Classifying Instagram Account

By: An Tran & Meggy Le

National University

Table of Contents

List of Tables

List of Figures

Abstract

The advancement of technology in these past decades has brought forth the growth of many different social media platforms. The most established definition of social media to the population is the applications that are installed on smartphones and tablets. Although social media are efficient platforms where people can obtain and share information quickly, this is also a place where that information could be jeopardized and manipulated via fraudulent activities. The goal of this study is to utilize the collected features of multiple Instagram accounts, build models that can predict whether an account is real or fake by applying various classification methodologies and in a way help combating fraudulent activities on the internet.

Chapter 1: Introduction

Instagram is one of the most popular social media platforms around the world. This application is installed on almost every teenager and young adult's smartphone. Through Instagram, they can quickly connect with other people whether it is a friend, an acquaintance or a complete stranger, and share photos of their life and interests. There are hackers out there who rely on the convenience of these social media platforms to illegally obtain and exploit other people's information. Everyone has had at least once experienced or know someone whose account had been taken over by a hacker. These hackers would post photos and links that when clicked on, could cause further damaged to your device via viral invasion and potentially expose your identity to the dark webs.

When you receive a friend request on Instagram, there are cases where you can immediately tell if it is someone you know or not. There are also cases where there is not enough information to determine if the request is from a friend or a stranger or possibly a hacker using a fake account. For example, it could be that the person does not have a profile picture or the picture they have is not clear, it could be that the person only displays their first name or do not show their name at all, etc. The more careful each friend request is examined, the less risk you are exposed to potentially spammer and faker.

With the rise of cyber fraudulent activities, not only users but the administrations of these social media applications should also involve to stop the manifestations of this disaster. Instagram administrators could proactively scan all of the accounts to detect any unusualness. However, fake accounts do not always transparently display evidences that could make them easily detectable by users and administrators. Experienced hackers could utilize the same known knowledge to enhance their fake accounts and prevent them from being identified. The question is how do we prevent this from happening while maintaining the utilization of these social media applications?

The goal of this study is to develop models that can predict whether an Instagram account is real or fake using multiple features that describe each account. Various classification algorithms will be utilized to build different models and compare the performance of each model among one another. Hopefully, these models could help the population successfully identify fake Instagram accounts and in the long run, they could help improve the safety of social media platforms. One thing to keep in mind is that as there is not sufficient data on how to distinguish those well masked fake accounts from the genuine ones, this could be one of the many setbacks that could potentially affect the performance of our models.

Chapter 2: Literature Review

Instagram is a phenomenon among teenagers and young adults. The social media platform attracts millions of users. However, it is also a playground for hackers due to its popularity and high usage. It is a rule of thumb to always be careful while using internet as your information could be jeopardized at any time without your permission and knowledge. This study aims to focus on how to improve the safety of Instagram by creating models that could predict a genuine account from a fake one and thus could help prevent users from being attacked by hackers. Although the rise of internet fraudulent activities is well known to the public, there is not much research done on this topic specifically regarding Instagram. Hopefully, the study could yield effective results and become a pioneer for future researches on this topic as well as other social media platforms.

Chapter 3: Methodology

The data set that will be used for this study contains features and label of multiple Instagram accounts and was obtained from Kaggle, a website that supplies numerous data sets for different topics and purposes. This data set also conveniently came with two divided subsets, training set and test set. The training set will be used to build and tune the models whereas the test set will be used to evaluate the performance of the models. Splitting the data set into training and test sets is a technique that helps achieve the best tuning parameters that would improve the performance of the model (Kuhn & Johnson, 2013). The variables included in this data set is summarized in table 1.

Table 3.1 Instagram data set variables

Variables	**Description**
profile pic	user has profile picture or not
nums/length username	ratio of number of numerical chars in username to its length
fullname words	full name in word tokens
nums/length fullname	ratio of number of numerical characters in full name to its length
name==username	are username and full name literally the same
description length	bio length in characters
external URL	has external URL or not
private	private or not
#posts	number of posts
#followers	number of followers
#follows	number of follows
fake	class (0 genuine, 1 fake)

Fake will be the outcome variable in this study. The rest of the variables will be the predictors for the models. The outcome variable is a binary variable that takes on two classes, 0 - a genuine account and 1- a fake account. This data set will be trained using three different algorithms: linear regression, non-linear regression and tree based.

3.0 Data Preparation

3.0.1 Zero/Near Zero variance predictors

Data sets sometimes come with predictors that consist of a single unique value across all samples (Zero-Variance). This kind of predictors is not only non-informative, they may also cause some models to crash or the fit to be unstable (Kuhn & Johnson, 2013). Similarly, there are predictors that have the number of unique values occurring with very low frequency (Near Zero-Variance). Such predictors are very common among data; and the data set used for this project is not an exception. These predictors may be excluded from the data set before the model training process.

In this case, "name==username" and "nums/length fullname**"** are considered as near zero-variance predictors since the number of unique values in each variable occurs with very low frequency as indicated in Figure 3.1. These two variables can be excluded from the dataset during the model training process.

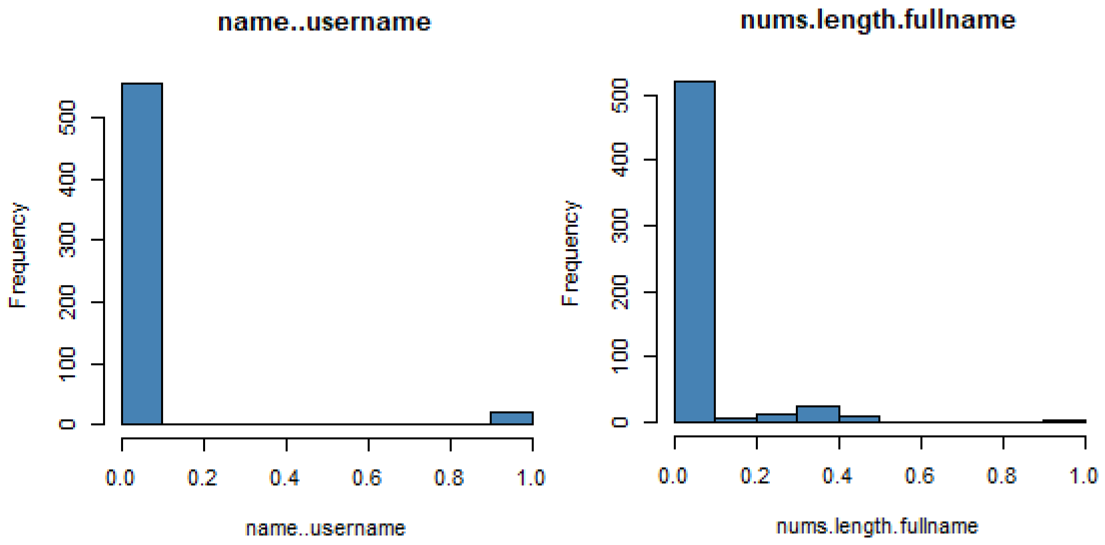

Figure 3.1: Zero Variance Variable Distributions

3.0.2 Multi-collinearity

Multi-collinearity occurs when predictors are intercorrelated instead of being independent. In other words, it refers to predictors that are correlated with other predictors in a model. Moderate multicollinearity might not be problematic; however, strong multicollinearity is a serious problem that needs to be taken care of.

Multicollinearity induces variance inflation when predictors are strongly intercorrelated. This results in wider Confidence Intervals for the true parameter estimates and makes models less reliable. Thus, detection of Multicollinearity is crucial to training models. A method called Variance Inflation Factor (VIF) can be used. As the name suggests; VIF measures how much the variance of estimated coefficients is inflated as compared to when the predictor variables are not related (Yoo et al., 2014). For a given predictor j, its VIF is defined as:

$$VIF = \frac{1}{1 - {R^2}_j} \tag{1}$$

(where ${R^2}_j$ is the coefficient of determination when predictor j is regressed against other predictors)

There is no limit on the magnitude of VIF. Table 3.2 suggests the general rule of thumb in order to detect multicollinearity.

Table 3.2 VIF Measurement

VIF	**Status of Predictors**
VIF = 1	Not Correlated
1 < VIF < 5	Moderately Correlated
VIF > 5	Highly Correlated

Applying VIF to the Instagram data set, there exists no multicollinearity among the continuous variables as seen in Figure 3.2. Most of the predictors' VIFs are slightly above 1; implying that these predictors are not intercorrelated or are moderately correlated at best; therefore, no further processing is needed regarding multicollinearity problem for this dataset.

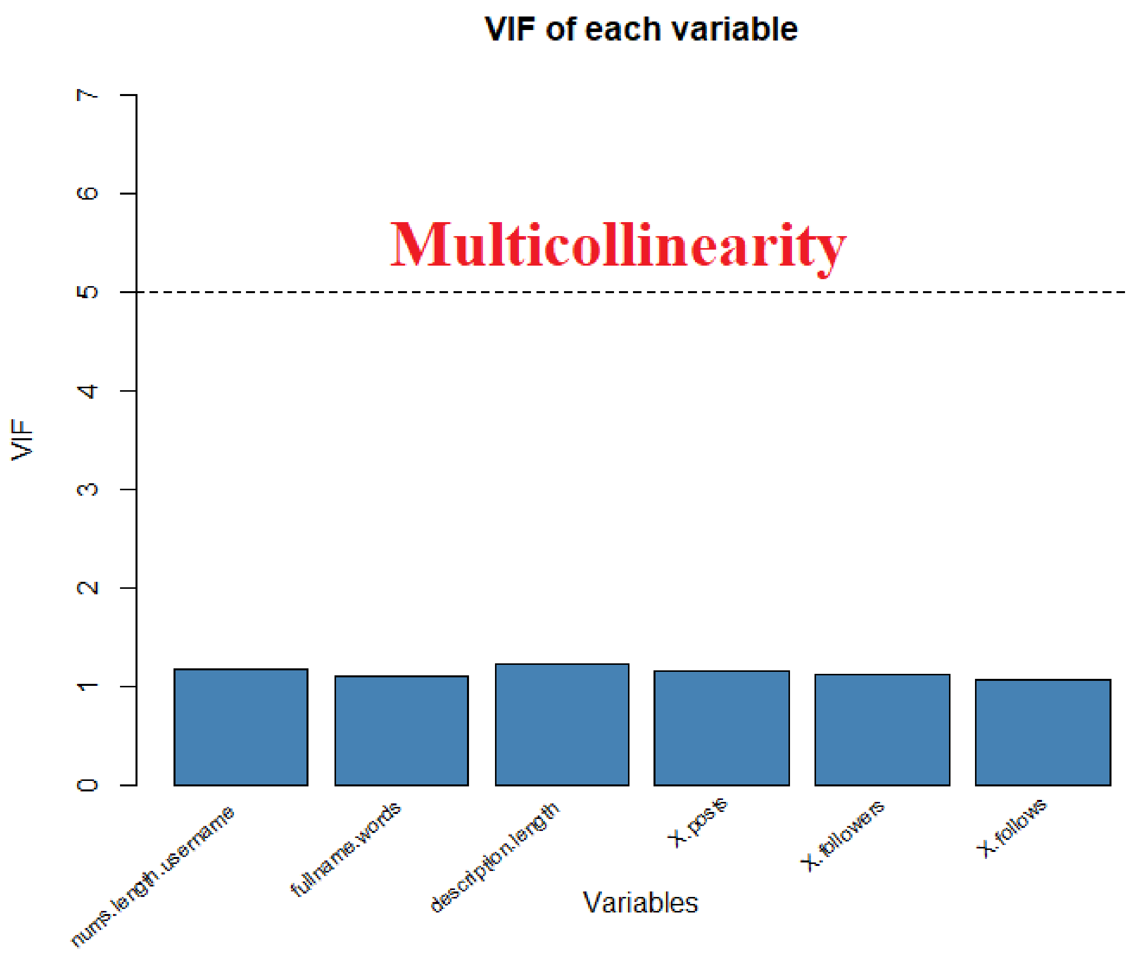

Figure 3.2: VIF Bar Chart

3.0.3 Transformation:

One of the most common assumptions when doing statistical analysis is normality. Predictors often do not meet the assumptions of parametric statistical test (such as ANOVA) because they are not normally distributed (McDonald, 2014). Therefore, using these statistical tests on such predictors might give incorrect or misleading results. In practice, transforming skewed or non-normal distributed predictors will make models fit statistical assumptions better.

Depending on whether a predictor is skewed left or right; appropriate transformation can be used to eliminate skewness. For instance, some popular transformations are log transformation, square root, inverse, Box-Cox, Yeo-Johnson, etc. For this project, Box-Cox transformation is used in order to transform data into normality. Not all the continuous variables will become normally distributed after the transformation due to some of them having extreme skewness or outliers; however, they will be less skewed at the least. Figure 3.3 shows the before and after Box-Cox transformations of continuous variables in this dataset.

