EE 559: Mathematical Pattern Recognition
Project Report

Data Set: APS Failure at Scania Trucks

Author: Nipun Manral (manral@usc.edu)

Date: April 30, 2019

1. Abstract

This project aims to develop a pattern recognition system for the "APS Failure at Scania Trucks" dataset in order to minimize the overall maintenance costs of the air pressure system in Scania trucks. Four different types of classification techniques (Support Vector Machines, Naïve Bayes, K Nearest Neighbor and Multilayer perceptron) were used to evaluate, compare and optimize the total cost. The Support Vector Machines classifier with Radial basis Function kernel, penalty parameter (C) = 1000 and kernel parameter (gamma) = 1 gave the best minimum total cost of 14,090$ with a ROC AUC score of 0.9805 using the down-sampled training dataset and 12,260$ with a ROC AUC score of 0.9865 using the complete training dataset.

2. Introduction

2.1 Problem Statement and Goals

The "APS Failure at Scania Trucks Dataset" was chosen for the project. The dataset contains data about the Air Pressure System (APS) in Scania trucks that generate pressurized air which are utilized in various functions in a truck, such as braking and gear changes. This is a two class problem in which the positive class in the dataset corresponds to component failures for a specific component of the APS system. The negative class corresponds to failures for components not related to the APS [1].

The cost-metric of mis-classification is based on the table provided below where Cost_1 = 10 and Cost_2 = 500 :

		True Class	
		Positive	Negative
Predicted Class	Positive	-	Cost_1
	Negative	Cost_2	-

Here, Cost_1 refers to the cost that an unnecessary checks needs to be done by a mechanic at a workshop (Type 1 - false positive) where as Cost_2 refers to the cost of missing a faulty truck, which may cause a breakdown (Type 2 - false negative). Hence, the total cost is calculated as:

Total Cost = Cost_1 * Type 1 failure instances + Cost_2 * Type 2 Failure instances

The goal of this project is to create a prediction model using classification techniques learnt in class that minimizes the total cost, which in turn would minimize the maintenance costs of Scania trucks.

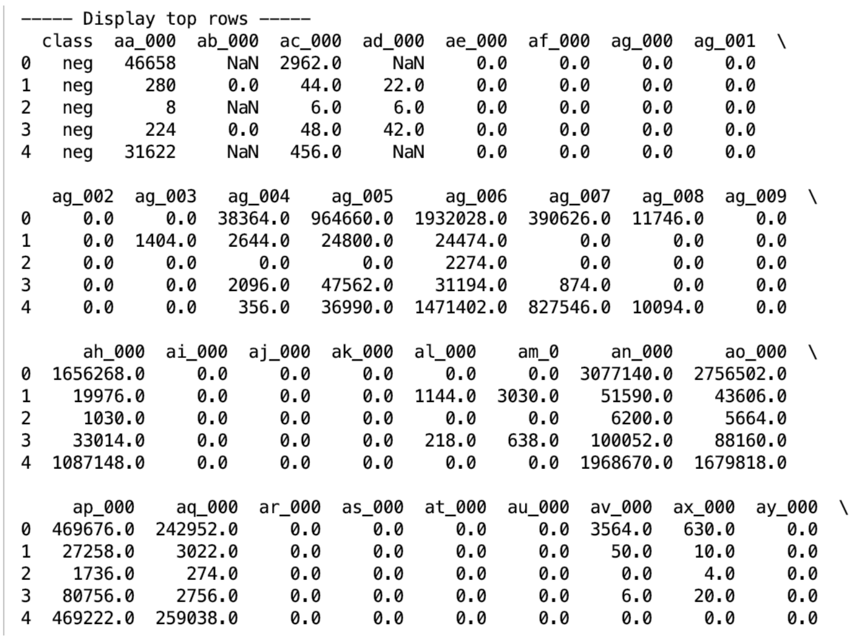
3. Approach and Implementation

3.1 Preprocessing

The training and test datasets were loaded into a data frame using pandas library in python for preprocessing. While importing, any attribute having the value 'na' was labelled as a missing value. The number of features in both the datasets (training and test dataset) were counted which indicated 171 columns representing different attributes including the class label.

Step 1 (Data Exploration):

The first few rows of the training dataset were previewed using the dataframe.head() command as shown below (for a few features):



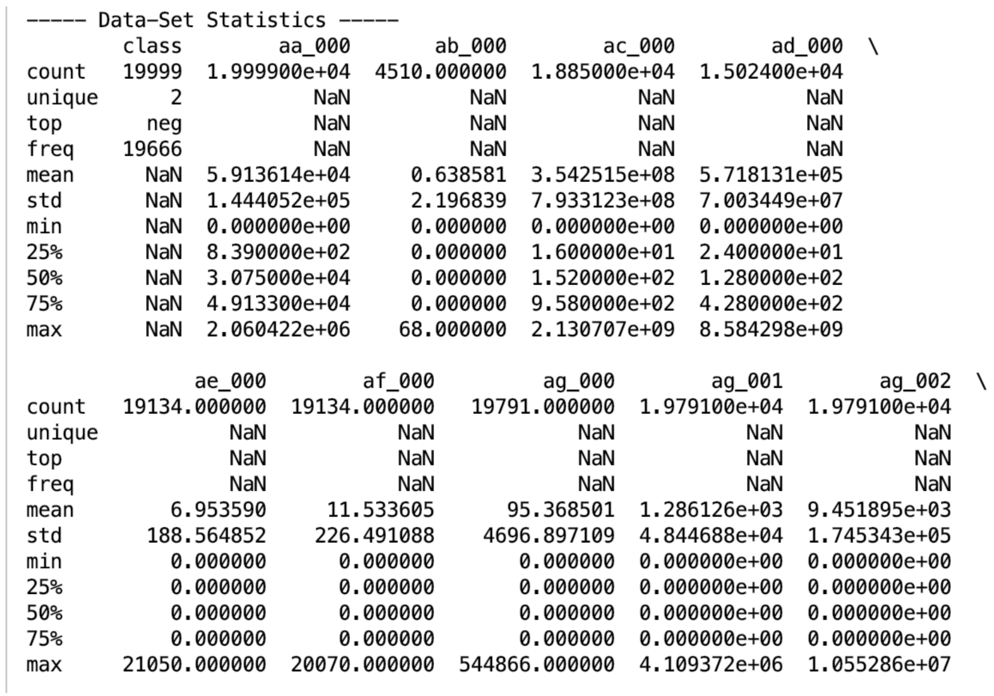
```
----- Display top rows -----
  class  aa_000  ab_000  ac_000  ad_000  ae_000  af_000  ag_000  ag_001  \
0   neg   46658     NaN  2962.0     NaN     0.0     0.0     0.0     0.0
1   neg     280     0.0    44.0    22.0     0.0     0.0     0.0     0.0
2   neg       8     NaN     6.0     6.0     0.0     0.0     0.0     0.0
3   neg     224     0.0    48.0    42.0     0.0     0.0     0.0     0.0
4   neg   31622     NaN   456.0     NaN     0.0     0.0     0.0     0.0

   ag_002  ag_003   ag_004    ag_005     ag_006    ag_007   ag_008  ag_009  \
0     0.0     0.0  38364.0  964660.0  1932028.0  390626.0  11746.0     0.0
1     0.0  1404.0   2644.0   24800.0    24474.0       0.0      0.0     0.0
2     0.0     0.0      0.0       0.0     2274.0       0.0      0.0     0.0
3     0.0     0.0   2096.0   47562.0    31194.0     874.0      0.0     0.0
4     0.0     0.0    356.0   36990.0  1471402.0  827546.0  10094.0     0.0

      ah_000  ai_000  aj_000  ak_000  al_000    am_0     an_000     ao_000  \
0  1656268.0     0.0     0.0     0.0     0.0     0.0  3077140.0  2756502.0
1    19976.0     0.0     0.0     0.0  1144.0  3030.0    51590.0    43606.0
2     1030.0     0.0     0.0     0.0     0.0     0.0     6200.0     5664.0
3    33014.0     0.0     0.0     0.0   218.0   638.0   100052.0    88160.0
4  1087148.0     0.0     0.0     0.0     0.0     0.0  1968670.0  1679818.0

     ap_000    aq_000  ar_000  as_000  at_000  au_000  av_000  ax_000  ay_000  \
0  469676.0  242952.0     0.0     0.0     0.0     0.0  3564.0   630.0     0.0
1   27258.0    3022.0     0.0     0.0     0.0     0.0    50.0    10.0     0.0
2    1736.0     274.0     0.0     0.0     0.0     0.0     0.0     4.0     0.0
3   80756.0    2756.0     0.0     0.0     0.0     0.0     6.0    20.0     0.0
4  469222.0  259038.0     0.0     0.0     0.0     0.0     0.0     0.0     0.0
```

