Beast Cancer Data Analysis Part 1

January 15, 2020

[3]:
```
import pandas as pd
import seaborn as sns # for data visualization
import matplotlib.pyplot as plt # for data visualization
%matplotlib inline
import numpy as np
import os
```

[4]:
```
df = pd.read_csv('Breast_cancer_data.csv')
```

[5]:
```
df.head()
```

[5]:
```
   mean_radius  mean_texture  mean_perimeter  mean_area  mean_smoothness  \
0        17.99         10.38          122.80     1001.0          0.11840
1        20.57         17.77          132.90     1326.0          0.08474
2        19.69         21.25          130.00     1203.0          0.10960
3        11.42         20.38           77.58      386.1          0.14250
4        20.29         14.34          135.10     1297.0          0.10030

   diagnosis
0          0
1          0
2          0
3          0
4          0
```

Data dictionary 1. diagnosis: The diagnosis of breast tissues (1 = malignant, 0 = benign) 2. mean_radius: mean of distances from center to points on the perimeter 3. mean_texture: standard deviation of gray-scale values 4. mean_perimeter: mean size of the core tumor 5. mean_area 6. mean_smoothness: mean of local variation in radius lengths

[6]:
```
df.isnull().sum()
```

[6]:
```
mean_radius        0
mean_texture       0
mean_perimeter     0
mean_area          0
mean_smoothness    0
```

```
diagnosis          0
dtype: int64
```

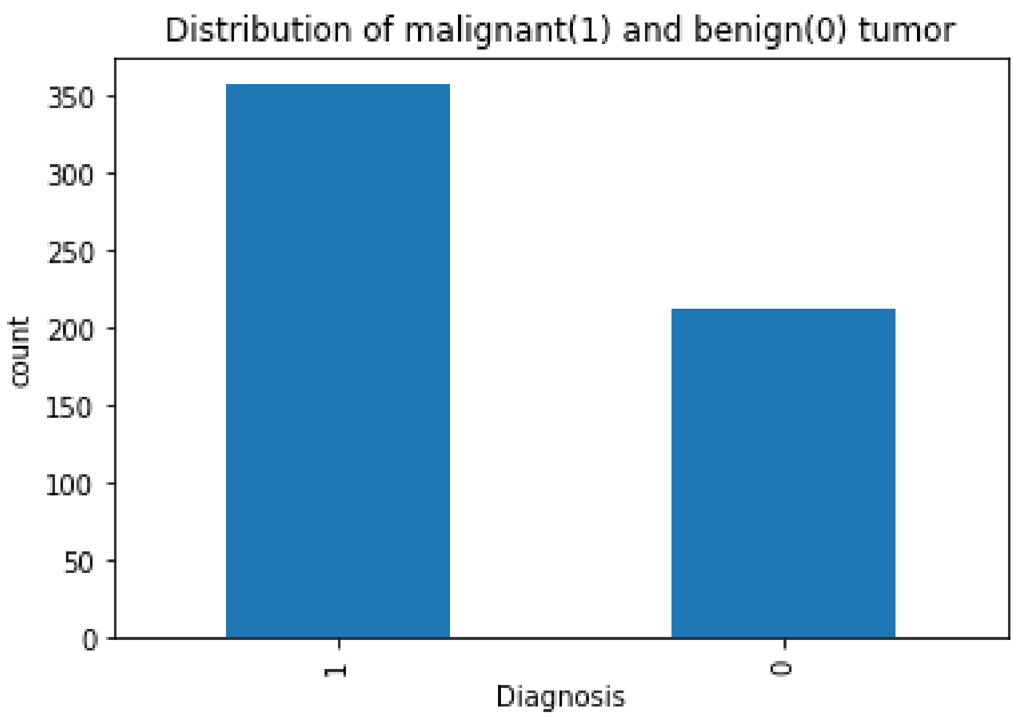
This data set is clean.

[7]:
```
count = df.diagnosis.value_counts()
count
```

[7]:
```
1    357
0    212
Name: diagnosis, dtype: int64
```

The distribution can be visualized as well by using a simple plot function of the matplotlib library.

