Gaussian Classification Model

Виконав
Левадний Микола, MI-4

Dataset

Breast Cancer Dataset
Binary Classification Prediction for type of Breast Cancer

About dataset:

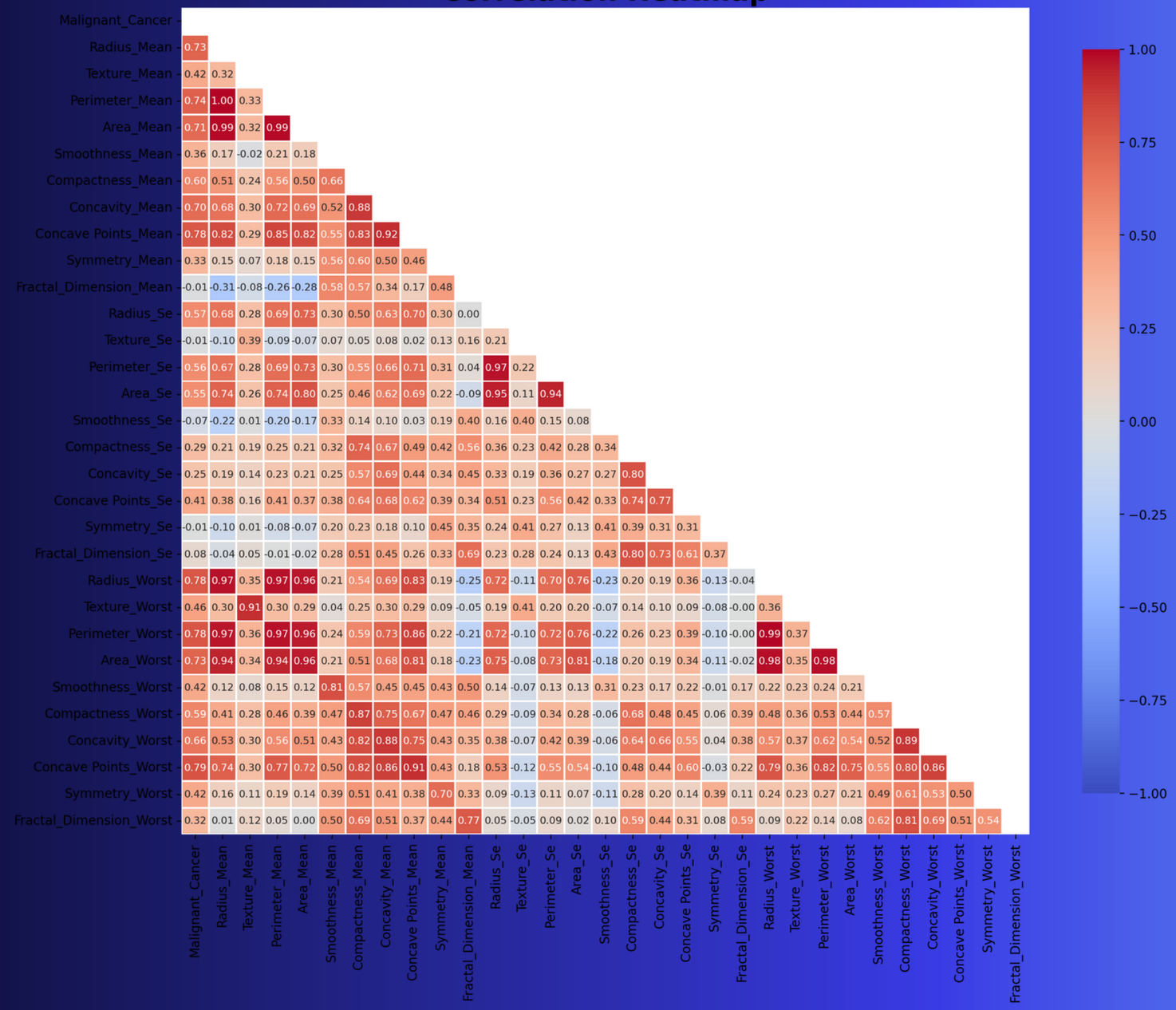
Breast cancer is the most common cancer amongst women in the world. It accounts for 25% of all cancer cases, and affected over 2.1 Million people in 2015 alone. It starts when cells in the breast begin to grow out of control. These cells usually form tumors that can be seen via X-ray or felt as lumps in the breast area.
The key challenges against it’s detection is how to classify tumors into malignant (cancerous) or benign(non cancerous). We ask you to complete the analysis of classifying these tumors using machine learning (with SVMs) and the Breast Cancer Wisconsin (Diagnostic) Dataset.

