Math 4323 – Final Project Report

Group members:

**3 Members for KNN:**  
Andreck Soto 1619873   
Johnny Chien 1897427   
Zainab Dadani 1304129

**3 Members for SVM:**  
Tola Ouk 0821843  
Mario Villarreal 1483988

May Chu 1609654

1. **Introduction**

The data set is called Admission\_Predict and the purpose of this data set is to predict the probability of an individual being admitted to a master’s program given the predictors GRE score, TOEFL score, University rating, Statement of Purpose and Letter of Recommendation Strength, Undergraduate GPA, Research Experience. The data set also has Serial Number, but this is strictly to identify the applicant.

The predictors are broken down such as:

1. GRE Scores which is out of 340
2. TOEFL Scores which is out of 120
3. University Rating which is out of 5
4. Statement of Purpose
5. Letter of Recommendation Strength which is out of 5
6. Undergraduate GPA which is out of 10
7. Research Experience which is either 0 or 1

There is a total of 9 columns in this data set and 401 rows, with 400 schools taken into consideration. The file is saved as Admission\_Predict.csv and is 12.6 KB of size. Overall, the purpose of this data set is to give students an idea of the possibility attending the graduate school of their choice.

With this data we are looking to accurately predict variable “admitmedian01” (where admitmedian01 is a yes or no of whether the student is likely to be admitted or not which is based on whether the Chance of admit is over the median or not for Chance of Admit) initially given predictors GRE score, TOEFL score, University rating, SOP, LOR, CGPA, and Research.

1. **Methodology**
2. K Nearest Neighbors

KNN is a supervised machine learning algorithm. KNN can be used both for classification and regression. In this case we are going to be using it for classification. KNN makes predictions based off the distance of the observations around it. KNN is a supervised learning method and has variety of distance types to choose from, including Euclidean distance and Manhattan. By doing it in R Studio we do not have to worry about the arithmetic of the distance. Advantages of KNN is that it is intuitive simple, high accuracy, and has no assumptions. For KNN, we are going to try to find a good k value to avoid underfitting and overfitting. For KNN, we are going to see the relationship of the data based on how close the observations are to each other in order to see the performance of our KNN.

Possible Formulas:

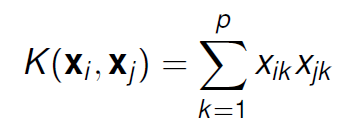
* Euclidean Distance:
* Manhattan Distance:

1. Supported Vector Machines

SVM is a supervised machine learning algorithm. SVM can be used both for classification and regression. There are different types of SVM and a large aspect of SVM revolves are dimension reduction and making predictions based off these dimensions. Advantages of SVM are that it allows us a general framework of enlarging the feature space for a support vector classifier in a way that leads to efficient computations. We will be using three types of SVM functions: linear, polynomial, and radial. For each SVM respectively we are looking for the best the best cost for linear, the best cost and degree for polynomial, and the best cost and gamma for radial. Afterwards we will see relationship of the data and find the best test error for kernels linear, polynomial, and radial in order to see the performance of SVM.

Kernels

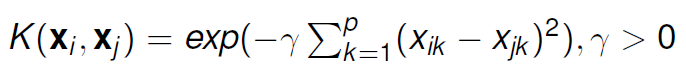
* Linear:



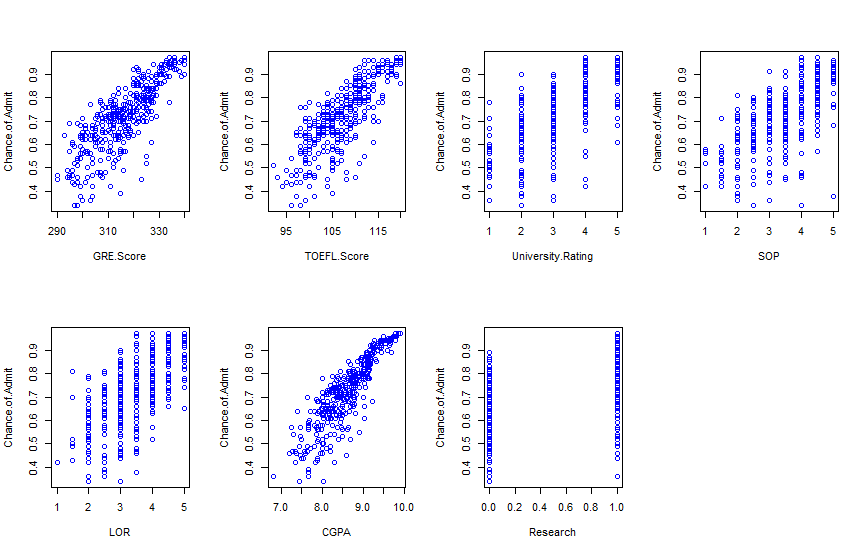
* Polynomial:



