Part IV: Final Report
Roger Federer Tennis Performance Report
Muhammad Husain (2140818)
Ashna Patel (1940541)

Introduction

In the evolving landscape of sports analytics, tennis is unique as it is marked by precision and intensity. Our group was inspired by the recently concluded US Open and wanted to extend the topic of tennis to our project. Two players, Serena Williams and Roger Federer, retired and did not participate in the 2023 US Open. We are big fans of these players but that did not stop us from enjoying this year's tournament as there were many great players. Fuelled by the excitement of watching the top world's players, we still wanted to reflect back on a previous player and see all their achievements. Our group focused on Roger Federer and decided to further explore his past winnings to have an insight to the world of tennis match prediction.

In the world of tennis that deals with analytics and match predictions, Roger Federer is the main focus of our project. Roger Federer is often called one of the greatest tennis players of all time and has left his mark in the sport of tennis. He is known for playing in an elegant and versatile playing style that is rare to see in players. Even though Federer retired, his career is not to be forgotten as he has a record breaking 20 Grand Slam single titles. His legacy lives on by his foundation that supports education and health in South Africa and Switzerland, where he originates.

Our chosen dataset provides information spanning nearly one decade of the insights of tennis matches. This project aims to use a dataset containing information about all tennis matches played by Roger Federer in the ATP World Tour from January 2000 to December 2010. The primary objective of this project is to create an accurate prediction model for determining the odds of a Roger Federer winning a tennis match. To achieve this goal, the dataset provides various variables like location of the match, the specific tournament, the type of court (eg., grass, clay, hard), the surface condition, the series of the tournament and other possible factors to help us predict the outcome of which player would win. Our response variable for this project will be win or lose as it will represent if Roger Federer will win or lose a tennis match before the match starts.

In summary, the project involves analyzing historical tennis match data about Roger Federer with a goal of building a predictive model that considers various and multiple variables. This model will then be used to estimate the likelihood of a Federer winning a tennis match and providing insights into the numerous variables that impact this statistic.

Methodology

In our project, our dataset is categorized into supervising learning. The two methods that we will use for dealing with supervising learning are K Nearest Neighbors (KNN) and Support Vector Machines (SVM).

- **K Nearest Neighbors (KNN)**
  - In this approach we will classify and decide on data points based on their closest neighbor. If most of its neighbors are in a certain category, then we will say our point is also in that corresponding category.
  - An advantage of this method is that KNN will handle the dataset no matter the data distribution or pattern. However, a disadvantage of KNN is that it is more sensitive to 'noisy' and outlier data.
  - The distance between the points is calculated using euclidean distance in the R function KNN. The equation for euclidean distance is as follows $\sqrt{\sum_{i}^{n}(x_i - y_i)^2}$
  - Leave One Out Cross Validation is a method for validation that is used in the KNN section of our assignment. The way that it works is we will train our model on all the data besides 1 observation which will be our test set. We compare our model to the 1 test set and then repeat the process until all the observations have been a test set.
- **Support Margin Classifier (SVM)**
  - Instead of looking at neighbors link KNN, SVM draws a line on the graph, called a hyperplane, to separate the data into different categories. It is like drawing a boundary between two teams on the tennis court. This hyperplane will help us calculate the 'margin cost' which is like a budget for how much space the margins can be. The data points are then labeled based on which side of the hyperplane they are on.
  - An advantage of this is that SVM is not as sensitive to noisy or outlier data. However, a disadvantage of SVM is that the data needs to be spread out so the hyperplane can neatly divide the data points.
  - We will be using 3 different kernels: linear, polynomial, radials, and the equations for them are as follow

    linear: $K(x_i, x_j) = \sum_{k=1}^{p} x_{ik} x_{jk}$

    polynomial: $\beta_0 + \sum_{i \in S} \alpha i K(x, x_i)$

    radial: $K(x_i + x_j) = exp(-\gamma \sum_{k=1}^{p} (x_{ik} - x_{jk})^2)$

In summary, K Nearest Neighbors (KNN) is making decisions based on the neighbors on specific data points which is beneficial when there is uneven data distribution. Support Margin Classifier (SVM) is drawing a hyperplane (line) to separate the data points and categorizing them. This method is beneficial when the data is more organized.

