Predictive Analytics for a Bank Marketing Campaign

Predictive Modeling Report

Robert Maheux

March 11 2019

Contents

[1. Introduction 3](#_Toc3126505)

[2. General Assumptions 3](#_Toc3126506)

[3. Predictive Models 4](#_Toc3126507)

[3.1 Logistic Regression 4](#_Toc3126508)

[3.2 Random Decision Forest 5](#_Toc3126509)

[3.3 K-Nearest Neighbours 6](#_Toc3126510)

[3.4 Support Vector Classification 6](#_Toc3126511)

[4. Performance Metrics 7](#_Toc3126512)

[4.1 Confusion Matrix 7](#_Toc3126513)

[4.2 Accuracy Score 9](#_Toc3126514)

[4.3 Precision Score 10](#_Toc3126515)

[4.4 Recall Score 11](#_Toc3126516)

[4.5 F1 Score 12](#_Toc3126517)

[4.6 Receiver Operating Characteristics Curve and Area Under the Curve 13](#_Toc3126518)

[5. Model Evaluation and Comparison 16](#_Toc3126519)

[6. Summary 17](#_Toc3126520)

[Appendix A – Results Obtained with Various Training/Testing Sets 18](#_Toc3126521)

[Appendix B – Listing of Python Commands Used for this Report 21](#_Toc3126522)

# 1. Introduction

This report describes the steps to build and test four different types of classification models. The Bank Marketing dataset will be used to make predictions. The performance of the models will be evaluated and compared using a combination of six metrics.

# 2. General Assumptions

The algorithms for the models assume the following:

* The categorical features have been converted into numerical features.
* The missing values have been removed or imputed.
* The features have the same weight. The distribution of the features has been normalized or standardized so that they are on equal footing.
* The outcome variable isn’t heavily unbalanced.
* There is no ordering to the data. All observations are to be treated the same.
* There are limited dependencies between the features in the dataset (i.e. - there are no strong correlations between the features).
* The split between the training and testing data was performed randomly.
* The data in the testing set is independent of the data in the training set.
* The distance between observations is assumed to be the *Euclidean* one.
* It’s acceptable to have the classifier algorithms weight the false positives and false negatives equally.
* The future will continue to be like the past. Of course, this isn’t realistic since many factors would cause the clients’ behaviour to change over time. The performance of the model will need to be reassessed on a regular basis.

# 3. Predictive Models

This section describes the four predictive models that will be used as well as the steps to build them.

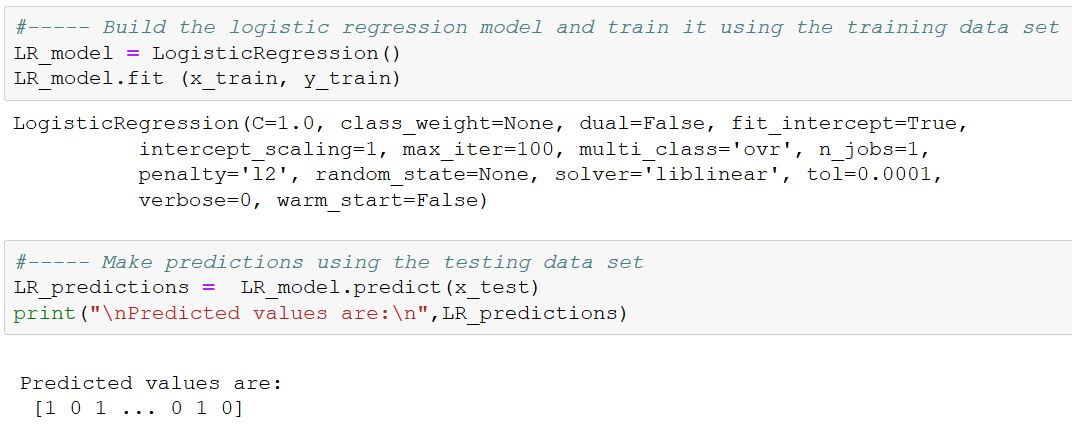
The models were selected because they are suitable for classification problems. They are:

1. Logistic Regression
2. Random Decision Forest
3. K-Nearest Neighbours
4. Support Vector Classification

## 3.1 Logistic Regression

The logistic regression algorithm accepts the features of an observation and predicts the probability that the output belongs to a certain class.  It is used to estimate discrete binary values (such as 0/1, yes/no, true/false) based on given set of independent variables. It plots the training observations using a *sigmoid* which is a mathematical function with a characteristic "S"-shaped curve. This allows it to determine a boundary between two different classes of outcomes. A new observation is plotted and its outcome is predicted depending on which side of the boundary it was mapped.

Build the Logistic Regression model, train it using the training data set, then make predictions using the testing data set:

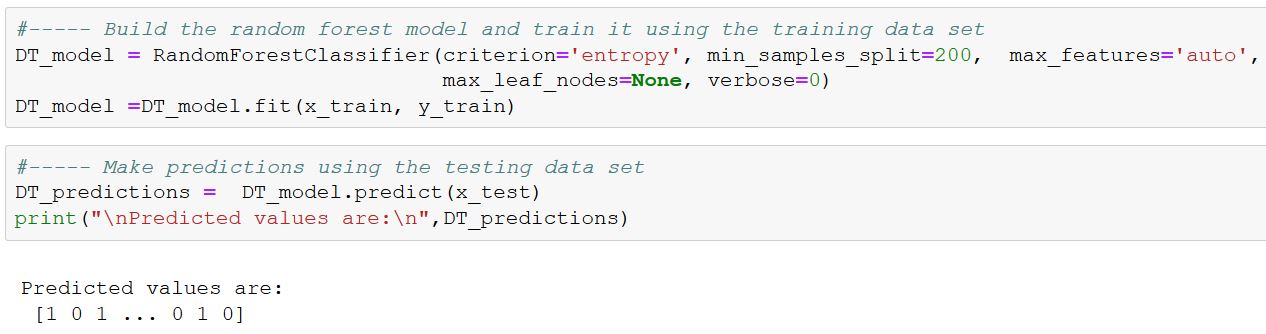


## 3.2 Random Decision Forest

A decision tree breaks down the training dataset into smaller and smaller subsets and builds a set of *if-then-else* decision rules in the form of a tree structure. The final result is a tree with several decision nodes leading down to leaf nodes. Leaf nodes represent a classification or outcome.

