Predicting a Pulsar Star

Data set describes a sample of pulsar candidates collected during the High Time Resolution Universe Survey.

Pulsars are a rare type of Neutron star that produce radio emission detectable here on Earth. They are of considerable scientific interest as probes of space-time, the inter-stellar medium, and states of matter.

As pulsars rotate, their emission beam sweeps across the sky, and when this crosses our line of sight, produces a detectable pattern of broadband radio emission. As pulsars rotate rapidly, this pattern repeats periodically. Thus pulsar search involves looking for periodic radio signals with large radio telescopes.

Each pulsar produces a slightly different emission pattern, which varies slightly with each rotation . Thus a potential signal detection known as a 'candidate', is averaged over many rotations of the pulsar, as determined by the length of an observation. In the absence of additional info, each candidate could potentially describe a real pulsar. However in practice almost all detections are caused by radio frequency interference (RFI) and noise, making legitimate signals hard to find.

Machine learning tools are now being used to automatically label pulsar candidates to facilitate rapid analysis. Classification systems in particular are being widely adopted, which treat the candidate data sets as binary classification problems. Here the legitimate pulsar examples are a minority positive class, and spurious examples the majority negative class.

The data set shared here contains 16,259 spurious examples caused by RFI/noise, and 1,639 real pulsar examples. These examples have all been checked by human annotators.

Each row lists the variables first, and the class label is the final entry. The class labels used are 0 (negative) and 1 (positive).

Attribute Information:

Each candidate is described by 8 continuous variables, and a single class variable. The first four are simple statistics obtained from the integrated pulse profile (folded profile). This is an array of continuous variables that describe a longitude-resolved version of the signal that has been averaged in both time and frequency . The remaining four variables are similarly obtained from the DM-SNR curve . These are summarised below:

1. Mean of the integrated profile

2. Standard deviation of the integrated profile

3. Excess kurtosis of the integrated profile

4. Skewness of the integrated profile

5. Mean of the DM-SNR curve
6. Standard deviation of the DM-SNR curve
7. Excess kurtosis of the DM-SNR curve
8. Skewness of the DM-SNR curve
9. Class

17,898 total examples. 1,639 positive examples. 16,259 negative examples.

```
In [36]: import numpy as np # linear algebra
         import pandas as pd # data processing
         import warnings
         warnings.filterwarnings("ignore")
         import matplotlib.pyplot as plt    # basic plotting library
         import seaborn as sns
```

Most of the code was taken from https://www.kaggle.com/efeergun96/prediciting-a-pulsar-star

Loading the dataset

```
In [38]: DataFrame = pd.read_csv("pulsar_stars.csv")
```

EDA

```
In [39]: DataFrame.head()    # first 5 rows of whole columns
```

```
Out[39]:     Mean of the integrated profile  \
         0                        140.562500
         1                        102.507812
         2                        103.015625
         3                        136.750000
         4                         88.726562

             Standard deviation of the integrated profile  \
         0                                       55.683782
         1                                       58.882430
         2                                       39.341649
         3                                       57.178449
         4                                       40.672225

             Excess kurtosis of the integrated profile  \
         0                                   -0.234571
         1                                    0.465318
         2                                    0.323328
```

```
3                                      -0.068415
4                                       0.600866

    Skewness of the integrated profile   Mean of the DM-SNR curve  \
0                             -0.699648                  3.199833
1                             -0.515088                  1.677258
2                              1.051164                  3.121237
3                             -0.636238                  3.642977
4                              1.123492                  1.178930

    Standard deviation of the DM-SNR curve  \
0                                19.110426
1                                14.860146
2                                21.744669
3                                20.959280
4                                11.468720

    Excess kurtosis of the DM-SNR curve   Skewness of the DM-SNR curve  \
0                              7.975532                      74.242225
1                             10.576487                     127.393580
2                              7.735822                      63.171909
3                              6.896499                      53.593661
4                             14.269573                     252.567306

   target_class
0             0
1             0
2             0
3             0
4             0
```

```
In [40]: DataFrame.info() # information about data types and amount of non-null rows of our Data
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 17898 entries, 0 to 17897
Data columns (total 9 columns):
 Mean of the integrated profile                  17898 non-null float64
 Standard deviation of the integrated profile    17898 non-null float64
 Excess kurtosis of the integrated profile       17898 non-null float64
 Skewness of the integrated profile              17898 non-null float64
 Mean of the DM-SNR curve                        17898 non-null float64
 Standard deviation of the DM-SNR curve          17898 non-null float64
 Excess kurtosis of the DM-SNR curve             17898 non-null float64
 Skewness of the DM-SNR curve                    17898 non-null float64
target_class                                     17898 non-null int64
dtypes: float64(8), int64(1)
memory usage: 1.2 MB
```

Data types are all numeric and non-null, I don't need to do any transformations or cleaning.

```
In [41]: DataFrame.describe()    # statistical information about our data
```

