Optimizing Recruitment Strategies For Teach For America: A Predictive Analytics Study

MAX 522

Predictive Analytics

Dr. Dinakar Jayarajan

Final Project

Developed By:

Aditya Raj Sharma

Part A - Analysis of the Case and Approach

Background

Teach For America (TFA), a renowned organization focused on educational equity, has encountered challenges in maintaining its application rates for recruiting teachers. This situation poses a significant threat to its mission of empowering low-income schools across the United States with quality education. Our group's analysis is aimed at leveraging predictive analytics to aid TFA in enhancing its recruitment and retention strategies.

Variables in the Dataset

The dataset we are analyzing encompasses a variety of variables that provide insights into the applicants' profiles and their engagement with TFA's recruitment process. The dataset has 37,134 rows of data. These include:

- *Personal Identifier (personid):* A unique numerical identifier for each applicant.
- *Application Year (appyear):* The academic year of application.
- *GPA:* Reflecting the applicant's academic performance.
- *STEM Background (stem)*: Binary indicator of a math, science, or engineering major or minor.
- *School Selectivity (schoolsel, schoolsel_chr)*: Categorical variables representing the selectivity of the applicant's undergraduate institution.
- *Major and Minor Fields (major1, major2, minor)*: Detailed information on the applicant's undergraduate major(s) and minor, if any.
- *Major Groupings (major1group, major2group, minorgroup)*: Categorization of majors and minors into broader groups.
- *Undergraduate University (undergrad_uni):* The institution where the applicant completed their undergraduate studies.
- *Essay Metrics (essay1length, essay2length, essay3length, essayuniquewords, essayssentiment):* Quantitative measures of the applicant's essays, including length, unique word count, and sentiment analysis.
- *Engagement and Timeline Metrics (signupdate, starteddate, appdeadline, submitteddate):* Data points reflecting the applicant's engagement timeline with the application process.
- *Event Attendance (attendedevent):* Indicator of whether the applicant attended a TFA event.
- *Completion of Admission Process (completedadm):* Binary variable indicating whether the applicant completed all steps of the admissions process.

Problem Statement

Our primary objective is to identify a predictive model that accurately forecasts whether a candidate will accept a position with TFA by best predicting the Completion of Admission Process (completedadm). This prediction is crucial for TFA to efficiently allocate resources, enhance recruitment efficiency, and ultimately have a more significant impact in underserved educational settings.

Our Approach to Model Testing
Our methodology involves a comparative analysis of various predictive models to ascertain the one that best suits TFA's needs. The models we are evaluating include:

- kNN (K-Nearest Neighbors): A simple yet effective model, particularly adept at handling smaller datasets with a clear margin of separation.
- Naive Bayes: Renowned for its efficiency in classification tasks, especially with text data or when the assumption of independence among features holds.
- Decision Trees: Offers easy interpretability, making it a practical choice for decision-making processes.
- ANN (Artificial Neural Networks): A more complex model that excels in capturing non-linear relationships in large datasets.
- SVM (Support Vector Machines): Known for its robustness, particularly in high-dimensional spaces.

Each model will be rigorously tested and evaluated based on performance metrics relevant to TFA's context.

Part B - Analysis of Predictive Models

(1.1) kNN

Approach
Our team implemented the k-Nearest Neighbors (kNN) algorithm to predict the completion of the admission process at Teach For America. Utilizing an 80-20 split for training and test data, we experimented with k values (1, 3, 5, 7, 11) to assess how the model's accuracy changes with varying degrees of neighbor consideration.

Analysis
Performance metrics for each k value were calculated as follows:

k Value	True Negatives	True Positives	False Positives	False Negatives	Accuracy
1	479	3939	593	361	82.24%
3	353	4135	719	165	83.54%
5	280	4218	792	82	83.73%
7	243	4252	829	48	83.67%
11	185	4283	887	17	83.17%

Fine-tuning

- Sensitivity to k: Increasing the value of k generally resulted in a decrease in true negatives and an increase in true positives. This suggests that as the model considers more neighbors, it becomes more likely to predict an applicant as completing the admission process.
- Class Imbalance: The model's tendency to predict a higher number of admissions ('1') indicates a class imbalance.

Result

- Model Performance: The model with k = 5 showed a high number of true positives and a fairly low number of false negatives, indicating it's better at correctly identifying applicants likely to complete the admission process.
- Potential Bias: A notable issue across all k values was the high number of false positives, suggesting a bias towards predicting completion of the admission process. This could lead to inefficiencies in accurately identifying applicants who might not complete the process.
- Practical Implications: While higher k values provide better accuracy in predicting admissions, the potential bias towards predicting completion could impact the admissions strategy at TFA. A careful balance between sensitivity and specificity is crucial to ensure a more accurate and efficient admissions process.

(1.2) Naive Bayes

Approach

Our analysis utilized the Naive Bayes algorithm to predict the likelihood of applicants completing the admission process. The continuous variables in the dataset were scaled, and an 80-20 split was employed to separate the dataset into training and testing sets.
The GPA variable was categorized into bins (Low, Medium, High, Very High) as part of the data preprocessing. This approach aimed to simplify the model's interpretation of GPA, allowing it to identify trends and patterns that may not be apparent in continuous data. Binning is a crucial step in preparing the data for modeling and can significantly impact the model's performance.