Decision trees are prone to overfitting but this can be corrected by using a collection of multiple decision trees called a *Random Decision Forest*. This classification method operates by constructing a multitude of decision trees with different predictions and combining the results of those individual trees to give the final outcome.

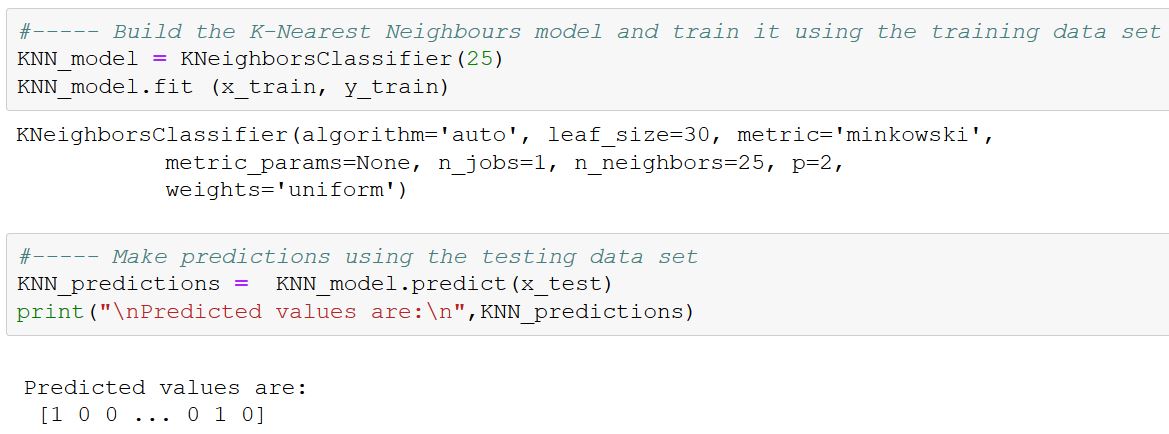
Build the Random Decision Forest model, train it using the training data set, then make predictions using the testing data set:



## 3.3 K-Nearest Neighbours

This algorithm compares a new observation with all the observations in the training set and finds the K observations that most closely resemble it. It then takes the majority of the outcomes of these “nearest neighbours” to be the predicted outcome for the new observation.

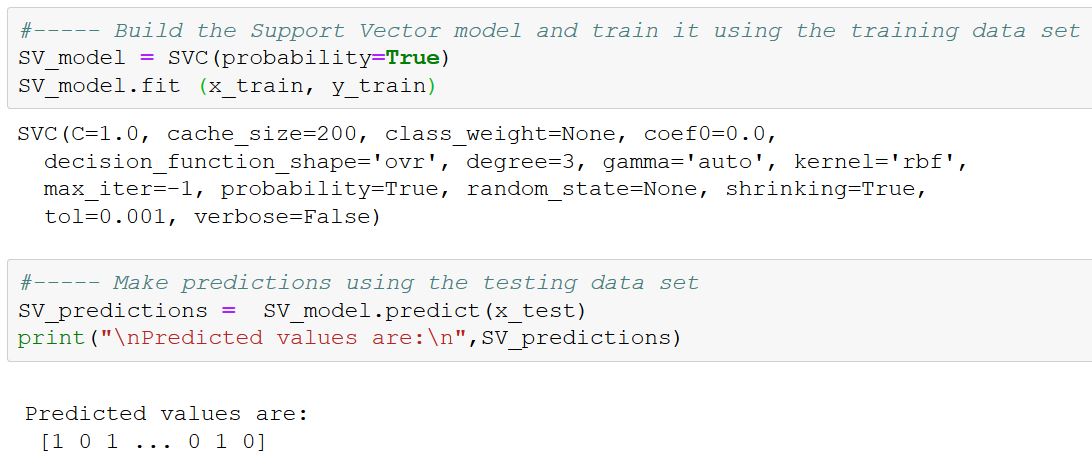
Build the K-Nearest Neighbours model (using K = 25 nearest neighbours), train it using the training data set, then make predictions using the testing data set:



## 3.4 Support Vector Classification

This algorithm creates a representation of the training observations as points in space. They are mapped and are divided into two categories by a clear gap that is as wide as possible. New observations are then mapped into that same space and predicted to belong to one of the categories based on which side of the gap they fall.

Build the Support Vector model, train it using the training data set, then make predictions using the testing data set:

****

# 4. Performance Metrics

The performance of the models will be evaluated and compared using a combination of six metrics that are suitable for classification algorithms.

The following terms and acronyms will be used in this section of the report:

**True Positive (TP)**: Observation correctly classified as ‘1’.

The model correctly predicted that the client would open a term bank account.

**False Positive (FP)**: Observation misclassified as ‘1’.

The model predicted that the client would open a term bank account, but this prediction was wrong.

**True Negative (TN)**: Observation correctly classified as ‘0’.

The model correctly predicted that the client would not open a term bank account.

**False Negative (FN)**: Observation misclassified as ‘0’.

The model predicted that the client would not open a term bank account, but this prediction was wrong.

## 4.1 Confusion Matrix

A Confusion Matrix is a clean and unambiguous way to summarize the prediction results of a classification model.

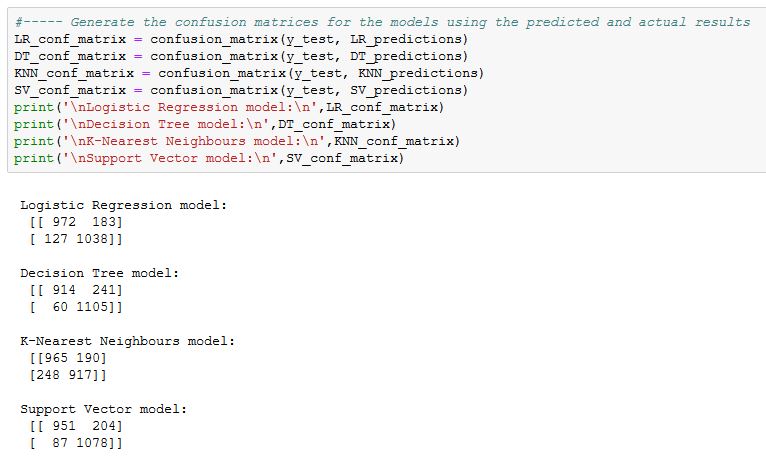
For a binary classification problem, the table has 2 rows and 2 columns. Across the top are the *observed* class labels and down the side are the *predicted* class labels. Each cell contains the number of predictions made by the model that fall into that cell.