The descriptive statistics such as mean, standard deviation of each feature were also observed using the dataframe.describe() command.



```
----- Data-Set Statistics -----
        class        aa_000       ab_000        ac_000        ad_000  \
count   19999  1.999900e+04  4510.000000  1.885000e+04  1.502400e+04
unique      2           NaN          NaN           NaN           NaN
top       neg           NaN          NaN           NaN           NaN
freq    19666           NaN          NaN           NaN           NaN
mean      NaN  5.913614e+04     0.638581  3.542515e+08  5.718131e+05
std       NaN  1.444052e+05     2.196839  7.933123e+08  7.003449e+07
min       NaN  0.000000e+00     0.000000  0.000000e+00  0.000000e+00
25%       NaN  8.390000e+02     0.000000  1.600000e+01  2.400000e+01
50%       NaN  3.075000e+04     0.000000  1.520000e+02  1.280000e+02
75%       NaN  4.913300e+04     0.000000  9.580000e+02  4.280000e+02
max       NaN  2.060422e+06    68.000000  2.130707e+09  8.584298e+09

              ae_000        af_000         ag_000        ag_001        ag_002  \
count   19134.000000  19134.000000   19791.000000  1.979100e+04  1.979100e+04
unique           NaN           NaN            NaN           NaN           NaN
top              NaN           NaN            NaN           NaN           NaN
freq             NaN           NaN            NaN           NaN           NaN
mean        6.953590     11.533605      95.368501  1.286126e+03  9.451895e+03
std       188.564852    226.491088    4696.897109  4.844688e+04  1.745343e+05
min         0.000000      0.000000       0.000000  0.000000e+00  0.000000e+00
25%         0.000000      0.000000       0.000000  0.000000e+00  0.000000e+00
50%         0.000000      0.000000       0.000000  0.000000e+00  0.000000e+00
75%         0.000000      0.000000       0.000000  0.000000e+00  0.000000e+00
max     21050.000000  20070.000000  544866.000000  4.109372e+06  1.055286e+07
```

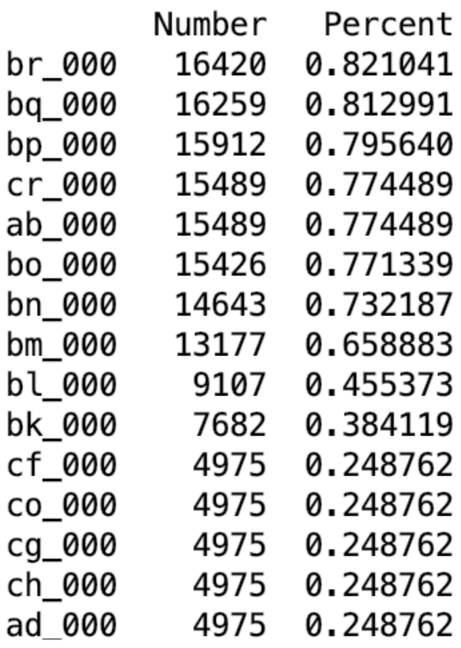
Both the above commands provide us an idea about the type of dataset that we are dealing with and enable us to identify if there is any missing data that we need to preprocess.

Step 2 (Class Labels):
The class labels were present as strings - “neg” and “pos” in both the datasets. These labels were converted into integer labels where the “neg” class is represented as 0 and the positive

class is represented as 1. The count of each class label was calculated to check for data imbalance as shown below:

Class	Count
0 (Neg)	19666
1 (Pos)	333
Total	19999

Step 3 (Missing Data):
The percentage of missing data for each feature in the training dataset was calculated. The features with the highest percentage of missing data in descending order are listed below:


```
        Number   Percent
br_000   16420  0.821041
bq_000   16259  0.812991
bp_000   15912  0.795640
cr_000   15489  0.774489
ab_000   15489  0.774489
bo_000   15426  0.771339
bn_000   14643  0.732187
bm_000   13177  0.658883
bl_000    9107  0.455373
bk_000    7682  0.384119
cf_000    4975  0.248762
co_000    4975  0.248762
cg_000    4975  0.248762
ch_000    4975  0.248762
ad_000    4975  0.248762
```

All features in the training dataset that had more than 50% of their values missing were removed from both the datasets using the dataframe.drop() command. This resulted in the removal of 8 features and hence we now had 162 features to work with.

Next, the missing values were imputed using the SimpleImputer() function in the scikit-learn library, with the imputation strategy as median i.e. the value which lies in the midpoint of the frequency distribution of the observed values. The median of the features over all class labels were first computed on the training dataset using SimpleImputer.fit() command and then applied to both the training and test dataset using the SimpleImputer.transform() command.

Step 4 (Standardization):
All real-valued features were standardized to have zero mean and unit standard deviation using the StandardScaler() function in the scikit-learn library. The mean and standard deviation were first calculated over all class labels on the training dataset using StandardScaler.fit() command and then applied to both the training and test dataset using the StandardScaler.transform() command.

Note: Both imputation and standardization techniques were applied only to the features and not the class labels.