* Radial:



1. **Data Analysis:**
2. **Data pre-processing**



**R – code in Appendix**

Judging from these plots, we believe that Research column provides relatively insignificant effect on the admission, and therefore will be omitted from the data.

New column admitmedian01 will be added to the data. It will be a binary (0-1) column. This column is based on the Chance.of.Admit column. The observations with value higher than the median will be assigned 1 (admitted) and observations with value below the median will be assigned 0 (not admitted).

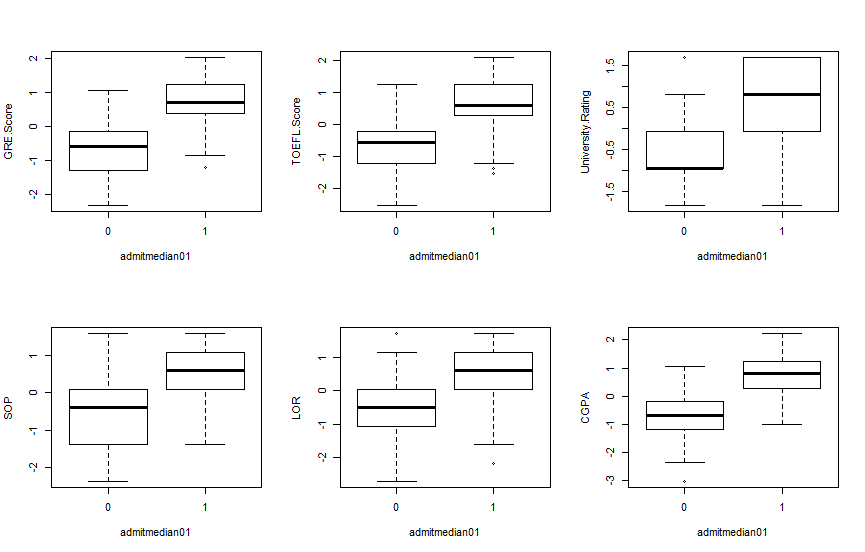
**R-code in Appendix**

In total 3 columns will be deleted:

* Serial No.: row number, not relevant in our data analysis.
* Research: no effect on admission rate.
* Chance.of.Admit: to be replaced with admitmedian01.

**R-code in Appendix**

New plots of admitmedian01 vs other variables:



**R-code can be found in appendix.**

**Scaling:**

Scaling the data is necessary because all the predictors are on different ranges. For example, GRE.score varies from 290 to 340, TOEFL.score varies from 92 to 120, while University Rating, SOP, LOR vary from 1 to 5.

**R-code for scaling is in appendix.**

**RNG Rounding:**

We also used

RNGkind(sample.kind = "Rounding")

at the beginning of the program.

1. **KNN and SVM**

**KNN:**

train() function is used to find optimal k for 10 fold cross validation knn

> model1

k-Nearest Neighbors

320 samples

6 predictor

2 classes: '0', '1'

No pre-processing

Resampling: Cross-Validated (10 fold)

Summary of sample sizes: 288, 288, 289, 287, 287, 289, ...

Resampling results across tuning parameters:

Accuracy was used to select the optimal model using the largest value.

The final value used for the model was k = 16.

> test.err <- mean(knn.10cross.predictions != test\_df$admitmedian01)

> test.err

[1] 0.1125

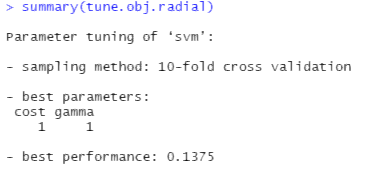
10-fold cross validation gives the optimal k of 16 for knn.

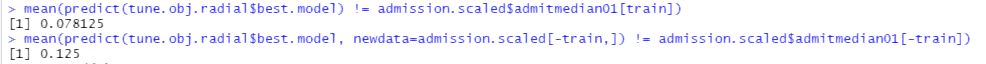
Using these best parameters on the set give us the following:

Testing error = 11.25%

**Radial SVM:**

Tune function is used to determine the optimal parameters, cost and gamma.