Analysis

The performance of the Naive Bayes model was evaluated as follows:

Dataset	True Negatives	True Positives	False Positives	False Negatives	Accuracy
Training	885	15,576	3,403	1,624	76.61%
Holdout	220	3,897	852	403	76.64%

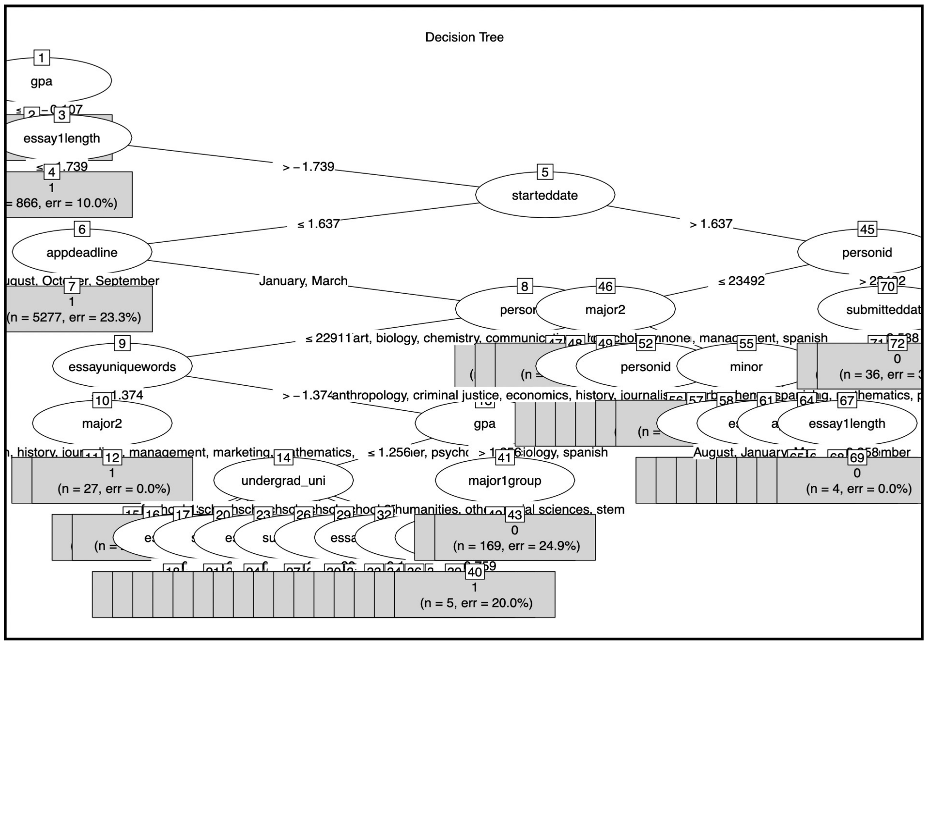
Results

- Model Performance: The Naive Bayes model demonstrated moderate accuracy with its strength in predicting true positives (applicants likely to complete the admission process). This is evidenced by the model's higher specificity in both the training and testing datasets.
- Potential Bias: A significant observation in the model's performance was the higher number of false positives. This suggests a bias towards predicting that applicants would complete the admission process when they might not. Such a bias can lead to an overestimation of successful admissions and may affect the overall efficiency of the admissions process.
- Practical Implications: The model's performance highlights the need for a balanced approach in predicting admission outcomes. While it is effective in identifying applicants likely to complete the process, the low number of positive predictions requires attention. Enhancing the model to better identify applicants who might not complete the admission process could lead to more accurate and efficient strategies in managing admissions. This balance is crucial for making informed decisions and allocating resources effectively in the context of admissions strategies.

(1.3) Decision Trees

Approach

Our team employed the Decision Tree algorithm, specifically using the C5.0 model, to predict the likelihood of candidates completing the admission process at Teach For America. We used various configurations of the model, particularly focusing on the number of trials, to understand its impact on prediction accuracy. The dataset was split into training and test sets following an 80-20 ratio, and models with different trial numbers (10, 8, and 25) were evaluated to gauge the best trial.



Analysis
The performance metrics for each model configuration were calculated as follows:

No. of Trials	True Negatives	True Positives	False Positives	False Negatives	Accuracy
Original	70	4609	1106	75	79.84%
10	108	4563	1068	121	79.70%
8	108	4572	1068	112	79.86%
25	184	4480	992	204	79.59%

Fine-tuning

- Sensitivity to Trials: Our analysis indicates that the number of trials in the Decision Trees model has a notable impact on the predictive performance. Specifically, increasing the trials generally improved the model's ability to correctly identify candidates unlikely to complete the admission process, as evidenced by an increase in true negatives.
- Class Imbalance and Prediction Bias: A consistent trend across all configurations was the model's inclination to predict a higher number of admissions ('1'). This tendency might suggest a class imbalance within the dataset. Addressing this could involve implementing strategies such as resampling or adjusting class weights in the model to achieve a more balanced prediction.

