MAX 522 / MSC 615 – Predictive Analytics

TEACH FOR AMERICA – Predictive Analytics Report

Predicting Applicant Withdrawal Risk

Presented By

Shruthi Rajeshwari Ganapathiappan Senthilkumar

1. Objective

The main goal of this predictive analytics project is to identify candidates who are likely to drop out during the Teach For America admissions process. By accurately predicting which applicants will complete or drop out, TFA can:

- Optimize recruitment efforts to allocate resources to high-risk applicants.
- Reduce the rate of applicant dropouts through targeted interventions and engagement strategies.
- Improve the efficiency of overall admissions by using data insights to prioritize communication and support.
- Inform strategic decision-making for long-term recruitment planning and talent pipeline management.

Our Goal is to create accurate, interpretable, and actionable predictive models for operational deployment.

2. Data Overview

This analysis is based on a dataset provided by TFA containing information on applicant demographic and academic background, and application behavior.

Dataset summary:

Attribute	Description
Rows	74,839 applicants
Target Variable	Completed Admissions Process (1 = Completed, 0 = Withdrew)
Predictor Variables	• academic data (GPA, major, minor) • application timelines (sign-up date, start date, submission date, deadline), • essay sentiment • event attendance • demographic information
Class Distribution	• 60,429 applicants - completed the application process (81%) • 14,410 applicants - withdrew (19%)

Observations:

- The dataset is imbalanced, and the minority class is withdrawals.
- Various features are date-based, thus enabling the creation of behavioural metrics such as days to start or submit.
- Categorical variables (majors, universities, regions) contain high cardinality features that must be pre-processed.
- Text-based features such as essay sentiment are already quantified but need to be converted to factors for modelling.

Initial class Distribution:

```
> cat("Original distribution:\n")
Original distribution:
> cat("  Completed (1):", nrow(completed), "\n")
  Completed (1): 60429
> cat("  Withdrew (0):", nrow(withdrew), "\n\n")
  Withdrew (0): 14410
```

Key Takeaways:

- Behavioural features (timing between steps) are likely strong predictors of completion.
- High class imbalance needs to be taken into consideration for effective modeling.
- The preprocessing will include handling missing values, factor encoding, and balancing the dataset.

3. Data Preparation & Feature Engineering

Preprocessing involved dataset balancing, creation of derived features, preprocessing categorical and numeric variables, and handling missing values to prepare the TFA dataset for predictive modeling.

Balanced Sampling:
The original dataset was highly imbalanced.

- Completed: 60,429
- Withdrew: 14,410

To avoid the bias of the majority class, we kept all samples from the minority class (Withdrew) and randomly sampled an equivalent number from the majority class (Completed).

```
> cat("Balanced sample created:\n")
Balanced sample created:
> cat("  Total rows:", nrow(tfa_data), "\n")
  Total rows: 28820
> cat("  Completed (1):", sum(tfa_data$`Completed Admissions Process` == 1), "\n")
  Completed (1): 14410
> cat("  Withdrew (0):", sum(tfa_data$`Completed Admissions Process` == 0), "\n\n")
  Withdrew (0): 14410
```

Feature Engineering:

Additional features that capture applicant behavior and timing patterns were derived as follows:

Feature	Description	R Code
days_signup_to_start	Days between sign-up and starting the application	as.numeric(difftime(Started Date, Sign-up Date, units="days"))
days_start_to_submit	Days between starting and submitting the application	as.numeric(difftime(Submitted Date, Started Date, units="days"))
days_signup_to_submit	Total days from sign-up to the submission	as.numeric(difftime(Submitted Date, Sign-up Date, units="days"))
days_before_deadline	Days submitted before the application deadline	as.numeric(difftime(Application Deadline, Submitted Date, units="days"))
submitted_early	1 if submission > 7 days before deadline	ifelse(days_before_deadline > 7, 1, 0)
quick_starter	1 if start ≤ 1 day after sign-up	ifelse(days_signup_to_start <= 1, 1, 0)
quick_completer	1 if submission ≤ 7 days after start	ifelse(days_start_to_submit <= 7, 1, 0)
signup_month, signup_year, submit_month	Extracted month/year from dates	month(Sign-up Date) / year(Sign-up Date)
Essays Sentiment	Converted to Positive/Negative factor	factor(ifelse(Essays Sentiment >= 0, "Positive", "Negative"))

Preprocessing

Raw datasets often come with high-cardinality categorical variables, missing values, and near-zero variance features. Preprocessing ensures that models are trained efficiently and do not overfit the data.

Steps Taken:

- High-Cardinality Reduction
    - Major 1, Major 2, Minor, and Undergraduate University are reduced to top 20 categories, the rest labeled as "Other".
    - It reduces model complexity and prevents overfitting.

- Missing Value Imputation
    - Numeric columns → median
    - Categorical columns → mode
- Converting Between Factors
    - Tree-based models handle factors naturally: Decision Tree, Naive Bayes.
    - Numeric models require dummy variables for categorical inputs: SVM, kNN, ANN.
- Remove Near-Zero Variance Features
    - Columns having very little or no variation are removed, hence preventing models from learning spurious patterns.

Train-Test Split

The data were split into 80% training and 20% testing subsets to evaluate the performance of the model on unseen data.

```
> cat("Train:", nrow(train_tree), "| Test:", nrow(test_tree), "\n\n")
Train: 23056 | Test: 5764
```

Size:

- Training: 23,056 rows
- Testing: 5,764 rows

Importance of viewing it:

- Ensures model generalization and prevents overfitting.
- Provides a dependable benchmark with which to compare multiple predictive models.

Key Takeaways

- Features on applicants' behaviour, such as time to start and submit, are some of the critical predictors for applicant withdrawal.
- It also enables balanced dataset modelling for predicting the minority class.
- Preprocessing steps ensure stability and efficiency of a model, particularly for high-dimensional categorical variables. Now the structured dataset is ready for several predictive models, such as Decision Trees, SVM, KNN, ANN, and Naive Bayes.