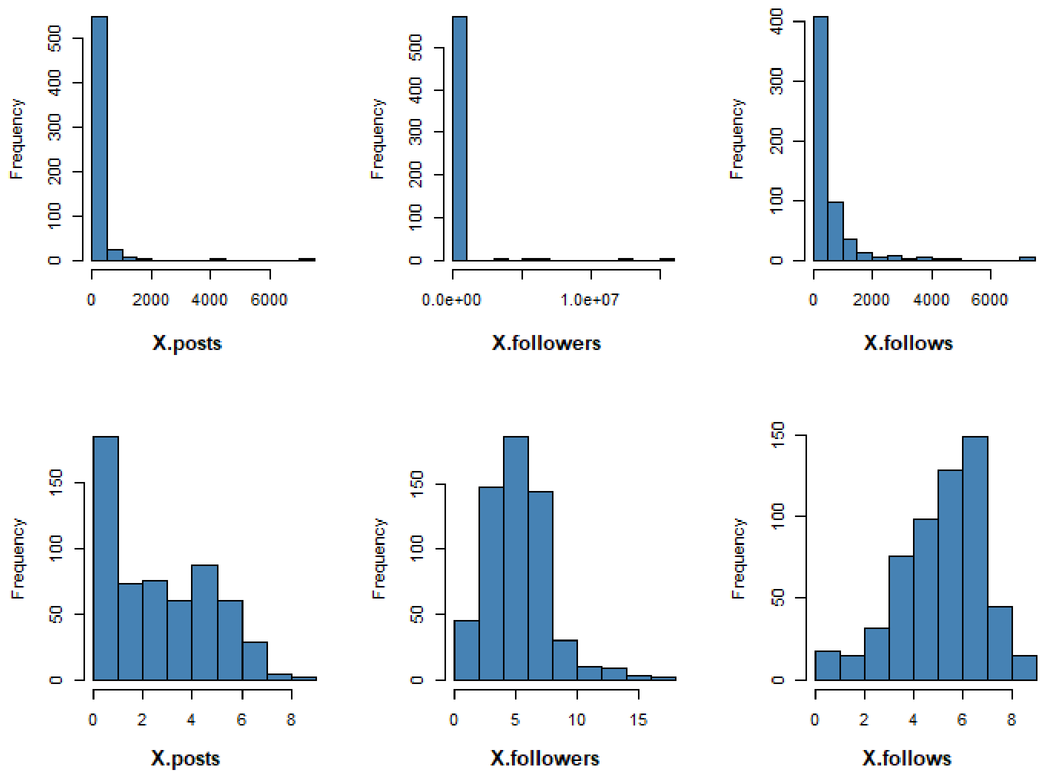

Figure 3.3: Continuous Variables Before Pre-processing

3.0.4 Standardization

Predictors may have different units (for example: scores, feet, dollars, hours); which in turn means that they have different scales. Specifically in this data set, *X.followers* (number of followers) and *fullname.word* (full name in words token) are most likely to have different scales.

Differences in scales across predictors may be problematic when training models. Large scale predictors can result in a model that learns large parameter estimates; which make those predictors seem more important and more predictive than the small-scale predictors; even when they have the same predictive power. In addition, models with large parameter estimates are often unstable and are extremely sensitive to a slight change in input value; which results in larger standard error.

A simple way to deal with dataset having differing scales across predictors is to standardize them in order for them to have the same ground. Standardizing a predictor refers to rescaling its distribution of values so that it has mean of 0 and standard deviation of 1; hence the name “center and scale”. An observation of a predictor j is standardized as follows:

$$y = \frac{x - mean_j}{std_j} \tag{2}$$

Figure 3.4 below is a comparison of before and after standardization of the continuous variable. The top three plots include the distributions of the predictors after Box-Cox transformations without being standardized yet. The bottom three plots show those variables after being standardized. As seen from the figure, their distributions are the same;

the only difference is the range in their x-axis where the standardized predictors now have centers at 0.

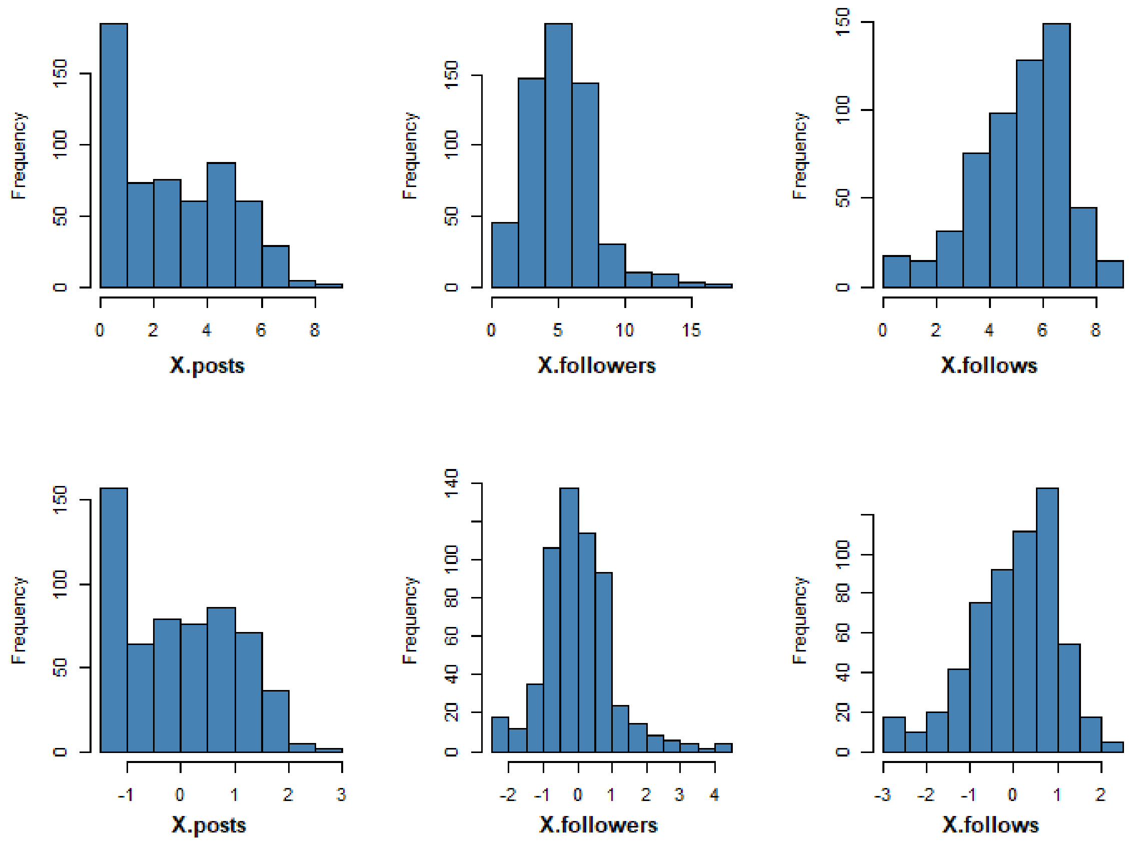

Figure 3.4: Continuous Variables After Pre-processing

3.0.5 Spatial Sign

In general, outliers are observations that are exceptionally far from the majority of the data. Outliers are defined differently depending on the context and business problems. Even with a thorough understanding of the data set, outliers can still be hard to define. However, we can identify suspected observations to be outliers by looking at figures, especially boxplots for this matter. Figure 3.5 shows boxplots of the distributions of all the continuous variables within this dataset.

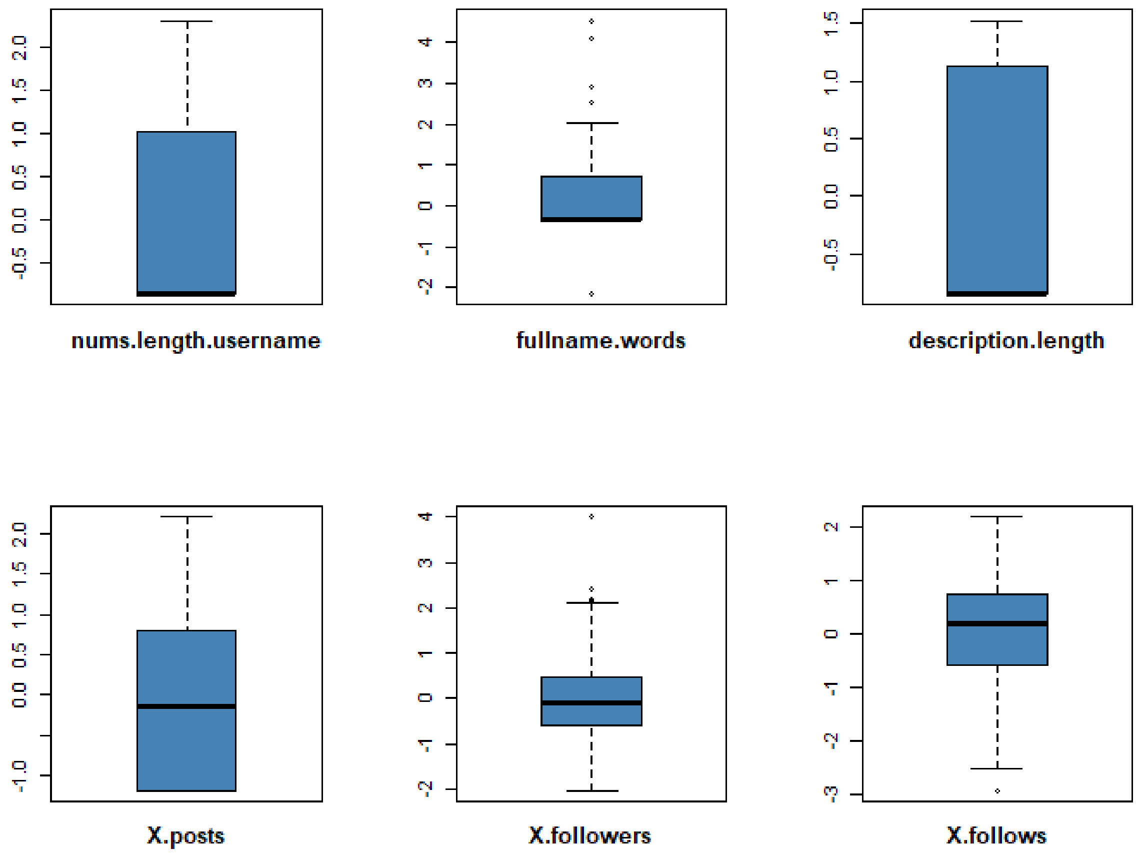

Figure 3.5: Continuous Variables Boxplots Before Spatial Sign Transformation

There are several algorithms that are robust to outliers, such as Tree-based classifiers and SVMs (Kuhn & Johnson, 2013). Unfortunately, algorithms such as Linear Regression and kNN are sensitive to outliers; therefore, removing them before training these models would increase their performance.

As seen in Figure 3.5, outliers reside across multiple variables such as *fullname.words*, *X.followers* and *X.follows*. These outliers can be excluded from the dataset before model training process. However, removing outliers would not be beneficial if the dataset is small such as this dataset because the amount of data would be insufficient for the models to learn. One data transformation that can minimize the problem is Spatial Sign. This transformation projects the data for a predictor to the unit circle in p-dimensions, where p is the number of predictors. In simple terms, this transformation brings the outliers towards the majority of the data. Mathematically, each sample is divided by its squared norm (Kuhn & Johnson, 2013):

$$x^*{}_{ij} = \frac{x_{ij}}{\sqrt{\sum_{j=1}^{P} x^2{}_{ij}}} \tag{3}$$

As it is not plausible to plot more than three dimensions, for demonstration purposes, Figure 3.6 takes into account only two predictors from the dataset, *X.follows* and *X.followers* to illustrates all data points using these predictors. There are at least four observations are far away from the majority of the data on the left-hand panel of the figure. The spatial sign transformation is shown on the right-hand panel where all the data points are projected to a circle where they share the same distance from the origin.

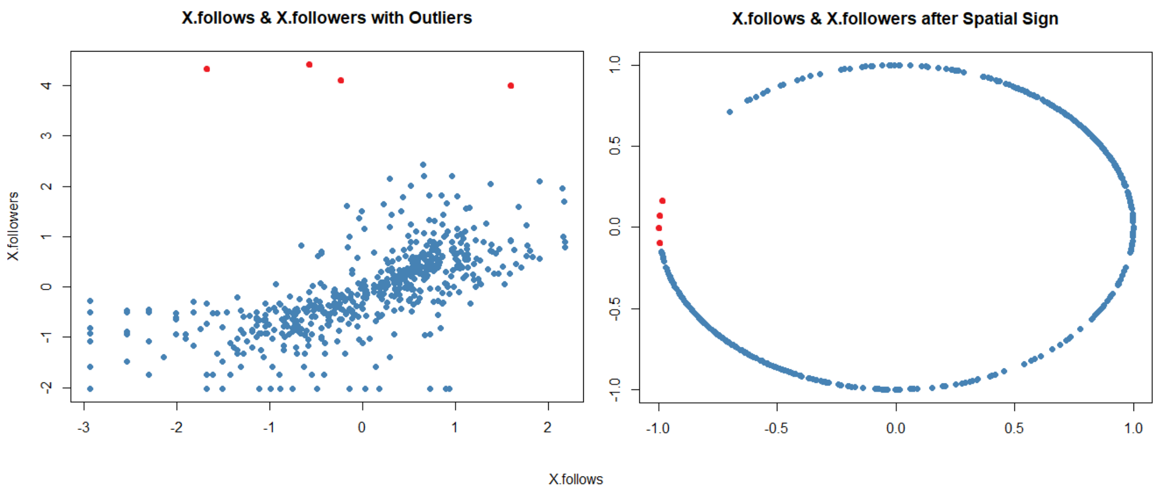

Figure 3.6: Follows and Followers Spatial Sign Plots

Figure 3.7 shows the boxplots of all continuous predictors after performing spatial sign transformation on them. As seen from the figure, there are no outliers in any of the predictors.