3.2 Compensation for unbalanced data

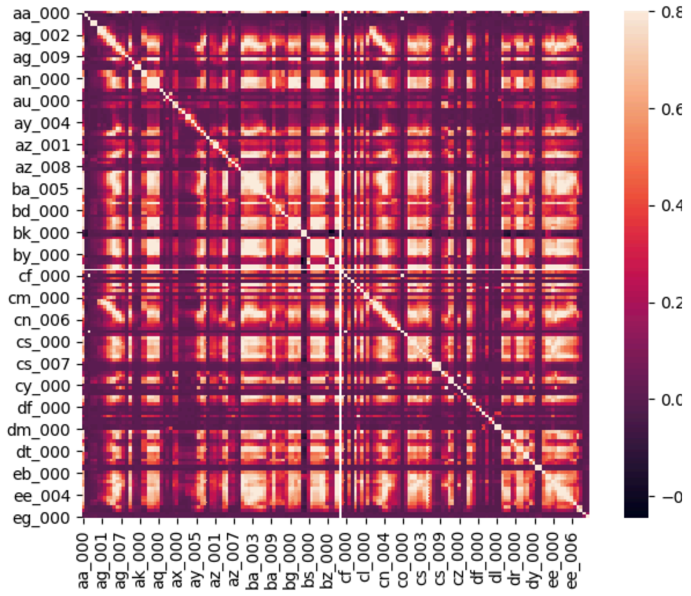
As seen in the previous subsection, we have approximately 98% data points belonging to class 0 and 2% data points belonging to class 1 which creates an unbalanced dataset. SMOTE (Synthetic Minority Oversampling Technique) was used to synthetically create additional data points from the minority class i.e. class 1. The steps involved in SMOTE are as follows:

- A sample was chosen from class 1 (minority) and then its 5 nearest neighbors were considered.
- A vector (difference) between the chosen sample and one of those k neighbors was considered.
- This vector was multiplied with a random number between 0 and 1 which was added to the chosen sample to create a new synthetic data point belonging to class 1
- This process was repeated till we had a balanced dataset.
  The over_sampling.SMOTE() function in the imblearn library was used to perform the SMOTE operation. After SMOTE, the number of data points belonging to each class were:

Class	Count
0 (Neg)	19666
1 (Pos)	19666
Total	39332

3.3 Feature extraction and dimensionality adjustment

The dataset initially contained 170 different features which were probably correlated and hence not required. We used Principal Component Analysis (PCA) which is a dimensionality reduction technique to reduce the number of features (possibly correlated) to a smaller set of uncorrelated features that contained most of the information of the dataset. Generally, the eigen vectors of the largest eigen values hold the most amount of information (maximum variance). A heatmap showing the correlation between the features are shown below:



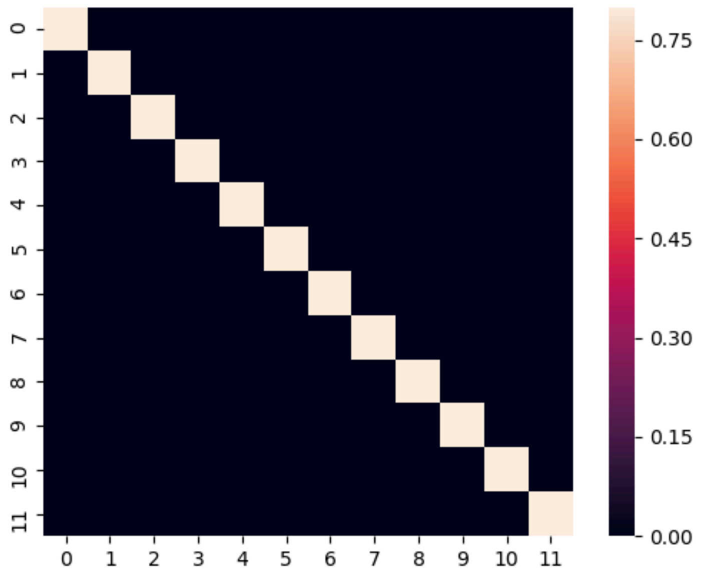
The PCA components were found using the decomposition.PCA(n_components) command in scikit-learn library. To begin, the value of n_components was set as 0.80 i.e. the number of

PCA components that were selected should contain atleast 80% of the variance of the dataset. The number of components that contained 80% of the variance was calculated as 15 by this function.

The 15 components obtained after PCA were used as input features for our three classifiers (Naïve Bayes, SVM, K Nearest Neighbor) to obtain a cost on the validation dataset. The cost for different selection of n_components is as shown in the table:

N_components Value (Maximum Variance)	Features	Support vector Machines (Cost)	Naïve Bayes (Cost)	K Nearest Neighbor (Cost)
0.80	15	4620	26790	5172
0.90	30	4706	28240	4900
0.75	12	4320	26320	4970
0.70	9	4926	24790	5335

From the above table, since 0.75 gave the overall best classification on the validation dataset, we selected the value of n_components as 0.75 i.e. the number of PCA components should contain atleast 75% of the maximum variance. A heatmap showing the correlation between the 12 components after PCA is as shown below:



3.4 Dataset usage

After SMOTE, the training dataset contained 39,332 data points. The test dataset contained 16,000 data points. The test dataset is set aside to evaluate the performance of our final model/classifier. From the training dataset, we created a validation dataset which is used to tune the hyperparameters of the classifiers.

We utilized cross-validation technique to split our training dataset into two parts - training subset and validation set, in order to train our classifier. We used model_selection.StratifiedKFold(n_splits =5, shuffle=True) command from the scikit-learn library to split the training dataset. The dataset is always shuffled to bring in randomness. Since we wanted to divide the training dataset into a 80:20 split, we divided the dataset into five parts where four parts were used as the training subset and one part was used as the validation dataset. Hence at any time, the number of data points in each set were:

Dataset	Data points
Training Dataset	31464
Validation Dataset	7868
Test Dataset	16,000

The validation dataset is used to tune our hyperparameters indicated below:

- C and Gamma parameters in Support Vector Machines Classifier
- Hidden Layer Configuration for Multilayer Perceptron Classifier
- K (Number of neighbors) in K Nearest Neighbor Classifier

Each classifier with a specific parameter combination was trained on the training subset (four parts) and its performance (statistics such as accuracy, F1 Score and total cost) was evaluated on the validation dataset (remaining part). This evaluation was performed 5 times (because number of folds = 5) with different permutations of the five parts in order to avoid overfitting. It is important to ensure that while training, our classifier does not train/access the validation dataset. We then average over the validation accuracy, F1 Score and total cost. The hyperparameters which gave the lowest total cost were selected as the best parameters for the classifier.

The best parameters for the classifier were then used to evaluate the performance of the final model on the test data. This test data should be completely unseen by our model/classifier during the training and validation phase. The test set is used just once for each classifier to evaluate the final performance.