4. Model Development & Evaluation

The goal of this stage is to develop predictive models that classify applicants into those who will complete or withdraw from the admissions process and assess their performance using relevant metrics.

Five models were developed:

- Decision Tree (C5.0)
- k-Nearest Neighbors (kNN)
- Naive Bayes
- Support Vector Machine (SVM)
- Artificial Neural Network (ANN)

The evaluation metrics used are Accuracy, Sensitivity (Recall), Specificity, Precision, F1-score, and ROC-AUC.

- Training Control Setup

  Model tuning was done with a 5-fold cross-validation strategy, using ROC as the optimization metric.

```
> ctrl <- trainControl(
+   method = "cv",
+   number = 5,
+   classProbs = TRUE,
+   summaryFunction = twoClassSummary
+ )
```

Model 1: Decision Tree (C5.0)

Decision Trees handle categorical and numeric data, are interpretable, and can capture non-linear relationships.

Best Parameters:

Parameter	Value
Trials	20
Model	Tree
Winnow	FALSE

```
--- DECISION TREE RESULTS ---
> print(dt_cm)
Confusion Matrix and Statistics

          Reference
Prediction  Withdrew Completed
  Withdrew      1935      1091
  Completed      947      1791

               Accuracy : 0.6464
                 95% CI : (0.6339, 0.6588)
    No Information Rate : 0.5
    P-Value [Acc > NIR] : < 2.2e-16

                  Kappa : 0.2929

 Mcnemar's Test P-Value : 0.001537

            Sensitivity : 0.6214
            Specificity : 0.6714
         Pos Pred Value : 0.6541
         Neg Pred Value : 0.6395
             Prevalence : 0.5000
         Detection Rate : 0.3107
   Detection Prevalence : 0.4750
      Balanced Accuracy : 0.6464

       'Positive' Class : Completed

> cat("\nPrecision:", sprintf("%.4f", dt_cm$byClass['Pos Pred Value']), "\n")

Precision: 0.6541
```

Performance on Test Set:

Metric	Value
Accuracy	64.6%
Sensitivity	62.1%
Specificity	67.1%
Precision	65.4%
Balanced Accuracy	64.6%

Interpretation:

- This model is moderately predictive and identifies 62% of the completed applicants.
- Balanced accuracy is a fair trade-off between both classes.
- Decision trees provide interpretability, thereby offering operational insights.

Model 2: k-Nearest Neighbors (kNN)

- KNN predicts the class based on the majority label of nearest neighbors in feature space.
- Sensitive to scaling and high-dimensional data.

```
--- k-NEAREST NEIGHBORS RESULTS ---
> print(knn_cm)
Confusion Matrix and Statistics

           Reference
Prediction  Withdrew Completed
  Withdrew      1596      1262
  Completed     1286      1620

               Accuracy : 0.5579
                 95% CI : (0.545, 0.5708)
    No Information Rate : 0.5
    P-Value [Acc > NIR] : <2e-16

                  Kappa : 0.1159

 Mcnemar's Test P-Value : 0.6486

            Sensitivity : 0.5621
            Specificity : 0.5538
         Pos Pred Value : 0.5575
         Neg Pred Value : 0.5584
             Prevalence : 0.5000
         Detection Rate : 0.2811
   Detection Prevalence : 0.5042
      Balanced Accuracy : 0.5579

       'Positive' Class : Completed

> cat("\nPrecision:", sprintf("%.4f", knn_cm$byClass['Pos Pred Value']), "\n")

Precision: 0.5575
```

Metric	Value
Accuracy	55.8%
Sensitivity	56.2%
Specificity	55.4%
Precision	55.8%
Balanced Accuracy	55.8%

Interpretation:

- Performance is below Decision Tree.
- kNN struggles with high-dimensional dummy-encoded features.
- Useful for a benchmark, but not optimal for TFA prediction.

Model 3: Naive Bayes

- Probabilistic model that assumes feature independence.
- Good for categorical data and text sentiment features.

```
--- NAIVE BAYES RESULTS ---
> print(nb_cm)
Confusion Matrix and Statistics

          Reference
Prediction  Withdrew Completed
  Withdrew       416       142
  Completed     2466      2740

               Accuracy : 0.5475
                 95% CI : (0.5346, 0.5604)
    No Information Rate : 0.5
    P-Value [Acc > NIR] : 2.794e-13

                  Kappa : 0.0951

 Mcnemar's Test P-Value : < 2.2e-16

            Sensitivity : 0.9507
            Specificity : 0.1443
         Pos Pred Value : 0.5263
         Neg Pred Value : 0.7455
             Prevalence : 0.5000
         Detection Rate : 0.4754
   Detection Prevalence : 0.9032
      Balanced Accuracy : 0.5475

       'Positive' Class : Completed

> cat("\nPrecision:", sprintf("%.4f", nb_cm$byClass['Pos Pred Value']), "\n")

Precision: 0.5263
```

Metric	Value
Accuracy	54.8%
Sensitivity	95.1%
Specificity	14.4%
Precision	52.6%
Balanced Accuracy	54.7%

Interpretation:

- Very high sensitivity means that nearly all completed applicants are detected.
- Extremely low specificity shows very poor performance in recognizing withdrawal symptoms.
- Over-predicts completion, limiting operational usefulness.
-

Model 4: Support Vector Machine (SVM)

- SVMs handle nonlinear decision boundaries well.

```
> confusionMatrix(predict(svm_mod, test_data), test_data$Completed.Admissions.Process, positive="1")
Confusion Matrix and Statistics

          Reference
Prediction    0    1
         0 1837 1111
         1 1045 1771

               Accuracy : 0.626
                 95% CI : (0.6133, 0.6385)
    No Information Rate : 0.5
    P-Value [Acc > NIR] : <2e-16

                  Kappa : 0.2519

 Mcnemar's Test P-Value : 0.1616

            Sensitivity : 0.6145
            Specificity : 0.6374
         Pos Pred Value : 0.6289
         Neg Pred Value : 0.6231
             Prevalence : 0.5000
         Detection Rate : 0.3073
   Detection Prevalence : 0.4885
      Balanced Accuracy : 0.6260

       'Positive' Class : 1
```

Metric	Value
Accuracy	62.6%
Sensitivity	61.45%
Specificity	63.7%
Precision	64.1%
Balanced Accuracy	62.6%

Interpretation:

- Performance similar to Decision Tree.
- Slightly higher specificity → better detection of withdrawals.
- Suitable for numerical-heavy feature sets.