Result

- Model Performance: The Decision Trees model with 8 trials emerged as the most balanced configuration. This model exhibited the highest overall accuracy at 79.86% and maintained a reasonable balance between true negatives and true positives.
- Although the model with 25 trials showed the highest number of true negatives, it also resulted in a lower overall accuracy and a significant increase in false negatives. In contrast, the model with 8 trials offers a superior blend of accuracy and balanced prediction.
- Potential Bias and Practical Implications: Despite improvements in identifying candidates unlikely to complete the admission process, a noticeable number of false positives persists across all trial configurations. This suggests a potential bias towards predicting the completion of the admission process. Such bias could introduce inefficiencies into TFA's recruitment strategy.
- To enhance the accuracy and efficiency of TFA's admissions process, it is crucial to maintain a balance between sensitivity (true positive rate) and specificity (true negative

rate). The Decision Trees model with 8 trials offers this balance, making it a compelling choice for TFA's predictive analytics needs.

(1.4) ANN

Approach:
We utilized an Artificial Neural Network (ANN) model to predict the likelihood of applicants completing the admission process at Teach For America. The training and testing split was maintained at 80-20, and the model used a wide range of input features, including academic records, essay attributes, and engagement levels.

Analysis:
The performance of the ANN model across different configurations was as follows:

Nodes	True Negatives	True Positives	False Positives	False Negatives	Accuracy
Original Model	2	4297	3	1070	80.03%
Hidden Nodes: 2	3	4294	6	1069	79.99%
Hidden Nodes: 3	4	4295	5	1068	80.03%

Fine-tuning:
- Fine-tuning of the ANN model involved experimenting with the number of hidden nodes to enhance its predictive performance. The key adjustments and observations were:
- Adding 2 Hidden Nodes: This configuration showed a slight improvement in identifying true negatives, with a marginal decrease in false negatives.
- Adding 3 Hidden Nodes: With three hidden nodes, there was a further increase in true negatives, suggesting improved identification of non-completions.
- Attempts to Increase Hidden Nodes Further: When trying to increase the number of hidden nodes beyond three, the model frequently encountered convergence issues. Specifically, R outputted the message "Algorithm did not converge in 1 of 1 repetition(s) within the stepmax." This indicates that the model's learning process was not settling on a stable solution, suggesting a potential overfitting issue or the need for a more complex network structure.

Results:

- The accuracy of the models is approximately 80%, but given the imbalance in sensitivity and specificity, this might be misleading. The high accuracy might be primarily due to the model's ability to predict true positives rather than its overall predictive performance.
- Model Performance: The ANN model, particularly with 3 hidden nodes in its final iteration, showed an improved ability to correctly identify true negatives. However, the model still faced challenges in reducing false negatives.
- Practical Implications: While the ANN model effectively identifies applicants likely to complete the admission process, its current configuration may not be optimal for accurately predicting non-completions. This highlights the importance of continuous model tuning and validation to ensure a balanced approach in predicting admissions outcomes for Teach for America.

(1.5) SVM

Approach:

The process involved data preprocessing, model training, and fine-tuning to optimize performance. Our approach was methodical, starting with preprocessing steps that included scaling numerical data and converting categorical data to factors. This prepared our dataset for SVM modeling. We initially applied a linear SVM, then experimented with a Radial Basis Function (RBF) kernel to capture nonlinear patterns.

Analysis:

Model Type	True Negatives	True Positives	False Positives	False Negatives	Accuracy
Linear Model	0	4300	0	1072	80.04%
RBF Kernel	2	4299	1	1070	80.06%

Linear Kernel SVM:

- The linear kernel SVM produced a confusion matrix that revealed it failed to predict any negative class instances correctly. This is evidenced by the lack of true negatives (TN = 0) and false negatives (FN = 0), with a total of 1072 false positives (FP) and 4300 true positives (TP).
- The accuracy was 0.8004, which is misleading because the model is simply predicting the majority class for all instances.

RBF Kernel SVM:

- A slight improvement was observed with the RBF kernel, which predicted 2 true negatives (TN = 2) but still missed many negative instances, as indicated by the large number of false negatives (FN = 1070).

- The accuracy increased marginally to 0.8006, and the model showed a slight capability to predict the negative class, but the performance was still heavily biased towards the positive class.
- These confusion matrices paint a clear picture of the SVM models' performance. The linear kernel SVM failed entirely to identify the negative class, while the RBF kernel SVM showed a minimal ability to distinguish between classes. Both models demonstrate high specificity but extremely low sensitivity, which suggests they are not effective for balanced classification tasks, particularly in the context of predicting unsuccessful admissions (the negative class) in the TFA dataset.

Fine-tuning:
To fine-tune our model, we experimented with different cost values, which manage the trade-off between classifying training points correctly and having a smoother decision boundary. Our fine-tuning process demonstrated that increasing the cost improved the model's accuracy on the training data. This is visualized in the attached table, which shows a positive trend between cost values and training accuracy.

Cost Value	Training Accuracy
1	0.8019
5	0.8141
10	0.8246
15	0.8352
20	0.8422
25	0.8482
30	0.8529
35	0.856
40	0.8601

This shows the relationship between the cost parameter and the SVM's training accuracy. It's evident from the table that as the cost increases, so does the training accuracy. This suggests that the SVM more accurately fits the training data with a higher penalty for misclassification. However, it's important to note that higher training accuracy does not always equate to better generalization; it's crucial to check the performance on a validation set to ensure the model isn't overfitting.