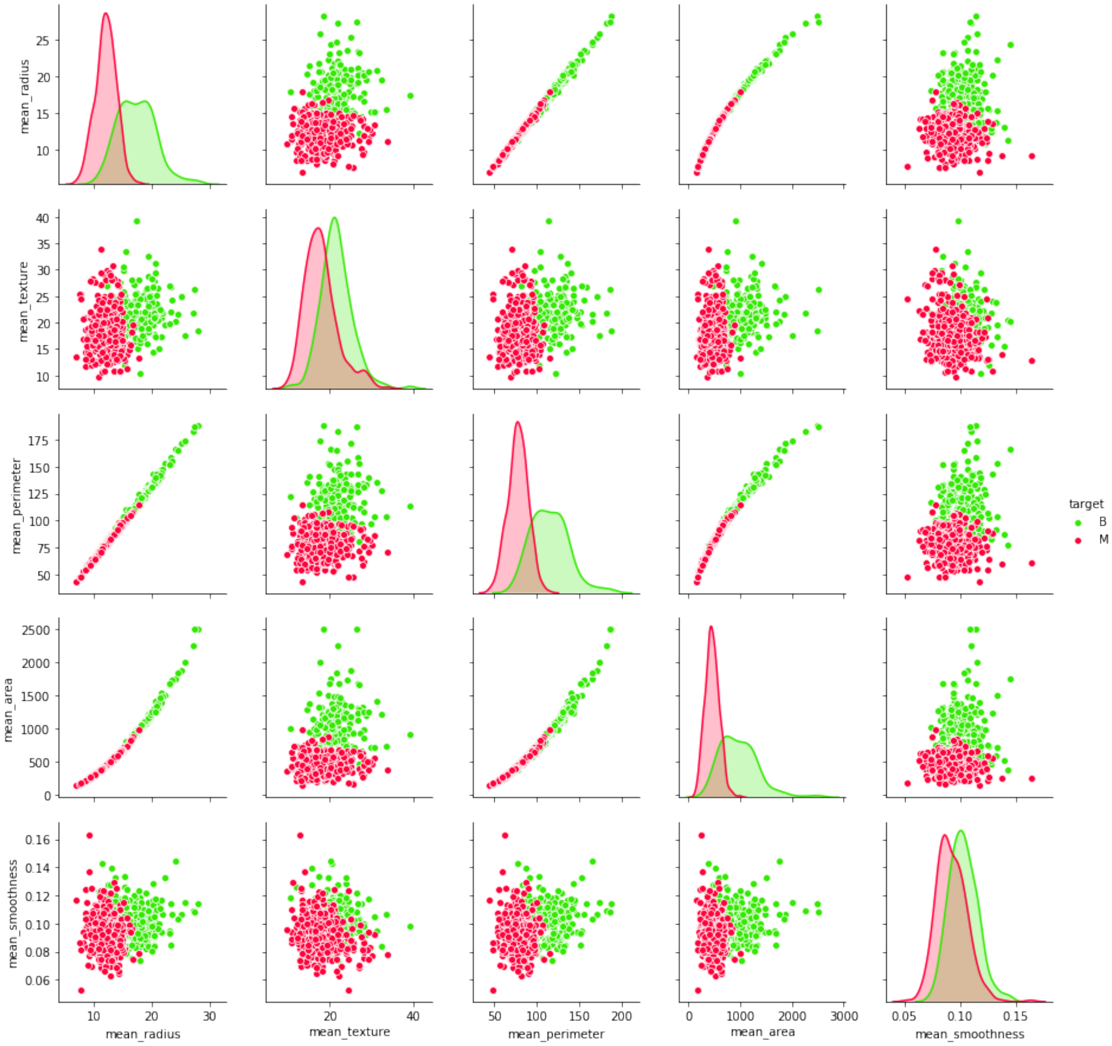
[8]:
```
count.plot(kind='bar')
plt.title("Distribution of malignant(1) and benign(0) tumor")
plt.xlabel("Diagnosis")
plt.ylabel("count");
```



1 Data Visualisation

Let us now plot out the pairplot of different features to determine which features are better at classifying the 2 classes of our problem.

[9]:
```
y_target = df['diagnosis']
df['target'] = df['diagnosis'].map({0:'B',1:'M'})
g = sns.pairplot(df.drop('diagnosis', axis = 1), hue="target", palette='prism');
```