Data Analysis

For our dataset, we decided against using the atp, date, b365w, b365l, and elo_predict, for our models due to the fact that b365w, b365l, and elo_predict were there initially for calculating betting odds and do not fit our goal of predicting if Roger Federer will win a match. Date and ATP were also removed because they do not really contribute to our goal of predicting if Roger Federer will win because the date is just the date while ATP is just the tournament number. Keeping these values will just create noise for our model which can create more errors and therefore we decided to remove them. Most of the variables we have in our dataset are categorical and the quantitative values we do have are: elo_winner, elo_loser. Due to this, we decided that scaling the data will not be necessary due to the fact that the elo variables should be scaled to each other already.

K Nearest Neighbors (Muhammad)

At the beginning, I separated the data into 80% training and 20% testing making sure my results were replicable by using RNGkind(sample.kind = "Rounding") and set.seed(1). Our dataset has 508 observations after removing all the rows that had empty values, so the training set will have 406 observations with the remaining 102 observations going into the test set. The values of K that I tested were: 1, 3, 5, 10, 15, 20, 25, 30 for both normal KNN and KNN with cross validation.
The following code is what I used for KNN and KNN with cross validation:

```
for (K in c(1,3,5,10,15,20,25,30)){
  set.seed(1)
  knn.pred <- knn(train = trainSet[-14],
                  test = testSet[-14],
                  cl = trainSet$Outcome,
                  k = K)
  print(mean(knn.pred != testSet$Outcome))
  print(table(Guess = knn.pred, Correct = testSet$Outcome))

}

for (K in c(1,3,5,10,15,20, 25, 30)){
  set.seed(1)
  knn.pred.cross <- knn.cv(train = dataset[-14],
                  cl = dataset$Outcome,
                  k = K)
  print(mean(knn.pred.cross != dataset$Outcome))
  print(table(Guess = knn.pred.cross,Correct = dataset$Outcome))

}
```

Out of the 8 values for K I tested, the best value for K in normal KNN was 25 with an error rate of 11.5 % and a false positive rate of 36%. (note that 1 means that Roger had lost the game while 2 means that Roger had won the game, so in this case 1 will be negative and 2 will be positive)

```
[1] 0.1153846
     Correct
Guess  1  2
    1 21  0
    2 12 71
```

For KNN with Leave One Out Cross Validation, the best K value was 5 with an error rate of 9.4% and a false positive rate of 20.5% and a false negative rate of 5.7%

```
[1] 0.09448819
     Correct
Guess   1   2
    1 101  22
    2  26 359
```

Due to the fact that Leave One Out Cross Validation better validates the data, it is not a surprise that there is a 2% decrease in error rate between the best K for KNN and the best K for KNN with LOOCV. A reason to explain the the higher false positive and false negative rates for KNN with LOOCV is due to the fact that KNN with LOOCV worked with the whole of the dataset and compared its results to the whole of the dataset while normal KNN only compared itself to 20% of the dataset, that being our testing set

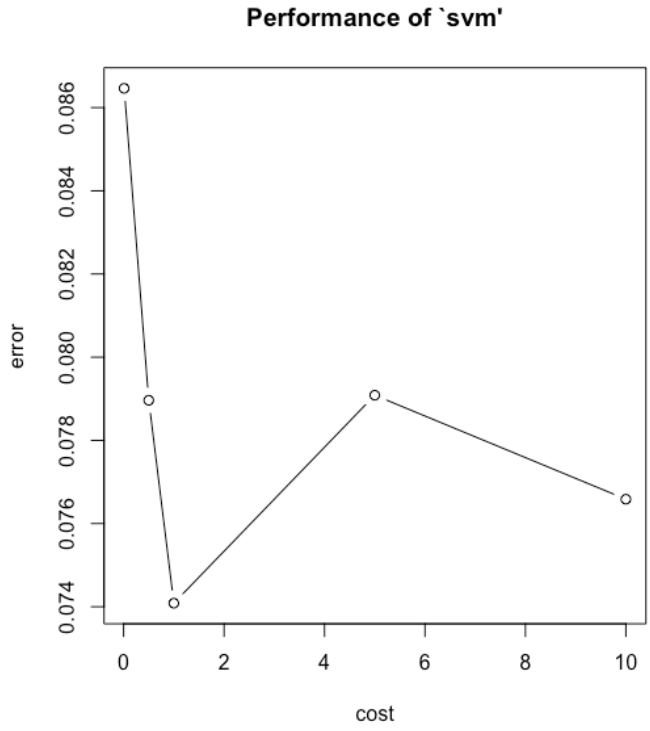
Support Vector Machine (Ashna)

The standard procedure to conduct SVM was used by randomly dividing the data for 80% training and 20% testing. I performed tuning to optimize SVM and used 3 kernels - Linear, Polynomial and Radial. Each kernel function defines parameters that significantly influence the model's decision boundary.

