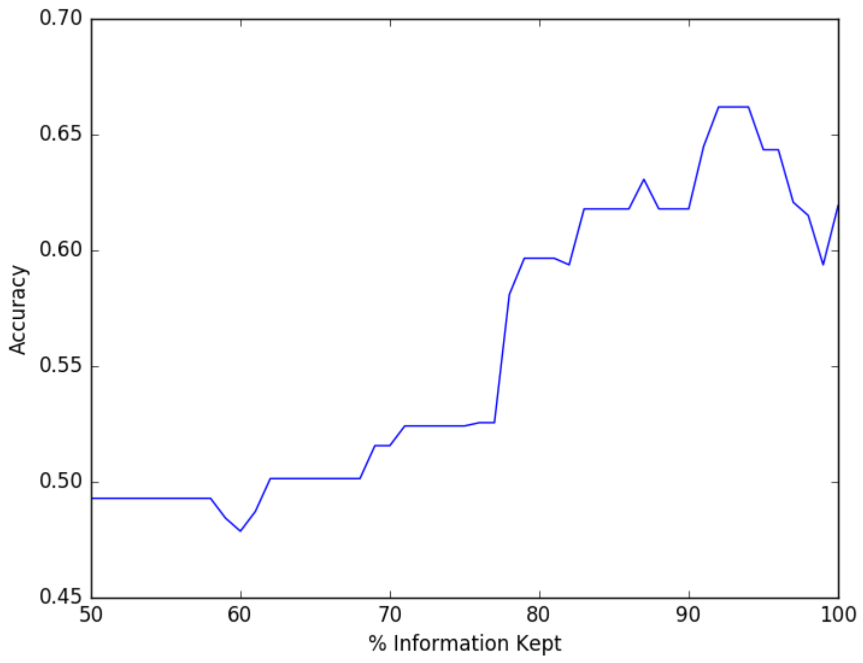

Figure 2: Decision Tree with PCA performance

0.3.4.2 Feature Selection Using A Genetic Algorithm

A Genetic Algorithm was used to search for the best set of features and to see if the accuracy could be improved by the removal of some subset of features. The population rate and mutation rate were both set to 0.2. Both population size and epochs were set to 100. The following fitness function was used;

$$fitness = (10^4 * a) + (0.4 * \#zeros)$$

Where, a = Average Accuracy Rate. The second part of the function promotes the chromosomes with the least number of 0s in their string. This is done because the desired result is to find the chromosomes that produce the highest scores with the least number of features possible. The GA was run with the DT classifier and the highest accuracy returned was 81.53% with only a subset of the features used. This score was achieved from the best performing chromosome using just 8 of the 14 features available. This was the highest accuracy rate in all the experiments done so far. With a reduced subset, the DT was able to perform better. The following table is the decoded chromosome and it shows which features were kept by the GA.

Table 5: Best feature set found by GA

Feature	Feature Description
-	-
-	-
FSW.1	First Serve Won - Player 1
-	-
SSW.1	Second Serve Won - Player 1
SSW.2	Second Serve Won - Player 2
WSP.1	Win Serve Percentage - Player 1
WSP.2	Win Serve Percentage - Player 2
-	-
-	-
-	-
ACC.2	Number of Aces - Player 2
BWP.1	Break Points Won Percentage - Player 1
BWP.2	Break Points Won Percentage - Player 2

This also allows for the DT to produce a tree with less depth and rules that are easier to interpret . Interestingly, one can see that the First Serve Percentages are both dropped from the feature set. Also, both constructed features WSP and BWP are kept by the GA. While other raw stats such as Double Faults and First Serve Percent are dropped.