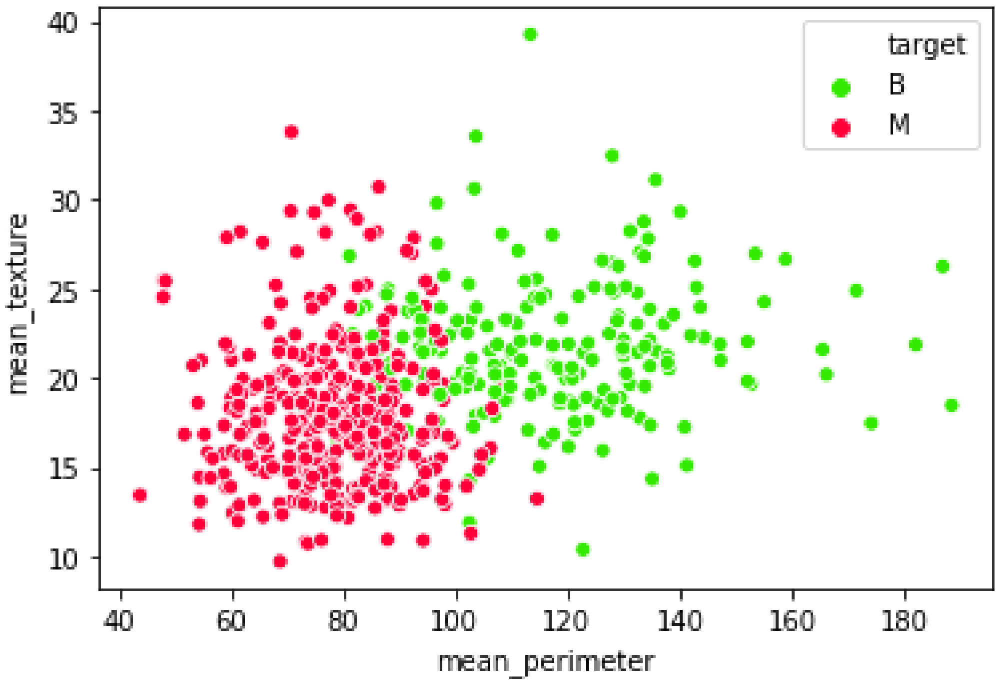
[10]:
```
# Correlation HeatMap

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

2 Logistic Regression

The features mean_perimeter and mean_texture seem to be most relevant from the pair plot above.

[11]:
```
sns.scatterplot(x='mean_perimeter', y = 'mean_texture', data = df, hue =␣
↪'target', palette='prism');
```



[12]:
```
# y_target = breast_cancer_data['diagnosis']
features = ['mean_perimeter', 'mean_texture']
X_feature = df[features]
```

[13]:
```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test= train_test_split(X_feature, y_target,␣
↪test_size=0.3, random_state = 42)
```

Binary classification using Logistic Regression Logistic Regression is mostly used for binary classifications where the dependent variable(target) which are dichotomous in nature(yes or no).

[14]:
```
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score
model1 = LogisticRegression()
model1.fit(X_train, y_train)
```

```
C:\Users\DELL\Anaconda3\lib\site-packages\sklearn\linear_model\logistic.py:432:
FutureWarning: Default solver will be changed to 'lbfgs' in 0.22. Specify a
solver to silence this warning.
  FutureWarning)
```

[14]:
```
LogisticRegression(C=1.0, class_weight=None, dual=False, fit_intercept=True,
                   intercept_scaling=1, l1_ratio=None, max_iter=100,
                   multi_class='warn', n_jobs=None, penalty='l2',
                   random_state=None, solver='warn', tol=0.0001, verbose=0,
```