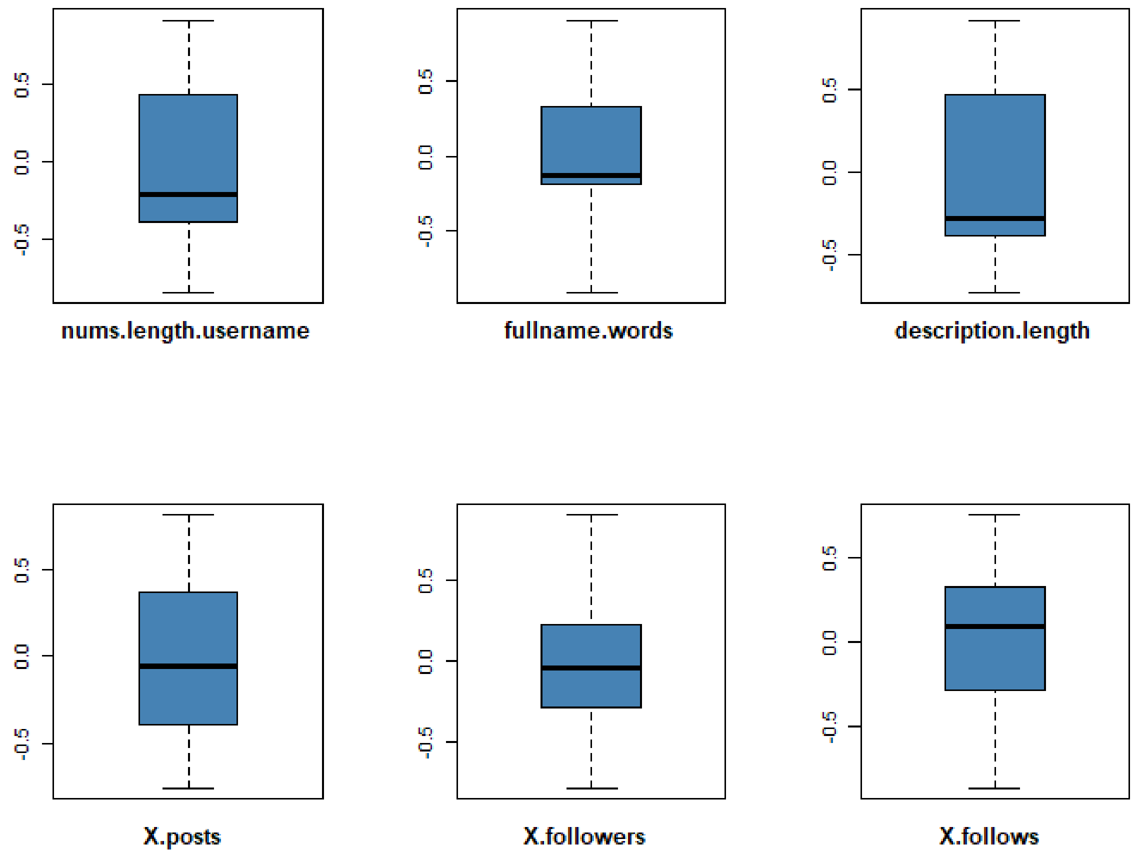

Figure 3.7: Continuous Variables Boxplots After Spatial Sign Transformation

3.1 Resampling method

Generally, resampling methods are used to estimate the model performance: a subset of data are used to fit a model and the remaining data are used to estimate the efficiency of that model. This process is repeated multiple times and the results are summarized (Kuhn & Johnson, 2013).

K-fold cross-validation is one of the most well-known resampling methods and is utilized in this study. Data are randomly partitioned into k sets of relatively equal sizes. A common number for k is usually 5 or 10. A model is trained using all data points except for the ones in the first subset (i.e. fold). This fold is used by the model for predictions and to estimate the performance measures. It is then returned to the training set and the process repeats with the second fold and so on. The estimates of performance of k folds are summarized and can be used to understand the changes between the tuning parameters and the models. Even if the models do not have tuning parameters, k-fold cross-validation is a great way to evaluate out-of-sample performance of the models. This study uses k = 10 for smaller bias. Figure 3.8 illustrates a 5-fold cross-validation.

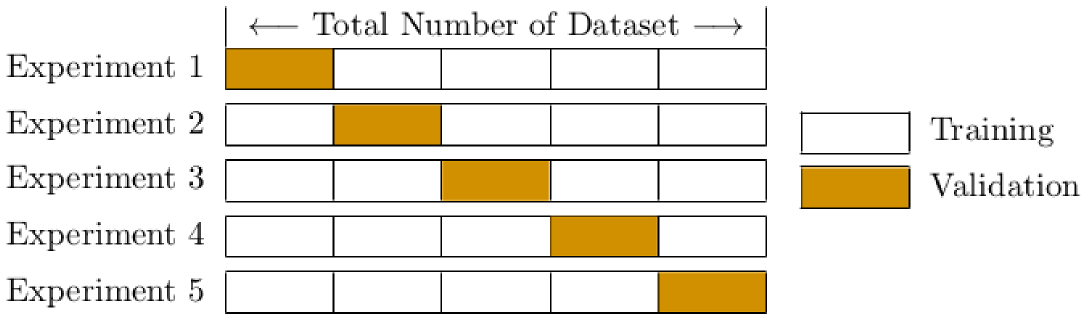

Figure 3.8: Cross Validation Diagram

3.2 Performance Metrics

In order to evaluate different models, performance measures of each model have to be compared. Confusion matrix is the simplest and most effective metric when it comes to model performance measurement. Various measures such as Accuracy, Kappa, F1 score, etc. can be obtained from a confusion matrix. A discussion of the concepts and different performance measure are as follows.

3.2.1 Confusion matrix

Confusion matrix is one of the most intuitive metrics used for finding the accuracy and misclassification of a model. It is used for Classification problem where the output can be of two or more types of classes. Figure 3.9 depicts an example of a simple confusion matrix.

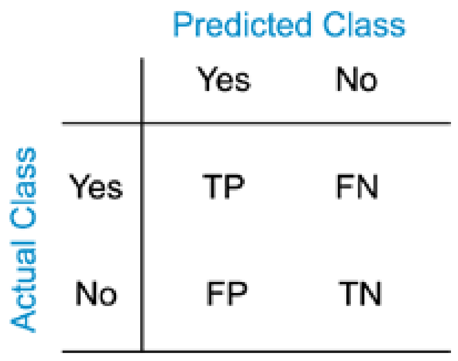

Figure 3.9 Confusion Matrix Diagram

True Positives (TP): these are cases when the actual class of the data point is ‘Yes’ and the predicted class of the same data point is also ‘Yes’.

True Negatives (TN): these are cases when the actual class of the data point is ‘No’ and the predicted class of the same data point is also ‘No’.

False Positives (FP): cases when the actual class of the data point is ‘No’ but the predicted value of the same data point is ‘Yes’.

False Negatives (FN): cases when the actual class of the data point is ‘Yes’ but the predicted value of the same data point is ‘No’.

An ideal model is a model that gives 0 False Positives and 0 False Negatives. However, this is hardly the case in real life as no model has perfect predictions. There is also no hard rule on what measure in the confusion matrix should be minimized. It all depends on the context of the problem and business goals. The confusion matrix itself is not a performance measure; however, almost all of the performance metrics are based on the numbers inside of it. A few of those are discussed as follows.

3.2.2 Accuracy

One of the easiest and most effective measure of performance is Accuracy. It is simply the number of correct predictions made by the model out of all predictions made. The higher the accuracy, the better a model is. Accuracy is a good measure when the target variable classes are balanced or nearly balanced. However, it is not a reliable measure when the classes are heavily imbalanced. Given a confusion matrix, accuracy is defined as:

$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN} \tag{4}$$

3.2.3 Precision

Precision is a measure that indicates what proportion of observations that are predicted as ‘Yes’ class by the model are actually from ‘Yes’ class. Given a confusion matrix, precision is defined as:

$$Precision = \frac{TP}{TP + FP} \tag{5}$$

3.2.4 Recall or Sensitivity

Recall/Sensitivity is a measure that indicates the proportion of observations that are actually from ‘Yes’ class are predicted as ‘Yes’ class by the mode. Recall/Sensitivity is defined as:

$$Recall = \frac{TP}{TP + FN} \tag{6}$$

In simple terms, Recall gives us information about a classifier’s performance with respect to False Negatives (how many did we miss); while Precision gives us information about a classifier’s performance with respect to False Positives (how many did we hit). Therefore, minimizing False Negatives or False Positives depends on the context and business problem.

3.2.5 F1 Score

It is not ideal to always have to look at both Precision and Recall when measuring performance of a model. Consequently, it is best to get a single score that represents both Precision and Recall. F1 Score calculates the harmonic mean of both Precision and Recall. As a result, F1 Score is much closer to the smaller number between the two (Derczynski, 2016). In other words, if either Precision or Recall is low; F1 Score would be closer to the lower measure and raises an alert that the model has low Precision or Recall. F1 Score is defined as follows:

$$F1\ Score = 2 * \frac{(Precision * Recall)}{(Precision + Recall)} \tag{7}$$

3.2.6 Kappa

Kappa is similar to Accuracy; however, it takes into account the accuracy that would have happened anyway through random predictions (Garrett & Viera, 2005). The kappa statistics can be used as a metrics to evaluate different classifiers because it takes into account random chance which is usually less misleading than observed accuracy. It is a more useful measure to use on problems that have imbalanced target variable classes. In other words, kappa measures how much better the classifier is comparing to a random guess of the target distribution. Kappa is defined as follows:

$$Kappa = \frac{(Observed\ Accuracy - Random\ Accuracy)}{(1 - Random\ Accuracy)} \tag{8}$$

where

$$Random\ Accuracy = \frac{((TN + FP) * (TN + FN) + (FN + TP) * (FP + TP))}{Total\ Observations^2}$$

3.2.7 ROC curve

The Receiving Operating Characteristic (ROC) is another measure of classifier performance. Using the proportion of positive data points that are correctly classified as positive and the proportion of negative data points that are mistakenly classified as positive (Davis & Goadrich, 2006). ROC curve is generated to depicts the tradeoff between the rate at which something can be correctly predicted (True Positive Rate) with the rate at which something can be incorrectly predicted (False Positive Rate).

The Area Under the ROC Curve (AUC) is a measure that combines the True Positive Rate and False Positive Rate and conveys a more meaningful metric. The higher the AUC, the better the model is in discriminating between the two classes in target variable. Figure 3.10 illustrates an example of a ROC – AUC plot can be used to compare between models.

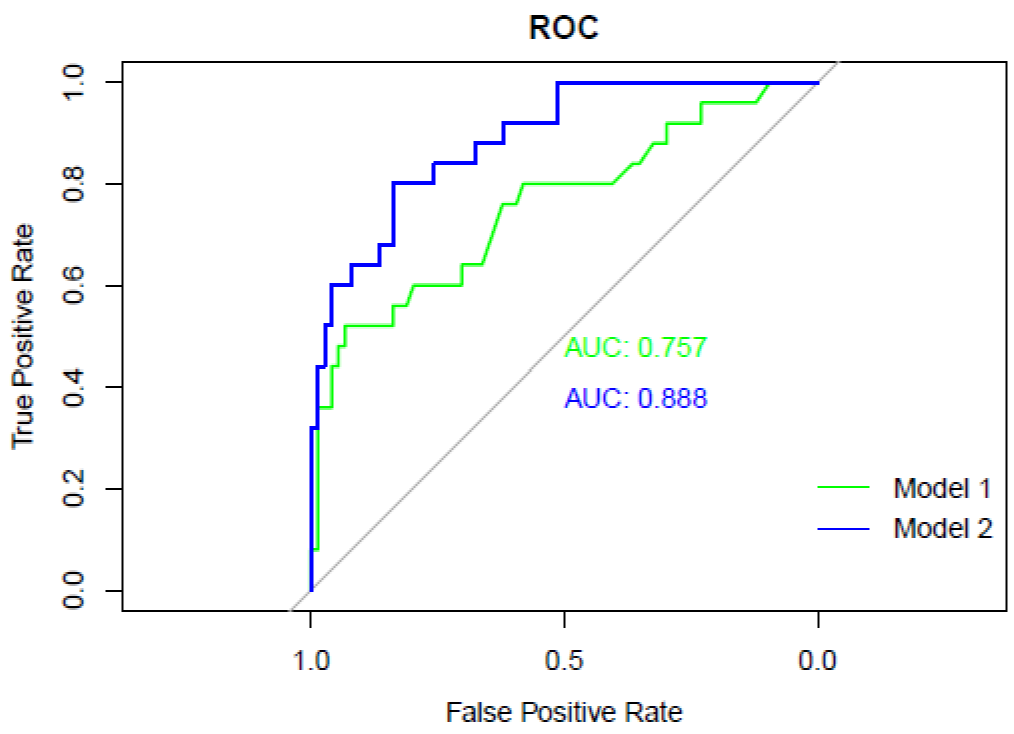

Figure 3.10: Multiple ROC in one plot with AUC

3.2.8 Lift Chart

A lift chart graphically represents the improvement that a model provides when compared against a random guess and measures the change in terms of a lift score. By comparing the lift scores for different models, the most effective model can be determined (Vuk & Curk, 2006). Figure 3.11 shows an example of a lift chart.