Model 5: Artificial Neural Network (ANN)

- Can capture complex, non-linear interactions between features.
- Scales well with multiple features.

```
--- ARTIFICIAL NEURAL NETWORK RESULTS ---
> print(ann_cm)
Confusion Matrix and Statistics

          Reference
Prediction  Withdrew Completed
  Withdrew      1819      1062
  Completed     1063      1820

               Accuracy : 0.6313
                 95% CI : (0.6187, 0.6438)
    No Information Rate : 0.5
    P-Value [Acc > NIR] : <2e-16

                  Kappa : 0.2627

 Mcnemar's Test P-Value : 1

            Sensitivity : 0.6315
            Specificity : 0.6312
         Pos Pred Value : 0.6313
         Neg Pred Value : 0.6314
             Prevalence : 0.5000
         Detection Rate : 0.3158
   Detection Prevalence : 0.5002
      Balanced Accuracy : 0.6313

       'Positive' Class : Completed

> cat("\nPrecision:", sprintf("%.4f", ann_cm$byClass['Pos Pred Value']), "\n")

Precision: 0.6313
```

Metric	Value
Accuracy	63.1%
Sensitivity	63.2%
Specificity	63.1%
Precision	63.1%
Balanced Accuracy	63.1%

Interpretation:

- Performance comparable to SVM and Decision Tree.
- While ANN can capture complex interactions, it requires more computations and is less interpretable.

Final Comparison of all the models using downsampling:

Model	Accuracy	Sensitivity	Specificity	Precision	Balanced Accuracy
Decision Tree	64.6%	62.1%	67.1%	65.4%	64.6%
kNN	55.8%	56.2%	55.4%	55.8%	55.8%
Naive Bayes	54.8%	95.1%	14.4%	52.6%	54.7%
SVM	63.8%	62.7%	64.9%	64.1%	63.8%
ANN	63.1%	63.2%	63.1%	63.1%	63.1%

- Decision Trees presents the best balance between interpretability and predictive performance.
- SVM and ANN are close in performance but less interpretable.
- Naive Bayes favours sensitivity at the cost of specificity, therefore over-predicting completions.
- KNN performs badly due to high-dimensional dummy variables.

Recommendation for TFA:

- Deploy the model as either Decision Tree or SVM for operational usage.
- Concentrate on features that include application timing, quick starter/completer flags, and essay sentiment to reduce withdrawals.

Up sampling Analysis

Objective:

Compare the performance of models when the minority class (Withdrew) is balanced with upsampling against the downsampling approach from the previous solution.

Methodology:

- All majority class records were kept – Completed.
- The minority class has been up sampled to match the number of majority classes.
- Models were retrained on the same features, preprocessing steps, and 5-fold cross-validation as in the down sampling experiment**.**
- To handle the class imbalance, up sampling was done: all records of the minority class (Withdrew) were kept and more samples were generated from the minority class to match the majority class (Completed). This will ensure that learning is fair between the two classes and enhances the detection of the withdrawals.

Results:

Decision Tree:

```
--- DECISION TREE RESULTS ---
> print(dt_cm)
Confusion Matrix and Statistics

          Reference
Prediction  Withdrew Completed
  Withdrew     11763      1104
  Completed      322     10981

               Accuracy : 0.941
                 95% CI : (0.938, 0.9439)
    No Information Rate : 0.5
    P-Value [Acc > NIR] : < 2.2e-16

                  Kappa : 0.882

 Mcnemar's Test P-Value : < 2.2e-16

            Sensitivity : 0.9086
            Specificity : 0.9734
         Pos Pred Value : 0.9715
         Neg Pred Value : 0.9142
             Prevalence : 0.5000
         Detection Rate : 0.4543
   Detection Prevalence : 0.4676
      Balanced Accuracy : 0.9410

       'Positive' Class : Completed
```

Decision Trees split the data into branches based on feature values, handling both numeric and categorical data. They are interpretable and capture non-linear relationships.

Interpretation:

- High sensitivity: 90.9% correctly identified completed applicants.

- High specificity: 97.3% correctly classifies almost all withdrawals.
- Balanced performance: High accuracy with balanced accuracy shows that the model performed effectively in distinguishing both classes.
- Operational insight: Provides interpretable rules that are actionable in targeting applicants and managing resources efficiently.

KNN:

- The original TFA dataset had more applicants with a status of "Completed" than "Withdrawn", so it created an imbalance in classes.
- Up sampling duplicates or synthetically generates minority cases of Withdrawn to balance the dataset.
- This ensures that in the feature space, each applicant's neighborhood has equal representation of both classes, reducing bias toward the majority class.

Interpretation:

```
--- k-NEAREST NEIGHBORS RESULTS ---
> print(knn_cm)
Confusion Matrix and Statistics

          Reference
Prediction  Withdrew Completed
  Withdrew     11157      5002
  Completed      928      7083

               Accuracy : 0.7547
                 95% CI : (0.7492, 0.7601)
    No Information Rate : 0.5
    P-Value [Acc > NIR] : < 2.2e-16

                  Kappa : 0.5093

 Mcnemar's Test P-Value : < 2.2e-16

            Sensitivity : 0.5861
            Specificity : 0.9232
         Pos Pred Value : 0.8842
         Neg Pred Value : 0.6905
             Prevalence : 0.5000
         Detection Rate : 0.2930
   Detection Prevalence : 0.3314
      Balanced Accuracy : 0.7547

       'Positive' Class : Completed
```

- High Specificity (92.3%): sensitive to missing withdrawals.
- Poor sensitivity, 58.6%: Many completed applicants are misclassified.
- Balanced Accuracy (Moderate 75.5%): Generally decent overall performance but slightly favors withdrawal detection.