10-fold cross validation gives the best parameters of cost = 1 and gamma = 1.

Using these best parameters on the set give us the following:

Testing error = 12.5%

**Linear SVM:**

Tune function is used to determine the optimal parameter, cost.

> summary(tune.lin)

Parameter tuning of ‘svm’:

- sampling method: 10-fold cross validation

- best parameters:

cost

1

- best performance: 0.15

> mean(predict(tune.lin$best.model, newdata=admission.scaled[-train,]) != admission.scaled$admitmedian01[-train])

[1] 0.125

10-fold cross validation gives the best parameters of cost = 1

Using these best parameters on the set give us the following:

Testing error = 12.5%

**Polynomial SVM:**

Tune function is used to determine the optimal parameters, cost and degree.

> summary(tune.polyn)

Parameter tuning of ‘svm’:

- sampling method: 10-fold cross validation

- best parameters:

cost degree

100 3

- best performance: 0.15

> mean(predict(tune.polyn$best.model, newdata=admission.scaled[-train,]) != admission.scaled$admitmedian01[-train])

[1] 0.1

10-fold cross validation gives the best parameters of cost = 100 and degree = 3.

Using these best parameters on the set give us the following:

Testing error = 10%

**Detailed result and R-Code in appendix**



Test error for knn 10 fold cross validation is 11.25%. 10 fold cross validation for SVM radial and linear have the same test error of 12.5 %. 10 fold cross validation for SVM polynomial has the lowest test error of 10 %.

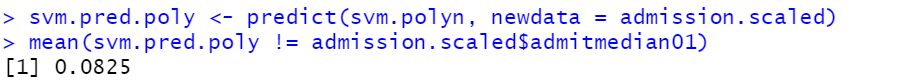
1. **Fit best model from part c) on the full data set.**

SVM function with a polynomial kernel was used on the full data set with cost = 100 and degree = 3 (from Part C)

Summary(svm.polyn) shows the following

Graphical user interface, text, application, email

Description automatically generated

****

Testing error for SVM polynomial for the full data set is 8.25%.

Below is one plot of our fitted boundary with respect to a pair of predictors (GRE.Score and CGPA) Chart, scatter chart

Description automatically generated

**Note: The rest of the plots will be in Appendix**



We achieved a respectable predictive performance since all of our test errors for all the methods that we used is low, close to 10 % for all of them.

1. **Conclusion**

The answer to the question in the introduction is yes since we can accurately predict admitmedian01 based on the predictors GRE score, TOEFL score, University rating, SOP, LOR, and CGPA. SVM polynomial was the best at accurately predicting admitmedian01 based on the predictors mentioned above. We had to clean the data to figure out which predictors was most useful for predicting so we had to exclude research because it wasn’t useful. We also had to figure out whether we had to scale the data so our predictions would be more accurate. We could try different splits for the training/testing sample which might improve our prediction rate for the methods.

1. **Reference**

**Mohan S Acharya, Asfia Armaan, Aneeta S Antony : A Comparison of Regression Models for Prediction of Graduate Admissions, IEEE International Conference on Computational Intelligence in Data Science 2019**

[**https://www.kaggle.com/mohansacharya/graduate-admissions**](https://www.kaggle.com/mohansacharya/graduate-admissions)

1. **Appendix:**

**Plots – Chance.of.Admit vs other variables**

attach(admission)

par(mfrow=c(2,4))

plot(Chance.of.Admit ~ GRE.Score)

plot(Chance.of.Admit ~ TOEFL.Score)

plot(Chance.of.Admit ~ University.Rating)

plot(Chance.of.Admit ~ SOP)

plot(Chance.of.Admit ~ LOR)

plot(Chance.of.Admit ~ CGPA)

plot(Chance.of.Admit ~ Research)

**Add new column admitmedian**

admission$admitmedian01 <- ifelse(admission$Chance.of.Admit > median(admission$Chance.of.Admit), 1, 0)

class(admission$admitmedian01)

admission$admitmedian01 <- as.factor(admission$admitmedian01)

class(admission$admitmedian01)