```
                        warm_start=False)
```

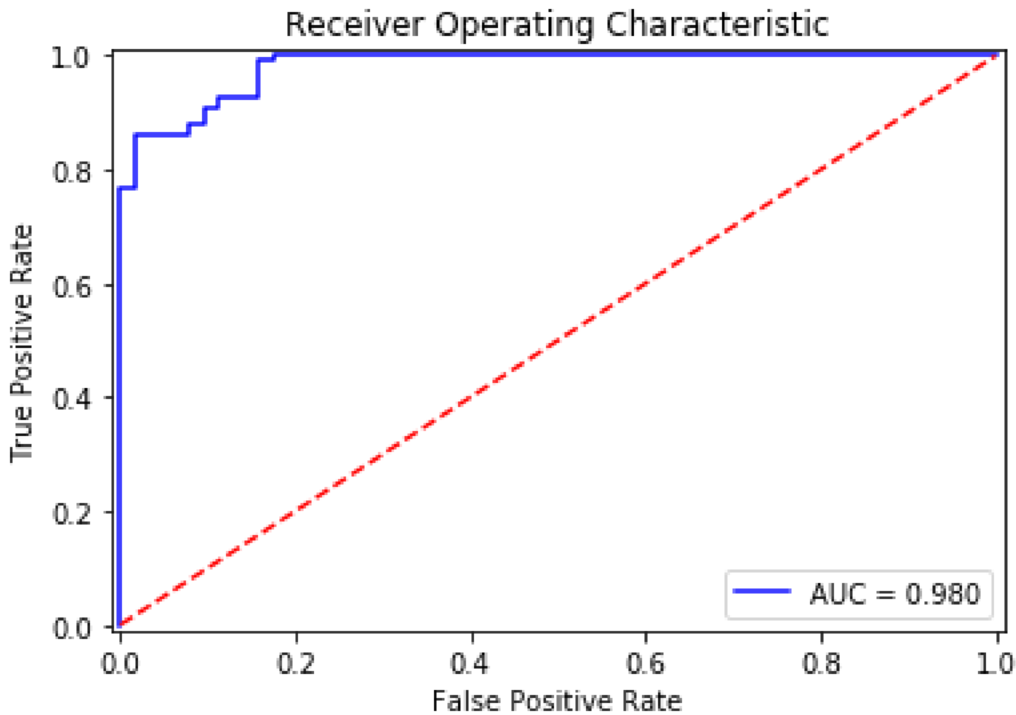
[15]:
```
y_pred1 = model1.predict(X_test)
```

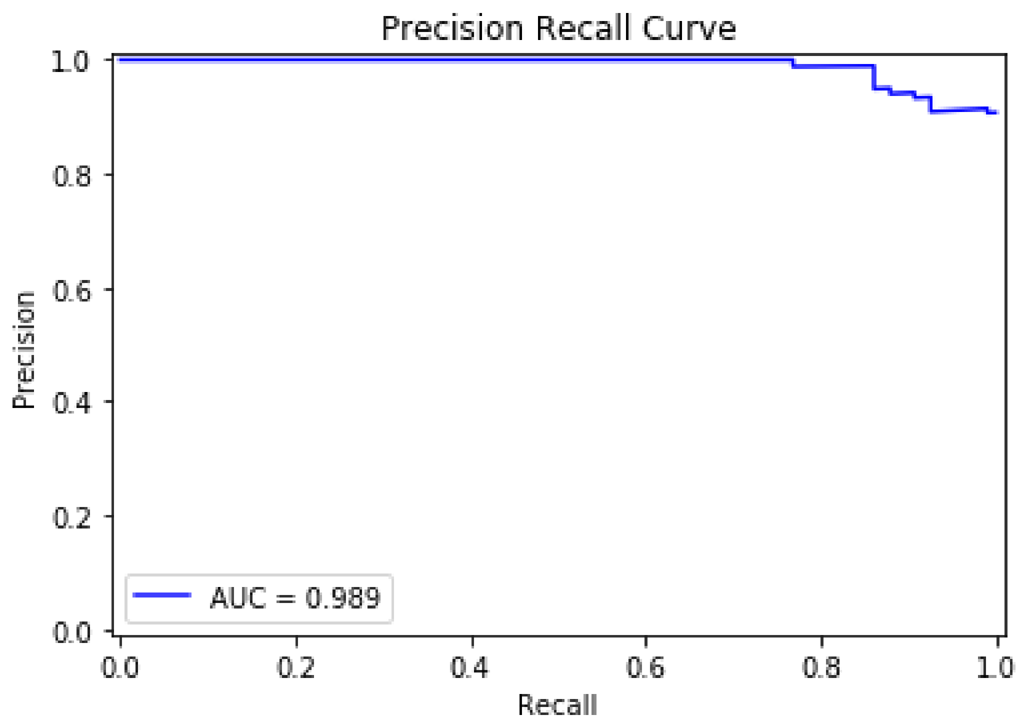
[16]:
```
import sklearn.metrics as metrics
# calculate the fpr and tpr for all thresholds of the classification
probs = model1.predict_proba(X_test) # probabilities for class 0,1
preds = probs[:,1] # probabilities for class 1
fpr, tpr, threshold = metrics.roc_curve(y_test, preds)
roc_auc = metrics.auc(fpr, 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()
```


Receiver Operating Characteristic
True Positive Rate
False Positive Rate
AUC = 0.980
0.0
0.2
0.4
0.6
0.8
1.0


Precision Recall Curve
Precision
Recall
AUC = 0.989
0.0
0.2
0.4
0.6
0.8
1.0

[17]:
```
acc = accuracy_score(y_test, y_pred1)
print("Accuracy score using Logistic Regression:", acc)
```