Linear Kernel

Linear Kernel is the simplest form of SVM as it assumes the variables are linear. The cost variables used are 0.01, 0.5, 1, 5 and 10. Using R, the algorithm produced the output below.

```
> set.seed(1)
> tune.out.linear=tune(svm,
+                Outcome~.,
+                data=trainFrame,
+                kernel="linear",
+                ranges=list(cost=c(0.01,0.5,1,5,10)))
>
> summary(tune.out.linear)

Parameter tuning of 'svm':

- sampling method: 10-fold cross validation

- best parameters:
 cost
    1

- best performance: 0.07408537

- Detailed performance results:
   cost      error dispersion
1  0.01 0.08646341 0.03532003
2  0.50 0.07896341 0.03980446
3  1.00 0.07408537 0.04372541
4  5.00 0.07908537 0.04189759
5 10.00 0.07658537 0.04736967
```



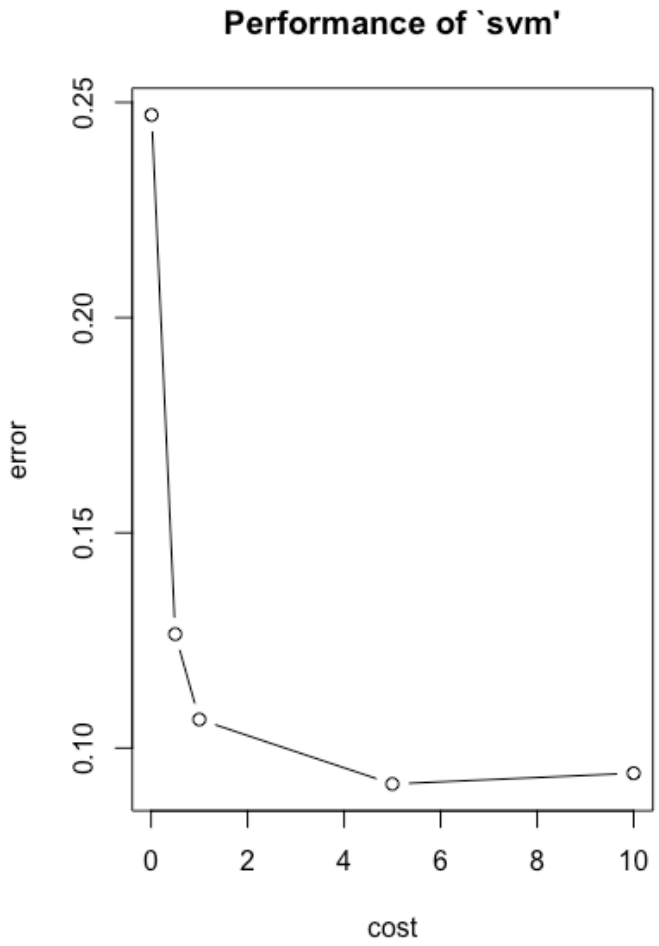

```
> ypred <- predict(tune.out.linear$best.model,testFrame)
> table(predict=ypred, truth=testFrame$Outcome)
       truth
predict  1  2
      1 17  4
      2 10 72
```

The best cost value is cost = 1 for our linear kernel with an error rate of 7.4% with a false positive rate of 37% and a false negative rate of 5.26% .

Polynomial Kernel

The polynomial kernel transforms the variables in a higher dimension by using polynomial functions for a complex decision boundary like curves. The cost variables used are 0.01, 0.5, 1, 5 and 10. Using R, the algorithm produced the output below.

```
> set.seed(1)
> tune.out.poly=tune(svm,
+                Outcome~.,
+                data=trainFrame,
+                kernel="polynomial",
+                ranges=list(cost=c(0.01,0.5,1,5,10)))
>
> summary(tune.out.poly)

Parameter tuning of ‘svm’:

- sampling method: 10-fold cross validation

- best parameters:
 cost
    5

- best performance: 0.09164634

- Detailed performance results:
   cost      error dispersion
1  0.01 0.24707317 0.05273895
2  0.50 0.12646341 0.06645515
3  1.00 0.10664634 0.05560230
4  5.00 0.09164634 0.05027607
5 10.00 0.09414634 0.06244979
```