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 569 entries, 0 to 568
Data columns (total 32 columns):
 #   Column                   Non-Null Count  Dtype  
---  ------                   --------------  -----  
 0   id                       569 non-null    int64  
 1   diagnosis                569 non-null    object 
 2   radius_mean              569 non-null    float64
 3   texture_mean             569 non-null    float64
 4   perimeter_mean           569 non-null    float64
 5   area_mean                569 non-null    float64
 6   smoothness_mean          569 non-null    float64
 7   compactness_mean         569 non-null    float64
 8   concavity_mean           569 non-null    float64
 9   concave points_mean      569 non-null    float64
 10  symmetry_mean            569 non-null    float64
 11  fractal_dimension_mean   569 non-null    float64
 12  radius_se                569 non-null    float64
 13  texture_se               569 non-null    float64
 14  perimeter_se             569 non-null    float64
 15  area_se                  569 non-null    float64
 16  smoothness_se            569 non-null    float64
 17  compactness_se           569 non-null    float64
 18  concavity_se             569 non-null    float64
 19  concave points_se        569 non-null    float64
 20  symmetry_se              569 non-null    float64
 21  fractal_dimension_se     569 non-null    float64
 22  radius_worst             569 non-null    float64
 23  texture_worst            569 non-null    float64
 24  perimeter_worst          569 non-null    float64
 25  area_worst               569 non-null    float64
 26  smoothness_worst         569 non-null    float64
 27  compactness_worst        569 non-null    float64
 28  concavity_worst          569 non-null    float64
 29  concave points_worst     569 non-null    float64
 30  symmetry_worst           569 non-null    float64
 31  fractal_dimension_worst  569 non-null    float64
dtypes: float64(30), int64(1), object(1)
memory usage: 142.4+ KB
```

Tumors characteristics

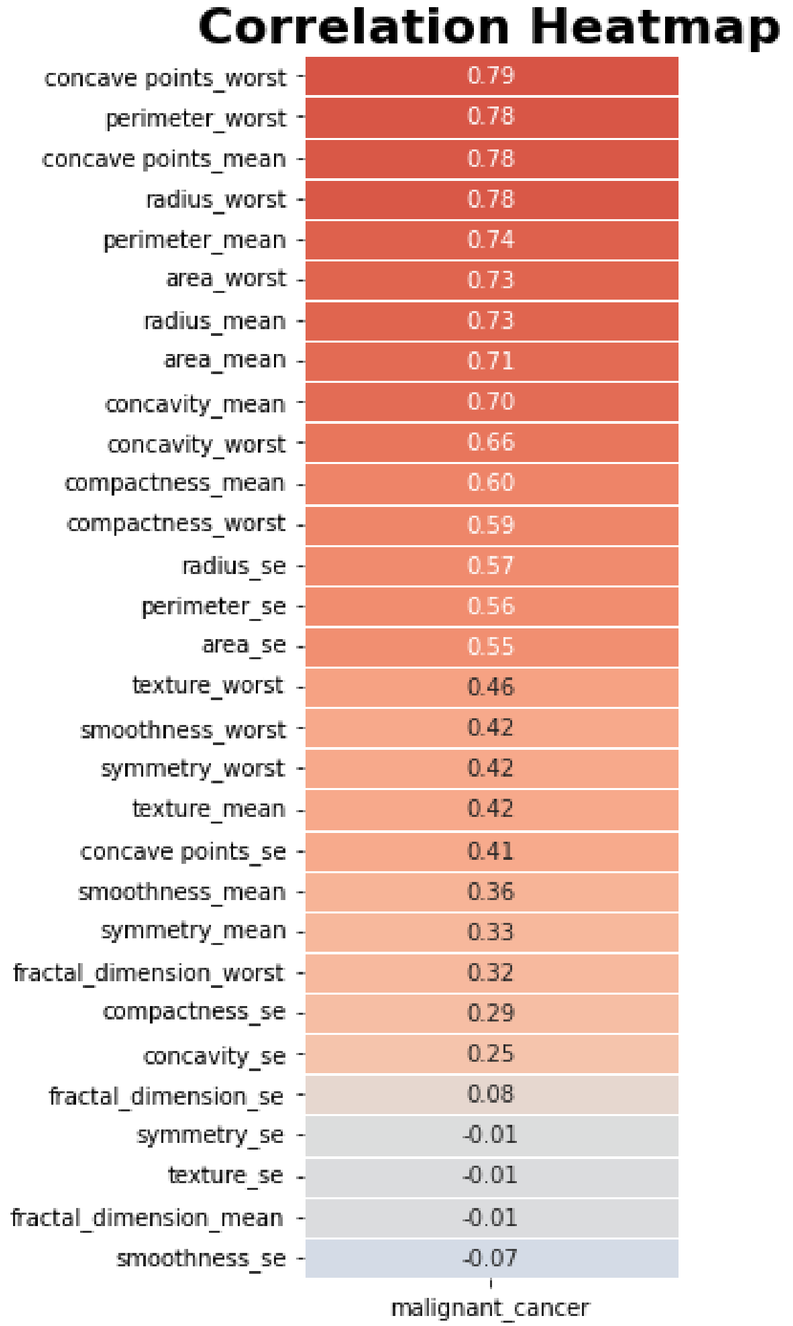
Correlation HeatMap



Correlation

We see that many features correlate strongly with breast cancer. This is a good thing. We can assume our ML model will work great knowing this.

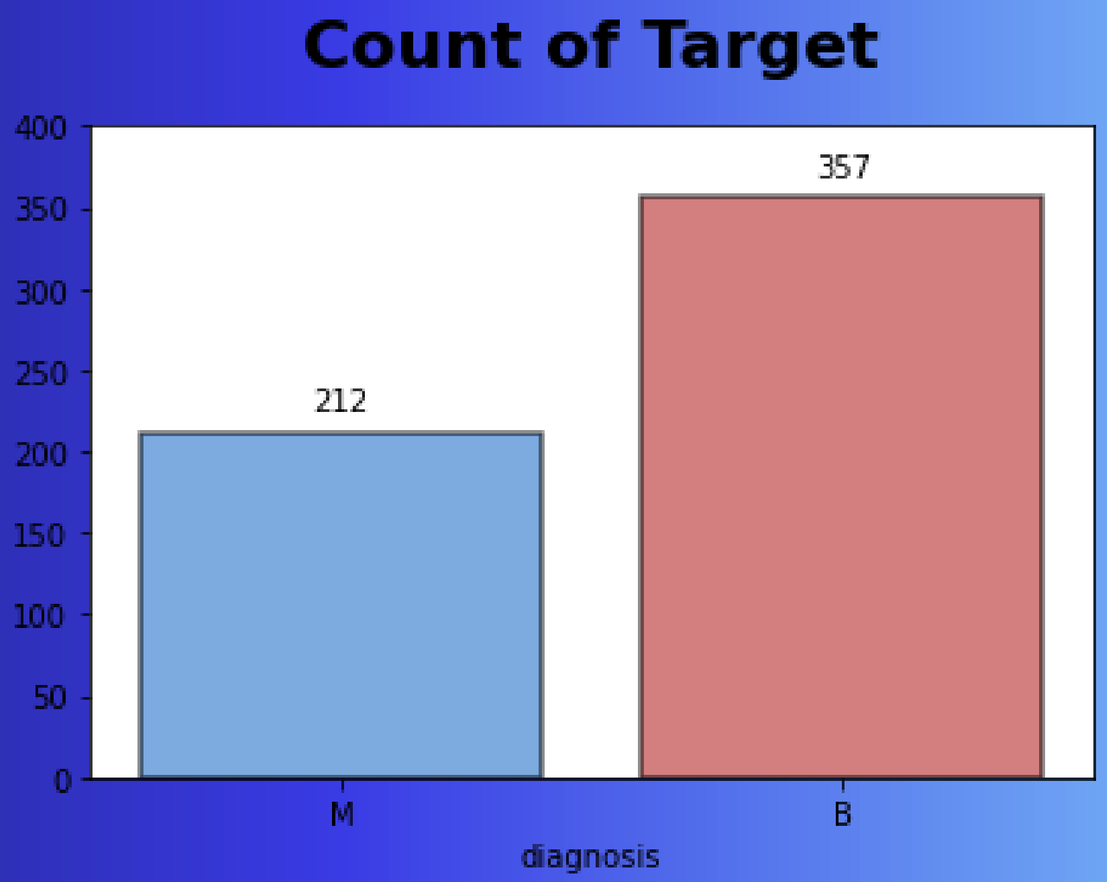
```
[23] fig, ax = plt.subplots(figsize=(3, 10))