```
Accuracy score using Logistic Regression: 0.9298245614035088
```

[18]:
```
from sklearn.metrics import classification_report
print(classification_report(y_test,y_pred1))
```

```
              precision    recall  f1-score   support

           0       1.00      0.81      0.89        63
           1       0.90      1.00      0.95       108

    accuracy                           0.93       171
   macro avg       0.95      0.90      0.92       171
weighted avg       0.94      0.93      0.93       171
```

[19]:
```
from sklearn.metrics import log_loss
from sklearn.metrics import f1_score
print("log_loss is", '%.03f' %log_loss(y_test, probs))
print("F1 is", '%.03f' %f1_score(y_test, y_pred1, average='weighted'))
```

```
log_loss is 0.294
F1 is 0.928
```

3 KNN

[20]:
```
from sklearn.neighbors import KNeighborsClassifier
knn = KNeighborsClassifier()
knn.fit(X_train, y_train)
```

[20]:
```
KNeighborsClassifier(algorithm='auto', leaf_size=30, metric='minkowski',
                     metric_params=None, n_jobs=None, n_neighbors=5, p=2,
                     weights='uniform')
```

[21]:
```
y_pred2 = knn.predict(X_test)
```

[22]:
```
acc = accuracy_score(y_test, y_pred2)
print("Accuracy score using KNN:", acc)
```

```
Accuracy score using KNN: 0.9239766081871345
```

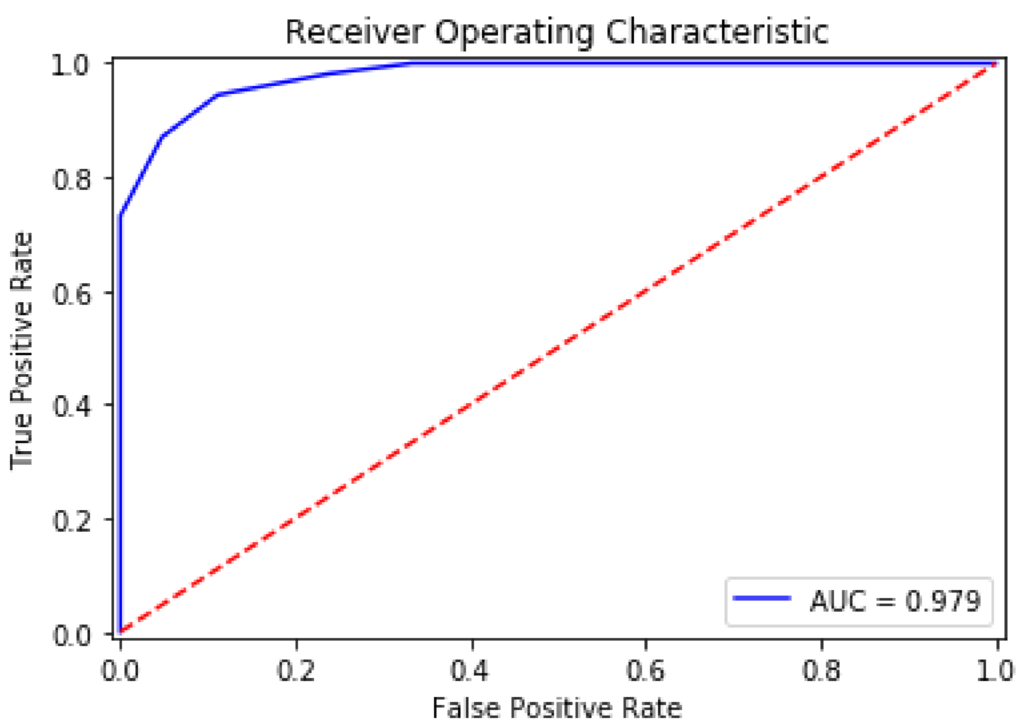
[23]:
```
import sklearn.metrics as metrics
# calculate the fpr and tpr for all thresholds of the classification
probs = knn.predict_proba(X_test) # probabilities for class 0,1
preds = probs[:,1] # probabilities for class 1
```

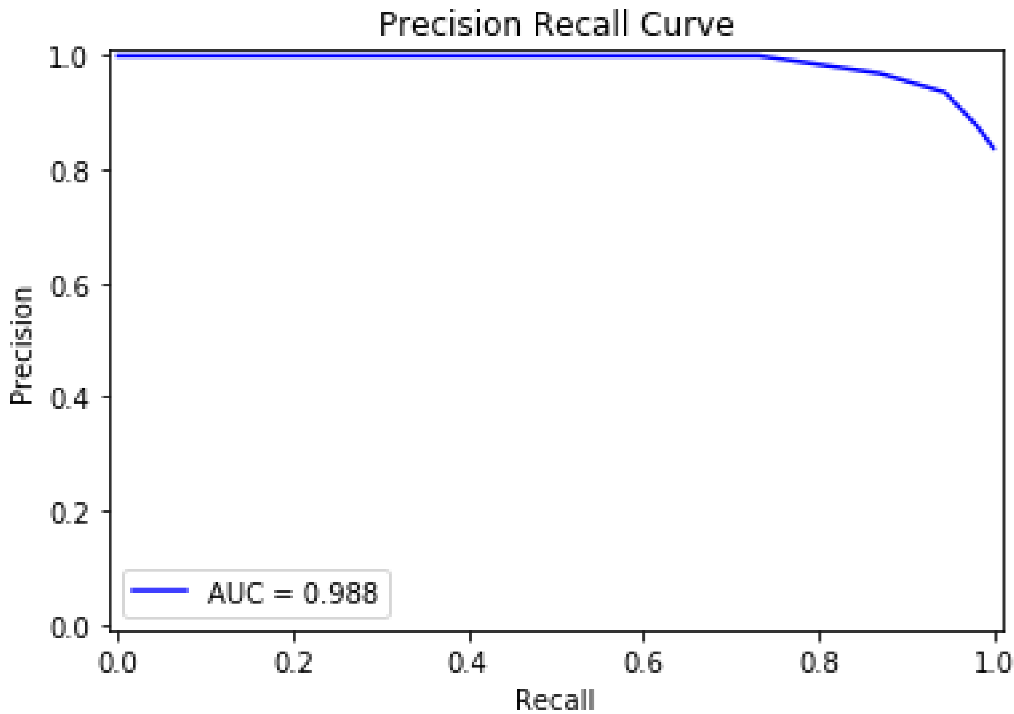
```python
fpr, tpr, threshold = metrics.roc_curve(y_test, preds)
roc_auc = metrics.auc(fpr, tpr)