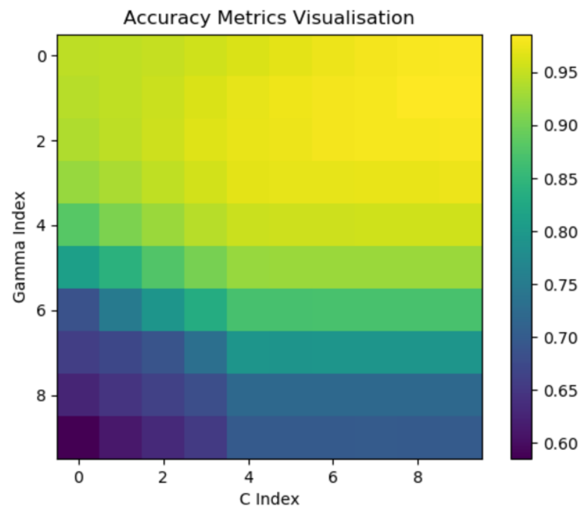
3.5 Training and classification

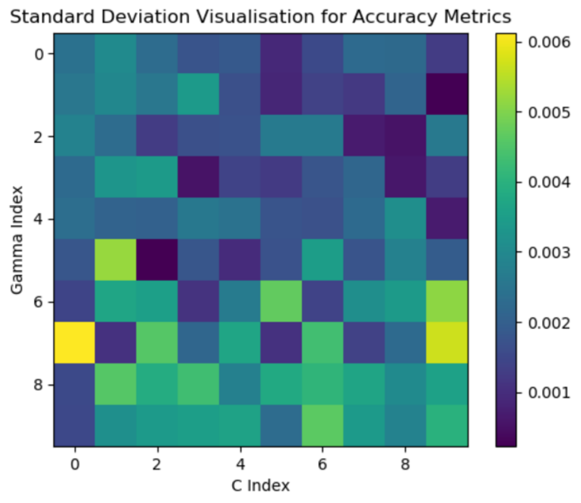
3.5.1 Support Vector Machines

We used Support vector machines (SVM) as a classification technique for the dataset, which uses a non-linear mapping such as phi-machine to map the original feature space into a higher dimension , where the data can be separated using a hyperplane with the largest margin. We used the RBF (Radial Basis Function) kernel, also called the Gaussian Kernel in the algorithm. The penalty parameter (C) and the RBF kernel parameter (gamma) were considered as hyperparameters in the SVM classifier.

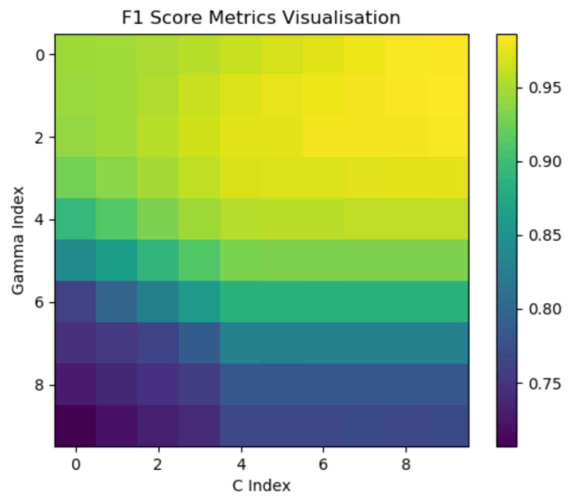
We used the svm.SVC(kernel, C, gamma) command from scikit-learn to classify the dataset using the SVM classifier. 100 different combinations of C and gamma were used to evaluate the performance of accuracy, F1 Score and total cost on the validation dataset using cross validation. The C and gamma parameters were chosen using the logarithmic scale between 10^{-2} and 10^{3}.

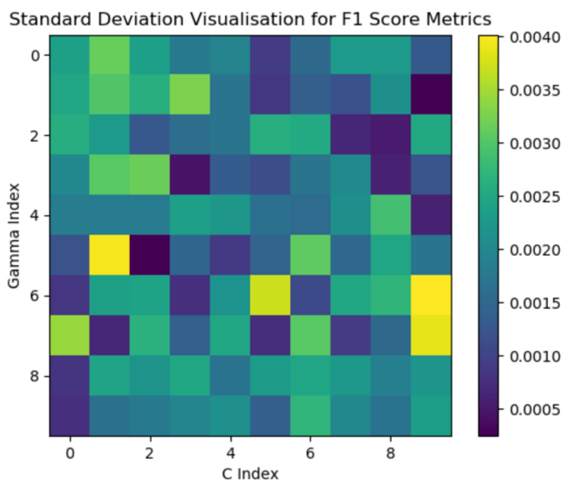
Accuracy: The best accuracy score obtained on the validation dataset was 0.9861 when C=1000 and gamma = 0.0359. The standard deviation of the best accuracy score using cross validation was 0.0002. Graph visualization of the accuracy and standard deviation of accuracy:





F1 Score: The best F1 Score obtained on the validation dataset was 0.9862 when C=1000 and gamma = 0.0359. The standard deviation of the best F1 Score using cross validation was 0.0002. Graph visualization of the F1 Score and standard deviation of F1 Score:

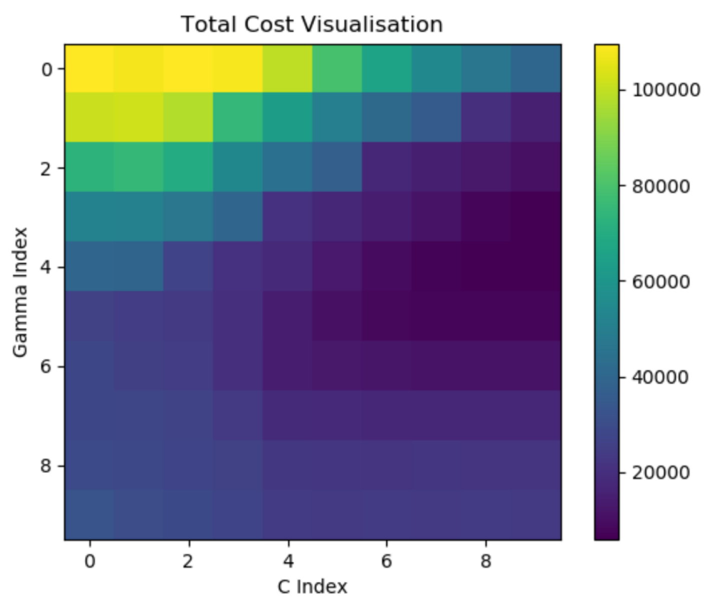




Total Cost: The total cost on the validation set for the different combinations of C and gamma are shown below:

Gamma Values	C Values									
	1.0e-02	3.5e-02	1.2e-01	4.6e-01	1.6e+00	5.9e+00	2.1e+01	7.7e+01	2.7e+02	1.0e+03
1.0e-02	109642	107928	109434	108150	99424	79470	65848	54970	46396	40502
3.5e-02	101442	102220	97634	75118	63620	50536	41880	35774	20196	15496
1.2e-01	72906	75346	69740	54042	44026	37138	17468	14988	12764	10532
4.6e-01	51870	51354	46956	40042	20996	17488	14710	11296	7580	6168
1.6e+00	40180	39414	26556	21240	18256	13434	9382	7338	5846	5834
5.9e+00	26354	24734	23332	20522	14626	10510	8474	7844	7774	7486
2.1e+01	28188	25568	24536	20290	14388	12770	12178	11468	11468	11180
7.7e+01	28300	28002	26972	23612	18516	18230	17504	17056	16990	17092
2.7e+02	29388	28568	27370	26396	22594	22160	21976	22112	21946	22012
1.0e+03	32640	30506	29162	27548	23890	23648	23712	23572	23704	23632

The distribution of these costs are visualized below:

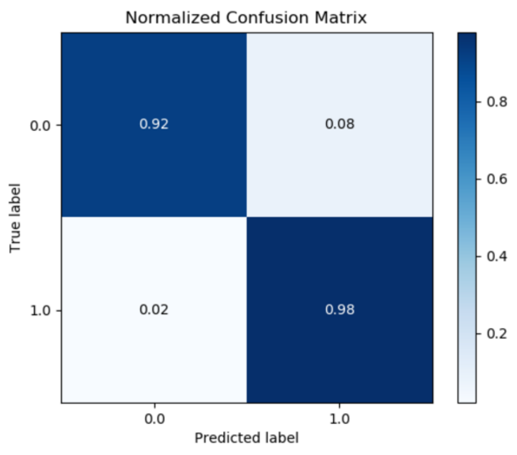


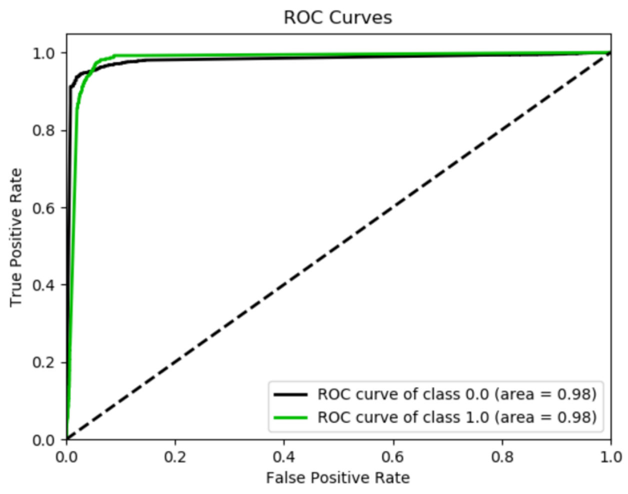
From the above table and graph for total cost, we observed that the lowest total cost on the validation set is 5834 which was obtained when C = 1000 and gamma = 1.6681.

Thus, based on the results of the cross validation, the best parameters for the support vector machine classifier were chosen as C = 1000 and gamma = 1.6681. Using these parameters, we ran the SVM classifier on the test dataset which gave a total cost of 15,210 with the below confusion matrix:

		True Class	
Predicted Class		Positive	Negative
	Positive	369	1221
	Negative	6	14404

The number of type 1 failures = 1221 and number of type 2 failures = 6. The accuracy using these combination of parameters on the test set was 0.923 and the F1 score was 0.375. The confusion matrix and ROC curve are visualized below:

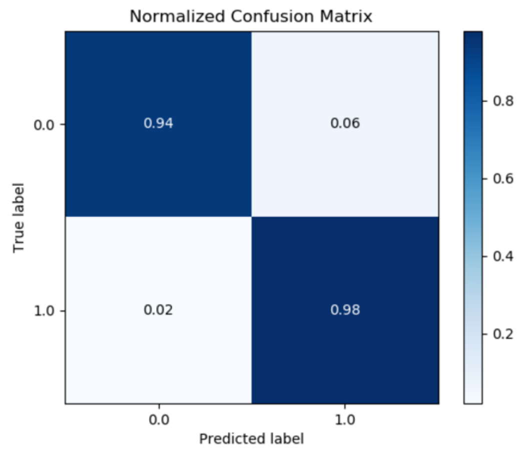


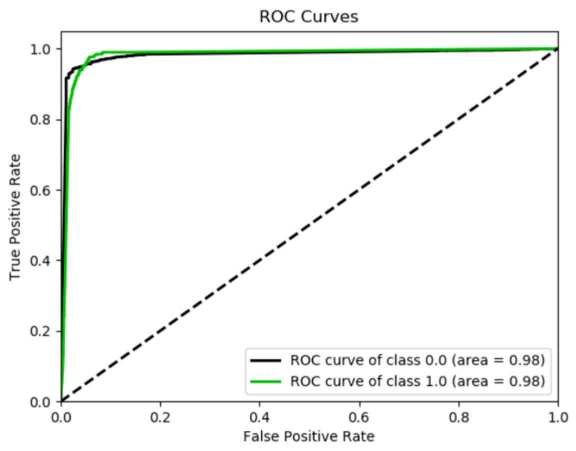


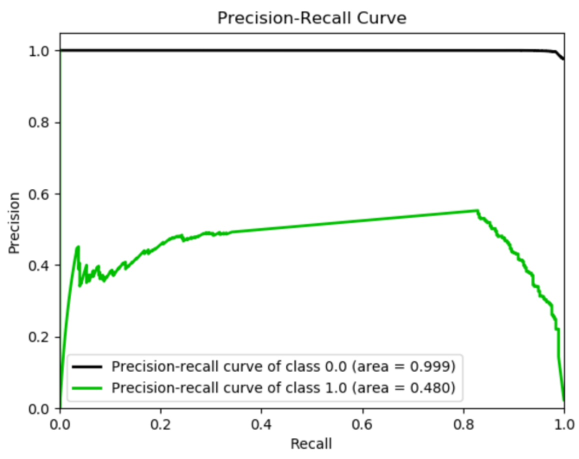
In order to get a even lower cost, we slightly tinkered around with a few more C and gamma parameters around the best parameters obtained from the validation dataset. We found that when C = 1000 and gamma = 1, we got an even lower cost of 14,090 on the test set with the below confusion matrix:

	True Class		
Predicted Class		Positive	Negative
	Positive	366	959
	Negative	9	14666

The number of type 1 failures = 959 and number of type 2 failures = 9. The accuracy on the test set was found to be 0.9395 and F1 score was 0.43. The ROC AUC score when C = 1000 and gamma = 1 was found to be 0.98054. The confusion matrix, ROC curve and precision recall are visualized below:







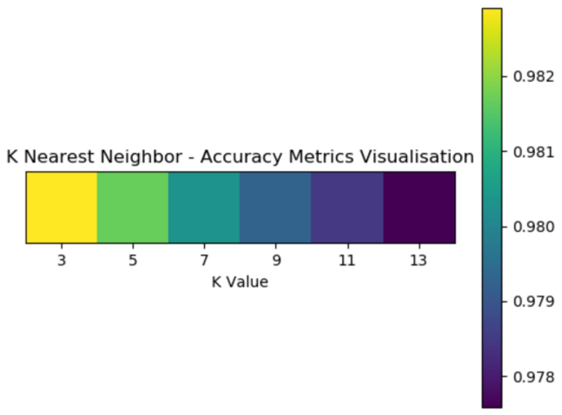
3.5.2 K Nearest Neighbor

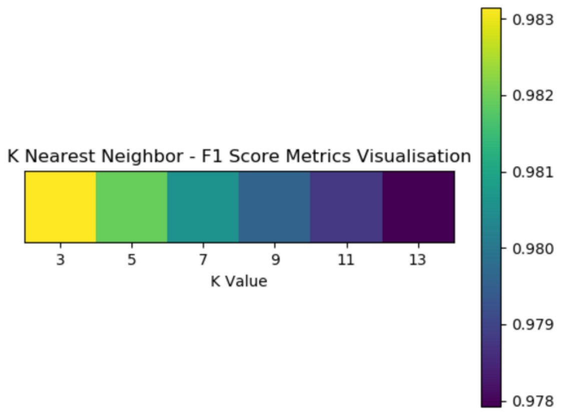
K Nearest Neighbor is a type of non-parametric classifier which assigns a class to an unknown sample based on the class which is most common among its nearest neighbors. The number of nearest neighbors to be considered is denoted as K.