- Operational insight: Useful when the priority is in detecting potential withdrawals, though less reliable for identifying applicants likely to complete.

Naive bayes:

```
--- NAIVE BAYES RESULTS ---
> print(nb_cm)
Confusion Matrix and Statistics

          Reference
Prediction  Withdrew Completed
  Withdrew        39         3
  Completed    12046     12082

               Accuracy : 0.5015
                 95% CI : (0.4952, 0.5078)
    No Information Rate : 0.5
    P-Value [Acc > NIR] : 0.3239

                  Kappa : 0.003

 Mcnemar's Test P-Value : <2e-16

            Sensitivity : 0.999752
            Specificity : 0.003227
         Pos Pred Value : 0.500746
         Neg Pred Value : 0.928571
             Prevalence : 0.500000
         Detection Rate : 0.499876
   Detection Prevalence : 0.998262
      Balanced Accuracy : 0.501489

       'Positive' Class : Completed
```

Interpretation:

- Extremely high sensitivity of 99.9%: Almost all completed applicants are detected.
- Extremely low specificity (0.3%): almost no withdrawals are correctly predicted.
- Accuracy close to 50%: Overall performance is near random.
- Operational insight: Overpredicts completion, failing to distinguish withdrawals; hence, it is not reliable for practical decision-making.

ANN:

```
--- ARTIFICIAL NEURAL NETWORK RESULTS ---
> print(ann_cm)
Confusion Matrix and Statistics

          Reference
Prediction  Withdrew Completed
  Withdrew      8782      4398
  Completed     3303      7687

               Accuracy : 0.6814
                 95% CI : (0.6755, 0.6873)
    No Information Rate : 0.5
    P-Value [Acc > NIR] : < 2.2e-16

                  Kappa : 0.3628

 Mcnemar's Test P-Value : < 2.2e-16

            Sensitivity : 0.6361
            Specificity : 0.7267
         Pos Pred Value : 0.6995
         Neg Pred Value : 0.6663
             Prevalence : 0.5000
         Detection Rate : 0.3180
   Detection Prevalence : 0.4547
      Balanced Accuracy : 0.6814

       'Positive' Class : Completed
```

Interpretation:

- Moderate sensitivity 63.6%, specificity 72.6%: Both completed and withdrawn applicants are represented better than by SVM but not as well as by Decision Tree.
- Accuracy: 68.1%, Balanced accuracy: 68.1%. It shows an overall improvement over down sampling.
- Operational insight: ANN can model complex non-linear interactions but requires more computation and is less interpretable than Decision Tree.

SVM:

```
> confusionMatrix(predict(svm_mod, test_data), test_data$Completed.Admissions.Process, positive="1")
Confusion Matrix and Statistics

          Reference
Prediction    0    1
         0 1837 1111
         1 1045 1771

               Accuracy : 0.626
                 95% CI : (0.6133, 0.6385)
    No Information Rate : 0.5
    P-Value [Acc > NIR] : <2e-16

                  Kappa : 0.2519

 Mcnemar's Test P-Value : 0.1616

            Sensitivity : 0.6145
            Specificity : 0.6374
         Pos Pred Value : 0.6289
         Neg Pred Value : 0.6231
             Prevalence : 0.5000
         Detection Rate : 0.3073
   Detection Prevalence : 0.4885
      Balanced Accuracy : 0.6260

       'Positive' Class : 1
```

- The sensitivity and specificity are balanced at about 61–64%, which implies a moderate performance for both the completed and withdrawn applicants.
- Accuracy and balanced accuracy (~62–63%): Stable but not high-performing.
- Operational insight: Works reasonably well with numerical features but performs less well compared to Decision Tree or ANN when data is highly imbalanced or heavy with categorical data.

Up sampling - Key Observations Across Models:

Model	Accuracy	Sensitivity (Recall)	Specificity	Precision (PPV)	Balanced Accuracy	Kappa
Decision Tree	94.1	90.9%	97.3%	97.1%	94.1%	0.882
kNN	75.5%	58.6%	92.3%	88.4%	75.5%	0.509
Naive Bayes	50.2%	99.9%	0.3%	50.1%	50.1%	0.003
SVM	62.6%	61.5%	63.7%	62.9%	62.6%	0.252
ANN	68.1%	63.6%	72.6%	70%	68.1%	0.363

- Upsampling improved performance across all models, most markedly for the minority class - withdrawals.

- Decision Tree yields the highest accuracy, sensitivity, specificity, and balanced accuracy, performing well and with great interpretability.
- kNN favors withdrawals, enhancing the specificity with a cost in sensitivity.
- Naive Bayes overpredicts completions, remaining operationally ineffective.
- Stable but moderate performance by SVM, with minor improvements over downsampling.
- ANN improves over downsampling, capturing complex patterns but less interpretable.

Final Model Comparison:

Model	Sampling	Accuracy	Sensitivity (Completed)	Specificity (Withdrawal)	Precision	Balanced Accuracy
Decision Tree	Downsampling	64.6%	62.1%	67.1%	65.4%	64.6%
kNN	Downsampling	55.8%	56.2%	55.4%	55.8%	55.8%
Naive Bayes	Downsampling	54.8%	95.1%	14.4%	52.6%	54.7%
SVM	Downsampling	63.8%	62.7%	64.9%	64.1%	63.8%
ANN	Downsampling	63.1%	63.2%	63.1%	63.1%	63.1%
Decision Tree	Upsampling	94.1	90.9%	97.3%	97.1%	94.1%
kNN	Up sampling	75.5%	58.6%	92.3%	88.4%	75.5%
Naive Bayes	Up sampling	50.2%	99.9%	0.3%	50.1%	50.1%
SVM	Upsampling	62.6%	61.5%	63.7%	62.9%	62.6%
ANN	Upsampling	68.1%	63.6%	72.6%	70%	68.1%

Model Comparison (Downsampling vs Upsampling)

Overall Trends

- Up sampling improves model performance consistently throughout almost all of the metrics compared to down sampling.
- It balances the classes between Completed vs Withdrawn cases, and this permits the models to predict better on the minority cases, which in this case are the withdrawals, crucial for any operational decisions.
- Down sampling realizes only moderate performance at the cost of sacrificing data from the majority class, thus seriously limiting predictive accuracy.