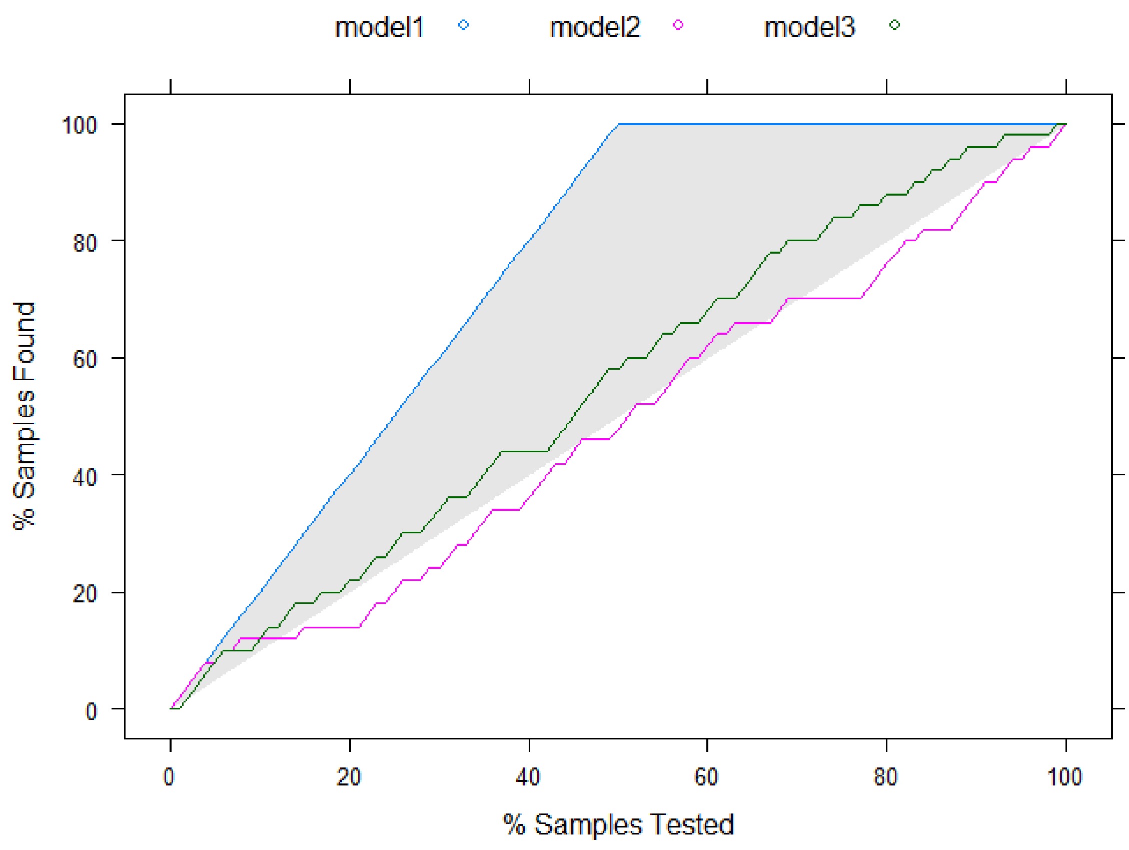

Figure 3.11: Lift Chart

3.3 Logistic Regression

The outcome variable of this study is a binary variable which means it only takes two classes as possible values. The appropriate algorithm to train this model needs to be capable of predicting the outcome value as either 0 or 1, or as a probability of the occurrence of an event between 0 and 1 (“Logistic Regression,” (n.d.)). Linear regression will not restrict the binary variable within the required range and therefore will not be used in this study. On the other hand, logistic regression, a sub class of generalized linear model, can achieve the desired model for binary outcome variable and will be used to build a model for this data set outcome variable “fake”. Figure 3.12 below is a visualization of the difference features between linear regression and logistic regression discussed above.

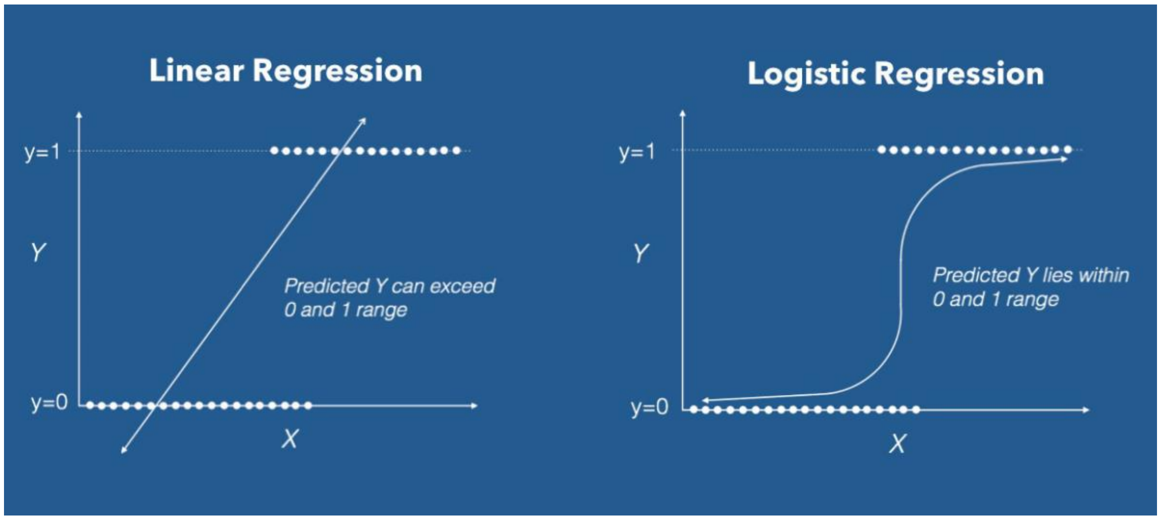
Linear Regression
y=1
Y
y=0
X
Predicted Y can exceed 0 and 1 range
Logistic Regression
y=1
Y
y=0
X
Predicted Y lies within 0 and 1 range

Figure 3.12: Linear Regression vs. Logistic Regression

With this, logistic regression can help predict whether the class of each account as either 0 - a genuine account or 1- a spammer account by computing a probability for each account. Using a threshold or cut off point between 0 and 1, if the probability is over this threshold, it is said that the class for that sample is 1 or fake. Similarly, if the probability is under this threshold, it is said that the class for that sample is 0 or genuine.

3.4 K-nearest neighbors

The next algorithm that will be used to train the model is k-nearest neighbors or often denoted as kNN. KNN uses the closest data points or closest neighbors to classify the outcome. The number of neighbors is defined by k and the closeness between the neighbors and the value is measured by distance. Since distance is being used for this model, it is imperative that all predictors should scaled and centered to prevent predictors with larger scales have bias weight on the predictions (Kuhn & Johnson, 2013). Figure 3.13 below will illustrate how KNN predicting samples using five closest neighbors.

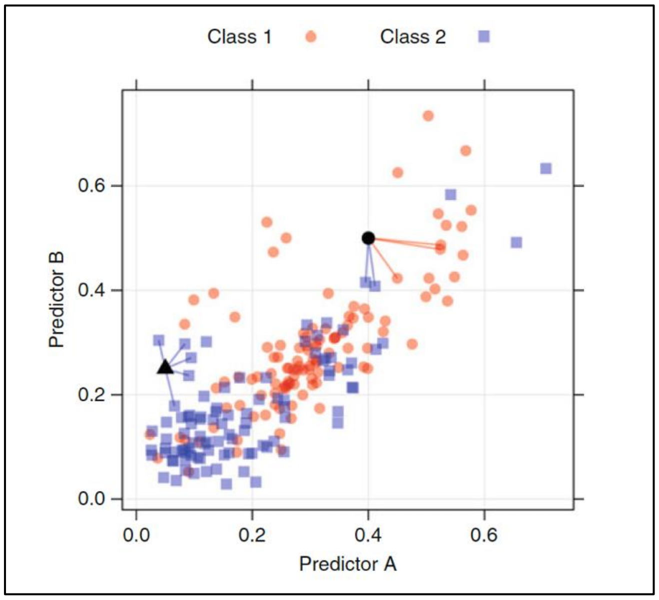

Figure 3.13: K-nearest neighbor classifying samples using five closest neighbors.

From the figure, there are two possible classes, circle and square. The first value that needs to be classified is a black triangle. As illustrated, the five closest points to this black triangle are squares so overall, the class of this value is a square. On the other hand, the other value is a black circle and the five closest neighbors to this point include two squares and three circles so overall, the class for this point will be a circle. For this data set, using the KNN algorithm will train a model that can help predict whether the class of each account is 0-genuine or 1-fake by examining the k-closest neighbors to each sample and tallying the majority votes.

3.5 Rule-Based Decision Tree

Tree-based models as the name suggested follow structure of trees which include roots, branches and leaves. Figure 3.14 below is a graphical representation of a tree model.

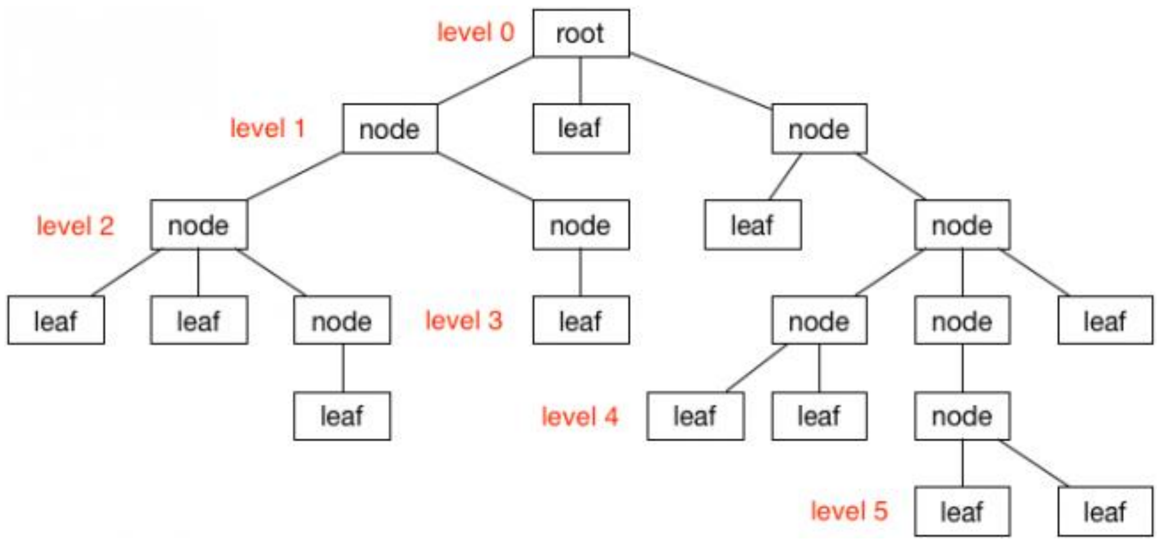

Figure 3.14: Decision Tree Diagram

Tree-based models consist of one or multiple nested if-then statements. The model would start at the root, train the data to follow a set of rules by following the nested if-then conditions down the tree nodes to get to the terminal node or leaf as noted in the diagram. Each tree node is represented by a predictor variable and the terminal node is one of the classes of the outcome variable that the model tries to predict. Tree-based models are easy to interpret and implement as the structure of the tree is constructed by logical conditions. In addition, these models do not require the pre-processing of predictor variables. (Kuhn and Johnson, 2013)

The rule-based tree algorithm that is used to train the Instagram data set for this study will specify a set of rules as distinct paths through a tree. Each path will lead through a set of nodes or predictors and finally arrive at one of the two possible classes of the outcome variable “Fake”. By following these rules down the tree, it will lead us to the predicted classified value of each account.