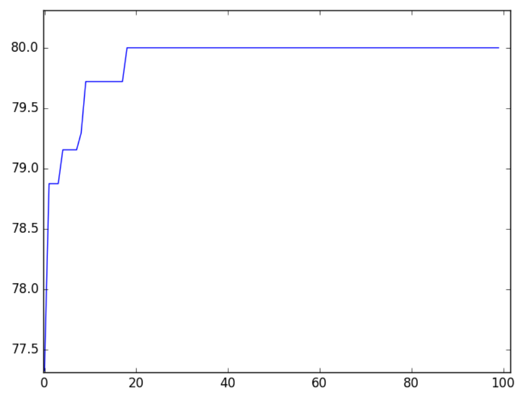

Figure 3: Best Chromosome Accuracy/Generation

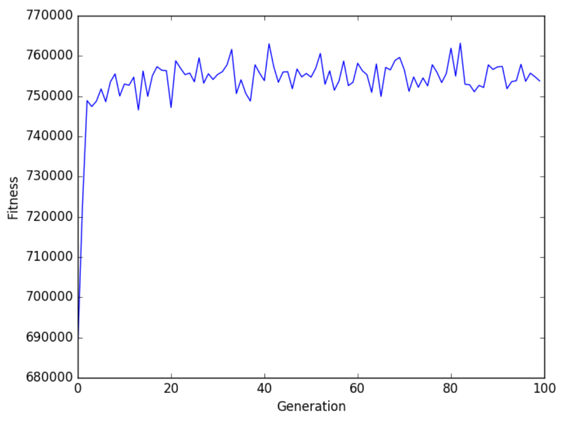

Figure 4: Average performance per Generation

0.3.4.3 Finding Best `max_depth` Of A Tree

Decision Trees can also output the result of a classification as a probability. So when classifying a Positive instance, the DT does not only output 0 or 1, but a degree of probability that that sample is a Positive or a Negative, such as for example [0.2, 0.8]. Meaning that the sample is classified as a Positive with a probability of 80%. The experiment carried out was to find the best `max_depth` by calculating the Area Under Curve(AUC) of the Receive Operating Characteristic(ROC) curves created by different `max_depth` values. The classifiers' performance with different `max_depth` between 1 and 10 was evaluated using cross-validation. From the results it was shown that `max_depth`=9 was the best performant classifier with a mean AUC of 0.87 and mean accuracy of 81.67%. The tree generated is the smallest one from all the other trees generated by other experiments. It is also the best performing classifier with the highest accuracy. The mean ROC curves with AUCs can be seen in the figure below.

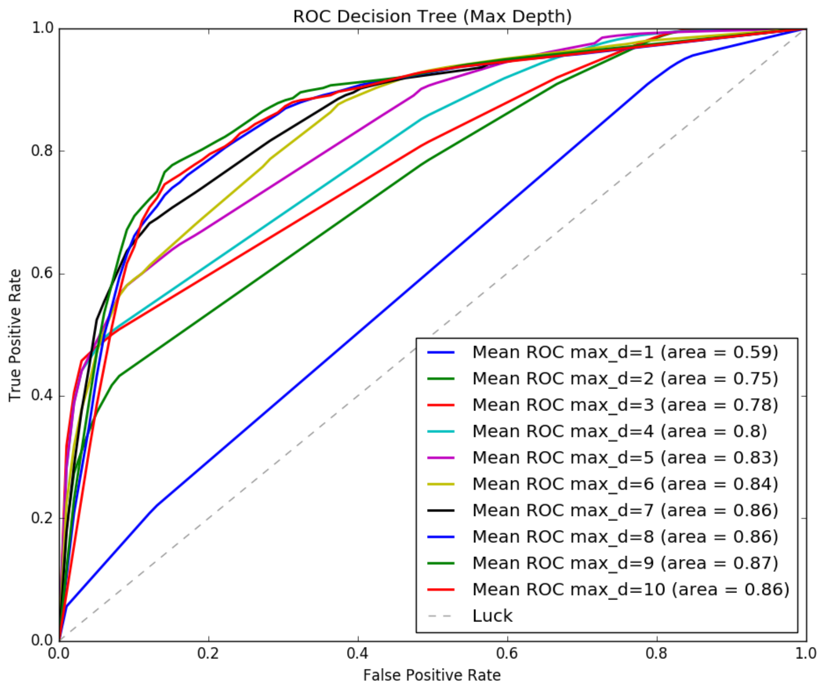

Figure 5: ROC Curves of DT with varying `max_depth` values.

0.3.5 Random Forest

Two experiments were carried out on the Random Forest model. First experiment was used to find the best number of estimators and the best `max_depth` of the trees generated. The best values for these were `max_depth` of 10 nodes and 50 estimator. These were found by cross validation for a different combination of `max_depth` and `n_estimators` with the same configuration as the previous experiments. The best model was found by getting the largest mean AUC of the mean ROC curve from the tests performed. The mean accuracy of this model on the validation set was 92.41%. This score breaks both the scores of the DT model with the reduced set of features, and with a `max_depth`=9. The figure below displays the ROC curve for the 10/50 model for each validation set and the mean ROC.