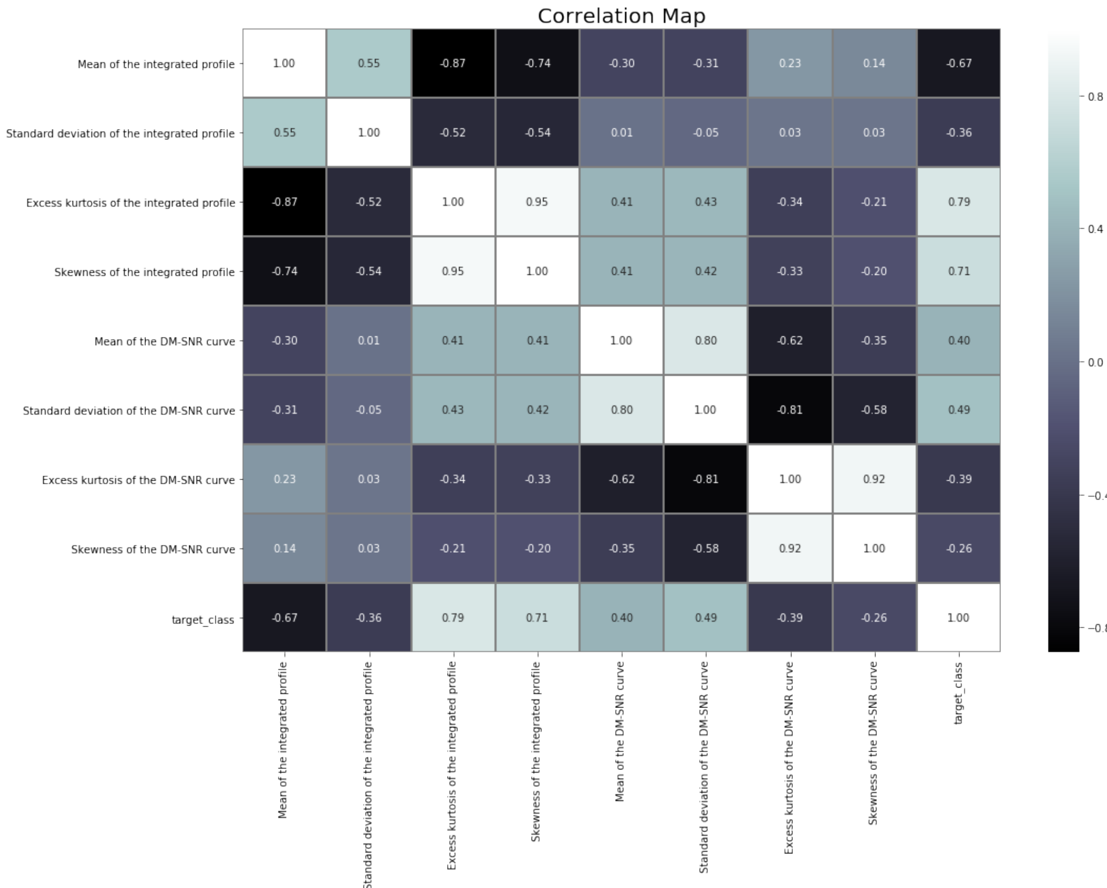
```
Out[41]:         Mean of the integrated profile  \
count                      17898.000000
mean                         111.079968
std                           25.652935
min                            5.812500
25%                          100.929688
50%                          115.078125
75%                          127.085938
max                          192.617188

         Standard deviation of the integrated profile  \
count                                    17898.000000
mean                                        46.549532
std                                          6.843189
min                                         24.772042
25%                                         42.376018
50%                                         46.947479
75%                                         51.023202
max                                         98.778911

         Excess kurtosis of the integrated profile  \
count                                 17898.000000
mean                                      0.477857
std                                       1.064040
min                                      -1.876011
25%                                       0.027098
50%                                       0.223240
75%                                       0.473325
max                                       8.069522

         Skewness of the integrated profile   Mean of the DM-SNR curve  \
count                          17898.000000               17898.000000
mean                               1.770279                  12.614400
std                                6.167913                  29.472897
min                               -1.791886                   0.213211
25%                               -0.188572                   1.923077
50%                                0.198710                   2.801839
75%                                0.927783                   5.464256
max                               68.101622                 223.392140

         Standard deviation of the DM-SNR curve  \
count                              17898.000000
mean                                  26.326515
std                                   19.470572
```

```
min                                       7.370432
25%                                      14.437332
50%                                      18.461316
75%                                      28.428104
max                                     110.642211

        Excess kurtosis of the DM-SNR curve   Skewness of the DM-SNR curve  \
count                          17898.000000                  17898.000000
mean                               8.303556                    104.857709
std                                4.506092                    106.514540
min                               -3.139270                     -1.976976
25%                                5.781506                     34.960504
50%                                8.433515                     83.064556
75%                               10.702959                    139.309331
max                               34.539844                   1191.000837

       target_class
count  17898.000000
mean       0.091574
std        0.288432
min        0.000000
25%        0.000000
50%        0.000000
75%        0.000000
max        1.000000
```

In [43]:
```
# PairPlot (each column is compared to others and itself)

sns.pairplot(data=DataFrame,
             palette="husl",
             hue="target_class",
             vars=[" Mean of the integrated profile",
                   " Excess kurtosis of the integrated profile",
                   " Skewness of the integrated profile",
                   " Mean of the DM-SNR curve",
                   " Excess kurtosis of the DM-SNR curve",
                   " Skewness of the DM-SNR curve"])

plt.suptitle("PairPlot of Data Without Std. Dev. Fields",fontsize=18)

plt.tight_layout()
plt.show()   # pairplot without standard deviaton fields of data
```

We can see that our data is quite separable on most of the columns.

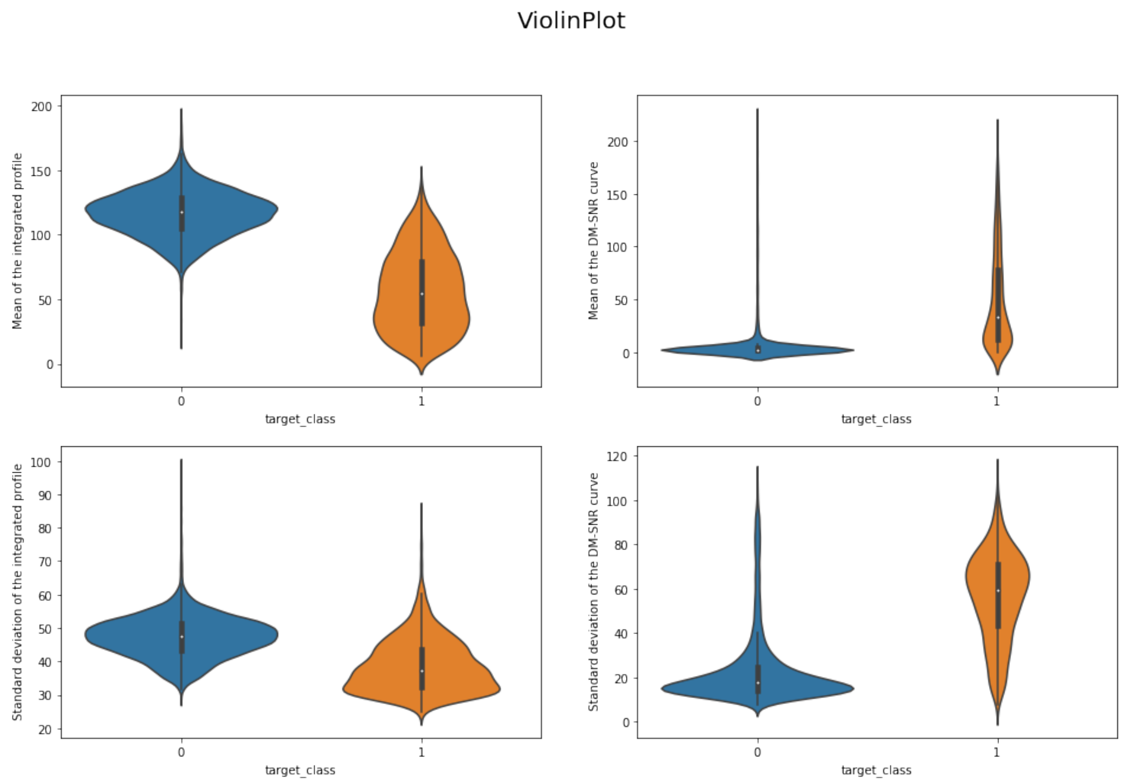
In [44]:
```
# Correlation HeatMap

plt.figure(figsize=(16,12))
sns.heatmap(data=DataFrame.corr(),annot=True,cmap="bone",linewidths=1,fmt=".2f",linecol
plt.title("Correlation Map",fontsize=20)
plt.tight_layout()
plt.show()      # lightest and darkest cells are most correlated ones
```