Result:
The result of our analysis and fine-tuning is a nuanced understanding that while the SVM model's training accuracy increases with higher cost values, the model's practical performance on the test data remains questionable. The increased training accuracy suggests that the model fits the training data better with a higher cost. Still, the testing phase highlighted the model's inability to detect the negative class effectively.

Part C - Recommendation and Conclusion

Conclusion:
After a detailed analysis of various predictive models for forecasting the completion of the admission process at Teach for America (TFA), we have observed unique strengths and limitations in each model. This analysis is particularly significant for TFA's goal to enhance recruitment efficiency and allocate resources effectively, ultimately impacting underserved educational settings.

- kNN (K-Nearest Neighbors): Exhibited high accuracy, particularly with k=5. However, it demonstrated a bias towards predicting positive outcomes, leading to a high rate of false positives. This model's simplicity is advantageous for smaller datasets, but the noted bias could affect balanced prediction needs.
- Naive Bayes: Achieved moderate accuracy, excelling in identifying true positives. Its tendency towards predicting applicants would complete the admission process resulted in numerous false positives, raising concerns about overestimation in successful admissions.
- Decision Trees (C5.0 model): Initially, the model with 25 trials seemed ideal, but a reassessment highlighted that 8 trials offer the best balance. This configuration showed the highest accuracy (79.86%) with a more equitable distribution of predictive outcomes. It effectively identified both successful and unsuccessful admissions with a reasonable trade-off between true negatives and true positives.
- ANN (Artificial Neural Networks): Maintained around 80% accuracy but struggled with imbalance in sensitivity and specificity. The model showed improved performance with added hidden nodes but also faced convergence issues and bias towards positive outcomes.
- SVM (Support Vector Machines): Demonstrated a heavy bias towards positive predictions with marginal improvements in accuracy. Both linear and RBF kernel SVMs showed minimal effectiveness in identifying the negative class, raising questions about their practical applicability.

Recommendation:
In the context of Teach For America's specific needs, both the kNN and Decision Tree models emerge as viable options. However, upon careful analysis, the **Decision Tree model with 8 trials is the superior choice**. This conclusion is drawn based on several key factors:

Parameters	**k-Nearest Neighbors (k=5)**	**Decision Trees (8 Trials)**
Interpretability	Less Interpretable – difficult to trace back the paths of the predictive model.	Highly Interpretable - allows for easier understanding and communication of how predictions are made.
Sensitivity to Imbalanced Data	Highly sensitive to class imbalances, affecting the accuracy and reliability of predictions.	Better equipped to handle class imbalances, offering more reliable predictions in diverse data scenarios.
Practicality for Complex Datasets	Performance may diminish with complex datasets containing a mix of categorical and numerical data.	More adept at handling complex datasets with various types of data, offering versatile applicability.
Scalability and Efficiency	Computationally intensive as dataset size grows, which might be challenging for TFA's expanding data needs.	Generally more scalable and efficient in processing large datasets, suitable for TFA's evolving data requirements.

In light of our comprehensive analysis, it is our strong recommendation to adopt the Decision Trees model with 8 trials for Teach For America's (TFA) predictive analytics requirements. This model distinctly surpasses others in critical aspects that align with TFA's strategic objectives. Its superior balanced accuracy ensures robust predictions for both successful and unsuccessful admissions, a vital factor given the unique characteristics of TFA's applicant dataset. Notably, its adept handling of class imbalances in the data highlights its suitability for TFA's context, where nuanced prediction capabilities are essential.

Moreover, the Decision Trees model offers exceptional interpretability, a key attribute that facilitates insightful, data-driven decision-making. This transparency in the model's decision-making process is invaluable for stakeholders to understand and leverage insights effectively. The model also demonstrates a sustainable advantage, being adaptable and maintainable over time.

Therefore, based on our rigorous evaluation, the Decision Trees model with 8 trials is not merely a technical choice, but a strategic one, poised to significantly enhance TFA's recruitment efficiency and contribute profoundly to its mission of advancing educational equity. This recommendation is made with the conviction that the model will serve as a robust, reliable, and insightful tool in TFA's pursuit of excellence in educational recruitment.