     cmap = sns.color_palette("coolwarm", as_cmap=True)
     sns.heatmap(data = data_cancer, annot=True, fmt=".2f", cmap=cmap, linewidths= 1,
                 vmin=-1, vmax=1, cbar= False)

     plt.title("Correlation Heatmap", fontsize=22, fontweight= "bold")

     plt.show()
```



Count of Targer

The difference is high but not too big. Based on this, we can not say that data is imbalanced. And we going to proceed normally.

```
plot= sns.countplot(data= df, x= "diagnosis", palette=['#0b6fe7',"#cf1112"],
              alpha= 0.6, edgecolor="0", linewidth=1.5)

for bar in plot.patches:
    plot.annotate(format(bar.get_height(), '.0f'),
        (bar.get_x() + bar.get_width() / 2,
        bar.get_height()), ha='center', va='center',
        size=10, xytext=(0, 10),
        textcoords='offset points')

plt.ylim(0, 400)
plt.title("Count of Target", fontsize=22, fontweight= "bold", pad= 20)
plt.ylabel("")

plt.show()
```



Data preparation

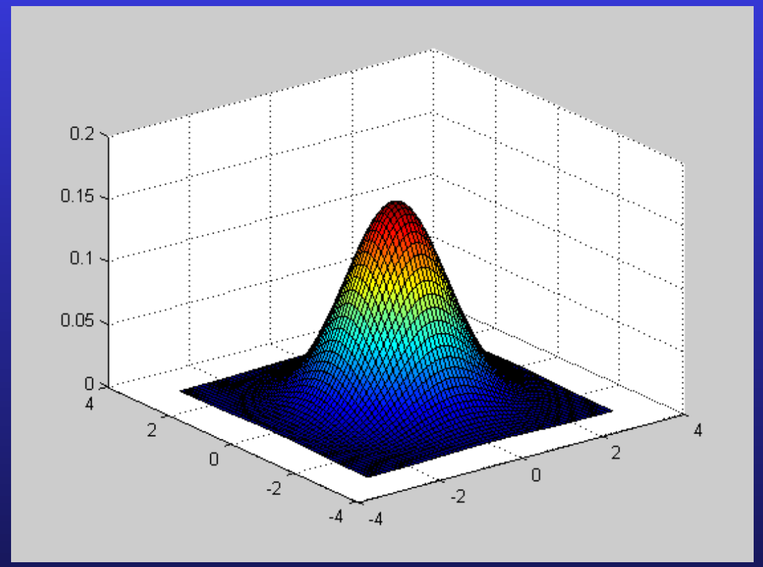
```
df= df.rename({"diagnosis": "malignant_cancer"}, axis= "columns")
X= df.drop(["id", "malignant_cancer"], axis= "columns")
y= df["malignant_cancer"]
y= y.map({"M": 1, "B": 0})
```

```
[26] X_train, X_test, y_train, y_test = train_test_split(
         X, y, test_size= 0.25, random_state= 100)
```

```
[27] scaler= StandardScaler()