Important Gains from Comparative Analysis

Up sampling outperforms down sampling:

- In general, up sampling results in higher accuracy, sensitivity, and specificity, and balanced accuracy in models.
- Decision Tree benefits most, with balanced accuracy going up from 64.6% → 94.1%.

Best Performing Model:

The strongest is obviously a C5.0 Decision Tree with up sampling:

- Highly sensitive, 90.9% will highlight most completed applicants.
- High specificity, 97.3% → it accurately detects withdrawals.
- High balanced accuracy of 94.1%: It handles class imbalance well.
- Interpretable rules for operational decision-making.

Other Models:

- ANN upsampling improves to a reasonable level, with a balanced accuracy of 68.1%, but is heavier computationally and less interpretable.
- kNN performs well for the upsampling of withdrawals, with high specificity (92.3%), but low sensitivity limits its use for predicting completions.
- SVM is stable but underperforms as compared to Decision Tree and ANN.
- Naive Bayes overpredicts completions with very low specificity; unsuitable despite upsampling.

Operational Takeaways:

- Decision Tree-upsampling is ideal in targeting resources efficiently, predicting completed applicants, and managing withdrawals.
- ANN may be used for an exploratory analysis of complex feature interactions.
- kNN and SVM are complementary tools, but not advisable for use as primary models.
- Naive Bayes has poor withdrawal detection and thus can be avoided.

Overall Conclusion:

- Upsampling clearly improves predictive performance across all models.
- Decision Tree with upsampling represents the preferable model for performing TFA predictive analytics because of its high accuracy, balanced sensitivity and specificity, and interpretability.
- ANN and kNN can serve as secondary or supplementary models depending on operational priorities.
- Naive Bayes suffers from extreme overprediction bias and hence cannot be used practically.