Part D - Appendix

KNN

```
# import the CSV file
data <- read.csv("pp.csv", stringsAsFactors = FALSE)

summary(data)
head(data)
str(data)
sum(is.na(data))

# Removing rows with NA values
data <- na.omit(data)

# Remove 'personid' and 'appyear' column
data$personid <- NULL
data$appyear <- NULL

# Convert chr variables into factors
categorical_columns <- c("schoolsel_chr", "major1", "major2", "minor", "major1group", "major2group",
"minorgroup", "undergrad_uni", "appdeadline")
data[categorical_columns] <- lapply(data[categorical_columns], as.factor)

# Convert 'stem', 'schoolsel', 'attendedevent', and 'completedadm' to factors
data$stem <- as.factor(data$stem)
data$schoolsel <- as.factor(data$schoolsel)
data$attendedevent <- as.factor(data$attendedevent)
data$completedadm <- as.factor(data$completedadm)
data$schoolsel_chr <- as.factor(data$schoolsel_chr)
data$major1 <- as.factor(data$major1)
data$major2 <- as.factor(data$major2)
data$minor <- as.factor(data$minor)
data$major1group <- as.factor(data$major1group)
data$major2group <- as.factor(data$major2group)
data$minorgroup <- as.factor(data$minorgroup)
data$undergrad_uni <- as.factor(data$undergrad_uni)
data$undergrad_uni <- as.factor(data$undergrad_uni)

# Converting catagorical variables into numeric variables
data[categorical_columns] <- lapply(data[categorical_columns], function(x) as.integer(as.factor(x)))

# Check the names of the columns in the dataset
print(colnames(data))

# View the first few rows of the completedadm column
print(head(data$completedadm))
```

```
# Check the number of unique values in the completedadm column
print(length(unique(data$completedadm)))
# Scale numeric variables
numeric_columns <- c('gpa', 'essay1length', 'essay2length', 'essay3length', 'essayuniquewords',
'essayssentiment', 'signupdate', 'starteddate', 'submitteddate')
data[, numeric_columns] <- scale(data[, numeric_columns])

#df <- data[, numeric_columns]

summary(data[c('gpa', 'essay1length', 'essay2length', 'essay3length', 'essayuniquewords',
'essayssentiment', 'signupdate', 'starteddate', 'submitteddate')])

set.seed(123)

# Create training and test data
# Randomly split data into training and holdout
idx <- sample(nrow(data), 0.8*nrow(data)) # 80% training, rest holdout
length(idx)

# Create training and holdout data sets
train_data <-
data[idx,c("gpa","stem","schoolsel","schoolsel_chr","major1","major2","minor","major1group","major2
group","minorgroup","undergrad_uni","essay1length","essay2length","essay3length",
"essayuniquewords","essayssentiment","signupdate","starteddate","appdeadline","submitteddate","atten
dedevent","completedadm")]
test_data <- data[-
idx,c("gpa","stem","schoolsel","schoolsel_chr","major1","major2","minor","major1group","major2grou
p","minorgroup","undergrad_uni","essay1length","essay2length","essay3length",
"essayuniquewords","essayssentiment","signupdate","starteddate","appdeadline","submitteddate","atten
dedevent","completedadm")]

head(train_data)
head(test_data)

# create labels for training and test data
train_labels <- data$completedadm[idx]
test_labels <- data$completedadm[-idx]

library(class)
library(gmodels)

test_pred <- knn(train = train_data, test = test_data, cl = train_labels, k = 1)

# Evaluating model performance by creating a cross tabulation of actual vs. predicted
table(x = test_labels, y = test_pred)
```

```
CrossTable(x = test_labels, y = test_pred,prop.chisq = FALSE)

test_pred <- knn(train = train_data, test = test_data, cl = train_labels, k = 3)
CrossTable(x = test_labels, y = test_pred, prop.chisq=FALSE)

test_pred <- knn(train = train_data, test = test_data, cl = train_labels, k = 5)
CrossTable(x = test_labels, y = test_pred, prop.chisq=FALSE)

test_pred <- knn(train = train_data, test = test_data, cl = train_labels, k = 7)
CrossTable(x = test_labels, y = test_pred, prop.chisq=FALSE)

test_pred <- knn(train = train_data, test = test_data, cl = train_labels, k = 11)
CrossTable(x = test_labels, y = test_pred, prop.chisq=FALSE)
```

Naive Bayes

```
library(caret)
library(e1071)
setwd("C:/Users/asinha27/Downloads")

# Load the dataset
nb <- read.csv('dataset.csv', stringsAsFactors = TRUE)
nb <- na.omit(nb)

# Convert 'completedadm' to a factor
nb$completedadm <- factor(nb$completedadm)

# Binning the 'gpa' variable into 4 bins for example
nb$gpa_binned <- cut(nb$gpa, breaks=4, labels=c("Low", "Medium", "High", "Very High"))
nb$gpa <- NULL
str(nb)

# Feature Scaling for numeric variables (excluding 'gpa' which we've binned)
numeric_columns <- c('essay1length', 'essay2length',
                  'essay3length', 'essayuniquewords',
                  'essayssentiment', 'signupdate',
                  'starteddate', 'submitteddate')
nb[, numeric_columns] <- scale(nb[, numeric_columns])

# Create a data partition for training and testing
set.seed(123)
idx <- createDataPartition(nb$completedadm, p=0.8, list=FALSE)
train.df <- nb[idx, ]
holdout.df <- nb[-idx, ]

# Train Naive Bayes model
```

```
adm.nb <- naiveBayes(completedadm ~ ., data = train.df)

# Compute Conditional Proportions
prop.table(table(train.df$completedadm), margin = 1)

# Predict probabilities
adm.pred.prob <- predict(adm.nb, newdata=holdout.df, type="raw")

# Predict class membership
adm.pred.class <- predict(adm.nb, newdata=holdout.df)

# Create a dataframe combining class and probabilities
admpred.df <- data.frame(actual=holdout.df$completedadm, predicted=adm.pred.class, adm.pred.prob)

# Confusion Matrix for Training Data------- Accuracy : 0.7661
confusionMatrix(predict(adm.nb, newdata=train.df), train.df$completedadm)

# Confusion Matrix for Test (Holdout)------Data Accuracy : 0.7664
confusionMatrix(predict(adm.nb, newdata=holdout.df), holdout.df$completedadm)
```