3.6 Random Forest

When many trees are being planted, a forest is created. This leads to the final algorithm that will be used in this study, random forest. Random forest as the name suggested consists of a large number of decision trees. Each tree in the forest will produce a prediction for the class. After combining the results of all the trees, the class prediction with the most counts will be the winner and also the model's prediction (Yiu, 2019). This concept can be seen illustrated in Figure 3.15 below.

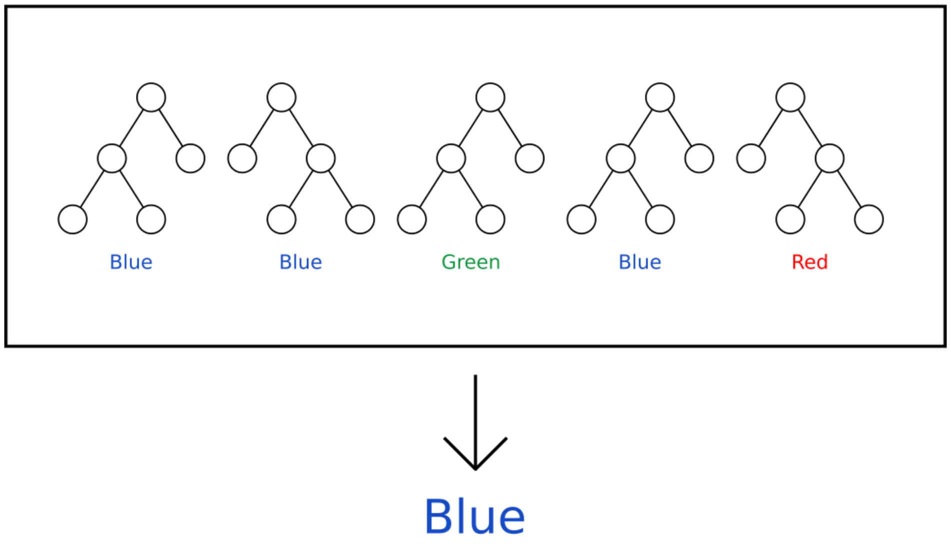

Figure 3.15: Random Forest Diagram

In Figure 3.15, there are five trees in this forest. Three of the trees spit out Blue as the class. One spits out Green and one spits out red. After tallying up the results, Blue has the most votes and hence it is the final class prediction.

Similar to the rule-based algorithm, random forest will be used to train the Instagram data set except that under this algorithm, there will be multiple trees and each tree will have its own set of rules. The predictions from all trees will then be tallied up and the class prediction, whether it is 0-genuine or 1-fake, with the most votes will be the winner.

Chapter 4: Results and Analysis

The metric used for evaluation of model is accuracy on test set. This dataset is overall balanced in terms of classes; therefore, accuracy should be a good measure for how models perform. Other measures are also calculated and illustrated as means of complementary metrics in addition to accuracy. But first of all, Apparent Performance of each model is compared to evaluate how each model performs on in-sample data.

4.1 Logistic regression

In spite of having no tuning parameters, applying 10-fold cross-validation on Logistic Regression is still beneficial because it gives an idea how the model performs on out-of-sample data. The final parameters for this model are given as:

$$\begin{aligned} logit(p) = 3.63 &- 4.17\, profile.picYes + 2.71\, nums.length.username \\ &- 0.68\, fullname.words - 0.94\, description.length \\ &- 1.19\, privateYes - 1.80\, X.post - 9.14\, X.followers \\ &+ 3.23\, X.follows \end{aligned} \tag{9}$$

These parameter estimates are in log-odds scale. In order for them to be more interpretable; they can be converted to odds scale instead by simply taking their exponential: $e^{coefficient}$. If the result of the exponential is less than 1; log-odds scale are interpreted as $\frac{1}{e^{coefficient}}$. For example, for *X.followers* predictor, since $e^{-9.14} \approx 0.0001$; this odds scale should be interpreted as $\frac{1}{e^{-9.14}} \approx 9320$. This translates to 1 unit increase in *X.followers,* an account is 9320 times less likely to be fake.

4.2 K-Nearest Neighbor (KNN)

As kNN is an algorithm with the number of neighbors k as tuning parameter; a grid search was utilized in order to figure out the optimal k. A sequence of k from one to fifteen is run on 10-fold cross-validation, the metrics that was used to optimize k is accuracy. Figure 4.1 shows the summarized accuracy for different values of k.

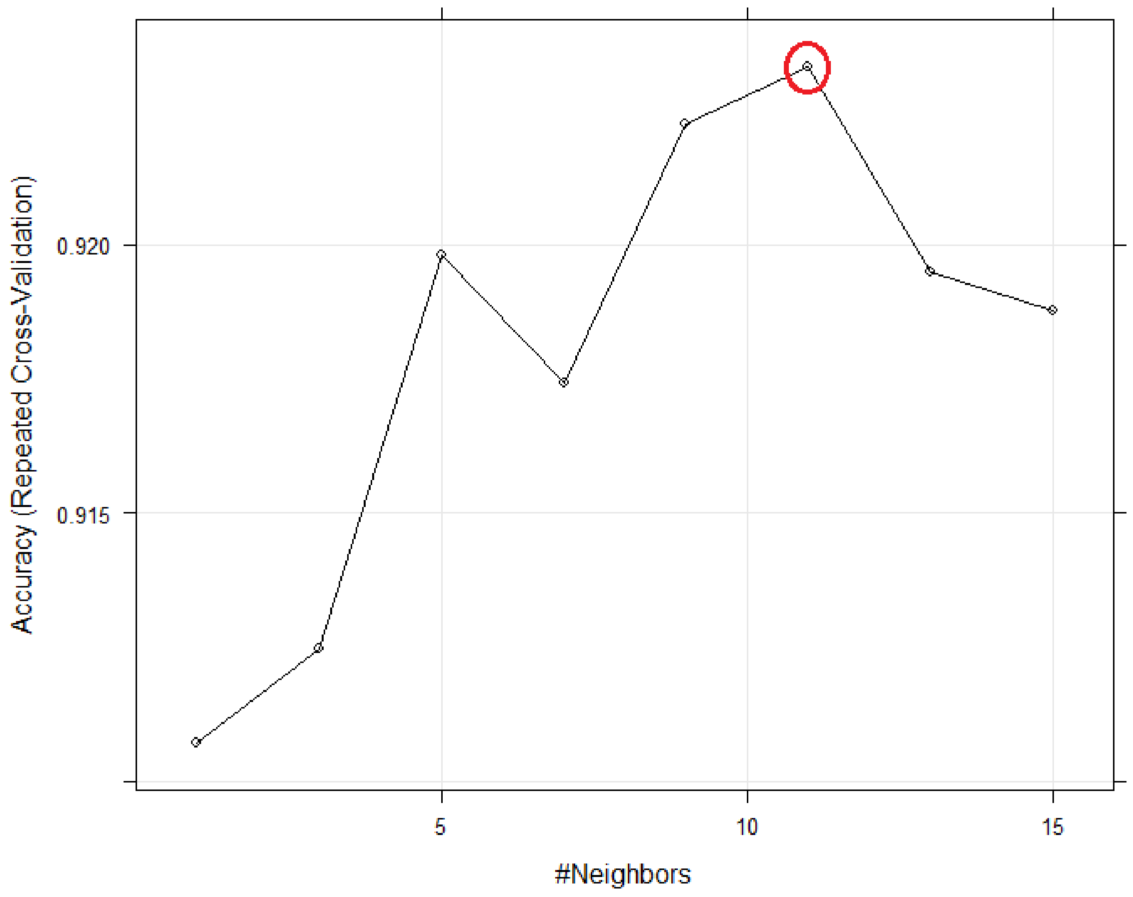

Figure 4.1: KNN Accuracy by Tuning Parameters Plot

As seen from Figure 4.1 above, k = 11 gives the best accuracy. With k greater than 11, the model is a result of underfitting due to decreasing accuracy on the validation tests within 10-fold cross-validation.

4.3 Decision Tree

Decision Tree is also an algorithm with a tuning hyperparameter 'complexity parameter' (cp). This hyperparameter controls how complex a tree is. The lower complexity parameter; the more complex. Figure 4.2 illustrates the summarized accuracy over different values of complexity parameters.

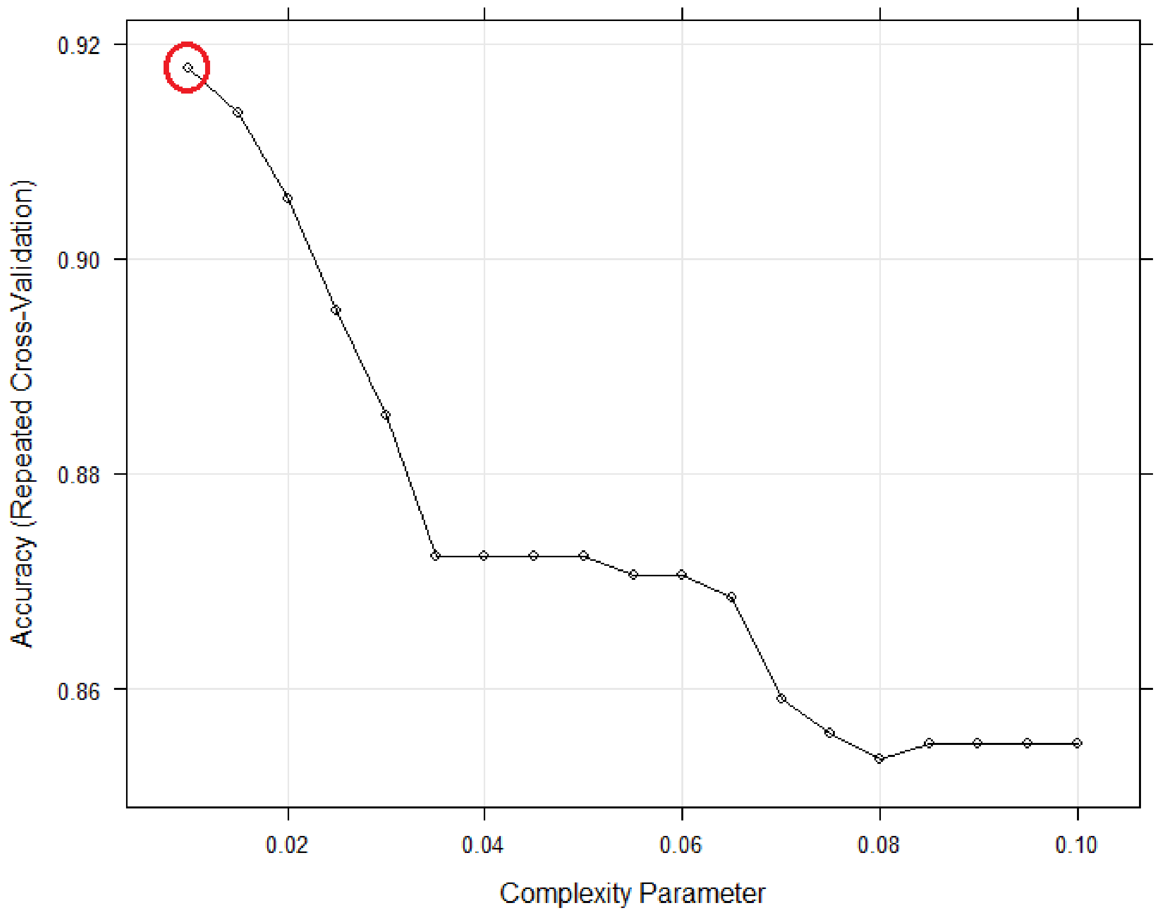

Figure 4.2: Decision Tree Accuracy by Tuning Parameters

As seen in Figure 4.2, the tree with the best performance is the most complex tree for this dataset. With complexity parameter = 0; this tree can capture the relationship between the variables to make its predictions. Figure 4.3 below show the actual plot of this tree.