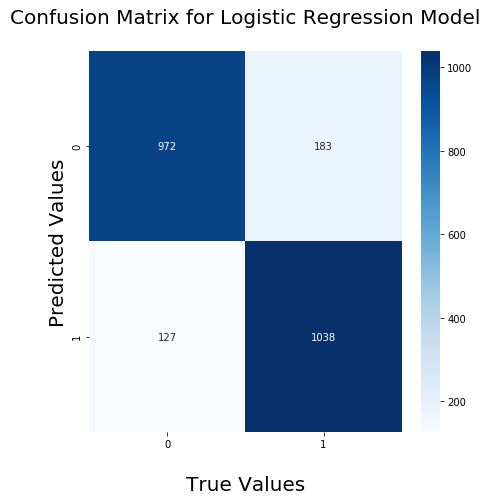
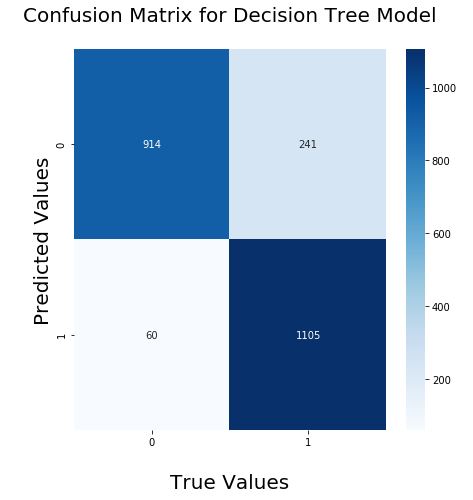
|  |  |  |
| --- | --- | --- |
|  | **Observed Positive** | **Observed Negative** |
| **Predicted Positive** | **TP** | **FP** |
| **Predicted Negative** | **FN** | **TN** |

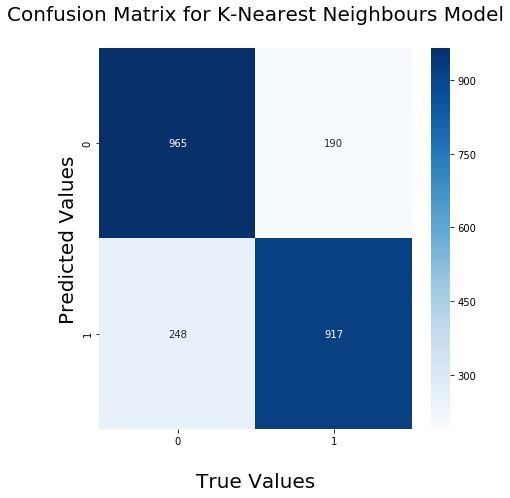
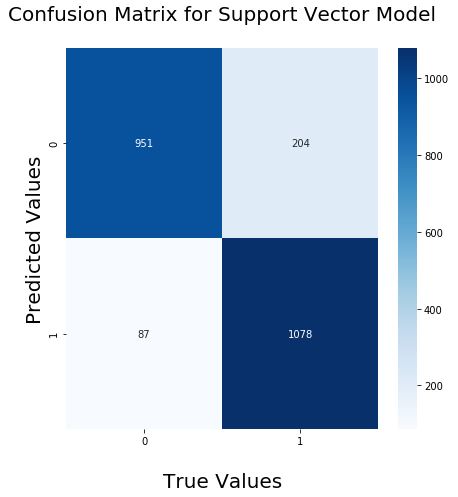
The correct predictions (TP and TN) are placed in the green cells and the incorrect predictions (FP and FN) are placed in the red cells.

These values are used to compute many of the metrics that measure the performance of the models.

The following screen snapshots show the steps taken to generate the confusion matrices for the models as well as their corresponding heat maps.



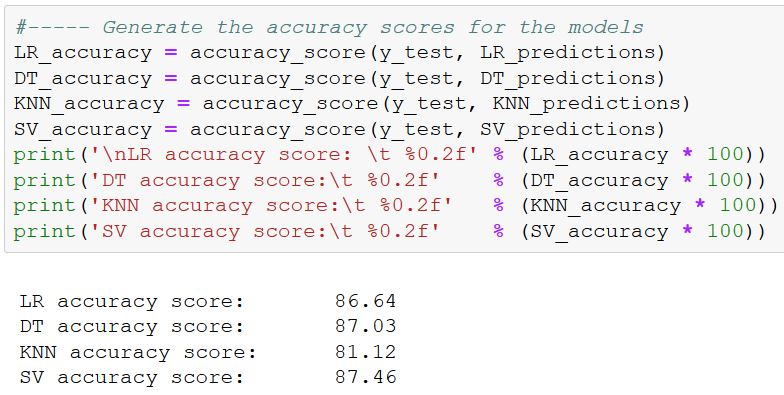
 

## 4.2 Accuracy Score

The Accuracy score is the simplest and most commonly used performance metric. It is simply the ratio of correct predictions divided by the total number of predictions:

The Accuracy score is useful when the goal is to produce a subset of high-confidence predictions. There may be some scenarios in which a model might have really good accuracy but fails to perform well for new unseen data points. This has been mitigated by ensuring that the distribution of the outcome variable has been balanced.

Generate the Accuracy scores for the models:

****

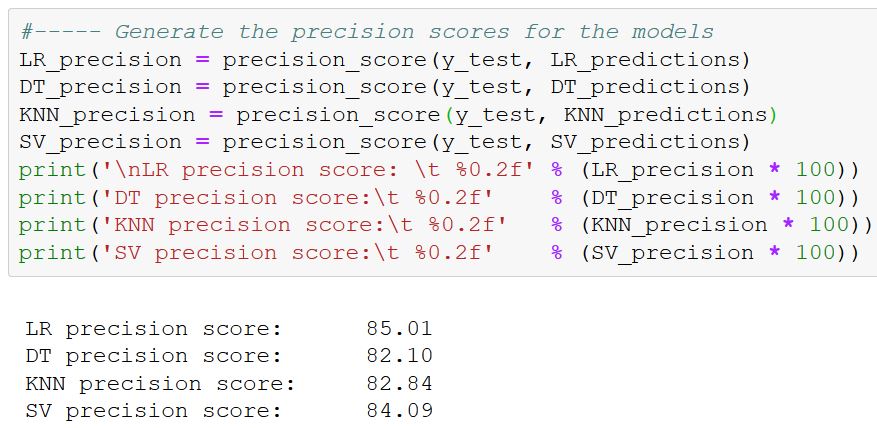
## 4.3 Precision Score

The Precision **score** tells us the proportion of clients that were correctly predicted to open a term account compared to *the total number that were predicted to do so*. It is the ratio of true positives to all predicted positives:

The Precision score can be thought of as a measure of a classifier model’s exactness. A low Precision score would indicate a large number of false positives.