Most of our Columns are already related or derived from one or another and we can see it clearly on some cells above.

```
In [45]: # ViolinPlot (act as a boxplot but we can see amounts too)

         plt.figure(figsize=(16,10))

         plt.subplot(2,2,1)
         sns.violinplot(data=DataFrame,y=" Mean of the integrated profile",x="target_class")

         plt.subplot(2,2,2)
         sns.violinplot(data=DataFrame,y=" Mean of the DM-SNR curve",x="target_class")

         plt.subplot(2,2,3)
         sns.violinplot(data=DataFrame,y=" Standard deviation of the integrated profile",x="targ

         plt.subplot(2,2,4)
         sns.violinplot(data=DataFrame,y=" Standard deviation of the DM-SNR curve",x="target_cla
```

```
plt.suptitle("ViolinPlot",fontsize=20)

plt.show()
```



We can see that our data has different kind of distributions which is helpful for training our models.

Data PreProcessing

In [46]:
```
# Splitting the Feature and Label fields

labels = DataFrame.target_class.values
DataFrame.drop(["target_class"],axis=1,inplace=True)
features = DataFrame.values
```

In [47]:
```
# Scaling the Features

from sklearn.preprocessing import MinMaxScaler
scaler = MinMaxScaler(feature_range=(0,1))
features_scaled = scaler.fit_transform(features)
```

In [48]:
```
# Splitting the Train and the Test rows

from sklearn.model_selection import train_test_split
x_train, x_test, y_train, y_test = train_test_split(features_scaled,labels,random_state
```

Machine Learning Models

Random Guess

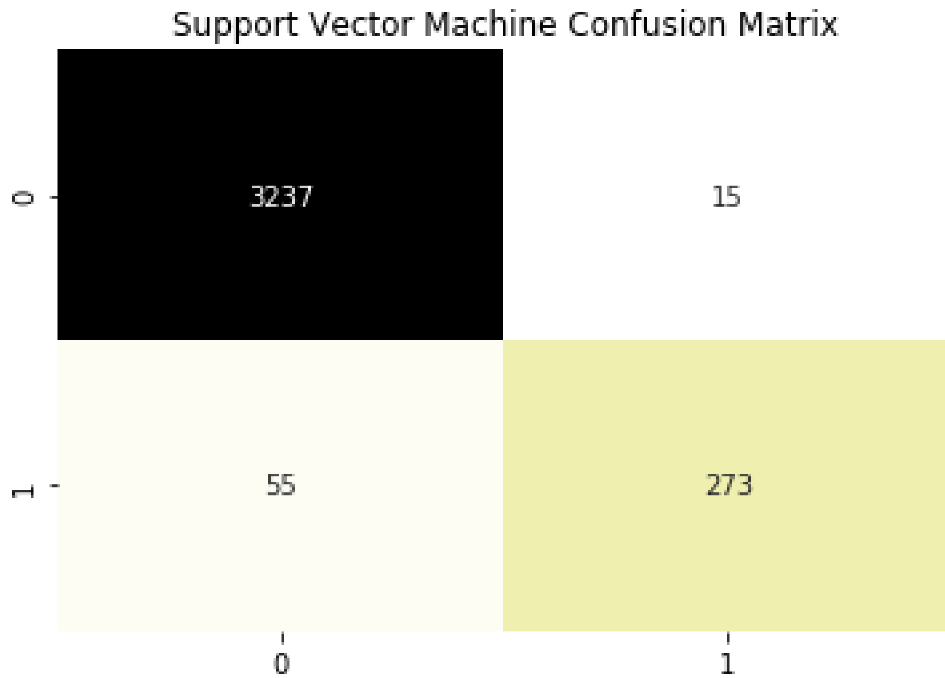
In [49]:
```
data_size = DataFrame.shape[0]
is_positive = np.sum(labels == 1)/data_size #probability that y=1
is_negative = np.sum(labels == 0)/data_size

# Accuracy
print("Accuracy is", '%.03f' %is_negative)

# AUROC
print("AUROC is 0.500")

#AUPRC
print("AUPRC is", '%.03f' %is_positive)
```

```
Accuracy is 0.908
AUROC is 0.500
AUPRC is 0.092
```