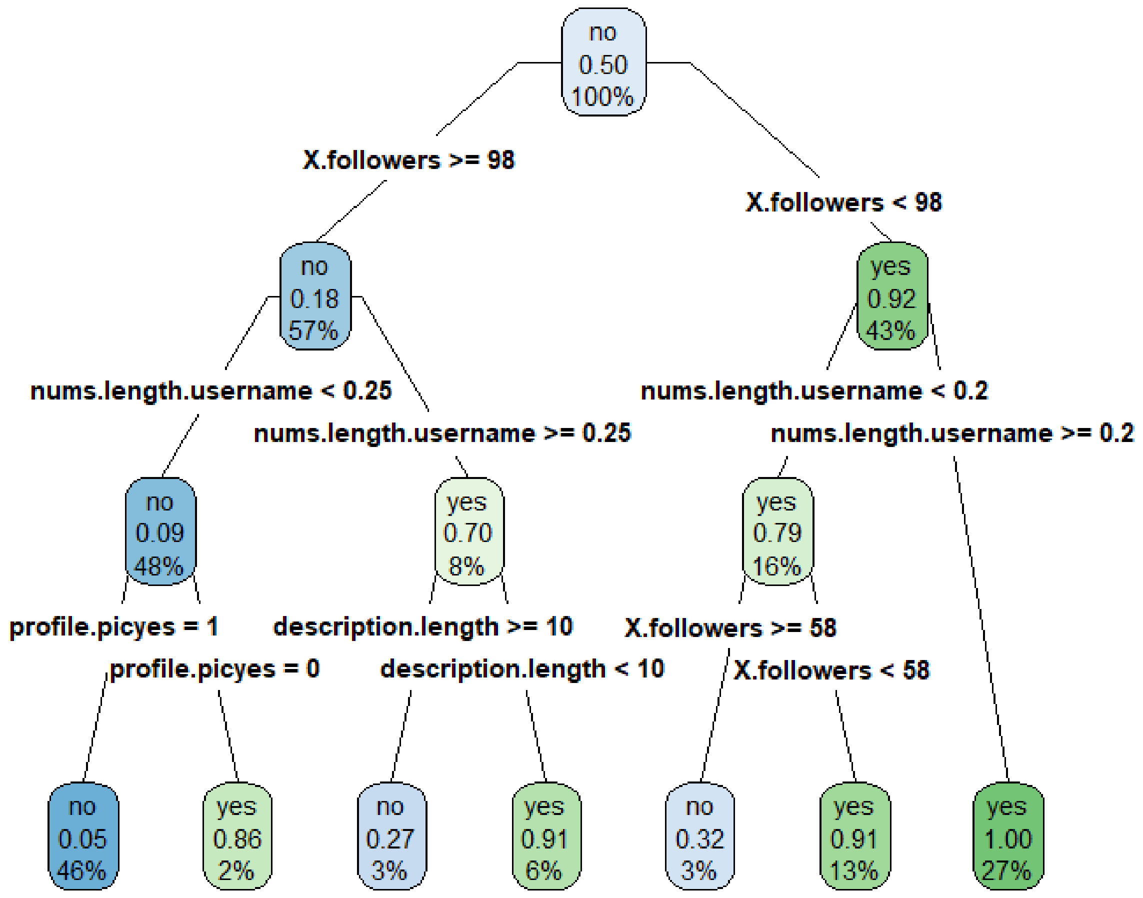

Figure 4.3: Decision Tree Diagram

4.4 Random Forest

The final algorithm random forest is also one with a tuning hyperparameter ‘mtry’. This hyperparameter is the number of variables randomly sampled as candidates at each split. As shown in Figure 4.4, mtry = 1 gives the best performance.

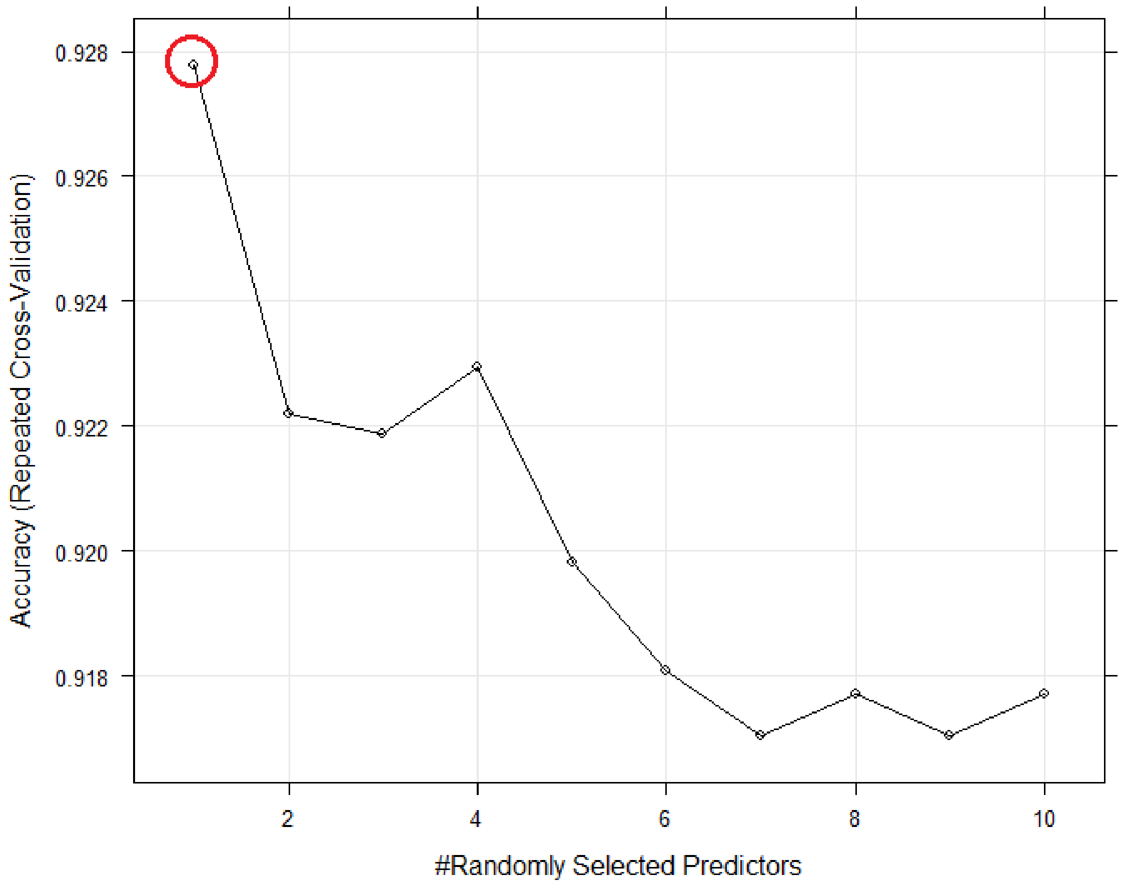

Figure 4.4: RF Accuracy by Tuning Parameters

4.5 In-sample Apparent Performance

Apparent performance is the performance of each model when they are predicting on the dataset they were trained on. It gives a relative idea of how each model performs before they are tested on the out-of-sample test dataset. Below is some apparent performance measure of the models.

Table 4.1 below shows the Accuracy, the Kappa and F1 score of each model sorted in a descending order followed by a bar plot of the models' accuracy. Random forest algorithm performs best out of all models based on the Apparent Performance.

Table 4.1 In-Sample Performance Metric by Model

	Accuracy	Kappa	F1 Score
Random Forest	**0.950**	**0.899**	**0.949**
Decision Tree	0.939	0.878	0.938
Logistic Regression	0.932	0.865	0.931
kNN	0.932	0.865	0.930

4.6 Out-of-sample performance

In order to evaluate how the models actually performs on what they have learned, out-of-sample data set is used. This is the test set that was partitioned at the very beginning of the entire process. As discussed in previous section, Random forest is the model that performs the best on the training set. If it can perform best on test set, it would be selected as the best model. Table 4.2 below illustrates the performance of every model on test set.

Table 4.2 Out-of-Sample Performance Metric by Model

	Accuracy	Kappa	F1 Score
Random Forest	**0.925**	**0.850**	**0.926**
Logistic Regression	0.908	0.817	0.911
kNN	0.900	0.800	0.898
Decision Tree	0.875	0.750	0.867

Not surprisingly, Random forest still performs the best out of all models in Accuracy, Kappa and F1 score measures. Figure 4.5 below shows the ROC curve of the models on the test set.

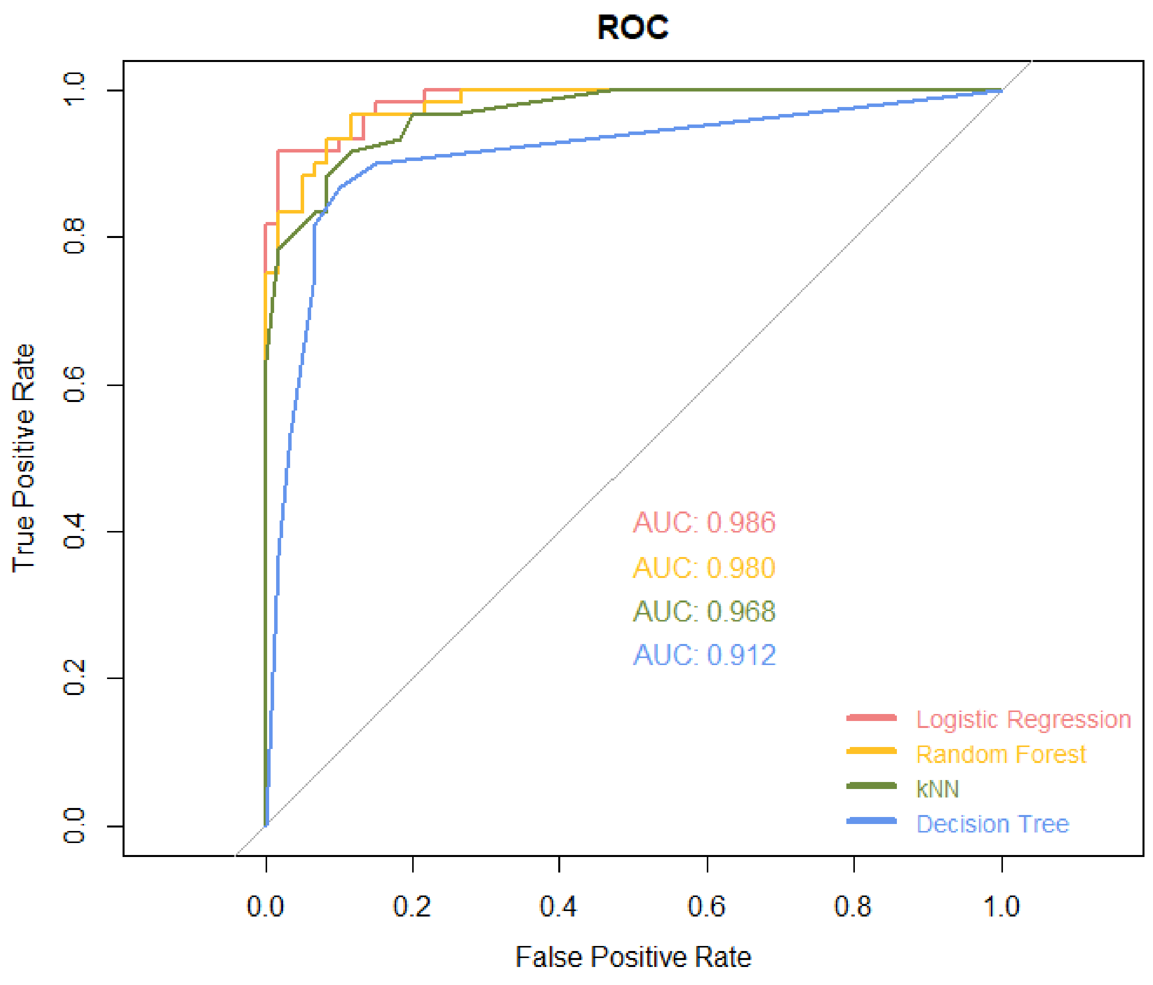

Figure 4.5: Test Set ROC and AUC by Model

From the ROC plot, Logistic Regression and Random Forest have the best AUC; thus, they perform a lot better than kNN and Decision Tree in terms correctly discriminating the two classes. Even though Logistic Regression performs the best, Random Forest is not that far behind it. Figure 4.6 below illustrates the last performance metrics; which is Lift Chart.