The business objective is for the bank to save time and money by focusing its’ telemarketing efforts on clients most likely to open a term bank account. A low Precision score (i.e. a large number of false positives) means that the marketing team would waste a lot of time contacting clients that aren’t likely to open a bank account. Given the bank’s business objective, the Precision score is an important measure to use.

Generate the Precision scores for the models:



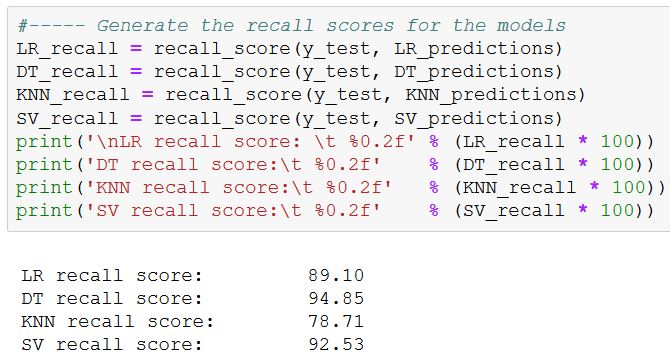
## 4.4 Recall Score

The Recall **score** tells us the proportion of clients that were correctly predicted to open a term account compared to *the total number that would actually do so*. It is the ratio of true positives to all *actual* positives:

The Recall score can be thought of as a measure of a classifier model’s completeness. A low Recall score would indicate a large number of False Negatives.

The business objective is for the bank to save time and money by focusing its’ telemarketing efforts on clients most likely to open a term bank account. A low Recall score (i.e. a large number of false negatives) means that the marketing team wouldn’t be informed about several clients that are likely to open a bank account. The model would erroneously direct the marketing team away from new business opportunities. Given the bank’s business objective, the Recall score is an important measure to use.

Generate the Recall scores for the models:

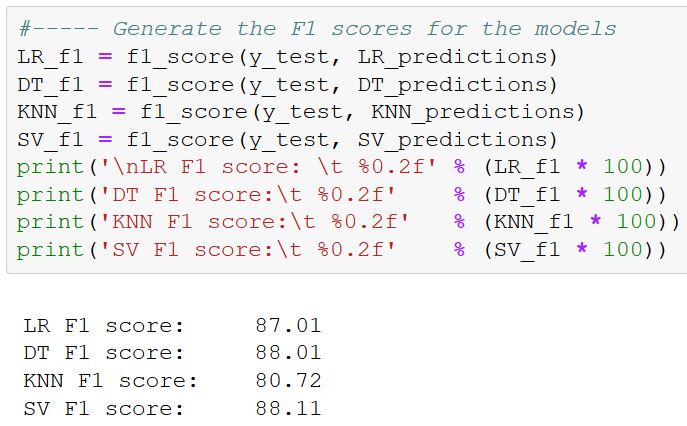
****

## 4.5 F1 Score

The F1 Score is the harmonic average of the precision and recall scores where both measures are weighted equally:

It conveys the balance between the precision and the recall scores. Given the bank’s business objective, the Precision and Recall scores are equally important so the F1 score will provide a weighted harmonic mean of both values.

Generate the F1 scores for the models:

****

## 4.6 Receiver Operating Characteristics Curve and Area Under the Curve

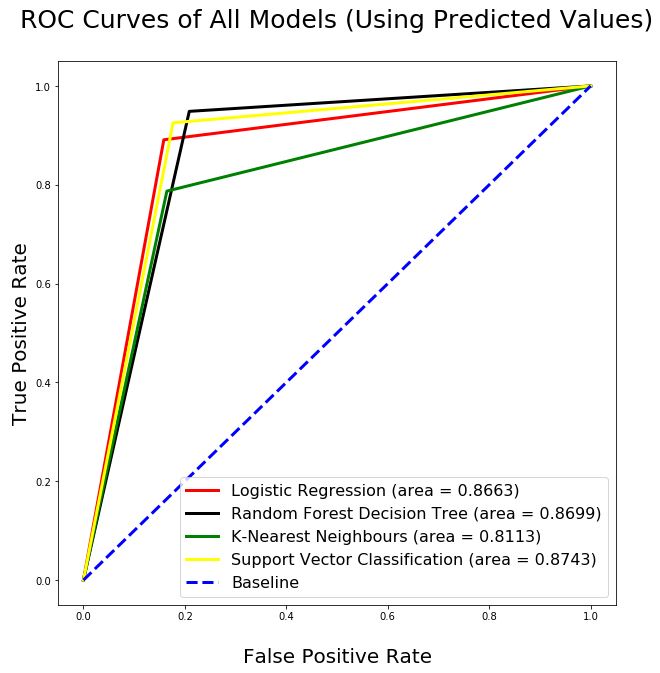
TheReceiver Operating Characteristics(ROC) curve provides a visual comparison of classification models. It shows the trade-off between the true positive and false positive rates.

The curves are plotted with the true positive rates on the Y axis and the false positive rates on the X axis. The top left corner of the plot is the *ideal* point - a false positive rate of zero, and a true positive rate of one. This means that a larger area under the curve (AUC) is usually better. The AUC summarizes the curve information in one number and can be used as a measure of the accuracy of the model.

The steepness of the ROC curve is also important since it is ideal to maximize the true positive rate while minimizing the false positive rate.