SVM

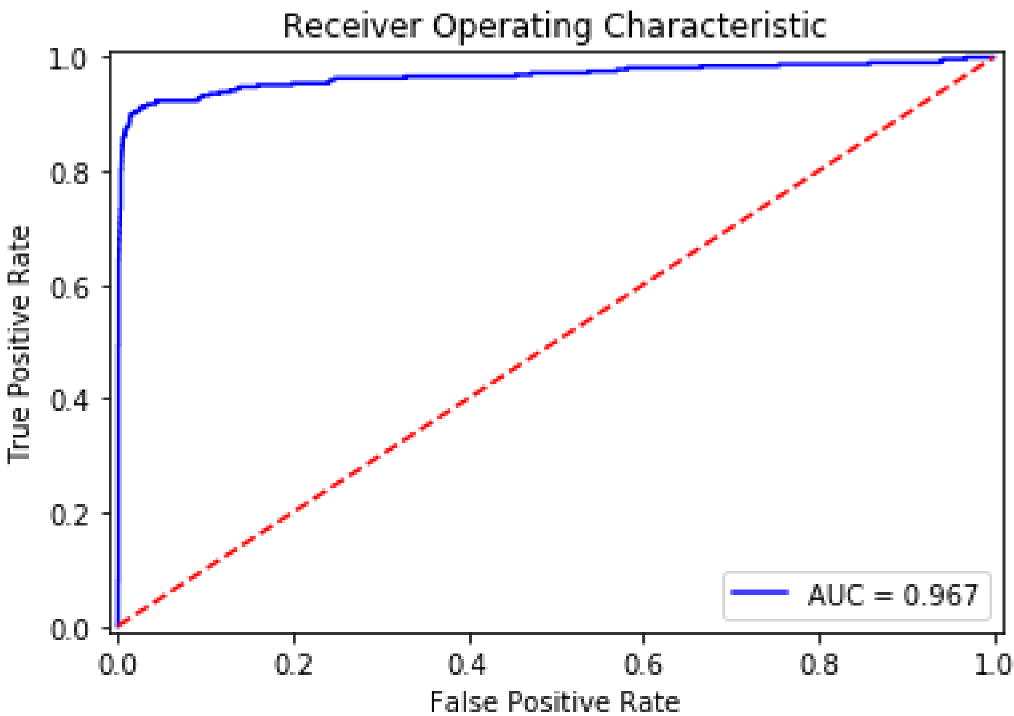
In [50]:
```
# Fitting SVM

from sklearn.svm import SVC
svm_model = SVC(random_state=42,C=250,gamma=1.6,kernel="poly",probability=True)
svm_model.fit(x_train,y_train)
y_head_svm = svm_model.predict(x_test)
svm_score = svm_model.score(x_test,y_test)
```

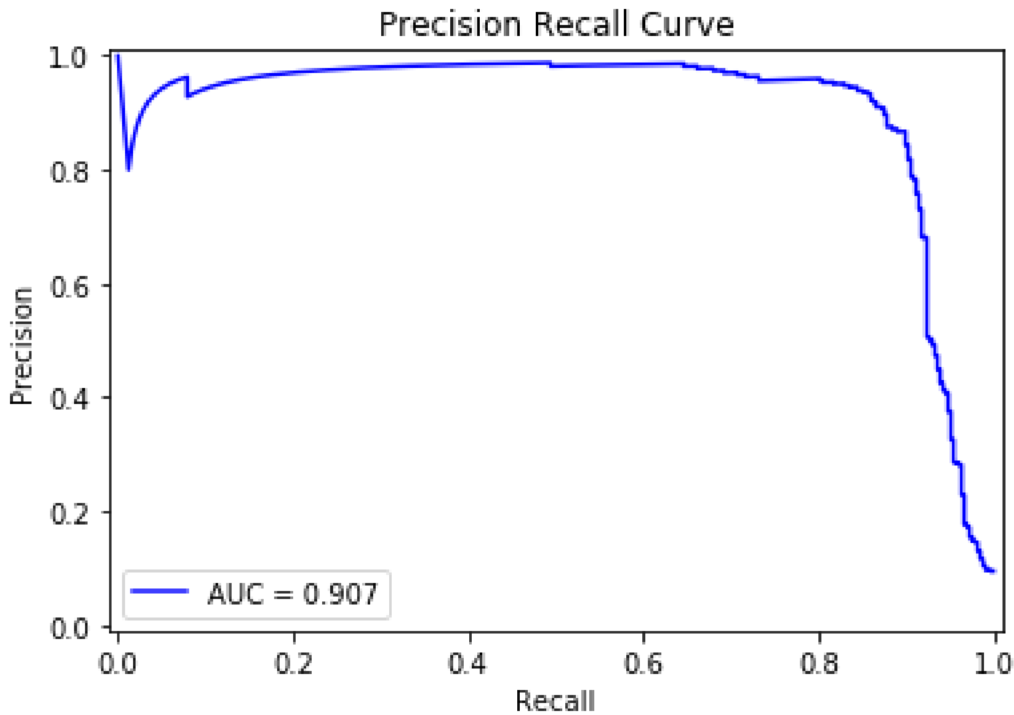
In [51]:
```
# Confusion Matrix

from sklearn.metrics import confusion_matrix
cm_svm = confusion_matrix(y_test,y_head_svm)

plt.title("Support Vector Machine Confusion Matrix")
sns.heatmap(cm_svm,cbar=False,annot=True,cmap="CMRmap_r",fmt="d")
plt.show()
```




```
In [52]: import sklearn.metrics as metrics
         # calculate the fpr and tpr for all thresholds of the classification
         probs = svm_model.predict_proba(x_test) # probabilities for class 0,1
         preds = probs[:,1] # probabilities for class 1
         fpr, tpr, threshold = metrics.roc_curve(y_test, preds)
         roc_auc = metrics.auc(fpr, tpr)

         plt.title('Receiver Operating Characteristic')
         plt.plot(fpr, tpr, 'b', label = 'AUC = %0.3f' % roc_auc)
         plt.legend(loc = 'lower right')
         plt.plot([0, 1], [0, 1],'r--')
         plt.xlim([-0.01, 1.01])
         plt.ylim([-0.01, 1.01])
         plt.ylabel('True Positive Rate')
         plt.xlabel('False Positive Rate')
         plt.show()

         from sklearn.metrics import precision_recall_curve
         from sklearn.metrics import auc
         precision, recall, thresholds = precision_recall_curve(y_test, preds)

         plt.title('Precision Recall Curve')
         plt.plot(recall, precision, 'b', label = 'AUC = %0.3f' % auc(recall, precision))
         plt.legend(loc = 'lower left')
         plt.xlim([-0.01, 1.01])
```

```
plt.ylim([-0.01, 1.01])
plt.ylabel('Precision')
plt.xlabel('Recall')
plt.show()
```