The K Nearest neighbor classifier was implemented using the neighbors.KNeighborsClassifier(n_neighbors) from the scikit-learn library. 6 different values of K(n_neighbors) = (3, 5, 7, 9, 11, 13) were used to evaluate the performance of accuracy, F1

Score and total cost on the validation dataset using cross validation. Unlike in SVM, the best accuracy, F1 Score and total cost for K Nearest Neighbors were all obtained when K = 3.

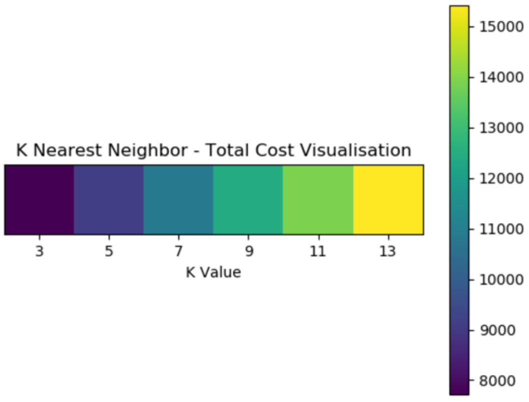
The best accuracy on the validation dataset was found to be 0.9829 and the best F1 Score was found to be 0.9831. The accuracy and F1 score for different K values are visualized below:





The total cost for different values of K on validation set is shown and visualized below:

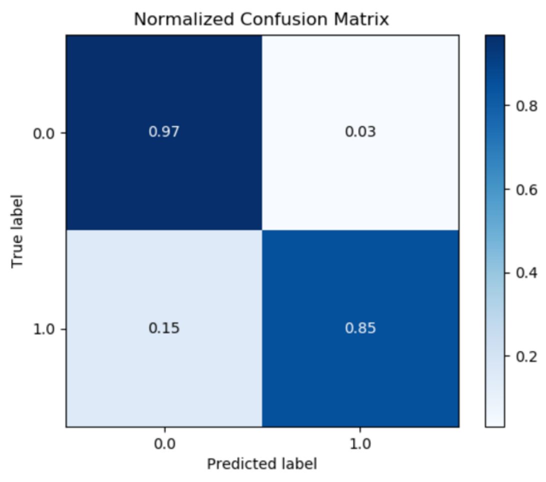
K Value	K = 3	K = 5	K = 7	K = 9	K = 11	K = 13
Total Cost	7714	9135	10892	12459	13885	15418

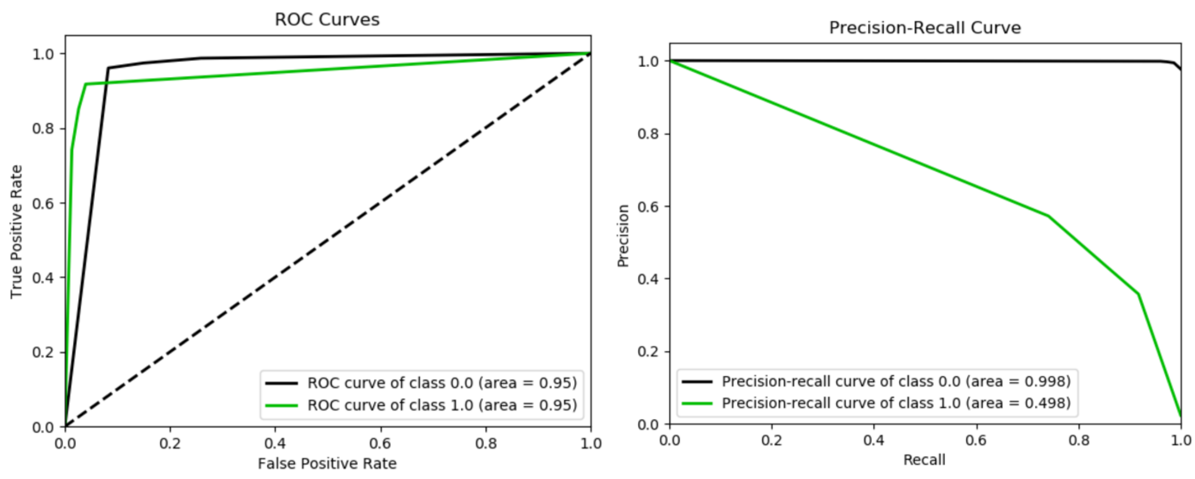


From the above table, we can see that the best cost obtained on validation data set is 7714 which occurs when K = 3. Using K = 3, we evaluated the performance of the K Nearest neighbor classifier on the test dataset. The total cost obtained on the test dataset was 32,090 with the below confusion matrix:

		True Class	
Predicted Class		Positive	Negative
	Positive	319	409
	Negative	56	15216

The number of type 1 failures = 409 and number of type 2 failures = 56. The accuracy on the test set was found to be 0.9709 and F1 score was 0.578. The ROC AUC score on the test set when K = 3 was found to be 0.9477 The confusion matrix, ROC curve and precision recall are visualized below:





3.5.3 Naïve Bayes

Naïve Bayes Classifier assumes conditional independence between the features and depends on the Bayes Theorem as shown below:

$$P(S/x) = \frac{P\left(\frac{x}{S}\right).P(S)}{P(x)}$$

where P(S/x) is the posterior probability, P(x/S) is the likelihood and P(S) is the prior.

We used the Gaussian Naïve Bayes for classification in which the likelihood of the features is assumed to be gaussian.

$$P(x/S) = \frac{1}{\sqrt{2\pi\sigma_y^2}} \exp\left(-\frac{(x_i - u_y)^2}{2\sigma_y^2}\right)$$

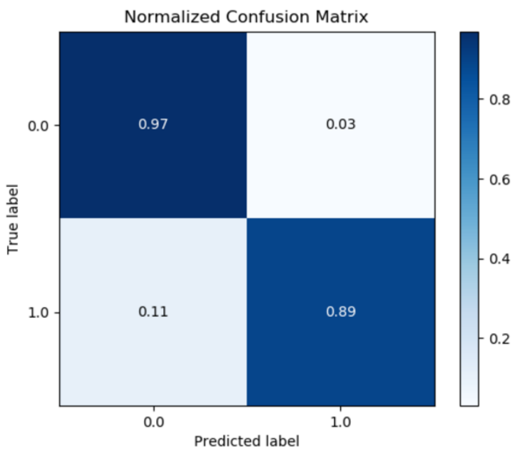
where σ_y and u_y are estimated using maximum likelihood

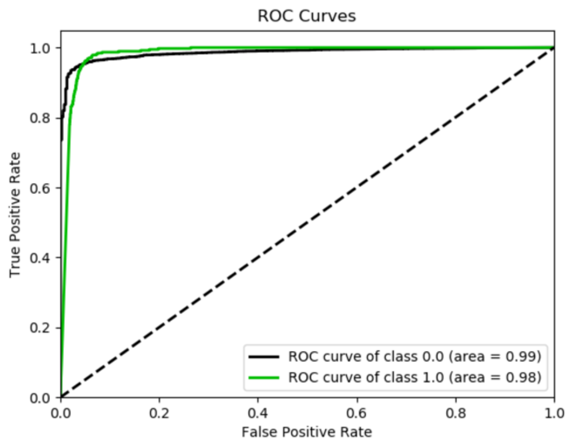
The Gaussian Naïve Bayes classifier is implemented using the naïve_bayes.GaussianNB() command from the scikit-learn library. The accuracy on the test set was found to be 0.9671 and the F1 Score was found to be 0.5584.