Decision Trees

```
library(caret)
library(rpart)
library(C50)
library(gmodels)
getwd()
setwd("C:/Users/asinha27/Downloads")
dataset_dt <- read.csv("dataset.csv", stringsAsFactors = TRUE)
str(dataset_dt)
head(dataset_dt)
dataset_dt <- na.omit(dataset_dt)
summary(dataset_dt)

# Convert outcome variable to factor
dataset_dt$completedadm <- factor(dataset_dt$completedadm)

# Feature Scaling for numeric variables
numeric_columns <- c('gpa', 'essay1length', 'essay2length',
                     'essay3length', 'essayuniquewords',
                     'essayssentiment', 'signupdate', 'starteddate',
                     'submitteddate')
dataset_dt[, numeric_columns] <- scale(dataset_dt[, numeric_columns])

# looking at the outcome variable
table(dataset_dt$completedadm)
```

```
# 0 - 5360    1 - 21500

# create a random sample for training and test data
set.seed(123) # use set.seed to use the same random number sequence as the demo

train_sample <- sample(26860, 21000)
str(train_sample)

# split the data frames
dataset.dt_train <- dataset_dt[train_sample, ]
dataset.dt_test <- dataset_dt[-train_sample, ]

# check the proportion of class variable
prop.table(table(dataset.dt_train$completedadm))
prop.table(table(dataset.dt_test$completedadm))

#Train - 0-0.1997021 1 - 0.8002979
#Test - 0- 0.1989587 1 - 0.8010413

## Training a model on the data -
dt_model <- C5.0(dataset.dt_train[c(-2,-24)], dataset.dt_train$completedadm)
# display simple facts about the tree
dt_model

# display detailed information about the tree
#summary(dt_model)

## Evaluating model performance -
# create a factor vector of predictions on test data
dt_model_pred <- predict(dt_model, dataset.dt_test)

# Display decision tree
plot(dt_model, type="s", main="Decision Tree")

# cross tabulation of predicted versus actual classes
CrossTable(dataset.dt_test$completedadm, dt_model_pred,
          prop.chisq = FALSE, prop.c = FALSE, prop.r = FALSE,
          dnn = c('actual', 'predicted'))
#false 1  -75 , false 0 - 1106
#actual 1  -4609 , actual 0 - 70

#Fine tuning the model
ctrl <- C5.0Control(noGlobalPruning = FALSE, winnow = TRUE)

# Train the model with new control settings, including trials
dt_model_tuned <- C5.0(x = dataset.dt_train[c(-2,-24)],
                    y = dataset.dt_train$completedadm,
                    trials = 10, control = ctrl)
# Summarize the model
summary(dt_model_tuned)

# Make predictions on the test set
dt_model_pred_tuned <- predict(dt_model_tuned, dataset.dt_test)

# Evaluate the performance
CrossTable(dataset.dt_test$completedadm, dt_model_pred_tuned,
          prop.chisq = FALSE, prop.c = FALSE, prop.r = FALSE,
          dnn = c('actual', 'predicted'))
#false 1  -121 , false 0 - 1068
#actual 1  -4563 , actual 0 - 108

#Trial 2
# Train the model with new control settings, including trials
dt_model_tuned2 <- C5.0(x = dataset.dt_train[c(-2,-24)],
                    y = dataset.dt_train$completedadm,
                    trials = 8, control = ctrl)
# Summarize the model
summary(dt_model_tuned2)

# Make predictions on the test set
dt_model_pred_tuned2 <- predict(dt_model_tuned2, dataset.dt_test)

# Evaluate the performance
CrossTable(dataset.dt_test$completedadm, dt_model_pred_tuned2,
          prop.chisq = FALSE, prop.c = FALSE, prop.r = FALSE,
          dnn = c('actual', 'predicted'))
#false 1  - 112  , false 0 - 1068
#actual 1  -4572  , actual 0 - 108

#Trial 3
# Train the model with new control settings, including trials
dt_model_tuned3 <- C5.0(x = dataset.dt_train[c(-2,-24)],
                    y = dataset.dt_train$completedadm,
                    trials = 25, control = ctrl)
# Summarize the model
summary(dt_model_tuned3)

# Make predictions on the test set
dt_model_pred_tuned3 <- predict(dt_model_tuned3, dataset.dt_test)

# Evaluate the performance
CrossTable(dataset.dt_test$completedadm, dt_model_pred_tuned3,
            prop.chisq = FALSE, prop.c = FALSE, prop.r = FALSE,
            dnn = c('actual', 'predicted'))
#false 1  -204  , false 0 - 992
#actual 1  -4480 , actual 0 - 184
```