Calculate the values for the ROC and AUC, and plot the curves for each model:

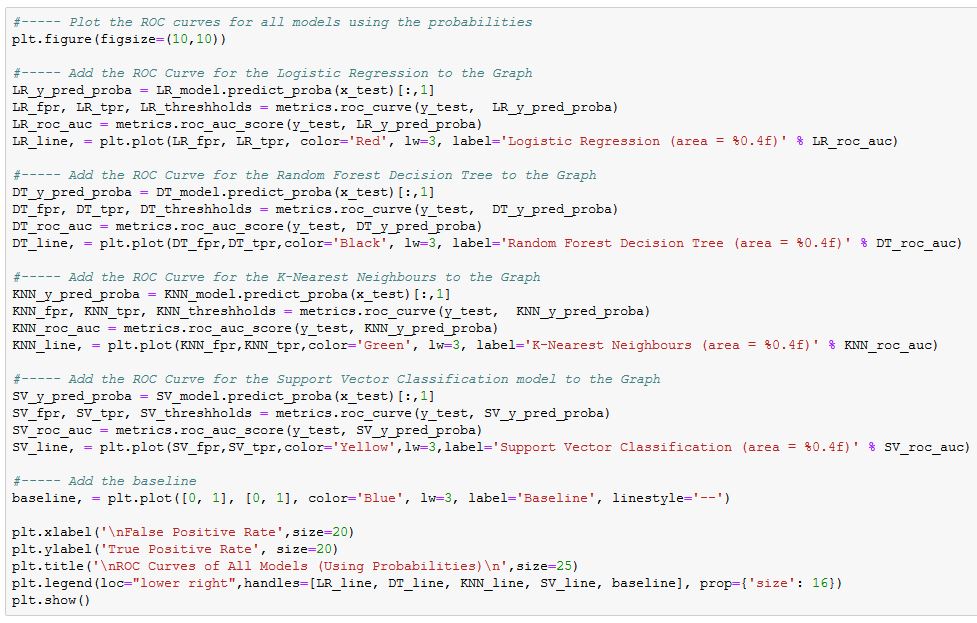


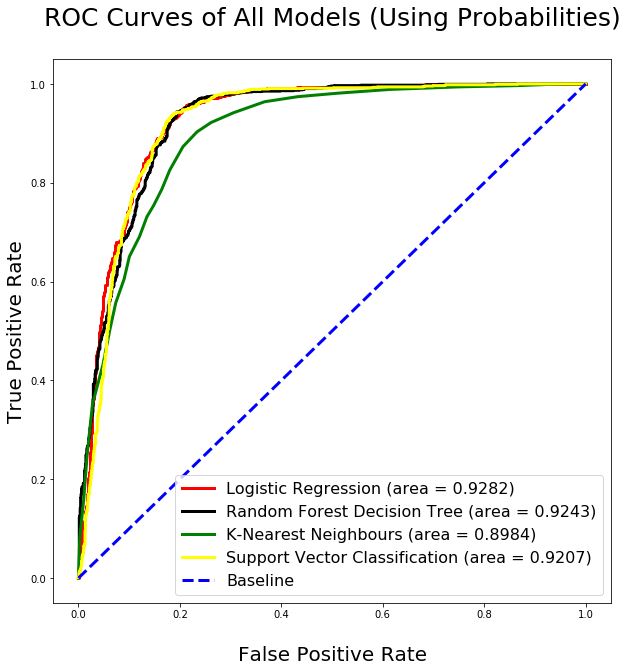


The number of points on the graph reflects the number of unique values in the input. The plotted lines have a triangular shape because the input vectors have only 2 unique values for the predictions.

When performing classifications, it is sometimes desirable to predict not only the outcome values but also the associated probabilities of predicted values (for each observation). These probabilities provide confidence on the predictions and can be plotted in the same way, as shown below. The area under the ROC curve that was created using the probabilities is a valuable metric to evaluate the performance of a model.

Calculate the values for the ROC and AUC using the probabilities, and plot the curves for each model:





# 5. Model Evaluation and Comparison

It’s difficult to compare models when several metrics are used because different metrics point to different models as being the “best one”. The following table is a compilation of the various scores obtained by each model. Identifying the one with the highest number of “best” and “second-best” scores can provide an overall ranking of each model. The highest scores for each column are shaded in green. The second highest scores are shaded in yellow.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  | **Accuracy Score** | **Precision Score** | **Recall Score** | **F1**  **Score** | **AUC**  **Using Predicted Values** | **AUC**  **Using Probabilities** |
| **Logistic Regression** | 86.64 | 85.01 | 89.10 | 87.01 | .8663 | .9282 |
| **Random Decision Forest** | 87.03 | 82.10 | 94.85 | 88.01 | .8699 | .9243 |
| **K-Nearest Neighbours** | 81.12 | 82.84 | 78.71 | 80.72 | .8113 | .8984 |
| **Support Vector** | 87.46 | 84.09 | 92.53 | 88.11 | .8743 | .9207 |

The overall scores in the table indicate that the Support Vector model is a strong performer, closely followed by the Random Forest model. The worst performer is the model using the K-Nearest Neighbours algorithm.

When the main dataset is split into training and testing datasets, the observations are divided randomly. By performing a new random split and re-executing the models on the new training and testing sets, different results are obtained. **Appendix A** lists the results for the same models built on 10 different versions of the training and testing sets. The Support Vector model was the best one in 70% of the cases while the Random Forest model was the best one in the remaining 30%. The K-Nearest Neighbours model was consistently the worst performer.

An alternative comparison method would be to focus on the metrics that are the most relevant for the problem being addressed. For the Bank Marketing problem, it’s extremely important for the model to have high Precision and Recall scores. A low Precision score would result in the marketing team wasting time contacting clients that are *false positives*. A low Recall score would mean that the marketing team would not be directed towards many clients that are *true positives*.

Since the F1 score is a harmonic average of the Precision and Recall scores, it may be acceptable to focus on that single metric to compare the performance of the models. The F1 score indicates that the Support Vector and Random Forest models are almost equal, with a slight edge in favour of the Support Vector model. Again, the K-Nearest Neighbours model was consistently the worst performer.

There are some techniques that can be used to see if one of the models can be improved so that it consistently outperforms the others, such as:

* Tuning the model parameters (amount of tree pruning, number of nearest neighbours used…)
* Removing some of the independent variables being used
* Using boosting algorithms

# 6. Summary

This report outlined the steps to build and test four different types of classification models. The Bank Marketing dataset was used to make predictions.

The performance of the models was evaluated and compared using a combination of six metrics that are suitable for classification algorithms.