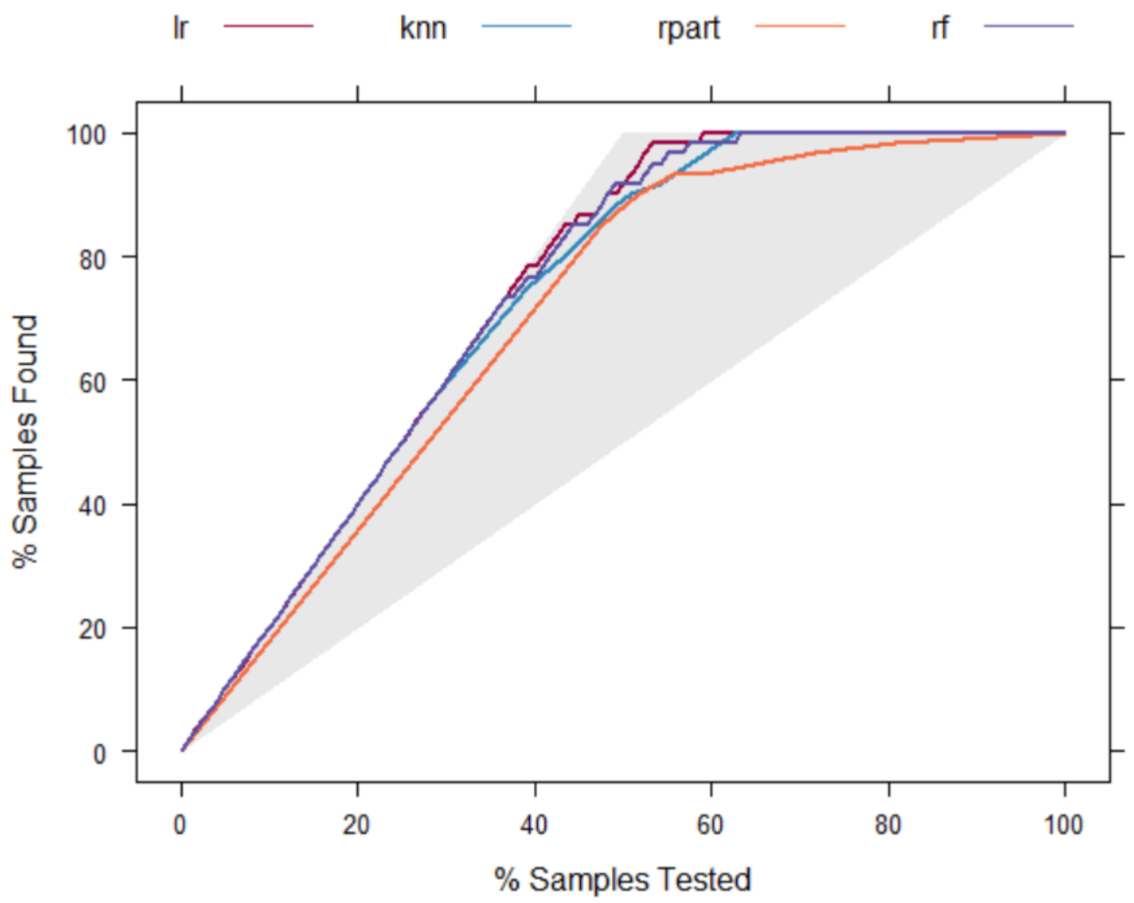

Figure 4.6: Test Set Lift Chart by Model

Figure 4.6 suggests that Logistic Regression and Random Forest again have the best lift of all the models. As a reminder, lift chart measures the improvement a model provides against making random guesses. The higher the improvement, the more lift. Since Logistic Regression and Random Forest have the highest lift, they have the best improvement against random guesses.

Chapter 5: Conclusions

Internet fraudulent activities on the rise have affected many people and jeopardized their personal information. Users of social media applications such as Instagram now have to be extra cautious to avoid having their account being exploited. This could potentially take some satisfactions away while using the application as users cannot fully enjoy all of the platforms' entertaining functions without fearing of getting their accounts hacked. However as seen in this study, machine learning algorithms can be deployed to help combat this disaster.

As the results show, the best algorithm in predicting whether an account is genuine or fake is Random Forest followed by Logistic Regression. Instagram and other social media platforms' administrators should take advantage of these techniques and proactively utilize them to identify fake accounts before they could harm their loyal users. Not only this could help decrease online fraudulent activities, it could also help these companies in retaining the number of users and in return could increase their profits overall.

References

Davis, J., & Goadrich, M. (2006). The relationship between Precision-Recall and ROC curves. ICML.

Derczynski, L. (2016). Complementarity, F-score, and NLP Evaluation. LREC.

Kuhn, M. & Johnson, K. (2013). Applied Predictive Modeling.

Logistic Regression – A Complete Tutorial With Examples in R (n.d.). Retrieved from https://www.machinelearningplus.com/machine-learning/logistic-regression-tutorial-examples-r/

McDonald, J.H. 2014. Handbook of Biological Statistics (3rd ed.). Sparky House Publishing, Baltimore, Maryland. 140-144.

Viera, A.J., & Garrett, J.M. (2005). Understanding interobserver agreement: the kappa statistic. Family medicine, 37 5, 360-3.

Vuk, M. (2006). ROC Curve, Lift Chart and Calibration Plot.

Yiu, T. (2019, June 12). Understanding Random Forest: How the Algorithm Works and Why It Is So Effective. Retrieved from https://towardsdatascience.com/understanding-random-forest-58381e0602d2.

Yoo, Wonsuk, et al. "A Study of Effects of MultiCollinearity in the Multivariable Analysis." International Journal of Applied Science and Technology, U.S. National Library of Medicine, Oct. 2014. Retrieved from www.ncbi.nlm.nih.gov/pmc/articles/PMC4318006/.