**Delete 3 columns:**

admission$Chance.of.Admit <- NULL

admission$Serial.No. <- NULL

admission$Research <- NULL

**Plots of admitmedian01 vs other variables:**

attach(admission)

par(mfrow=c(2,3))

plot(GRE.Score ~ admitmedian01)

plot(TOEFL.Score ~ admitmedian01)

plot(University.Rating ~ admitmedian01)

plot(SOP ~ admitmedian01)

plot(LOR ~ admitmedian01)

plot(CGPA ~ admitmedian01)

**Scaling:**

admission.scaled = data.frame(scale(admission[, 1:7]))

admission.scaled <- cbind(admission.scaled, admitmedian01 = admission$admitmedian01)

**Training/Test split code**

**Radial SVM:**

set.seed(1)

train=sample(400,320)

set.seed(1)

tune.obj.radial <- tune(svm,

admitmedian01 ~ .,

data = admission.scaled[train,],

kernel="radial",

ranges = list(cost=c(0.001,0.01,0.1,1,5,10,100),gamma=c(0.1, 0.2, 0.3, 0.4,0.5,0.7, 1,2,3,4)))

tune.obj.radial

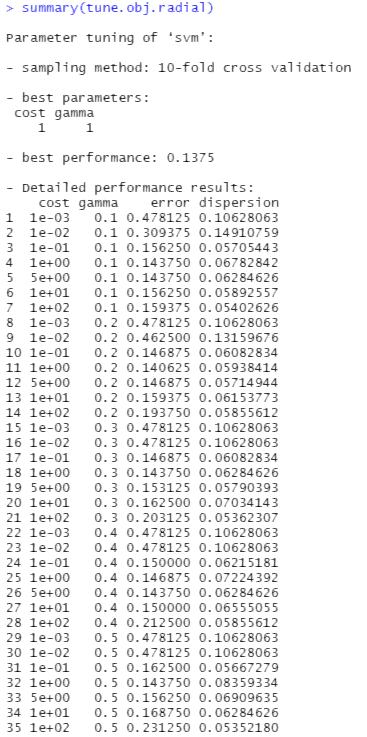
summary(tune.obj.radial)

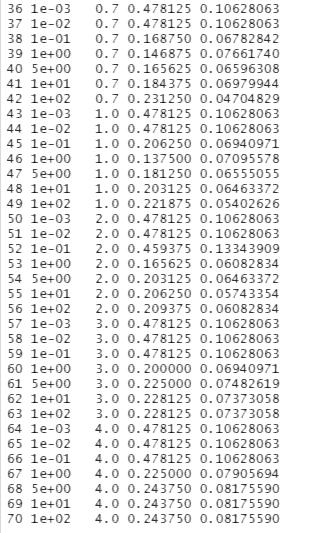
# for the training error

mean(predict(tune.obj.radial$best.model) != admission.scaled$admitmedian01[train])

# for the test error

mean(predict(tune.obj.radial$best.model, newdata=admission.scaled[-train,]) != admission.scaled$admitmedian01[-train])





**# for the training error**

mean(predict(tune.polyn$best.model) != admission.scaled$admitmedian01[train])

**# for the test error**

mean(predict(tune.polyn$best.model, newdata=admission.scaled[-train,]) != admission.scaled$admitmedian01[-train])

**KNN:**

#10 fold cross validation for knn

library(caret)

library(e1071)

set.seed(1)

index <- sample(400, 320)

train\_df <- admission.scaled[index, ]

test\_df <- admission.scaled[-index, ]

ctrlspecs <- trainControl(method="cv", number=10)

set.seed(1)

model1 <- train(admitmedian01~GRE.Score+TOEFL.Score+University.Rating+SOP + LOR+CGPA,

data = train\_df,

method = "knn", tuneGrid = expand.grid(k = 1:30),

trControl = ctrlspecs,

metric = "Accuracy")

model1

knn.10cross.predictions <- predict(model1, newdata = test\_df)

**# for the test error**

test.err <- mean(knn.10cross.predictions != test\_df$admitmedian01)

test.err

> model1

k-Nearest Neighbors

320 samples

6 predictor

2 classes: '0', '1'

No pre-processing

Resampling: Cross-Validated (10 fold)

Summary of sample sizes: 288, 288, 289, 287, 287, 289, ...