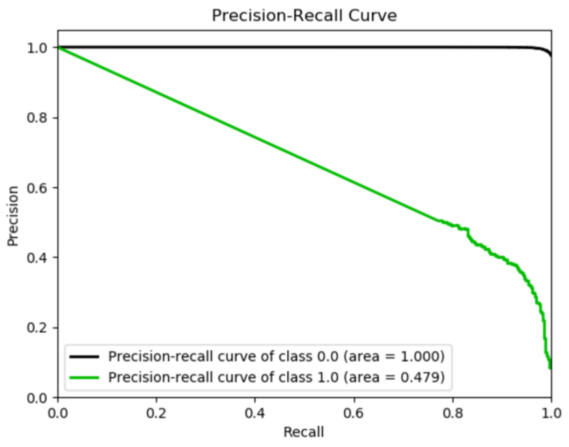
The total cost on the test set was calculated as 26,320 with the below confusion matrix:

		True Class	
Predicted Class		Positive	Negative
	Positive	332	482
	Negative	43	15143

The number of type 1 failures = 482 and number of type 2 failures = 43. The ROC AUC score on the test set was found to be 0.9833 The confusion matrix, ROC curve and precision recall are visualized below:







3.5.4 Multilayer Perceptron Classifier

A multilayer perceptron classifier is a type of artificial neural network which consists of an input layer, hidden layers and an output layer. Each layer consists of neurons which use a nonlinear activation function. Multilayer perceptron are a type of supervised learning which are trained using a technique called backpropagation.

We implemented the multilayer perceptron classifier using the neural_network.MLPClassifier(hidden_layer_sizes) from the scikit-learn library. We used ReLu (Rectified Linear Unit) as the activation function and stochastic gradient descent as the solver for optimization.

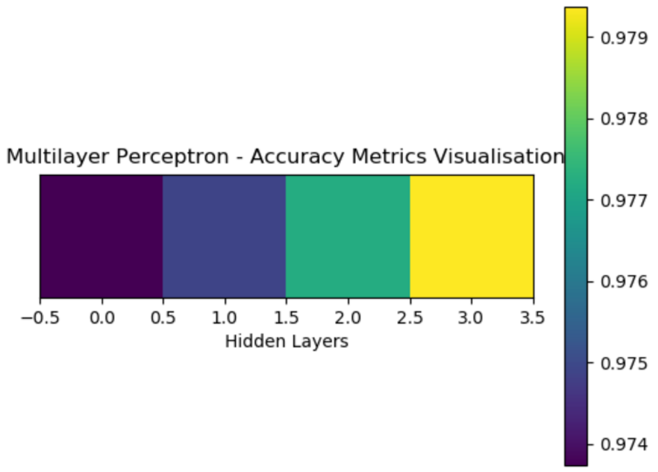
4 different hidden layer configurations were used to evaluate the performance of accuracy, F1 Score and total cost on the validation dataset using cross validation as shown below:

- 1 Hidden layer with 100 neurons (Index 0)

- 1 Hidden layer with 200 neurons (Index 1)
- 2 Hidden layers with 200 neurons in first layer & 100 neurons in second layer (Index 2)
- 2 Hidden layers with 200 neurons in first layer & 200 neurons in second layer (Index 3)

The best accuracy on the validation data set was found to be 0.9793 for the hidden layer configuration (200, 200) i.e. Index 3. The best F1 Score was found to be 0.9793 for the hidden layer configuration (200, 200) i.e. Index 3. The total cost on the validation data set for different hidden layer configuration is shown and visualized below:

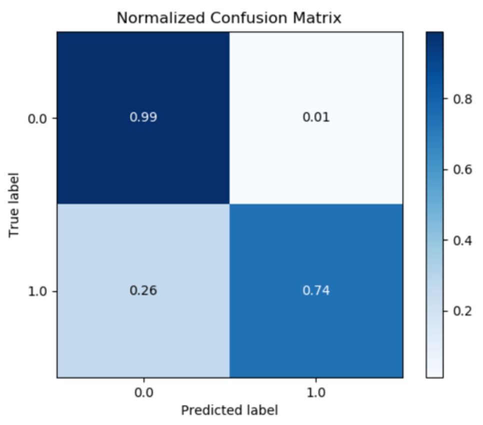
Hidden Layer Configurations	Index 0	Index 1	Index 2	Index 3
Total Cost	63904	58960	52554	46580

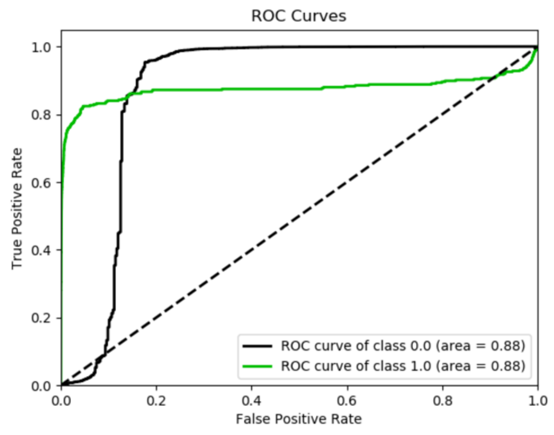


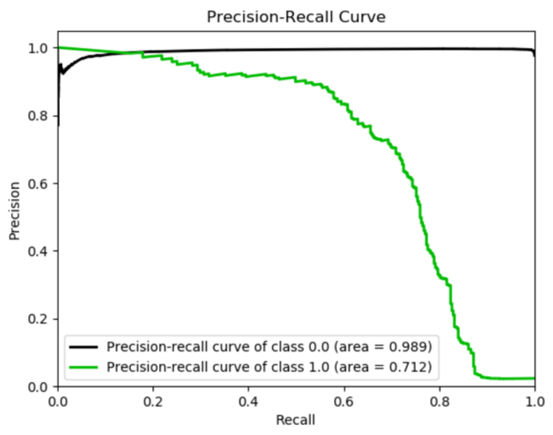
From the above table, we find that the best total cost on the validation data set is 46,580 which is obtained for a hidden layer configuration of (200, 200) i.e. Index 3. Hence this hidden layer configuration of 2 layers with 200 neurons each is used as the parameter for the Multilayer Perceptron classifier to test its performance on the test data set. The total cost obtained on the test dataset using this hidden layer configuration is 51,200 with the below confusion matrix:

	True Class		
Predicted Class		Positive	Negative
	Positive	276	170
	Negative	99	15455

The number of type 1 failures = 170 and number of type 2 failures = 99. The accuracy on the test set was found to be 0.9831 and F1 score was 0.672. The ROC AUC score on the test set was found to be 0.8753 The confusion matrix, ROC curve and precision recall are visualized below:







4. Comparison, Results and Interpretation

A baseline model/classifier was designed which assigned the majority class label as the class label to any unknown sample. This kind of a classifier had a very high accuracy of 0.9765 since it always decided the negative class (majority) by default, however it had a poor F1 score and a very high cost of 187,500 due to large number of type 2 failures. The table below shows a comparison of the baseline classifier with all the other classifiers on the test dataset that we have used so far:

	Classifiers				
Test Dataset	Baseline Model	Support Vector Machine	K Nearest Neighbor	Naïve Bayes	Multilayer Perceptron
Total Cost	187,500	14,090	32,090	26,320	51,200
Type 1 Failure	0	959	409	482	170
Type 2 Failure	375	9	56	43	99
Accuracy	0.9765	0.9395	0.9709	0.9671	0.9831
F1 Score	0	0.43	0.578	0.5584	0.672
ROC AUC Score	-	0.9805	0.9477	0.9833	0.8753
Best Parameter	-	C = 1000, Gamma = 1	K = 3	-	Layers = (200,200)

Since our primary metric for classification was the total cost, we can observe that the Support Vector Machines classifier with Radial Basis Function kernel gave the lowest total cost of 14,090$ on the test dataset, thus performing the best. Comparing this with the past usage of the dataset in the Industrial Challenge 2016, we can conclude that we have performed reasonably well in comparison to the total cost of 9,920$ obtained by the top contestant.

We can also evaluate the performance of the classifier using the Receiver Operating Characteristics graph/score. The ROC graph is obtained by plotting the true positive rate against the false positive rate. The true positive rate is given by:

$$True\ Positive\ Rate = \frac{True\ Positive}{True\ Positive + False\ Negative}$$

The false positive rate is given by:

$$False\ Positive\ Rate = \frac{False\ Positive}{False\ Positive + True\ Negative}$$

The Area Under Curve (AUC) is the area under the ROC curve and a higher area is always preferred. We can notice that both Support Vector Machines and Naïve Bayes have a very high ROC-AUC score which indicates that our classifiers have performed well on the test data set.

The F1 score is the harmonic mean of precision and recall. Precision is the number of correct positive results divided by the number of all positive results returned by the classifier and Recall is the number of correct positive results divided by the number of samples that should have been identified as positive. The best value of F1 Score is considered to be 1, however we can observe that our classifiers do not have a high F1 Score because we chose our model evaluation criteria as the cost-metric for mis-classification. Among all the classifiers, the highest F1 score while keeping the cost minimum was observed to be 0.672 using the multilayer perceptron classifier. In the previous sections (3.5.1, 3.5.2, 3.5.3 and 3.5.4,) we can observe from the classifier visualization graphs for F1 Score and Cost, that we can obtain high F1 scores at the expense of increasing the total cost.

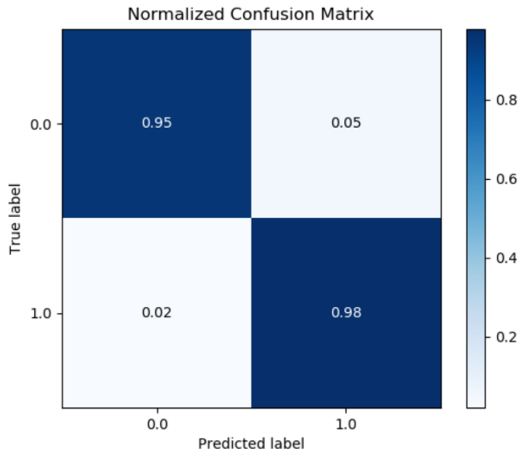
Multilayer Perceptron seems to have performed the fourth best on the test dataset in terms of cost. However it is important to note that such types of artificial neural networks are really powerful in classification tasks and due to a computation limit we have not been able to evaluate with higher number of layers and neurons. With an increased number of neurons and layers, multilayer perceptron would have performed equally well, if not better on the test data set.

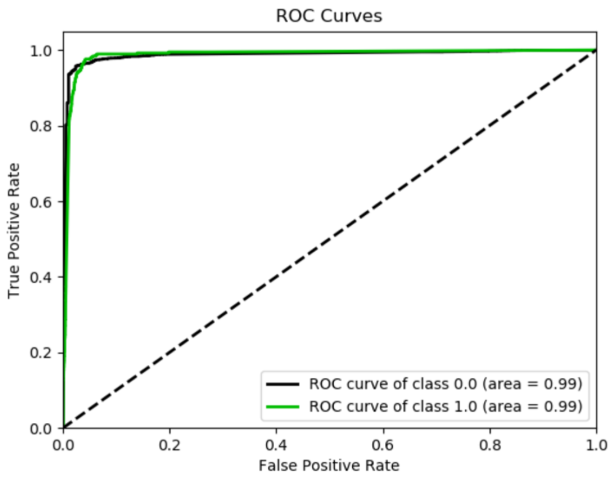
The four classifiers with their chosen best hyper parameters were also trained on the complete training dataset containing 60000 points and then evaluated on the test set. During PCA, similar to before, 75% of the maximum variance was captured which resulted in 15 components. The performance of the classifiers is as shown in the table below:

Test Data Set	Support Vector Machines	K Nearest Neighbor	Naïve Bayes	Multilayer Perceptron
Total Cost	12260	33930	25640	40390
Type 1 Failure	776	343	464	139
Type 2 Failure	9	61	42	78
Accuracy	0.9509	0.9747	0.968	0.9864
F1 Score	0.4825	0.6085	0.5682	0.7324
ROC AUC Score	0.9865	0.9402	0.9836	0.9003

From the above table, we can notice that the total cost has reduced for all the classifiers except K nearest neighbors on the complete dataset thus showing that increasing the amount of data points will improve the performance of the classifiers. The support vector machines achieved the lowest cost of 12,260$ on the complete dataset with a ROC AUC score of 0.9865. The confusion matrix for the SVM classifier is provided below along with the ROC curve:

		True Class	
Predicted Class		Positive	Negative
	Positive	366	776
	Negative	9	14849





5. Summary and Conclusions

The APS Failure at Scania Trucks dataset was evaluated using different classifiers on the test dataset and among all the classifiers, support vector machines with radial basis function kernel achieved the lowest total cost of 14,090$ using the down-sampled training dataset and 12,260$ using the complete training dataset.

We also observed that all our four classifiers performed better than the baseline model. For further follow-on work, we can investigate the impact of increasing the number of hidden layers and the number of neurons in Multilayer Perceptron as that could improve the performance of our classifier. It would also be interesting to perform feature engineering on the existing features and evaluate the performance on the test set using other classifiers such as Random Forest Classifier.

In completing this project, I was able to utilize my theoretical knowledge in pattern recognition to solve classification problems on real world datasets. During this process, I learnt various techniques that helped me in data exploration, dimensionality reduction and data synthesis. I was also able to observe the computation speed involved in running different classifiers. Naïve Bayes typically performed the fastest classification followed by K Nearest Neighbors. The computation time for Multilayer perceptron increased non-linearly with an increase in the number of layers and neurons. The computation time for support vector machines also increased non-linearly as the number of input features and data points increased.

6. References

1. https://archive.ics.uci.edu/ml/datasets/APS+Failure+at+Scania+Trucks
2. R. O. Duda, P. E. Hart, and D. G. Stork, Pattern Classification, Second Edition
3. https://scikit-learn.org/stable/documentation.html