The Support Vector and Random Forest models performed almost equally, with a slight edge in favour of the Support Vector model. The model that used the K-Nearest Neighbours algorithm was consistently the worst performer. Additional techniques should be attempted to see if one of the models can be improved so that it consistently outperforms the others

# Appendix A – Results Obtained with Various Training/Testing Sets

When the main dataset is split into training and testing datasets, the observations are divided randomly. By performing a new random split and re-executing the models on the new training and testing sets, different results are obtained.

This appendix lists the results obtained when the models were built on 10 versions of the training and testing sets.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  | **Accuracy Score** | **Precision Score** | **Recall Score** | **F1**  **Score** | **AUC**  **Using Predicted Values** | **AUC**  **Using Probabilities** |
| **Logistic Regression** | 87.28 | 86.86 | 87.98 | 87.42 | .8728 | .9397 |
| **Random Decision Forest** | 87.84 | 83.88 | 93.82 | 88.57 | .8782 | .9332 |
| **K-Nearest Neighbours** | 82.50 | 84.85 | 79.31 | 81.99 | .8251 | .9087 |
| **Support Vector** | 88.19 | 85.22 | 92.53 | 88.72 | .8817 | .9344 |

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  | **Accuracy Score** | **Precision Score** | **Recall Score** | **F1**  **Score** | **AUC**  **Using Predicted Values** | **AUC**  **Using Probabilities** |
| **Logistic Regression** | 87.72 | 88.06 | 87.38 | 87.72 | .8772 | .9473 |
| **Random Decision Forest** | 89.31 | 85.08 | 95.45 | 89.97 | .8928 | .9400 |
| **K-Nearest Neighbours** | 83.79 | 86.29 | 80.52 | 83.30 | .8381 | .9165 |
| **Support Vector** | 88.71 | 86.68 | 91.59 | 89.07 | .8869 | .9437 |

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  | **Accuracy Score** | **Precision Score** | **Recall Score** | **F1**  **Score** | **AUC**  **Using Predicted Values** | **AUC**  **Using Probabilities** |
| **Logistic Regression** | 87.76 | 87.17 | 88.67 | 87.91 | .8775 | .9403 |
| **Random Decision Forest** | 88.53 | 83.92 | 95.45 | 89.32 | .8850 | .9359 |
| **K-Nearest Neighbours** | 82.54 | 84.67 | 79.66 | 82.09 | .8256 | .9095 |
| **Support Vector** | 87.72 | 85.03 | 91.67 | 88.23 | .8770 | .9294 |

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  | **Accuracy Score** | **Precision Score** | **Recall Score** | **F1**  **Score** | **AUC**  **Using Predicted Values** | **AUC**  **Using Probabilities** |
| **Logistic Regression** | 87.41 | 86.59 | 88.67 | 87.62 | .8741 | .9398 |
| **Random Decision Forest** | 87.03 | 84.18 | 91.33 | 87.61 | .8701 | .9334 |
| **K-Nearest Neighbours** | 82.76 | 85.38 | 79.23 | 82.19 | .8277 | .9075 |
| **Support Vector** | 87.50 | 84.53 | 91.93 | 88.08 | .8748 | .9319 |

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  | **Accuracy Score** | **Precision Score** | **Recall Score** | **F1**  **Score** | **AUC**  **Using Predicted Values** | **AUC**  **Using Probabilities** |
| **Logistic Regression** | 86.34 | 85.27 | 87.98 | 86.61 | .8633 | .9332 |
| **Random Decision Forest** | 87.07 | 82.69 | 93.91 | 87.94 | .8704 | .9281 |
| **K-Nearest Neighbours** | 81.98 | 83.32 | 80.17 | 81.71 | .8199 | .8984 |
| **Support Vector** | 87.16 | 83.74 | 92.36 | 87.84 | .8713 | .9230 |

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  | **Accuracy Score** | **Precision Score** | **Recall Score** | **F1**  **Score** | **AUC**  **Using Predicted Values** | **AUC**  **Using Probabilities** |
| **Logistic Regression** | 87.89 | 87.52 | 88.50 | 88.01 | .8789 | .9412 |
| **Random Decision Forest** | 87.46 | 84.63 | 91.67 | 88.01 | .8744 | .9322 |
| **K-Nearest Neighbours** | 83.84 | 86.04 | 80.94 | 83.41 | .8385 | .9149 |
| **Support Vector** | 88.45 | 85.34 | 92.96 | 88.99 | .8843 | .9349 |

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  | **Accuracy Score** | **Precision Score** | **Recall Score** | **F1**  **Score** | **AUC**  **Using Predicted Values** | **AUC**  **Using Probabilities** |
| **Logistic Regression** | 86.81 | 85.70 | 88.50 | 87.08 | .8680 | .9358 |
| **Random Decision Forest** | 87.59 | 82.36 | 95.79 | 88.57 | .8755 | .9263 |
| **K-Nearest Neighbours** | 82.03 | 84.06 | 79.23 | 81.57 | .8204 | .9069 |
| **Support Vector** | 87.93 | 84.54 | 92.96 | 88.55 | .8791 | .9337 |

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  | **Accuracy Score** | **Precision Score** | **Recall Score** | **F1**  **Score** | **AUC**  **Using Predicted Values** | **AUC**  **Using Probabilities** |
| **Logistic Regression** | 87.93 | 87.40 | 88.76 | 88.07 | .8793 | .9459 |
| **Random Decision Forest** | 86.51 | 83.92 | 90.47 | 87.07 | .8649 | .9274 |
| **K-Nearest Neighbours** | 83.15 | 85.44 | 80.09 | 82.68 | .8316 | .9124 |
| **Support Vector** | 88.41 | 85.33 | 92.88 | 88.94 | .8839 | .9412 |

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  | **Accuracy Score** | **Precision Score** | **Recall Score** | **F1**  **Score** | **AUC**  **Using Predicted Values** | **AUC**  **Using Probabilities** |
| **Logistic Regression** | 87.63 | 87.14 | 88.41 | 87.77 | .8763 | .9433 |
| **Random Decision Forest** | 88.15 (tie) | 83.11 | 95.88 | 89.04 | .8811 | .9363 |
| **K-Nearest Neighbours** | 82.72 | 85.05 | 79.57 | 82.22 | .8273 | .9110 |
| **Support Vector** | 88.15 (tie) | 85.54 | 91.93 | 88.62 | .8813 | .9398 |
|  | **Accuracy Score** | **Precision Score** | **Recall Score** | **F1**  **Score** | **AUC**  **Using Predicted Values** | **AUC**  **Using Probabilities** |
| **Logistic Regression** | 87.72 | 87.16 | 88.58 | 87.87 | .8771 | .9459 |
| **Random Decision Forest** | 88.36 | 84.24 | 94.51 | 89.08 | .8834 | .9372 |
| **K-Nearest Neighbours** | 82.97 | 85.39 | 79.74 | 82.47 | .8299 | .9115 |
| **Support Vector** | 88.62 | 85.84 | 92.62 | 89.10 | .8860 | .9395 |