Resampling results across tuning parameters:

k Accuracy Kappa

1 0.7909244 0.5811940

2 0.7718719 0.5440653

3 0.8191502 0.6373737

4 0.8129002 0.6255316

5 0.8220858 0.6436741

6 0.8250214 0.6499493

7 0.8373320 0.6750608

8 0.8373259 0.6747905

9 0.8496365 0.7000453

10 0.8527676 0.7065177

11 0.8528623 0.7061969

12 0.8434873 0.6878295

13 0.8590237 0.7188759

14 0.8528684 0.7069096

15 0.8593200 0.7191837

16 0.8653745 0.7315120

17 0.8594147 0.7195340

18 0.8530578 0.7065620

19 0.8626344 0.7257403

20 0.8627352 0.7259039

21 0.8626344 0.7254982

22 0.8626344 0.7256764

23 0.8563844 0.7132154

24 0.8594147 0.7193376

25 0.8595155 0.7195404

26 0.8500336 0.7002433

27 0.8535496 0.7073068

28 0.8596102 0.7196192

29 0.8596102 0.7194410

30 0.8564852 0.7132753

Accuracy was used to select the optimal model using the largest value.

The final value used for the model was k = 16.

**SVM polynomial:**

set.seed(1)

train=sample(400,320)

set.seed(1)

tune.polyn <- tune(svm,

admitmedian01~ .,

data = admission.scaled[train,],

kernel="polynomial",

ranges = list(cost = c(0.001,0.01, 0.1, 1, 5, 10, 100),

degree = c(2,3,4)))

summary(tune.polyn)

print(tune.polyn$best.parameters)#optimal cost value

print(tune.polyn$best.performance)#optimal model's cross-validation error

**# for the test error**

mean(predict(tune.polyn$best.model, newdata=admission.scaled[-train,]) != admission.scaled$admitmedian01[-train])

> summary(tune.polyn)

Parameter tuning of ‘svm’:

- sampling method: 10-fold cross validation

- best parameters:

cost degree

100 3

- best performance: 0.15

- Detailed performance results:

cost degree error dispersion

1 1e-03 2 0.478125 0.10628063

2 1e-02 2 0.478125 0.10628063

3 1e-01 2 0.481250 0.12430362

4 1e+00 2 0.446875 0.10729672

5 5e+00 2 0.443750 0.11102427

6 1e+01 2 0.443750 0.08814248

7 1e+02 2 0.440625 0.06979944

8 1e-03 3 0.368750 0.15224022

9 1e-02 3 0.290625 0.12846753

10 1e-01 3 0.231250 0.09793883

11 1e+00 3 0.178125 0.08468949

12 5e+00 3 0.153125 0.07284222

13 1e+01 3 0.156250 0.06750772

14 1e+02 3 0.150000 0.06555055

15 1e-03 4 0.478125 0.10628063

16 1e-02 4 0.500000 0.12147816

17 1e-01 4 0.487500 0.12689535

18 1e+00 4 0.418750 0.12943520

19 5e+00 4 0.428125 0.11604185

20 1e+01 4 0.434375 0.09821541

21 1e+02 4 0.456250 0.07822910

**SVM linear:**

set.seed(1)

train <- sample(400,320)

set.seed(1)

tune.lin <- tune(svm,

admitmedian01~ .,

data = admission.scaled[train,],

kernel="linear",

ranges = list(cost = c(0.001,0.01, 0.1, 1, 5, 10, 100)))

tune.lin

summary(tune.lin)

print(tune.lin$best.parameters)

print(tune.lin$best.performance)

**# for the test error**

mean(predict(tune.lin$best.model, newdata=admission.scaled[-train,]) != admission.scaled$admitmedian01[-train])

> summary(tune.lin)

Parameter tuning of ‘svm’:

- sampling method: 10-fold cross validation

- best parameters:

cost

1

- best performance: 0.15

- Detailed performance results:

cost error dispersion

1 1e-03 0.156250 0.05511982

2 1e-02 0.159375 0.06327643

3 1e-01 0.153125 0.06327643

4 1e+00 0.150000 0.07186745

5 5e+00 0.156250 0.06588078

6 1e+01 0.156250 0.06588078

7 1e+02 0.156250 0.06588078

**Code and Plots for Section 3, Part D**

Screenshot of Code:

Text

Description automatically generated

Plots:

Chart, scatter chart

Description automatically generated

Chart, scatter chart

Description automatically generated

Chart

Description automatically generatedChart, histogram

Description automatically generatedChart, histogram

Description automatically generatedChart, scatter chart

Description automatically generatedChart

Description automatically generatedChart, scatter chart

Description automatically generatedChart, scatter chart

Description automatically generated