ANN

```
library(fastDummies)
library(caret)
library(neuralnet)

# import the CSV file
data <- read.csv("pp.csv", stringsAsFactors = FALSE)

summary(data)
head(data)
str(data)
sum(is.na(data))

# Removing rows with NA values
data <- na.omit(data)

# Remove 'personid' and 'appyear' column
data$personid <- NULL
data$appyear <- NULL

# Convert chr variables into factors
categorical_columns <- c("schoolsel_chr", "major1", "major2", "minor", "major1group", "major2group",
"minorgroup", "undergrad_uni", "appdeadline")
data[categorical_columns] <- lapply(data[categorical_columns], as.factor)

# Convert 'stem', 'schoolsel', 'attendedevent', and 'completedadm' to factors
data$schoolsel_chr <- as.factor(data$schoolsel_chr)
data$major1 <- as.factor(data$major1)
data$major2 <- as.factor(data$major2)
data$minor <- as.factor(data$minor)
data$major1group <- as.factor(data$major1group)
data$major2group <- as.factor(data$major2group)
data$minorgroup <- as.factor(data$minorgroup)
data$undergrad_uni <- as.factor(data$undergrad_uni)

# Converting catagorical variables into numeric variables

data[categorical_columns] <- lapply(data[categorical_columns], function(x) as.integer(as.factor(x)))

# Check the names of the columns in the dataset
print(colnames(data))
```

```
# View the first few rows of the completedadm column
print(head(data$completedadm))

# Check the number of unique values in the completedadm column
print(length(unique(data$completedadm)))
head(data$completedadm)

# Scale numeric variables
numeric_columns <- c('gpa', 'essay1length', 'essay2length', 'essay3length', 'essayuniquewords',
'essayssentiment', 'signupdate', 'starteddate')
data[, numeric_columns] <- scale(data[, numeric_columns])

#df <- data[, numeric_columns]

summary(data[c('gpa', 'essay1length', 'essay2length', 'essay3length', 'essayuniquewords',
'essayssentiment', 'signupdate', 'starteddate')])

set.seed(123)

# Create training and test data
# Randomly split data into training and holdout
idx <- sample(nrow(data), 0.8*nrow(data)) # 80% training, rest holdout
length(idx)

# Create training and holdout data sets
train_data <-
data[idx,c("gpa","stem","schoolsel","schoolsel_chr","major1","major2","minor","major1group","major2
group","minorgroup","undergrad_uni","essay1length","essay2length","essay3length",
"essayuniquewords","essayssentiment","signupdate","starteddate","appdeadline","submitteddate","atten
dedevent","completedadm")]
test_data <- data[-
idx,c("gpa","stem","schoolsel","schoolsel_chr","major1","major2","minor","major1group","major2grou
p","minorgroup","undergrad_uni","essay1length","essay2length","essay3length",
"essayuniquewords","essayssentiment","signupdate","starteddate","appdeadline","submitteddate","atten
dedevent","completedadm")]

str(train_data )
str(test_data)
train_data$completedadm
test_data$completedadm

# create labels for training and test data
train_labels <- data$completedadm[idx]
test_labels <- data$completedadm[-idx]
```

```
# Train the NN
nn <- neuralnet(completedadm ~ ., data=train_data)

# Predict using the trained neural network
ann_pred.test <- predict(nn, test_data)
str(test_data)
head(test_data$completedadm)

# Assign class based on the prediction
ann_class.test <- ifelse(ann_pred.test > 0.5, 1, 0)
ann_class.test <- as.integer(ann_class.test)
ann_class.test

levels(factor(ann_class.test))
levels(factor(test_data$completedadm))

# Create confusion matrix
confusionMatrix(factor(ann_class.test), factor(test_data$completedadm))

# Improve the NN by adding hidden nodes
imp_nn <- neuralnet(completedadm ~ ., data=train_data, hidden=2)
# Predict and compute CM
pred.test <- predict(imp_nn, test_data)
class.test <- ifelse(pred.test > 0.5, 1, 0)
confusionMatrix(factor(class.test), factor(test_data$completedadm))

# Improve the NN by adding hidden nodes
imp_nn <- neuralnet(completedadm ~ ., data=train_data, hidden=3)

# Predict and compute CM
pred.test <- predict(imp_nn, test_data)
class.test <- ifelse(pred.test > 0.5, 1, 0)
confusionMatrix(factor(class.test), factor(test_data$completedadm))

# Improve the NN by adding hidden nodes
imp_nn <- neuralnet(completedadm ~ ., data=train_data, hidden=4)

# Predict and compute CM
pred.test <- predict(imp_nn, test_data)
class.test <- ifelse(pred.test > 0.5, 1, 0)
confusionMatrix(factor(class.test), factor(test_data$completedadm))
```