Appendix. R Code

```
library(purrr)

library(tidyr)

library(ggplot2)

library(caret)

library(plyr)

library(dplyr)

library(grid)

library(gridExtra)

library(pROC)

library(e1071)

library(olsrr)

library(rpart.plot)

library(rpart)


# Load in data

train <- read.csv('train.csv', stringsAsFactors = F)

test <- read.csv('test.csv', stringsAsFactors = F)


###### EDA/Visualizations ######

# histogram matrix of numeric variables

X <- train[,1:11] # create a subset dataframe consist of only the predictors

par(mfrow = c(3, 4)) # split the graph into 9 subplots with 3 rows & 3 columns
```

```
for (i in 1:ncol(X)) {
  hist(X[ ,i], xlab = names(X[i]), main = paste(names(X[i])), col="darkgray")
}

# distribution of labels
table(train$fake)

# # plot of labels against 'X.followers'
# ggplot(data=train, aes(x=X.followers, y=fake)) +
# geom_point() # all the fake accounts are the ones with low number of followers
##
# plot of labels against 'X.posts'
# ggplot(data=train, aes(x=X.posts, y=fake)) +
# geom_point() # fake accounts have very low amount of posts
##
# plot of labels against 'X.follows'
# ggplot(data=train, aes(x=X.follows, y=fake)) +
# geom_point() # fake and real accounts have roughly the same amount of follows

###### Models Training: Logistic Regression (Linear) ######
# Model 1: without any processing and using all variables
# relevel 'fake' for later purpose
train$fake <- ifelse(train$fake==1, 'yes', 'no')
```

```
test$fake <- ifelse(test$fake==1, 'yes', 'no')


# relevel 'profile.pic'

train$profile.pic <- ifelse(train$profile.pic==1, 'yes', 'no')

test$profile.pic <- ifelse(test$profile.pic==1, 'yes', 'no')


# relevel 'private'

train$private <- ifelse(train$private==1, 'yes', 'no')

test$private <- ifelse(test$private==1, 'yes', 'no')


# make categorical variables factors

train$fake <- as.factor(train$fake)

test$fake <- as.factor(test$fake)


train$profile.pic <- as.factor(train$profile.pic)

test$profile.pic <- as.factor(test$profile.pic)


train$private <- as.factor(train$private)

test$private <- as.factor(test$private)


# train model with all variables

lr <- glm(data=train,

        fake~.,
        family='binomial') # need to take out or transform some variables
summary(lr) # these coefficients are in log-odds scale

# exp(coef(lr)) # these are in odds scale (more interpretable): for 1 unit increase in each
variable;
# an account is X times more likely to be fake.
# If exp(coef(lr)) < 1: interpret by 1/exp(coef(lr)) - X times less likely to be fake.
# predict
predictions <- predict(lr, type='response')
# confusion matrix
confusion.matrix <- table(train$fake, predictions > 0.5)
confusion.matrix

# accuracy
accuracy <- (confusion.matrix[1] + confusion.matrix[4]) / sum(confusion.matrix)
accuracy

###### Pre-processing (transformation, standardization, etc.) ######
# excluding 'external.URL' (this variable causes perfect separation problem)

# identify Near Zero Variance variables
nzv <- nearZeroVar(train, saveMetrics=T)
nzv # 'nums.length.fullname' & 'name..username' are near zero variance variables.
```

```
# can consider excluding these variables.
# exclude the near 0 variance variables
no_nzv <- preProcess(train, method='nzv')
train_no_nzv <- predict(no_nzv, train)
test_no_nzv <- predict(no_nzv, test)

# check for multi-collinearity among variables
# same the continuous variable separately
vars <- c('nums.length.username', 'fullname.words', 'description.length',
        'X.posts', 'X.followers', 'X.follows')
# correlation matrix
varCor <- cor(train_no_nzv[vars])
round(varCor, 2)

# Print the number of correlated predictors higher than a correlation threshold
highCorr <- sum(abs(varCor[lower.tri(varCor)]) > 0.5)
highCorr # no highly correlated variables

# use VIF to indicate multicollinearity
# VIF <= 1: not correlated. 1 < VIF < 5: moderately correlated. VIF > 5: highly correlated
vif <- ols_coll_diag(lm(data=train_no_nzv,
              fake~. -profile.pic -external.URL -private)) # exclude the categorical variables
```

```
vif.table <- data.frame('Variables'=colnames(train_no_nzv[vars]), 'VIF'=vif$vif_t[3]) # no
multicollinearity

# plot bar chart of each variable VIF with multicollinearity threshold
barplot(height=vif.table$VIF,
        width=0.5,
        main='VIF of each variable',
        xlab='Variables',
        ylab='VIF',
        ylim=c(0, 7),
        names.arg=vif.table$Variables)
abline(h=5, lty=2)

# Transform some variables using Box-Cox:
# 'nums.length.username', 'fullname.words',
# 'description.length', 'X.posts', 'X.followers', 'X.follows'
cleaned.train <- train_no_nzv
cleaned.test <- test_no_nzv

# apply to train set
cleaned.train[vars] <- cleaned.train[vars] + 1 # add 1 to all these variables because cannot
take log(0)
cleaned.test[vars] <- cleaned.test[vars] + 1
```

```
# boxcox
boxcox <- preProcess(cleaned.train[vars], method='BoxCox')
cleaned.train <- predict(boxcox, cleaned.train) # train set
cleaned.test <- predict(boxcox, cleaned.test) # test set

# center and scale (standardization) the predictors
standardization <- preProcess(cleaned.train[vars], method=c('center', 'scale'))
cleaned.train <- predict(standardization, cleaned.train) # train set
cleaned.test <- predict(standardization, cleaned.test) # test set

# plot the new transformed variables
X <- cleaned.train[vars] # create a subset dataframe consist of only the predictors
par(mfrow = c(2, 3)) # split the graph into 9 subplots with 3 rows & 3 columns
for (i in 1:ncol(X)) {
  hist(X[, i], xlab = names(X[i]), main = paste(names(X[i])), col="darkgray")
}

# Outliers: using Spatial Sign - bring outliers closer to the majority of the data
X <- cleaned.train[-c(203, 41, 185, 6, 35, 46, 42, 166, 60, 259, 28, 44), vars] # create a subset dataframe consist of only the continuous predictors
                                                        # exclude some rows for demonstration purpose
par(mfrow = c(2, 3)) # split the graph into 6 subplots with 2 rows & 3 columns
```

```
for (i in 1:ncol(X)) {
  boxplot(X[, i], xlab = names(X[i]), main = paste(names(X[i])), col="darkgray")
} # some variables still have some outliers


# scatter plot of X.posts & X.follows
plot(x=cleaned.train$X.follows[-c(203, 41, 185, 6, 35, 46, 42, 166, 60, 259, 28, 44)], #
exclude these rows
     y=cleaned.train$X.followers[-c(203, 41, 185, 6, 35, 46, 42, 166, 60, 259, 28, 44)], # for
demonstration purpose
     main = "X.follows against X.followers",
     xlab = "X.follows", ylab = "X.followers",
     pch = 19)


# spatial sign
cleaned.train2 <- spatialSign(cleaned.train[vars]) # train set
cleaned.train2 <- data.frame(cleaned.train2)


cleaned.test2 <- spatialSign(cleaned.test[vars]) # test set
cleaned.test2 <- data.frame(cleaned.test2)


# put the variables after doing spatial sign into the data frame
cleaned.train[vars] <- cleaned.train2[vars] # train set
cleaned.test[vars] <- cleaned.test2[vars]
```

```
# replot the boxplot of our continuous variables after doing spatial sign
X <- cleaned.train[vars]
par(mfrow = c(2, 3)) # split the graph into 6 subplots with 2 rows & 3 columns
for (i in 1:ncol(X)) {
  boxplot(X[, i], xlab = names(X[i]), main = paste(names(X[i])), col="darkgray")
} # no more outliers

###### Model Training: lr, knn, decision tree, random forest ######
# logistic regression using using the processed data
lr <- glm(data=cleaned.train,
          fake~. -external.URL,
          family=binomial)
summary(lr)

# predict
new.predictions <- predict(lr, type='response')
# confusion matrix
confusion.matrix <- table(cleaned.train$fake, new.predictions > 0.5)
confusion.matrix

# accuracy
new.accuracy <- (confusion.matrix[1]+confusion.matrix[4])/sum(confusion.matrix)
new.accuracy # accuracy is improved compared to the model without any processing


# # Let's try exclude 'fullname.words' because this variable is not significant

# # based on the summary in previous section

# lr <- glm(data=cleaned.train,

#          fake~. -external.URL -fullname.words,

#          family=binomial)

# summary(lr)

#

# # predictions

# new.predictions2 <- predict(lr, type='response')

# # confusion matrix

# confusion.matrix <- table(cleaned.train$fake, new.predictions2 > 0.5)

# confusion.matrix

#

# # accuracy

# new.accuracy2 <- (confusion.matrix[1] + confusion.matrix[4]) / sum(confusion.matrix)

# new.accuracy2 # accuracy is improved a little bit


# using Cross-validation with 'train' by 'caret' package

control <- trainControl(method='repeatedcv',

                     number=10, # 10 folds cv

                     repeats=5) # repeat 5 times
```

```
                    #classProbs=T,
                    #summaryFunction = twoClassSummary) # so we can set different metric
other than Accuracy
metric <- 'Accuracy'

fit.lr <- train(data=cleaned.train,
             fake~. -external.URL,
             method='glm',
             trControl=control,
             metric=metric)

summary(fit.lr)
fit.lr

# predictions
lr.predictions <- predict(fit.lr, type='raw')

lr.predictionsProb <- round(predict(fit.lr, type='prob'), 8) # need this for ROC curve

# confusion matrix of model trained by cross-validation
lr.confusion <- confusionMatrix(data=lr.predictions,
                          reference=cleaned.train$fake,
                          positive='yes',
                             mode='everything')
lr.confusion


###### Model Training: kNN (Non-linear) ######
fit.knn <- train(data=cleaned.train,
                fake ~ . -external.URL,
                method='knn',
                trControl=control,
                tuneGrid=expand.grid(k=seq(1, 15, by=2)),
                metric=metric)
fit.knn


# plot in-sample accuracy with different k neighbors
plot(fit.knn)


# predictions
knn.predictions <- predict(fit.knn, type='raw')


knn.predictionsProb <- round(predict(fit.knn, type='prob'), 8) # need this for ROC curve


# confusion matrix of model trained by cross-validation
knn.confusion <- confusionMatrix(data=knn.predictions,
                               reference=cleaned.train$fake,
                          positive='yes',
                          mode='everything')
knn.confusion


###### Model Training: Decision Tree (rpart) ######
# Because tree models do not require pre-processing, we can use original data to fit model
fit.rpart <- train(data=train,
               fake ~ .,
               method='rpart',
               trControl=control,
               tuneGrid=expand.grid(cp=seq(0.01, 0.1, by=0.005)),
               metric=metric)
fit.rpart


# plot in-sample accuracy with different complexity parameters
plot(fit.rpart) # cp=0.01: best model (the lower cp; the more complex the tree)


# plot the selected features by our rpart model
plot(varImp(fit.rpart)) # important attributes in the classification process used in the tree)


# predictions
rpart.predictions <- predict(fit.rpart, type='raw')
rpart.predictionsProb <- round(predict(fit.rpart, type='prob'), 8) # need this for ROC curve


# confusion matrix of model trained by cross-validation
rpart.confusion <- confusionMatrix(data=rpart.predictions,
                            reference=train$fake,
                            positive='yes',
                            mode='everything')
rpart.confusion


# plot final decision tree
rpart.plot(fit.rpart$finalModel,
        type=4,
        clip.right.labs=F,
        branch=.3)


###### Model Training: Random Forest ######
# mtry: number of randomly selected variables used at each tree. The higher, the less
random
# ntree: number of trees in this Random Forest
tuneGrid.rf <- expand.grid(mtry=c(1:10))
fit.rf <- train(data=cleaned.train,
            fake ~ . -external.URL -fullname.words,
            method='rf',
            trControl=control,
            tuneGrid=tuneGrid.rf,
            metric=metric)
fit.rf

plot(varImp(fit.rf)) # important variables used by random forest model

# predictions
rf.predictions <- predict(fit.rf, type='raw')

rf.predictionsProb <- round(predict(fit.rf, type='prob'), 8) # need this for ROC curve

# confusion matrix of model trained by cross-validation
rf.confusion <- confusionMatrix(data=rf.predictions,
                        reference=cleaned.train$fake,
                        positive='yes',
                        mode='everything')
rf.confusion



###### Performance Measure ######
###### Apparent Performance of each model ######
# summarize accuracy of models from cross-validation
```

```
results <- resamples(list('Decision Tree'=fit.rpart,
                 'Logistic Regression'=fit.lr,
                 'Random Forest'=fit.rf,
                 'kNN'=fit.knn))
summary(results)


# compare accuracy of models (Apparent Performance)
dotplot(results)


# accuracy
metrics.accuracy <- round(rbind(lr.confusion$overall[1],
                     knn.confusion$overall[1],
                     rpart.confusion$overall[1],
                     rf.confusion$overall[1]), 3)
# kappa
metrics.kappa <- round(rbind(lr.confusion$overall[2],
                   knn.confusion$overall[2],
                   rpart.confusion$overall[2],
                   rf.confusion$overall[2]), 3)
# f1 score
metrics.f1 <- round(rbind(lr.confusion$byClass[7],
                 knn.confusion$byClass[7],
                 rpart.confusion$byClass[7],
                       rf.confusion$byClass[7]), 3)
# metrics table
metrics <- data.frame(cbind(metrics.accuracy, metrics.kappa, metrics.f1))

# rename the rows to the names of each model
rownames(metrics) <- c('Logistic Regression',
                     'kNN',
                     'Decision Tree',
                     'Random Forest')

# barplot of the metrics of each model
# accuracy
p.accuracy <- barplot(metrics$Accuracy,
                    main='Accuracy',
                    xlab='Models',
                    ylab='Accuracy',
                    names.arg=rownames(metrics),
                    ylim=c(0, 1.1))
text(x=p.accuracy, y=metrics$Accuracy, labels=metrics$Accuracy, pos=3, cex=1.5,
col='blue')

# Kappa
p.kappa <- barplot(metrics$Kappa,
            main='Kappa',
            xlab='Models',
            ylab='Kappa',
            names.arg=rownames(metrics),
            ylim=c(0, 1.1))
text(x=p.kappa, y=metrics$Kappa, labels=metrics$Kappa, pos=3, cex=1.5, col='blue')


# F1 score
p.f1 <- barplot(metrics$F1,
          main='F1 Score',
          xlab='Models',
          ylab='F1',
          names.arg=rownames(metrics),
          ylim=c(0, 1.1))
text(x=p.f1, y=metrics$F1, labels=metrics$F1, pos=3, cex=1.5, col='blue')


###### ROC ######
# decision tree
roc(cleaned.train$fake, rpart.predictionsProb$yes, plot = T,
    legacy.axes=T, xlab="False Positive Rate", ylab="True Positive Rate",
    main="ROC", col="cornflowerblue", print.auc=T, print.auc.y=0.43)
# add logistic regression
plot.roc(cleaned.train$fake, lr.predictionsProb$yes, add=T,
        col="lightcoral", print.auc=T, print.auc.y=0.37)
# add rf
plot.roc(cleaned.train$fake, rf.predictionsProb$yes, add=T,
        col="goldenrod1", print.auc=T, print.auc.y=0.31)
# add knn
plot.roc(cleaned.train$fake, knn.predictionsProb$yes, add=T,
        col="darkolivegreen4", print.auc=T, print.auc.y=0.25)
# add legend
legend("bottomright",  legend=c("Decision  Tree","Logistic  Regression","Random
Forest","kNN"),
      col=c("cornflowerblue","lightcoral","darkolivegreen4"), lwd=4, cex = 0.85,
      text.col=c("cornflowerblue","lightcoral","goldenrod1","darkolivegreen4"), bty="n")

###### Lift Chart ######
## Generate the test set results
lift_results <- data.frame(fake = cleaned.test$fake)
lift_results$lr <- predict(fit.lr, newdata=cleaned.test, type = "prob")[, 'no'] # only use the
probability of 'fake' class (probability observation being fake)
lift_results$knn <- predict(fit.knn, newdata=cleaned.test, type = "prob")[, 'no']
lift_results$rpart <- predict(fit.rpart, newdata=test, type = "prob")[, 'no']
lift_results$rf <- predict(fit.rf, newdata=test, type = 'prob')[, 'no']
head(lift_results)
# plot lift chart
```

```
trellis.par.set(caretTheme())
lift_obj <- lift(fake ~ lr + knn + rpart + rf, data = lift_results)
plot(lift_obj, values = 60, auto.key = list(columns = 4,
                                      lines = TRUE,
                                      points = FALSE))


###### Test set ######
# Logistic Regression
test.lr.predictions <- predict(fit.lr, newdata=cleaned.test, type='raw')

test.lr.Prob <- round(predict(fit.lr, newdata=cleaned.test, type='prob'), 8) # need this for ROC curve

# confusion matrix of Logistic Regression
test.lr.confusion <- confusionMatrix(data=test.lr.predictions,
                                reference=cleaned.test$fake,
                                positive='yes',
                                mode='everything')
test.lr.confusion

# knn
test.knn.predictions <- predict(fit.knn, newdata=cleaned.test, type='raw')
test.knn.Prob <- round(predict(fit.knn, newdata=cleaned.test, type='prob'), 8) # need this
for ROC curve

# confusion matrix of kNN
test.knn.confusion <- confusionMatrix(data=test.knn.predictions,
                                      reference=cleaned.test$fake,
                                      positive='yes',
                                      mode='everything')
test.knn.confusion

# Decision Tree
test.rpart.predictions <- predict(fit.rpart, newdata=test, type='raw')

test.rpart.Prob <- round(predict(fit.rpart, newdata=test, type='prob'), 8) # need this for ROC
curve

# confusion matrix of Decision Tree
test.rpart.confusion <- confusionMatrix(data=test.rpart.predictions,
                                        reference=test$fake,
                                        positive='yes',
                                        mode='everything')
test.rpart.confusion
```

```
# Random Forest

test.rf.predictions <- predict(fit.rf, newdata=test, type='raw')


test.rf.Prob <- round(predict(fit.rf, newdata=test, type='prob'), 8) # need this for ROC curve


# confusion matrix of Decision Tree

test.rf.confusion <- confusionMatrix(data=test.rf.predictions,

                                    reference=test$fake,

                                    positive='yes',

                                    mode='everything')

test.rf.confusion
```

```
# ###### Decision Boundaries of each model for visualization ######
# # Put everything in functions for plotting decision boundary of each model
# nbp <- 1000 # resolution for plotting decision line. The higher the smoother the line
# PredA <- seq(min(cleaned.train$X.follows), max(cleaned.train$X.follows), length = nbp)
# PredB <- seq(min(cleaned.train$X.followers), max(cleaned.train$X.followers), length =
nbp)
# Grid <- expand.grid(X.follows = PredA, X.followers = PredB)
#
# # function to plot the scatterplot and decision boundary line
# PlotGrid <- function(pred,title) {
#
#   pts <- (ggplot(data = cleaned.train, aes(x = X.follows, y = X.followers,
#                                  color = fake)) +
#           geom_contour(data = cbind(Grid, fake = pred), aes(z = as.numeric(fake)),
#                        color = "red", breaks = c(1.5)) +
#           geom_point(size = 4, alpha = .5) +
#           ggtitle("Decision boundary") +
#           theme(legend.text = element_text(size = 10))) +
#     scale_x_continuous() +
#     scale_y_continuous()

#

#   grid.arrange(pts, top = textGrob(title, gp = gpar(fontsize = 20)))

#

# }

#

# # function to train models with 10-fold cv repeats 5 times

# seed <- 1234

# folds <- 10

# repeats <- 5

# control <- trainControl(method = "repeatedcv",

#                         number = folds,

#                         repeats = repeats)

#

# # function to store accuracy of each iteration in cross-validation

# accuracy <- function(Model, Name) {

#     accuracy.df  <-  data.frame(t(postResample(predict(Model,  newdata  =  cleaned.train),
cleaned.train[["fake"]])),

#                     Resample = "None", model = Name)

#   rbind(accuracy.df, data.frame(Model$resample, model = Name))

# }

#

# accuracy.df <- data.frame()
```

```
#
# # function to train and plot decision boundary of each model in 1 step
# train.display <- function (accuracy.df, Name, Formula, Method, ...) {
#   set.seed(seed)
#   Model <- train(as.formula(Formula), data = cleaned.train, method = Method, trControl
= control, ...)
#   Pred <- predict(Model, newdata = Grid)
#   PlotGrid(Pred, Name)
#   accuracy.df <- rbind(accuracy.df, accuracy(Model, Name))
# }
#
#
#
#
#
#
#
#
#
# ###### Train and Plot decision boundary of each model ######
# ###### (only using 2 variables for showcasing - cannot plot more than 3 variables)
######
#
```

```
# # Logistic Regression
# lr.boundary <- train.display(accuracy.df, "Logistic Regression", "fake ~ X.follows+X.followers", "glm")
#
# # Decision Tree
# rpart.boundary <- train.display(accuracy.df, "Decision Tree", "fake ~ X.follows+X.followers", "rpart")
#
# # kNN
# acc.kNN <- data.frame()
# kNN <- c(3, 5, 7, 9)
# for (k in kNN) {
#   acc.kNN <- train.display(acc.kNN, sprintf("k-NN with k=%i", k),
#                       "fake ~ X.follows+X.followers", "knn", tuneGrid = data.frame(k = c(k)))
# }
#
# # Random Forest
# rf.boundary <- train.display(accuracy.df, "Random Forest", "fake ~ X.follows+X.followers", "rf")
#
```