Appendix:

```
# Load Libraries
library(readxl)
```

```
library(dplyr)
library(lubridate)
library(caret)
library(C50)
library(partykit)
library(class)
library(e1071)
library(kernlab)
library(nnet)
library(NeuralNetTools)
library(fastDummies)

set.seed(1947)

# Load data
cat("STEP 1: Loading Data...\n")
setwd("C:/Users/Nantika/Downloads/R data files")
tfa_data <- read_xlsx("Group Assignment R.xlsx")
cat("Data loaded:", nrow(tfa_data), "rows\n\n")

# CREATE BALANCED SAMPLE

cat("STEP 2: Creating Balanced Sample (All Minority + Equal Majority)...\n")
tfa_data <- tfa_data %>% filter(!is.na(`Completed Admissions Process`))

completed <- tfa_data[tfa_data$`Completed Admissions Process` == 1, ]
withdrew <- tfa_data[tfa_data$`Completed Admissions Process` == 0, ]

cat("Original distribution:\n")
cat("  Completed (1):", nrow(completed), "\n")
cat("  Withdrew (0):", nrow(withdrew), "\n\n")

# Identify minority and majority classes
if (nrow(completed) < nrow(withdrew)) {
  minority_class <- completed
  majority_class <- withdrew
  minority_label <- "Completed"
  majority_label <- "Withdrew"
} else {
  minority_class <- withdrew
  majority_class <- completed
  minority_label <- "Withdrew"
  majority_label <- "Completed"
}


cat("Minority class:", minority_label, "with", nrow(minority_class), "samples\n")
cat("Majority class:", majority_label, "with", nrow(majority_class), "samples\n\n")

# Take ALL minority class samples
n_minority <- nrow(minority_class)

# Sample EQUAL number from majority class
set.seed(1947)
sample_majority <- majority_class[sample(nrow(majority_class), n_minority), ]

# Combine datasets
tfa_data <- rbind(minority_class, sample_majority)
tfa_data <- tfa_data[sample(nrow(tfa_data)), ]  # Shuffle
```

```
cat("Balanced sample created:\n")
cat("  Total rows:", nrow(tfa_data), "\n")
cat("  Completed (1):", sum(tfa_data$`Completed Admissions Process` == 1), "\n")
cat("  Withdrew (0):", sum(tfa_data$`Completed Admissions Process` == 0), "\n\n")

# FEATURE ENGINEERING
cat("STEP 3: Feature Engineering...\n")

date_cols <- c("Sign-up Date", "Started Date", "Application Deadline", "Submitted Date")
tfa_data[date_cols] <- lapply(tfa_data[date_cols], ymd)

tfa_data <- tfa_data %>%
  mutate(
    days_signup_to_start = as.numeric(difftime(`Started Date`, `Sign-up Date`, units = "days")),
    days_start_to_submit = as.numeric(difftime(`Submitted Date`, `Started Date`, units = "days")),
    days_signup_to_submit = as.numeric(difftime(`Submitted Date`, `Sign-up Date`, units = "days")),
    days_before_deadline = as.numeric(difftime(`Application Deadline`, `Submitted Date`, units =
"days")),
    signup_month = month(`Sign-up Date`),
    signup_year = year(`Sign-up Date`),
    submit_month = month(`Submitted Date`),
    submitted_early = ifelse(days_before_deadline > 7, 1, 0),
    quick_starter = ifelse(days_signup_to_start <= 1, 1, 0),
    quick_completer = ifelse(days_start_to_submit <= 7, 1, 0)
  )

# Convert Essays Sentiment to binary factor
tfa_data$`Essays Sentiment` <- ifelse(tfa_data$`Essays Sentiment` >= 0, 0,
                                      ifelse(tfa_data$`Essays Sentiment` < 0, 1,
                                             tfa_data$`Essays Sentiment`))
tfa_data$`Essays Sentiment` <- factor(tfa_data$`Essays Sentiment`,
                                      levels = c(0, 1),
                                      labels = c("Positive", "Negative"))

tfa_data <- tfa_data %>% select(-all_of(date_cols))
cat("Feature engineering complete\n\n")

# ==================== PREPROCESSING ==================== #
cat("STEP 4: Preprocessing...\n")

high_card_cols <- c("Major 1", "Major 2", "Minor", "Undergraduate University")
for (col in high_card_cols) {
  if (col %in% names(tfa_data)) {
    freq_table <- sort(table(tfa_data[[col]]), decreasing = TRUE)
    top_categories <- names(freq_table)[1:min(20, length(freq_table))]
    tfa_data[[col]] <- ifelse(tfa_data[[col]] %in% top_categories,
                              tfa_data[[col]], "Other")
  }
}

tfa_data$`Person Id` <- NULL
```

```
num_cols <- names(tfa_data)[sapply(tfa_data, is.numeric)]
for (col in num_cols) {
  if (sum(is.na(tfa_data[[col]])) > 0) {
    tfa_data[[col]][is.na(tfa_data[[col]])] <- median(tfa_data[[col]], na.rm = TRUE)
  }
}

cat_cols <- names(tfa_data)[sapply(tfa_data, is.character)]
for (col in cat_cols) {
  if (sum(is.na(tfa_data[[col]])) > 0) {
    mode_val <- names(sort(table(tfa_data[[col]]), decreasing = TRUE))[1]
    tfa_data[[col]][is.na(tfa_data[[col]])] <- mode_val
  }
}

factor_cols <- c(
  "App Started Year (RTAT)",
  "Is Math, Sci, or Eng Major Minor",
  "Major 1",
  "Major 2",
  "Minor",
  "Undergraduate University",
  "School Selectivity",
  "Region Preference Level",
  "Attended Event",
  "Essays Sentiment",
  "submitted_early",
  "quick_starter",
  "quick_completer"
)