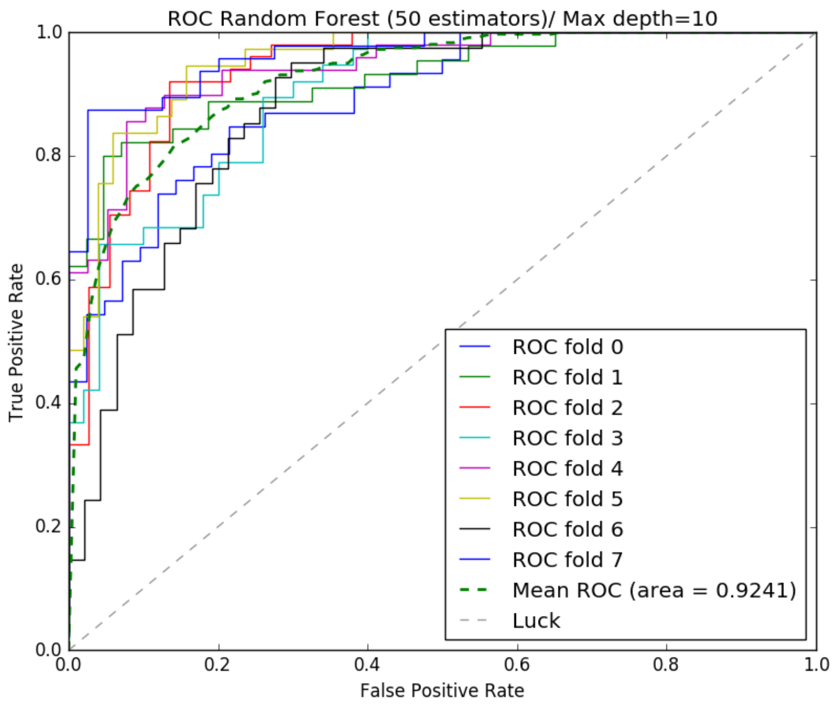

Figure 6: ROC Curves of RF with

0.3.6 Neural Network

A Feed-Forward Neural Network(NN) with Back-Propagation was also trained on the dataset. The number of units in the input and the output layer are dependent on the number of features(14) and the number of classes(2) respectively. The network was constructed with one hidden layer. One hyper-parameter of a NN is the number of hidden units that should be present in the hidden layer. Cross validation was used to determine the best number of hidden units. For each experiment the number of hidden units in the hidden layer was changed. Maximum number of epoch as set to 1000.

Table 6: Mean Accuracy achieved by Neural Networks with different hidden Units

#Hidden Neurones	Mean Accuracy(%)
7	71.88
8	71.44
9	69.89
10	70.45
11	70.60
12	73.01
13	71.59
14	7102

0.3.7 Testing and Evaluating the Best Models

The best models obtained in each experiment were then trained one last time on the whole training/validation set and then were tested on the unseen data of 202 samples from a specific tournament. This test checks if the model learnt is able to generalise enough to accurately classify samples that have never been seen before by the classifier. To measure the performance of the different models, Accuracy rate was measured. Below is the table of results achieved from the models. For the Neural Network the data was normalised before it was trained and tested. Total number of epochs was this time set to 5000.

Table 7: Summary of Results on The Test Set

Model	Mean Validation Acc.(%)	Train Error(%)	Accuracy(%)
Decision Tree	79.54	0.00	78.71
Decision Tree(GA reduced feature set)	81.54	0.00	76.24
Decision Tree(`max_depth`=9)	81.67	11.39	71.78
Random Forest(10/50)	92.41	7.59	**81.19**
Neural Network(12HU)	73.01	17.3	72.77

The Random Forest classifier was the best model to accurately classify the unseen samples. One can see that where no maximum tree depth was used for the Decision Trees, the Tree overfitted to the training example achieving 0% percent error on the training examples. All the experiments were done in Python and the libraries used were scikit-learn[2] and pybrain[3] .

0.4 Conclusions

In this assignment, the problem of predicting the winner of a tennis match by learning models from a data set was explored. The classifiers chosen were the Decision Tree, Random Forest and Neural Network. Several other techniques were used to reduce the dimensionality of the feature space and reduce the feature space by the use of a Genetic Algorithm. Cross validation was used to find the best hyper-parameters of the models and evaluation metrics such as accuracy and AUC of ROC curves were used to measure the quality of the output of a model. From the experiments it was shown the DT with PCA performed poorly and it mat be the case that the relationships between attributes are non-linear. The Decision Tree with reduced features performed very well during the validation stage however, it did not perform well on the test set. The drop of performance of the Decision Tree with maximum depth of 9 on the test error was not expected. The best performance throughout all the experiments was that of the Random Forest. It was also the model that achieved the best accuracy on the test set. The NN never performed that well and it may be the case that the architecture of the network might need to further investigated to achieve higher scores. Such as one more hidden layer might have been introduced. The performance of the reduced feature set had also dropped and this might be the case because of no

[2]`http://scikit-learn.org/stable/index.html`
[3]`http://pybrain.org/`

maximum was set for tree depth was specified. As can be seen the training error rate for the reduced tree was 0% and so over-fitting might have occurred.

REFERENCES

[1] - Morteza Mashayekhi and Robin Gras, ?Rule Extraction from Random Forest: The RF+HC Methods,? in U28th Canadian Conference on Artificial Intelligence, Canadian AI 2015, Halifax, Nova Scotia, Canada, June 2-5, 2015, Proceedings

[2] - Michal Sipko, Machine Learning for the Prediction of Professional Tennis Matches, Imperial College London ,June 15, 2015