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()
```





[24]:
```
from sklearn.metrics import classification_report
print(classification_report(y_test,y_pred2))
```

```
              precision    recall  f1-score   support

           0       0.90      0.89      0.90        63
           1       0.94      0.94      0.94       108

    accuracy                           0.92       171
   macro avg       0.92      0.92      0.92       171
weighted avg       0.92      0.92      0.92       171
```

[25]:
```
from sklearn.metrics import log_loss
from sklearn.metrics import f1_score
print("log_loss is", '%.03f' %log_loss(y_test, probs))
print("F1 is", '%.03f' %f1_score(y_test, y_pred2, average='weighted'))
```

```
log_loss is 0.165
F1 is 0.924
```

4 Naive Bayes Classifier

[27]:
```python
from sklearn.naive_bayes import GaussianNB
model3 = GaussianNB()
model3.fit(X_train,y_train)
y_pred3 = nb_model.predict(X_test)
```

```
        ---------------------------------------------------------------------------

        NotFittedError                            Traceback (most recent call last)

        <ipython-input-27-e8efa5923dfb> in <module>
          2 model3 = GaussianNB()
          3 model3.fit(X_train,y_train)
    ----> 4 y_pred3 = nb_model.predict(X_test)


        ~\Anaconda3\lib\site-packages\sklearn\naive_bayes.py in predict(self, X)
         63             Predicted target values for X
         64         """
    ---> 65         jll = self._joint_log_likelihood(X)
         66         return self.classes_[np.argmax(jll, axis=1)]
         67


        ~\Anaconda3\lib\site-packages\sklearn\naive_bayes.py in␣
↪_joint_log_likelihood(self, X)
        426
        427     def _joint_log_likelihood(self, X):
    --> 428         check_is_fitted(self, "classes_")
        429
        430         X = check_array(X)


        ~\Anaconda3\lib\site-packages\sklearn\utils\validation.py in␣
↪check_is_fitted(estimator, attributes, msg, all_or_any)
        912
        913     if not all_or_any([hasattr(estimator, attr) for attr in␣
↪attributes]):
    --> 914         raise NotFittedError(msg % {'name': type(estimator).__name__})
        915
        916


        NotFittedError: This GaussianNB instance is not fitted yet. Call 'fit'␣
↪with appropriate arguments before using this method.
```