# Appendix B – Listing of Python Commands Used for this Report

**Logistic Regression**

#----- Build the logistic regression model and train it using the training data sets

LR\_model = LogisticRegression()

LR\_model.fit (x\_train, y\_train)

#----- Make predictions using the testing data set

LR\_predictions = LR\_model.predict(x\_test)

print("\nPredicted values are:\n",LR\_predictions)

**Random Forest (Decision Tree)**

#----- Build the random forest model and train it using the training data sets

DT\_model = RandomForestClassifier(criterion='entropy', min\_samples\_split=200, max\_features='auto',

max\_leaf\_nodes=None, verbose=0)

DT\_model =DT\_model.fit(x\_train, y\_train)

#----- Make predictions using the testing data set

DT\_predictions = DT\_model.predict(x\_test)

print("\nPredicted values are:\n",DT\_predictions)

**K-Nearest Neighbours**

#----- Build the K-Nearest Neighbours model and train it using the training data sets

KNN\_model = KNeighborsClassifier(25)

KNN\_model.fit (x\_train, y\_train)

#----- Make predictions using the testing data set

KNN\_predictions = KNN\_model.predict(x\_test)

print("\nPredicted values are:\n",KNN\_predictions)

**Support Vector Classification**

#----- Build the Support Vector model and train it using the training data sets

SV\_model = SVC(probability=True)

SV\_model.fit (x\_train, y\_train)

#----- Make predictions using the testing data set

SV\_predictions = SV\_model.predict(x\_test)

print("\nPredicted values are:\n",SV\_predictions)

**Generate the Metrics for the Models**

#----- Generate the accuracy scores for the models

LR\_accuracy = accuracy\_score(y\_test, LR\_predictions)

DT\_accuracy = accuracy\_score(y\_test, DT\_predictions)

KNN\_accuracy = accuracy\_score(y\_test, KNN\_predictions)

SV\_accuracy = accuracy\_score(y\_test, SV\_predictions)

print('\nLR accuracy score: \t %0.2f' % (LR\_accuracy \* 100))

print('DT accuracy score:\t %0.2f' % (DT\_accuracy \* 100))

print('KNN accuracy score:\t %0.2f' % (KNN\_accuracy \* 100))

print('SV accuracy score:\t %0.2f' % (SV\_accuracy \* 100))

#----- Generate the precision scores for the models

LR\_precision = precision\_score(y\_test, LR\_predictions)

DT\_precision = precision\_score(y\_test, DT\_predictions)

KNN\_precision = precision\_score(y\_test, KNN\_predictions)

SV\_precision = precision\_score(y\_test, SV\_predictions)

print('\nLR precision score: \t %0.2f' % (LR\_precision \* 100))

print('DT precision score:\t %0.2f' % (DT\_precision \* 100))

print('KNN precision score:\t %0.2f' % (KNN\_precision \* 100))

print('SV precision score:\t %0.2f' % (SV\_precision \* 100))

#----- Generate the recall scores for the models

LR\_recall = recall\_score(y\_test, LR\_predictions)

DT\_recall = recall\_score(y\_test, DT\_predictions)

KNN\_recall = recall\_score(y\_test, KNN\_predictions)

SV\_recall = recall\_score(y\_test, SV\_predictions)

print('\nLR recall score: \t %0.2f' % (LR\_recall \* 100))

print('DT recall score:\t %0.2f' % (DT\_recall \* 100))

print('KNN recall score:\t %0.2f' % (KNN\_recall \* 100))

print('SV recall score:\t %0.2f' % (SV\_recall \* 100))

#----- Generate the F1 scores for the models

LR\_f1 = f1\_score(y\_test, LR\_predictions)

DT\_f1 = f1\_score(y\_test, DT\_predictions)

KNN\_f1 = f1\_score(y\_test, KNN\_predictions)

SV\_f1 = f1\_score(y\_test, SV\_predictions)

print('\nLR F1 score: \t %0.2f' % (LR\_f1 \* 100))

print('DT F1 score:\t %0.2f' % (DT\_f1 \* 100))

print('KNN F1 score:\t %0.2f' % (KNN\_f1 \* 100))

print('SV F1 score:\t %0.2f' % (SV\_f1 \* 100))

#----- Generate the confusion matrices for the models using the predicted and actual results

LR\_conf\_matrix = confusion\_matrix(y\_test, LR\_predictions)

DT\_conf\_matrix = confusion\_matrix(y\_test, DT\_predictions)

KNN\_conf\_matrix = confusion\_matrix(y\_test, KNN\_predictions)

SV\_conf\_matrix = confusion\_matrix(y\_test, SV\_predictions)

print('\nLogistic Regression model:\n',LR\_conf\_matrix)

print('\nDecision Tree model:\n',DT\_conf\_matrix)

print('\nK-Nearest Neighbours model:\n',KNN\_conf\_matrix)

print('\nSupport Vector model:\n',SV\_conf\_matrix)

#----- Generate a heat map of the confusion matrix for the Logistic Regression model

LR\_heatmap = sns.heatmap(LR\_conf\_matrix, cmap="Blues", annot=True, fmt="g")

plt.title('\nConfusion Matrix for Logistic Regression Model\n',size=20)

plt.xlabel('\nTrue Values',size=20)

plt.ylabel('Predicted Values',size=20)

#----- Generate a heat map of the SNS confusion matrix for the Decision Tree model

DT\_heatmap = sns.heatmap(DT\_conf\_matrix, cmap="Blues", annot=True, fmt="g")

plt.title('\nConfusion Matrix for Decision Tree Model\n',size=20)

plt.xlabel('\nTrue Values',size=20)

plt.ylabel('Predicted Values',size=20)

#----- Generate a heat map of the confusion matrix for the K-Nearest Neighbours model