     scaler.fit(X_train)
     X_train_scaled= scaler.transform(X_train)
     X_test_scaled= scaler.transform(X_test)
```

1. drop unused column
2. make dummy column
3. standardizes a features by subtracting the mean and then scaling to unit variance

Out-of-box classifier



The main idea is to build a classifier under the assumption that the function p(x|y) (the so-called likelihood function, i.e. the distribution of objects with a fixed answer y) is known for each class and is equal to the density of the multivariate normal (Gaussian) distribution:

$$p(x|y) = N(\mu_y, \Sigma_y) = \frac{1}{\sqrt{(2\pi)^D |\det(\Sigma_y)|}} \exp\left(-\frac{1}{2}(x-\mu_y)^T \Sigma_y^{-1}(x-\mu_y)\right),$$

$y \in \{1, 2, \ldots, C\}$.

Σ_y – covariance matrix

μ_y – expectation vector

N – number of objects

D – feature space dimension

Python implementation

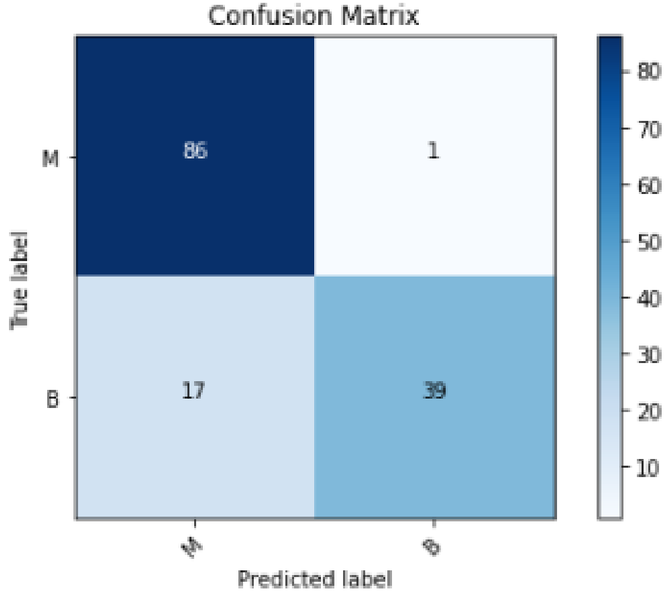
Gaussian classifier for simple binary classification in Python. We create GaussianClassifier class with two methods — train() and predict().

```python
class GaussianClassifier:
  def train(self,X,y):
    #class-conditional density - MLE estimate
    self.mean_0 = np.average(X[y==0], axis=0)
    self.mean_1 = np.average(X[y==1], axis=0)
    self.sigma_0 = np.cov(X[y==0], rowvar=False)
    self.sigma_1 = np.cov(X[y==0], rowvar=False)

    #class prior
    self.pi_0 = y[y==0].shape[0]/y.shape[0]
    self.pi_1 = 1.0 - self.pi_0

    #multivariate class-conditional density function
    self.prob_0 = multivariate_normal(self.mean_0,self.sigma_0,allow_singular=True)
    self.prob_1 = multivariate_normal(self.mean_1,self.sigma_1,allow_singular=True)

  def predict(self, X):
    preds = []
    for x_input in X:
      #class posterior
      if self.prob_0.pdf(x_input)*self.pi_0 > self.prob_1.pdf(x_input)*self.pi_1:
        preds.append(0)
      else:
        preds.append(1)

    return np.array(preds)
```

Our model train and test

```
clf = GaussianClassifier()
clf.train(X_train_scaled, y_train)

ypred_test = clf.predict(X_test_scaled)
print("Out-of-box prediction accuracy of test data:")
print(accuracy_score(y_test,ypred_test))

print("Out-of-box confution matrix of test data:")
plot_confusion_matrix(cm=confusion_matrix(y_test,ypred_test), classes=cm_plot_labels, title='Confusion Matrix')
```

```
Out-of-box prediction accuracy of test data:
0.8741258741258742
Out-of-box confution matrix of test data:
```

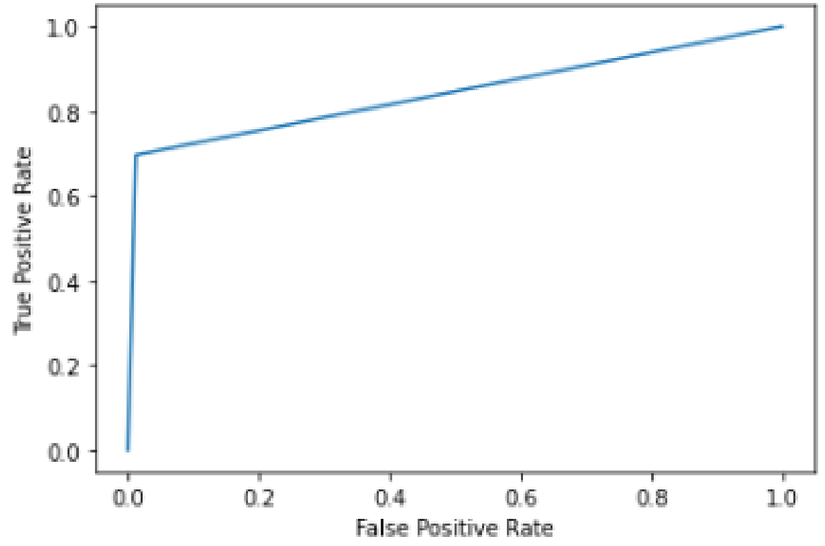


ROC curve

The more that the curve hugs the top left corner of the plot, the better the model does at classifying the data into categories.

As we can see from the plot, our model does a great job of classifying the data into categories.