# Create dataset for tree-based models (keeps factors)
data_tree <- tfa_data
data_tree[factor_cols] <- lapply(data_tree[factor_cols], as.factor)
data_tree$`Completed Admissions Process` <- factor(
  data_tree$`Completed Admissions Process`,
  levels = c(0, 1),
  labels = c("Withdrew", "Completed")
)

# Create dataset for numeric models (dummy variables)
data_numeric <- dummy_cols(
  tfa_data,
  select_columns = factor_cols,
  remove_first_dummy = TRUE,
  remove_selected_columns = TRUE
)
data_numeric$`Completed Admissions Process` <- factor(
  data_numeric$`Completed Admissions Process`,
  levels = c(0, 1),
  labels = c("Withdrew", "Completed")
)
```

```
# Remove near-zero variance
nzv <- nearZeroVar(data_numeric, saveMetrics = FALSE)
if (length(nzv) > 0) {
  data_numeric <- data_numeric[, -nzv]
}

cat("Preprocessing complete\n\n")

# TRAIN-TEST SPLIT
train_idx_tree <- createDataPartition(data_tree$`Completed Admissions Process`,
                                      p = 0.8, list = FALSE)
train_tree <- data_tree[train_idx_tree, ]
test_tree <- data_tree[-train_idx_tree, ]

train_idx_num <- createDataPartition(data_numeric$`Completed Admissions Process`,
                                     p = 0.8, list = FALSE)
train_numeric <- data_numeric[train_idx_num, ]
test_numeric <- data_numeric[-train_idx_num, ]

# Separate X and y for kNN
train_x <- train_numeric %>% select(-`Completed Admissions Process`)
test_x <- test_numeric %>% select(-`Completed Admissions Process`)
train_y <- train_numeric$`Completed Admissions Process`
test_y <- test_numeric$`Completed Admissions Process`

cat("Train:", nrow(train_tree), "| Test:", nrow(test_tree), "\n\n")

# TRAINING CONTROL
ctrl <- trainControl(
  method = "cv",
  number = 5,
  classProbs = TRUE,
  summaryFunction = twoClassSummary
)

# MODEL 1: DECISION TREE
dt_model <- train(
  `Completed Admissions Process` ~ .,
  data = train_tree,
  method = "C5.0",
  trControl = ctrl,
  tuneGrid = expand.grid(
    model = "tree",
    winnow = c(TRUE, FALSE),
    trials = c(1, 10, 20, 30)
  ),
  metric = "ROC"
)

print(dt_model)
cat("\nBest Parameters:\n")
```

```
print(dt_model$bestTune)

dt_pred <- predict(dt_model, test_tree)
dt_cm <- confusionMatrix(dt_pred, test_tree$`Completed Admissions Process`,
                         positive = "Completed")

cat("\n--- DECISION TREE RESULTS ---\n")
print(dt_cm)
cat("\nPrecision:", sprintf("%.4f", dt_cm$byClass['Pos Pred Value']), "\n")

# MODEL 2: k-NEAREST NEIGHBORS
k_values <- c(3, 5, 7, 11)
best_k <- 5
best_f1 <- 0

for (k in k_values) {
  pred <- knn(train_x, test_x, cl = train_y, k = k)
  cm_temp <- confusionMatrix(pred, test_y, positive = "Completed")
  if (cm_temp$byClass['F1'] > best_f1) {
    best_f1 <- cm_temp$byClass['F1']
    best_k <- k
  }
}

cat("Best k:", best_k, "\n\n")

knn_pred <- knn(train_x, test_x, cl = train_y, k = best_k)
knn_cm <- confusionMatrix(knn_pred, test_y, positive = "Completed")

cat("\n--- k-NEAREST NEIGHBORS RESULTS ---\n")
print(knn_cm)
cat("\nPrecision:", sprintf("%.4f", knn_cm$byClass['Pos Pred Value']), "\n")


# MODEL 3: NAIVE BAYES

nb_model <- train(
  `Completed Admissions Process` ~ .,
  data = train_tree,
  method = "naive_bayes",
  trControl = ctrl,
  tuneGrid = data.frame(
    laplace = c(0, 1, 2),
    usekernel = TRUE,
    adjust = 1
  ),
  metric = "ROC"
)

print(nb_model)

nb_pred <- predict(nb_model, test_tree)
nb_cm <- confusionMatrix(nb_pred, test_tree$`Completed Admissions Process`,
```

```
                    positive = "Completed")

cat("\n--- NAIVE BAYES RESULTS ---\n")
print(nb_cm)
cat("\nPrecision:", sprintf("%.4f", nb_cm$byClass['Pos Pred Value']), "\n")

#  MODEL 4: SUPPORT VECTOR
svm_model <- train(
  `Completed Admissions Process` ~ .,
  data = train_numeric,
  method = "svmRadial",
  trControl = ctrl,
  preProcess = c("center", "scale"),
  tuneLength = 3,
  metric = "ROC"
)

print(svm_model)
svm_pred <- predict(svm_model, test_numeric)
svm_cm <- confusionMatrix(svm_pred, test_numeric$`Completed Admissions Process`,
                    positive = "Completed")

cat("\n--- SUPPORT VECTOR MACHINE RESULTS ---\n")
print(svm_cm)
cat("\nPrecision:", sprintf("%.4f", svm_cm$byClass['Pos Pred Value']), "\n")

# MODEL 5: ARTIFICIAL NEURAL NETWORK

ann_model <- train(
  `Completed Admissions Process` ~ .,
  data = train_numeric,
  method = "nnet",
  trControl = ctrl,
  preProcess = c("center", "scale"),
  tuneGrid = expand.grid(
    size = c(5, 10),
    decay = c(0.1, 0.3)
  ),
  metric = "ROC",
  maxit = 200,
  trace = FALSE
)

print(ann_model)

ann_pred <- predict(ann_model, test_numeric)
ann_cm <- confusionMatrix(ann_pred, test_numeric$`Completed Admissions Process`,
                    positive = "Completed")

cat("\n--- ARTIFICIAL NEURAL NETWORK RESULTS ---\n")
print(ann_cm)
cat("\nPrecision:", sprintf("%.4f", ann_cm$byClass['Pos Pred Value']), "\n")
```