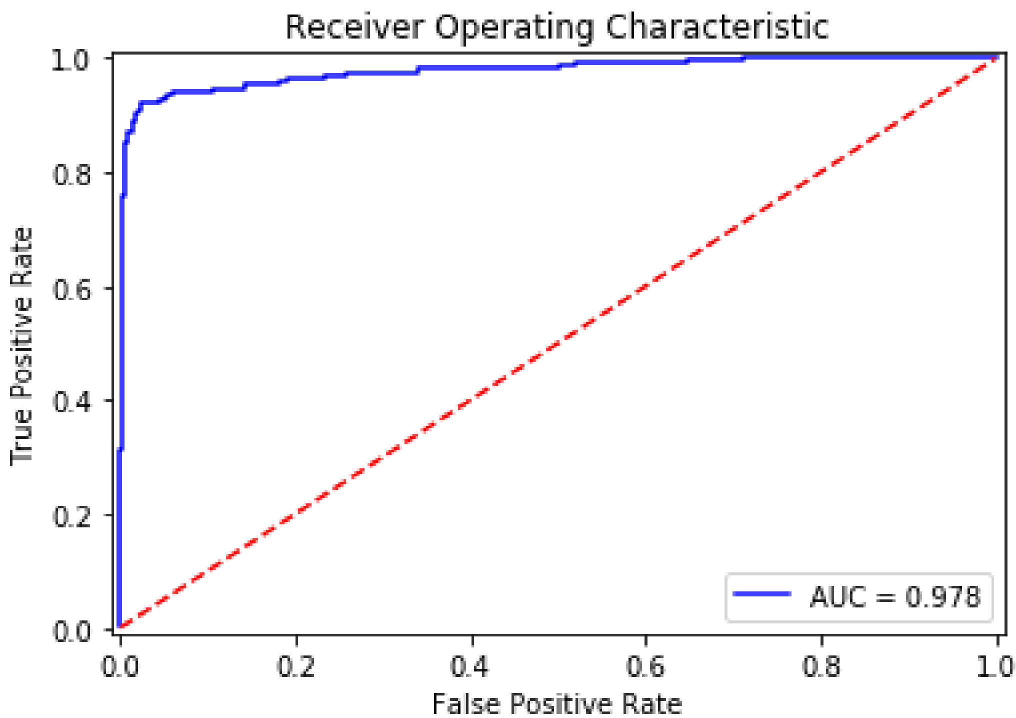


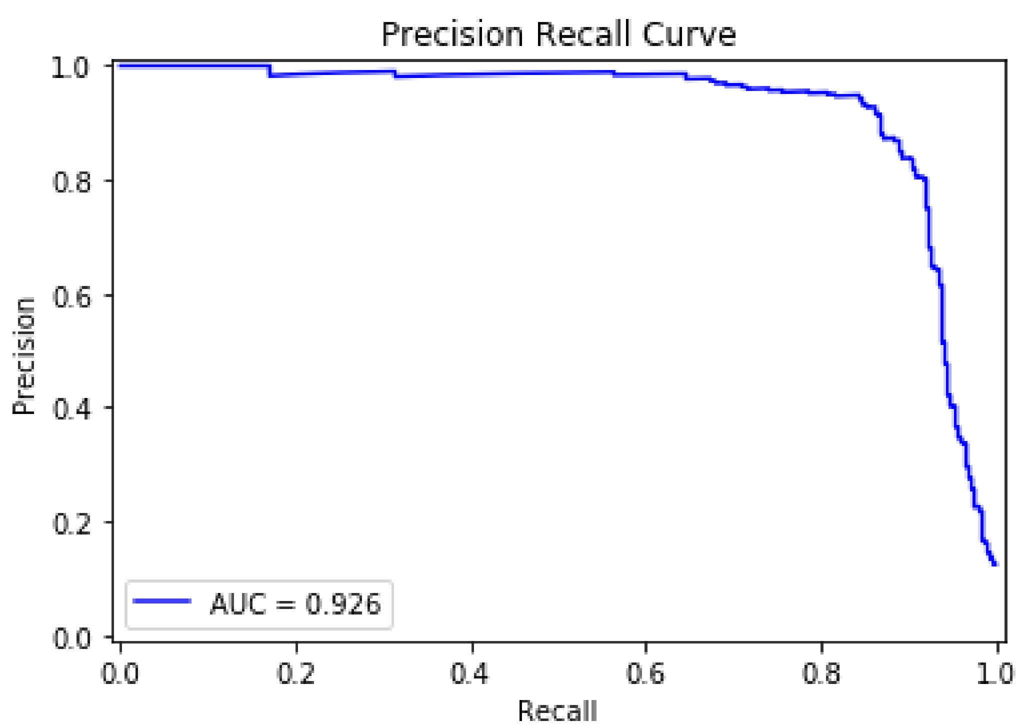

```
In [53]: # accuracy, log_loss and F1

         from sklearn.metrics import log_loss
         from sklearn.metrics import f1_score
         print("Accuracy is", '%.03f' %svm_score)
         print("log_loss is", '%.03f' %log_loss(y_test, probs))
         print("F1 is", '%.03f' %f1_score(y_test, y_head_svm))
```

```
Accuracy is 0.980
log_loss is 0.092
F1 is 0.886
```

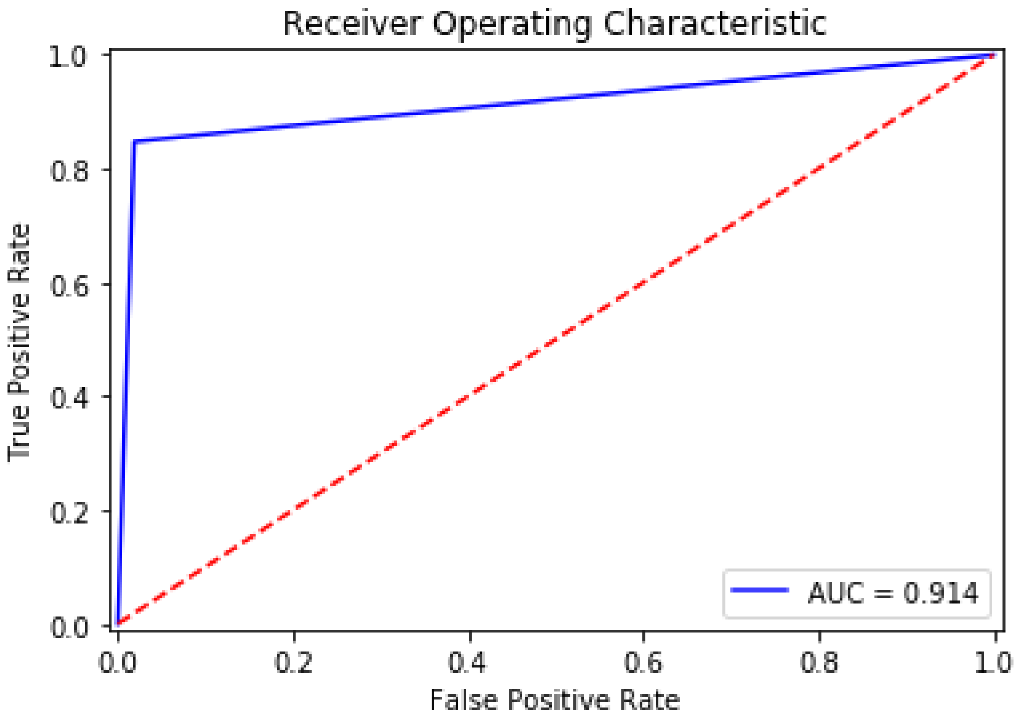
From this section on I will leave out the code as it is very similar to the one above. The full code can be found on GitHub in pulsar_star folder.