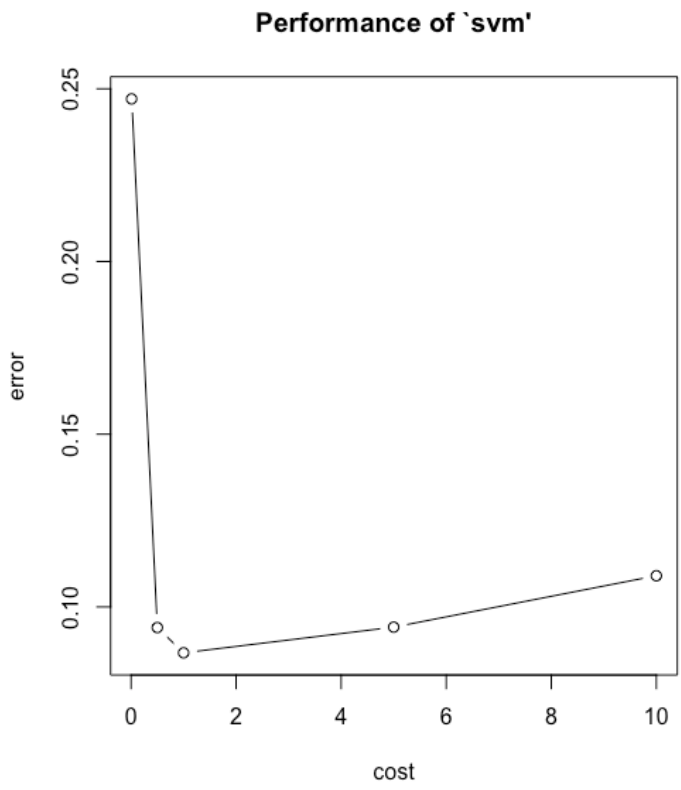

```
> ypred <- predict(tune.out.poly$best.model,testFrame)
> table(predict=ypred, truth=testFrame$Outcome)
       truth
predict  1  2
      1 14  4
      2 13 72
```

The best cost value is cost = 5 for our polynomial kernel with an error rate of 9.16% with a false positive rate of 48% and a false negative rate of 5.26% .

Radial Kernel

The radial kernel is the last one used for our project. The variables are mapped into an infinite dimensional space for creating a circular decision boundary.

```
> set.seed(1)
> tune.out.radial=tune(svm,
+                Outcome~.,
+                data=trainFrame,
+                kernel="radial",
+                ranges=list(cost=c(0.01,0.5,1,5,10)))
>
> summary(tune.out.radial)

Parameter tuning of 'svm':

- sampling method: 10-fold cross validation

- best parameters:
 cost
    1

- best performance: 0.08670732

- Detailed performance results:
   cost      error dispersion
1  0.01 0.24707317 0.05273895
2  0.50 0.09402439 0.05366223
3  1.00 0.08670732 0.05447594
4  5.00 0.09414634 0.06860423
5 10.00 0.10902439 0.06904038
```




```
> ypred <- predict(tune.out.radial$best.model,testFrame)
> table(predict=ypred, truth=testFrame$Outcome)
       truth
predict  1  2
      1 15  2
      2 12 74
```

The best cost value is cost = 1 for our polynomial kernel with an error rate of 8.67% with a false positive rate of 44% and a false negative rate of 2.63% .

From running all the algorithms for linear, polynomial and radial, we can infer that the lowest error rate is 7.4% with the cost being 1 which was produced by the linear kernel.

Conclusion

The goal of this project is to see which model will best help us predict if Federer will win a tennis match. Since our dataset uses supervised learning, we used 2 methods which are K Nearest Neighbor (KNN) and Support Vector Margin (SVM). The model that came out to have the lowest error rate is the SVM model with a linear kernel that produced an error rate of 7.4%. This output was very accurate as a result it helped us succeed in our goal.

Problems with the project that we were running into was the data. One problem was, we had to edit the data to only include matches that Federer played and remove the rest. Another problem we had was that our models did not work unless we made our categorical variables into factors. A possible procedure that could be improved was the analysis by making it so there was a variable that was the difference in elo ranking between Federer and his opponent. A variable like this would have made our analysis better and it could have been made with the information we were given.

References

The website where we found our dataset:
https://www.kaggle.com/code/edouardthomas/beat-the-bookmakers-with-machine-learning-tennis