Up sampling code:

```
# TEACH FOR AMERICA PREDICTIVE ANALYTICS
# Predictive Modeling Workflow (Upsampling)

# Load Libraries
library(readxl)
library(dplyr)
library(lubridate)
library(caret)
library(C50)
library(partykit)
library(class)
library(e1071)
library(kernlab)
library(nnet)
library(NeuralNetTools)
library(fastDummies)

set.seed(1947)

# Load Data
tfa_data <- read_excel("D:/Illinois Tech/Predictive Analytics - DJ/Project/Assignment_Dataset.xlsx")
tfa_data <- tfa_data %>% filter(!is.na(`Completed Admissions Process`))

completed <- tfa_data[tfa_data$`Completed Admissions Process` == 1, ]
withdrew <- tfa_data[tfa_data$`Completed Admissions Process` == 0, ]

cat("Original distribution:\n")
cat(" Completed (1):", nrow(completed), "\n")
cat(" Withdrew (0):", nrow(withdrew), "\n\n")

# Identify Minority and Majority Classes
if (nrow(completed) < nrow(withdrew)) {
  minority_class <- completed
  majority_class <- withdrew
  minority_label <- "Completed"
  majority_label <- "Withdrew"
} else {
  minority_class <- withdrew
  majority_class <- completed
  minority_label <- "Withdrew"
  majority_label <- "Completed"
}

cat("Minority class:", minority_label, "with", nrow(minority_class), "samples\n")
cat("Majority class:", majority_label, "with", nrow(majority_class), "samples\n\n")

# Upsample Minority Class
n_majority <- nrow(majority_class)
```

```
n_minority <- nrow(minority_class)

upsampled_minority <- minority_class[sample(nrow(minority_class), n_majority, replace = TRUE), ]

# Combine datasets and shuffle
tfa_data <- rbind(majority_class, upsampled_minority)
tfa_data <- tfa_data[sample(nrow(tfa_data)), ]

cat("Upsampled dataset created:\n")
cat(" Total rows:", nrow(tfa_data), "\n")
cat(" Completed (1):", sum(tfa_data$`Completed Admissions Process` == 1), "\n")
cat(" Withdrew (0):", sum(tfa_data$`Completed Admissions Process` == 0), "\n\n")

# Feature Engineering
date_cols <- c("Sign-up Date", "Started Date", "Application Deadline", "Submitted Date")
tfa_data[date_cols] <- lapply(tfa_data[date_cols], ymd)

tfa_data <- tfa_data %>%
  mutate(
    days_signup_to_start = as.numeric(difftime(`Started Date`, `Sign-up Date`, units = "days")),
    days_start_to_submit = as.numeric(difftime(`Submitted Date`, `Started Date`, units = "days")),
    days_signup_to_submit = as.numeric(difftime(`Submitted Date`, `Sign-up Date`, units = "days")),
    days_before_deadline = as.numeric(difftime(`Application Deadline`, `Submitted Date`, units =
"days")),
    signup_month = month(`Sign-up Date`),
    signup_year = year(`Sign-up Date`),
    submit_month = month(`Submitted Date`),
    submitted_early = ifelse(days_before_deadline > 7, 1, 0),
    quick_starter = ifelse(days_signup_to_start <= 1, 1, 0),
    quick_completer = ifelse(days_start_to_submit <= 7, 1, 0)
  )

# Convert Essays Sentiment to binary factor
tfa_data$`Essays Sentiment` <- ifelse(tfa_data$`Essays Sentiment` >= 0, 0, 1)
tfa_data$`Essays Sentiment` <- factor(tfa_data$`Essays Sentiment`, levels = c(0,1), labels =
c("Positive","Negative"))

# Remove original date columns
tfa_data <- tfa_data %>% select(-all_of(date_cols))

cat("Feature engineering complete\n\n")

# Preprocessing
high_card_cols <- c("Major 1", "Major 2", "Minor", "Undergraduate University")

# Reduce high-cardinality categories
for (col in high_card_cols) {
  if (col %in% names(tfa_data)) {
    freq_table <- sort(table(tfa_data[[col]]), decreasing = TRUE)
    top_categories <- names(freq_table)[1:min(20, length(freq_table))]
    tfa_data[[col]] <- ifelse(tfa_data[[col]] %in% top_categories, tfa_data[[col]], "Other")
  }
```

```
}

tfa_data$`Person Id` <- NULL

# Handle numeric missing values
num_cols <- names(tfa_data)[sapply(tfa_data, is.numeric)]
for (col in num_cols) {
  if (sum(is.na(tfa_data[[col]])) > 0) {
    tfa_data[[col]][is.na(tfa_data[[col]])] <- median(tfa_data[[col]], na.rm = TRUE)
  }
}

# Handle categorical missing values
cat_cols <- names(tfa_data)[sapply(tfa_data, is.character)]
for (col in cat_cols) {
  if (sum(is.na(tfa_data[[col]])) > 0) {
    mode_val <- names(sort(table(tfa_data[[col]]), decreasing = TRUE))[1]
    tfa_data[[col]][is.na(tfa_data[[col]])] <- mode_val
  }
}

# Factor conversion
factor_cols <- c("App Started Year (RTAT)", "Is Math, Sci, or Eng Major Minor", "Major 1", "Major 2",
                 "Minor", "Undergraduate University", "School Selectivity", "Region Preference Level",
                 "Attended Event", "Essays Sentiment", "submitted_early", "quick_starter",
"quick_completer")

# Create Datasets
data_tree <- tfa_data
data_tree[factor_cols] <- lapply(data_tree[factor_cols], as.factor)
data_tree$`Completed Admissions Process` <- factor(data_tree$`Completed Admissions Process`,
                                        levels = c(0,1), labels = c("Withdrew","Completed"))

data_numeric <- dummy_cols(tfa_data, select_columns = factor_cols, remove_first_dummy = TRUE,
remove_selected_columns = TRUE)
data_numeric$`Completed Admissions Process` <- factor(data_numeric$`Completed Admissions
Process`,
                                          levels = c(0,1), labels = c("Withdrew","Completed"))

# Remove near-zero variance predictors
nzv <- nearZeroVar(data_numeric, saveMetrics = FALSE)
if (length(nzv) > 0) { data_numeric <- data_numeric[, -nzv] }

cat("Preprocessing complete\n\n")

# Train-Test Split
train_idx_tree <- createDataPartition(data_tree$`Completed Admissions Process`, p = 0.8, list =
FALSE)
train_tree <- data_tree[train_idx_tree, ]
test_tree <- data_tree[-train_idx_tree, ]
```

```
train_idx_num <- createDataPartition(data_numeric$`Completed Admissions Process`, p = 0.8, list =
FALSE)
train_numeric <- data_numeric[train_idx_num, ]
test_numeric <- data_numeric[-train_idx_num, ]

train_x <- train_numeric %>% select(-`Completed Admissions Process`)
train_y <- train_numeric$`Completed Admissions Process`
test_x <- test_numeric %>% select(-`Completed Admissions Process`)
test_y <- test_numeric$`Completed Admissions Process`

cat("Train:", nrow(train_tree), "| Test:", nrow(test_tree), "\n\n")

# Training Control
ctrl <- trainControl(method = "cv", number = 5, classProbs = TRUE, summaryFunction =
twoClassSummary)

# MODEL 1: Decision Tree (C5.0)
dt_model <- train(`Completed Admissions Process` ~ ., data = train_tree,
                  method = "C5.0",
                  trControl = ctrl,
                  tuneGrid = expand.grid(model = "tree", winnow = c(TRUE,FALSE), trials = c(1,10,20,30)),
                  metric = "ROC")
print(dt_model)
cat("\nBest Parameters:\n")
print(dt_model$bestTune)

dt_pred <- predict(dt_model, test_tree)
dt_cm <- confusionMatrix(dt_pred, test_tree$`Completed Admissions Process`, positive =
"Completed")
print(dt_cm)

# MODEL 2: k-Nearest Neighbors (kNN)
k_values <- c(3,5,7,11)
best_k <- 5
best_f1 <- 0

for (k in k_values) {
  pred <- knn(train_x, test_x, cl = train_y, k = k)
  cm_temp <- confusionMatrix(pred, test_y, positive = "Completed")
  if (cm_temp$byClass['F1'] > best_f1) {
    best_f1 <- cm_temp$byClass['F1']
    best_k <- k
  }
}

cat("Best k:", best_k, "\n\n")
knn_pred <- knn(train_x, test_x, cl = train_y, k = best_k)
knn_cm <- confusionMatrix(knn_pred, test_y, positive = "Completed")
print(knn_cm)

# MODEL 3: Naive Bayes
nb_model <- train(`Completed Admissions Process` ~ ., data = train_tree,
```

```
                method = "naive_bayes",
                trControl = ctrl,
                tuneGrid = data.frame(laplace = c(0,1,2), usekernel = TRUE, adjust = 1),
                metric = "ROC")
print(nb_model)

nb_pred <- predict(nb_model, test_tree)
nb_cm <- confusionMatrix(nb_pred, test_tree$`Completed Admissions Process`, positive =
"Completed")
print(nb_cm)

# MODEL 4: Support Vector Machine (SVM)
svm_model <- train(`Completed Admissions Process` ~ ., data = train_numeric,
                method = "svmRadial",
                trControl = ctrl,
                preProcess = c("center","scale"),
                tuneLength = 3,
                metric = "ROC")
print(svm_model)

svm_pred <- predict(svm_model, test_numeric)
svm_cm <- confusionMatrix(svm_pred, test_numeric$`Completed Admissions Process`, positive =
"Completed")
print(svm_cm)

# MODEL 5: Artificial Neural Network (ANN)
ann_model <- train(`Completed Admissions Process` ~ ., data = train_numeric,
                method = "nnet",
                trControl = ctrl,
                preProcess = c("center","scale"),
                tuneGrid = expand.grid(size = c(5,10), decay = c(0.1,0.3)),
                metric = "ROC",
                maxit = 200,
                trace = FALSE)
print(ann_model)

ann_pred <- predict(ann_model, test_numeric)
ann_cm <- confusionMatrix(ann_pred, test_numeric$`Completed Admissions Process`, positive =
"Completed")
print(ann_cm)
```