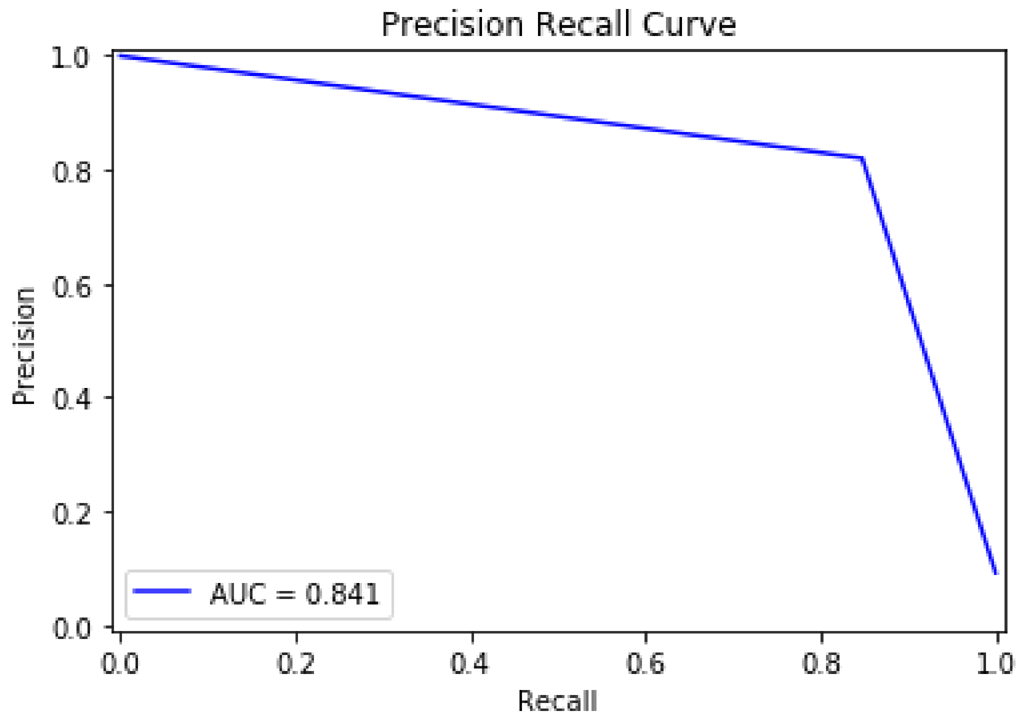
Logistic Regression



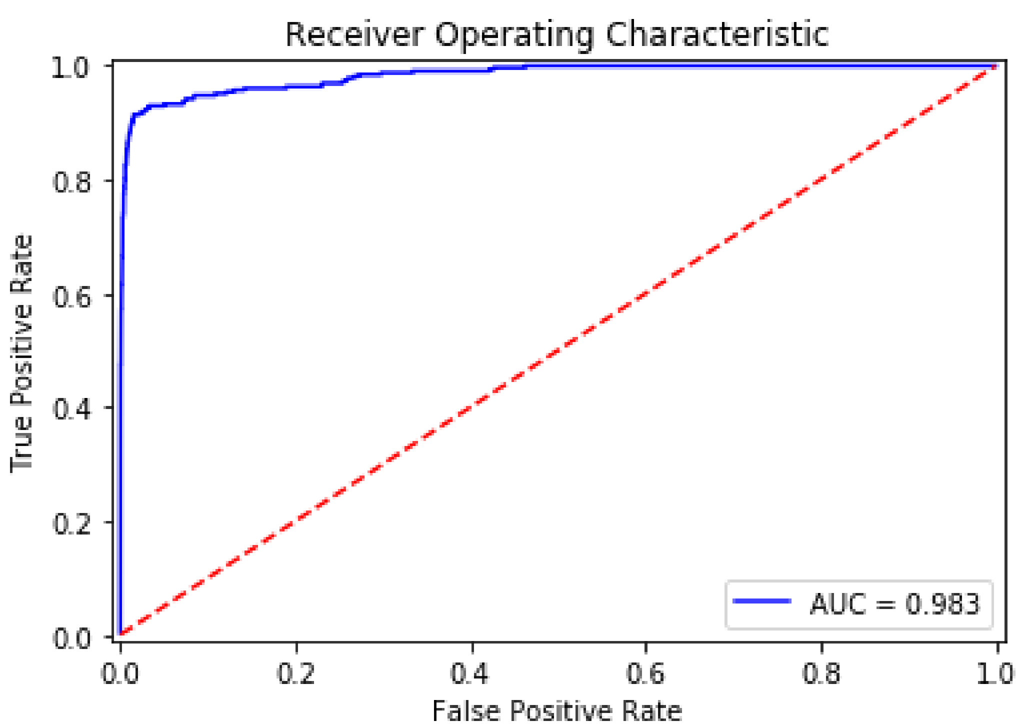


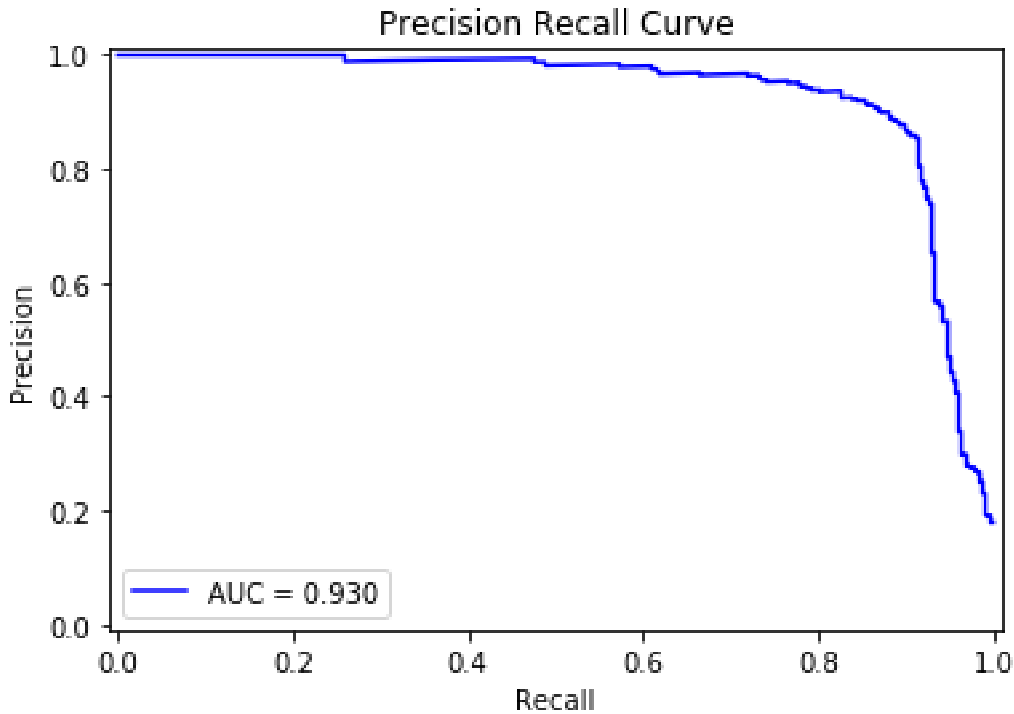
Decision Tree Classifier



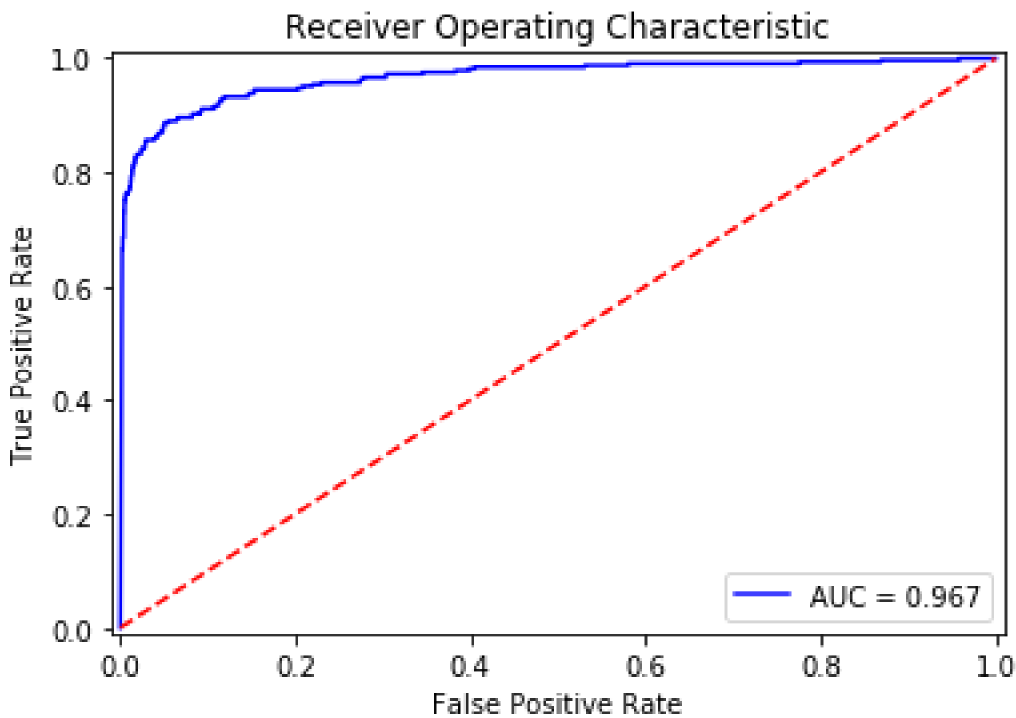


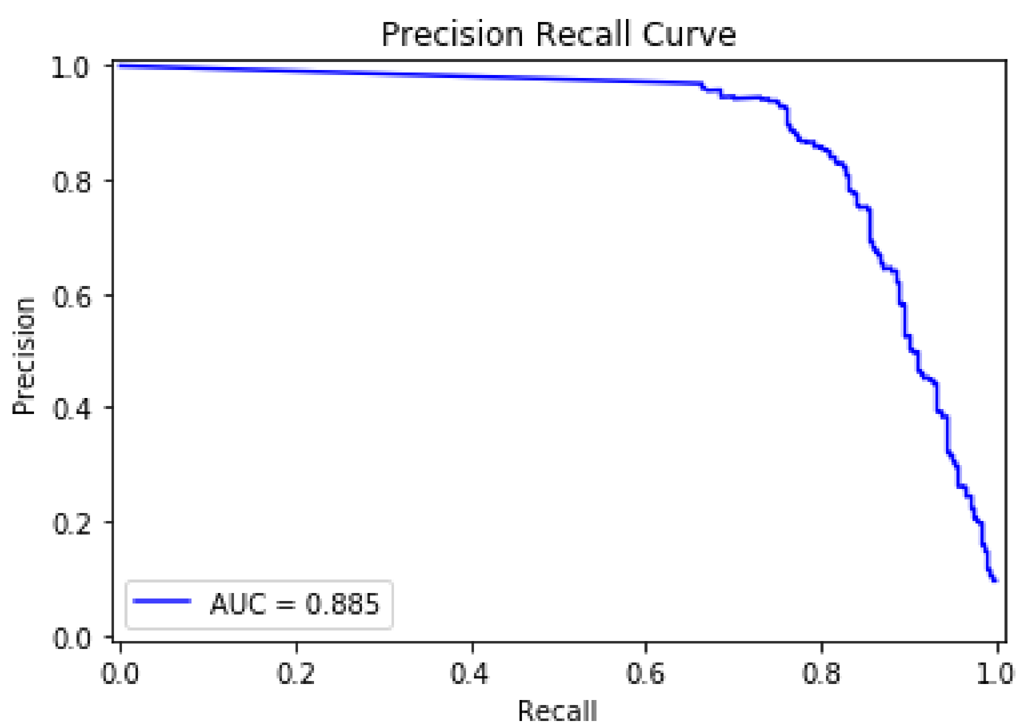
Random Forest Classifier



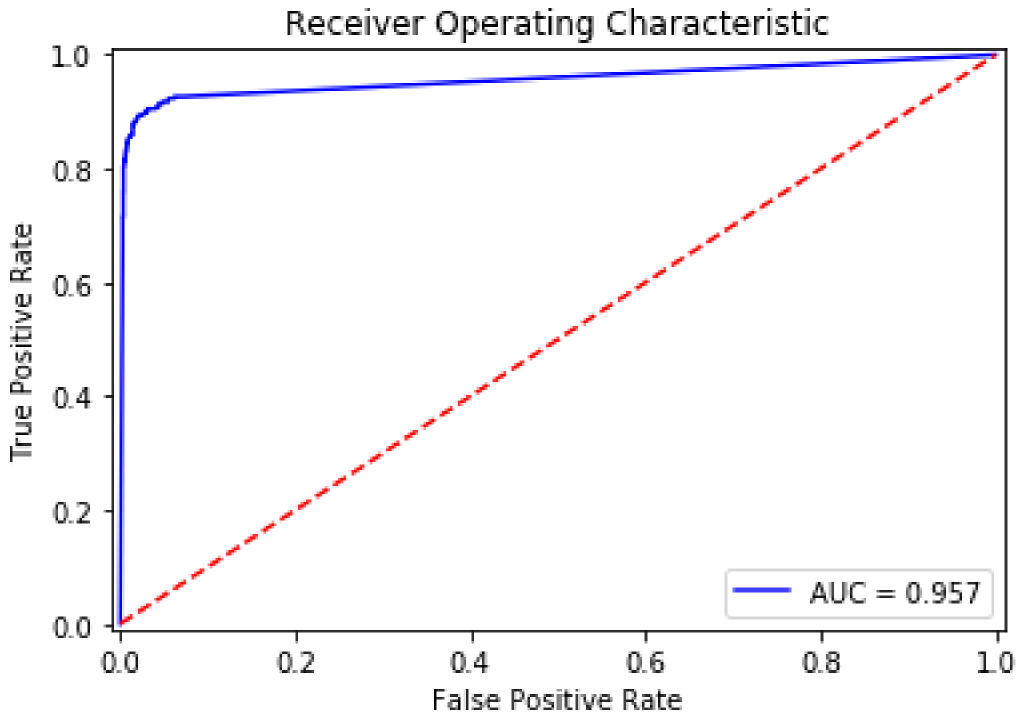

Precision Recall Curve
Precision
Recall
1.0
0.8
0.6
0.4
0.2
0.0
0.0
0.2
0.4
0.6
0.8
1.0
AUC = 0.930

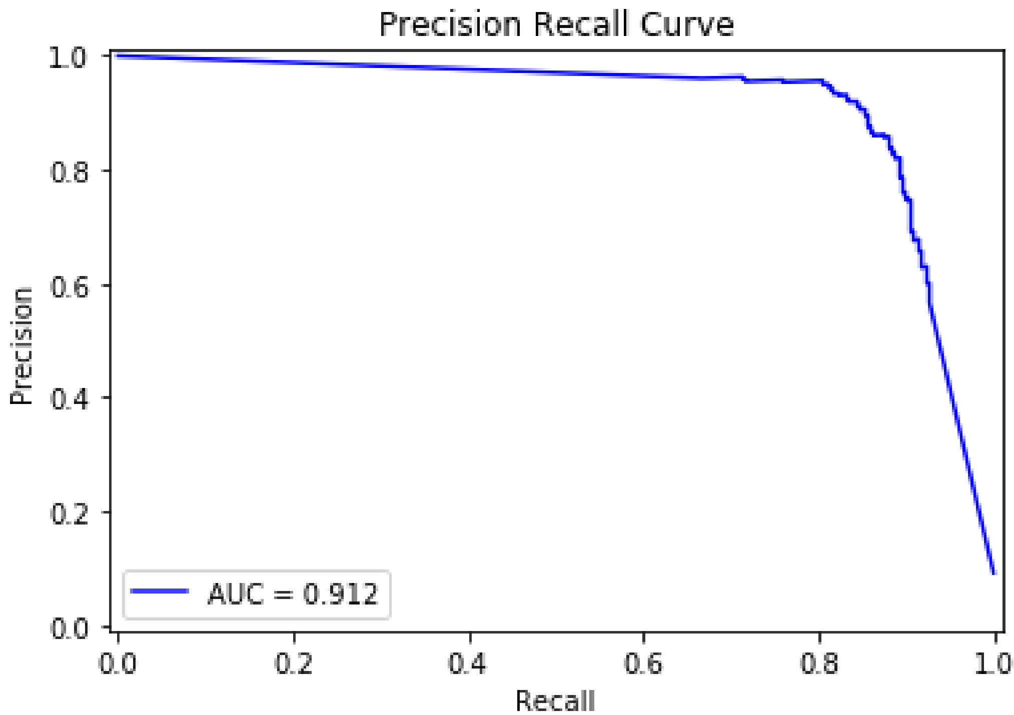
Naive Bayes Classifier





K Nearest Neighbors





Below is the classifier comparison with five different performance measures:

	accuracy	AUROC	AUPRC	log loss	F1
SVM	0.980	0.967	0.907	0.092	0.886
Logistic Regression	0.980	0.978	0.926	0.074	0.883