KNN\_heatmap = sns.heatmap(KNN\_conf\_matrix, cmap="Blues", annot=True, fmt="g")

plt.title('\nConfusion Matrix for K-Nearest Neighbours Model\n',size=20)

plt.xlabel('\nTrue Values',size=20)

plt.ylabel('Predicted Values',size=20)

#----- Generate a heat map of the confusion matrix for the Support Vector model

SV\_heatmap = sns.heatmap(SV\_conf\_matrix, cmap="Blues", annot=True, fmt="g")

plt.title('\nConfusion Matrix for Support Vector Model\n',size=20)

plt.xlabel('\nTrue Values',size=20)

plt.ylabel('Predicted Values',size=20)

#----- Compare the Receiver Operating Characteristic (ROC) Curves of All Models

#----- Calculate the true and false positive rates for the models

LR\_fpr, LR\_tpr, LR\_threshholds = roc\_curve(y\_test, LR\_predictions,drop\_intermediate=False)

DT\_fpr, DT\_tpr, DT\_threshholds = roc\_curve(y\_test, DT\_predictions, drop\_intermediate=False)

KNN\_fpr, KNN\_tpr, KNN\_threshholds = roc\_curve(y\_test, KNN\_predictions)

SV\_fpr, SV\_tpr, SV\_threshholds = metrics.roc\_curve(y\_test, SV\_predictions, drop\_intermediate=False)

#----- Calculate the area under the curve for the models

LR\_roc\_auc = metrics.auc(LR\_fpr, LR\_tpr)

DT\_roc\_auc = metrics.auc(DT\_fpr, DT\_tpr)

KNN\_roc\_auc = metrics.auc(KNN\_fpr, KNN\_tpr)

SV\_roc\_auc = metrics.auc(SV\_fpr, SV\_tpr)

#----- Plot the ROC curves for all models

plt.figure(figsize=(10,10))

#----- Add the ROC Curve for the Logistic Regression to the Graph

LR\_line, = plt.plot(LR\_fpr, LR\_tpr, color='Red', lw=3, label='Logistic Regression (area = %0.4f)' % LR\_roc\_auc)

#----- Add the ROC Curve for the Random Decision Forest to the Graph

DT\_line, = plt.plot(DT\_fpr, DT\_tpr, color='Black', lw=3, label='Random Decision Forest (area = %0.4f)' % DT\_roc\_auc)

#----- Add the ROC Curve for the K-Nearest Neighbours to the Graph

KNN\_line, = plt.plot(KNN\_fpr, KNN\_tpr, color='Green', lw=3, label='K-Nearest Neighbours (area = %0.4f)' % KNN\_roc\_auc)

#----- Add the ROC Curve for the Support Vector Classification model to the Graph

SV\_line, = plt.plot(SV\_fpr,SV\_tpr, color='Yellow',lw=3,label='Support Vector Classification (area = %0.4f)' % SV\_roc\_auc)

#----- Add the baseline

baseline, = plt.plot([0, 1], [0, 1], color='Blue', lw=3, label='Baseline', linestyle='--')

plt.xlabel('\nFalse Positive Rate',size=20)

plt.ylabel('True Positive Rate', size=20)

plt.title('\nROC Curves of All Models (Using Predicted Values)\n',size=25)

plt.legend(loc="lower right",handles=[LR\_line, DT\_line, KNN\_line, SV\_line, baseline], prop={'size': 16})

plt.show()

#----- Plot the ROC curves for all models using the probabilities

plt.figure(figsize=(10,10))

#----- Add the ROC Curve for the Logistic Regression to the Graph

LR\_y\_pred\_proba = LR\_model.predict\_proba(x\_test)[:,1]

LR\_fpr, LR\_tpr, LR\_threshholds = metrics.roc\_curve(y\_test, LR\_y\_pred\_proba)

LR\_roc\_auc = metrics.roc\_auc\_score(y\_test, LR\_y\_pred\_proba)

LR\_line, = plt.plot(LR\_fpr, LR\_tpr, color='Red', lw=3, label='Logistic Regression (area = %0.4f)' % LR\_roc\_auc)

#----- Add the ROC Curve for the Random Decision Forest to the Graph

DT\_y\_pred\_proba = DT\_model.predict\_proba(x\_test)[:,1]

DT\_fpr, DT\_tpr, DT\_threshholds = metrics.roc\_curve(y\_test, DT\_y\_pred\_proba)

DT\_roc\_auc = metrics.roc\_auc\_score(y\_test, DT\_y\_pred\_proba)

DT\_line, = plt.plot(DT\_fpr,DT\_tpr,color='Black', lw=3, label='Random Decision Forest (area = %0.4f)' % DT\_roc\_auc)

#----- Add the ROC Curve for the K-Nearest Neighbours to the Graph

KNN\_y\_pred\_proba = KNN\_model.predict\_proba(x\_test)[:,1]

KNN\_fpr, KNN\_tpr, KNN\_threshholds = metrics.roc\_curve(y\_test, KNN\_y\_pred\_proba)

KNN\_roc\_auc = metrics.roc\_auc\_score(y\_test, KNN\_y\_pred\_proba)

KNN\_line, = plt.plot(KNN\_fpr,KNN\_tpr,color='Green', lw=3, label='K-Nearest Neighbours (area = %0.4f)' % KNN\_roc\_auc)

#----- Add the ROC Curve for the Support Vector Classification model to the Graph

SV\_y\_pred\_proba = SV\_model.predict\_proba(x\_test)[:,1]

SV\_fpr, SV\_tpr, SV\_threshholds = metrics.roc\_curve(y\_test, SV\_y\_pred\_proba)

SV\_roc\_auc = metrics.roc\_auc\_score(y\_test, SV\_y\_pred\_proba)

SV\_line, = plt.plot(SV\_fpr,SV\_tpr,color='Yellow',lw=3,label='Support Vector Classification (area = %0.4f)' % SV\_roc\_auc)

#----- Add the baseline

baseline, = plt.plot([0, 1], [0, 1], color='Blue', lw=3, label='Baseline', linestyle='--')

plt.xlabel('\nFalse Positive Rate',size=20)

plt.ylabel('True Positive Rate', size=20)

plt.title('\nROC Curves of All Models (Using Probabilities)\n',size=25)

plt.legend(loc="lower right",handles=[LR\_line, DT\_line, KNN\_line, SV\_line, baseline], prop={'size': 16})

plt.show()