SVM

```
# import the CSV file
data <- read.csv("pp.csv", stringsAsFactors = FALSE)
```

```
summary(data)
head(data)
str(data)
sum(is.na(data))

# Removing rows with NA values
data <- na.omit(data)

# Remove 'personid' and 'appyear' column
data$personid <- NULL
data$appyear <- NULL

# Convert chr variables into factors
categorical_columns <- c("schoolsel_chr", "major1", "major2", "minor", "major1group", "major2group",
"minorgroup", "undergrad_uni", "appdeadline")
data[categorical_columns] <- lapply(data[categorical_columns], as.factor)

# Convert 'stem', 'schoolsel', 'attendedevent', and 'completedadm' to factors
data$stem <- as.factor(data$stem)
data$schoolsel <- as.factor(data$schoolsel)
data$attendedevent <- as.factor(data$attendedevent)
data$completedadm <- as.factor(data$completedadm)
data$schoolsel_chr <- as.factor(data$schoolsel_chr)
data$major1 <- as.factor(data$major1)
data$major2  <- as.factor(data$major2)
data$minor <- as.factor(data$minor)
data$major1group <- as.factor(data$major1group)
data$major2group <- as.factor(data$major2group)
data$minorgroup <- as.factor(data$minorgroup)
data$undergrad_uni <- as.factor(data$undergrad_uni)
data$undergrad_uni <- as.factor(data$undergrad_uni)

# Converting catagorical variables into numeric variables

data[categorical_columns] <- lapply(data[categorical_columns], function(x) as.integer(as.factor(x)))
# Check the names of the columns in the dataset
print(colnames(data))

# View the first few rows of the completedadm column
print(head(data$completedadm))

# Check the number of unique values in the completedadm column
print(length(unique(data$completedadm)))

# Scale numeric variables
```

```
numeric_columns <- c('gpa', 'essay1length', 'essay2length', 'essay3length', 'essayuniquewords',
'essayssentiment', 'signupdate', 'starteddate', 'submitteddate')
data[, numeric_columns] <- scale(data[, numeric_columns])

#df <- data[, numeric_columns]

summary(data[c('gpa', 'essay1length', 'essay2length', 'essay3length', 'essayuniquewords',
'essayssentiment', 'signupdate', 'starteddate', 'submitteddate')])

set.seed(123)

# Create training and test data
# Randomly split data into training and holdout
idx <- sample(nrow(data), 0.8*nrow(data)) # 80% training, rest holdout
length(idx)
length(-idx)

# Create training and holdout data sets
train_data <-
data[idx,c("gpa","stem","schoolsel","schoolsel_chr","major1","major2","minor","major1group","major2
group","minorgroup","undergrad_uni","essay1length","essay2length","essay3length",
"essayuniquewords","essayssentiment","signupdate","starteddate","appdeadline","submitteddate","atten
dedevent","completedadm")]
test_data <- data[-
idx,c("gpa","stem","schoolsel","schoolsel_chr","major1","major2","minor","major1group","major2grou
p","minorgroup","undergrad_uni","essay1length","essay2length","essay3length",
"essayuniquewords","essayssentiment","signupdate","starteddate","appdeadline","submitteddate","atten
dedevent","completedadm")]

str(train_data)
str(test_data)

head(train_data)
head(test_data)

# create labels for training and test data
train_labels <- data$completedadm[idx]
test_labels <- data$completedadm[-idx]

# Proceed with model development for SVM
library(kernlab)

# begin by training a simple SVM with linear kernel
completedadm_classifier <- ksvm(completedadm ~ ., data = train_data, kernel = "vanilladot")

# look at basic information about the model
```

```
completedadm_classifier

# predictions on testing dataset
completedadm_predictions <- predict(completedadm_classifier, test_data)

head(completedadm_predictions)

table(completedadm_predictions, test_data$completedadm)

# look only at agreement vs. non-agreement
# construct a vector of TRUE/FALSE indicating correct/incorrect predictions
agreement <- completedadm_predictions == test_data$completedadm
table(agreement)

## Improving model performance ----

# change to a RBF kernel
set.seed(123)
completedadm_classifier_rbf <- ksvm(completedadm ~ ., data = train_data, kernel = "rbfdot")
completedadm_predictions_rbf <- predict(completedadm_classifier_rbf, test_data)

agreement_rbf <- completedadm_predictions_rbf == test_data$completedadm
table(agreement_rbf)
prop.table(table(agreement_rbf))

library(caret)
library(e1071)

# Calculate Accuracy, Precision, Recall, and F1 Score for Linear Kernel
conf_matrix_linear <- confusionMatrix(as.factor(completedadm_predictions),
as.factor(test_data$completedadm))
print(conf_matrix_linear)

# Calculate Accuracy, Precision, Recall, and F1 Score for RBF Kernel
conf_matrix_rbf <- confusionMatrix(as.factor(completedadm_predictions_rbf),
as.factor(test_data$completedadm))
print(conf_matrix_rbf)

# Define a range of cost values
cost_values <- c(1, seq(from = 5, to = 40, by = 5))

# Initialize a vector to store accuracy values
accuracy_values <- numeric(length(cost_values))

# Loop over cost values, train SVM, and calculate accuracy
for (i in seq_along(cost_values)) {
  svm_model <- ksvm(completedadm ~ ., data = train_data, kernel = "rbfdot", C = cost_values[i])
  predictions <- predict(svm_model, train_data)
  accuracy_values[i] <- sum(predictions == train_data$completedadm) / nrow(train_data)
}

# Print accuracy values
print(accuracy_values)

# Plot Cost vs. Accuracy
plot(cost_values, accuracy_values, type = "b", xlab = "Cost Value", ylab = "Training Accuracy", main =
"SVM Training Accuracy vs. Cost")
```