Decision Tree	0.969	0.914	0.841	1.071	0.834
Random Forest	0.979	0.983	0.930	0.072	0.878
Naive Bayes	0.948	0.967	0.885	0.489	0.755
KNN	0.979	0.957	0.912	0.350	0.877
Random Guess	0.908	0.500	0.092		

Performance Comparison

Overall Analysis

From the table above we can see that all classifiers perform very well.

According to all performance measures we can say that Decision Tree and Naive Bayes perform the worst out of these classifiers. However, AUROC is the only measure where KNN perform worse than Naive Bayes, in fact, Naive Bayes seems to be one of the better classifiers according to AUROC. Though, in other measures, KNN is one of the best classifiers (according to accuracy, AUPRC and F1). Log_loss also agrees with AUROC, KNN is the third worse.

Accuracy and F1 seems to give us similar classifier comparison, with SVM being the best classifier. On the other hand, all other measures show that Random Forest is the best. Random

Forest is the third best according to F1.

In conclusion, AUROC, AUPRC and log_loss agree on the best and the worst classifier. However, there are big disagreements between third best and second worst classifiers. Log_loss gave us the most diverse scores for each classifier. All other measures have very high (and similar) scores for all classifiers, suggesting that this classification problem is easy. In my opinion, log_loss is the best measure in this case.