```
fpr, tpr, _ = metrics.roc_curve(y_test,  ypred_test)

#create ROC curve
plt.plot(fpr,tpr)
plt.ylabel('True Positive Rate')
plt.xlabel('False Positive Rate')
plt.show()
```

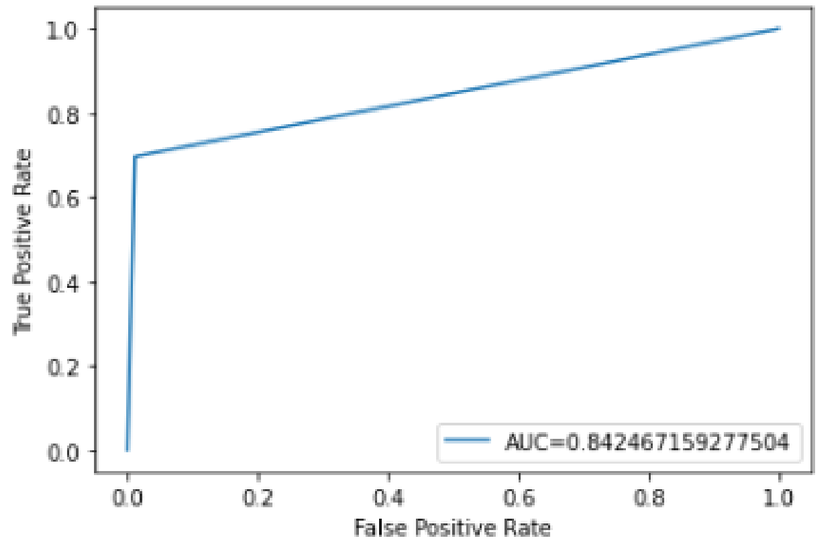


AUC

AUC - area under the curve - which tells us how much of the plot is located under the curve.
The closer AUC is to 1, the better the model. A model with an AUC equal to 0.5 is no better than a model that makes random classifications.

The AUC for out model turns out to be 0.842. Since this is pretty close to 1, this confirms that the model does a nice job of classifying data.

```
auc = metrics.roc_auc_score(y_test, ypred_test)

#create ROC curve
plt.plot(fpr,tpr,label="AUC="+str(auc))
plt.ylabel('True Positive Rate')
plt.xlabel('False Positive Rate')
plt.legend(loc=4)
plt.show()
```



In-box classifier

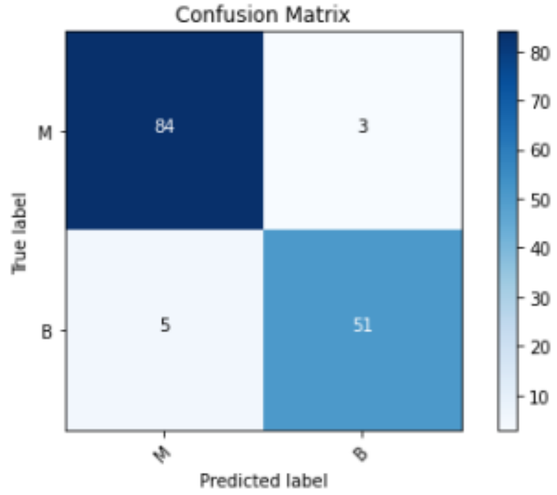
model train and test

```
[47] from sklearn.naive_bayes import GaussianNB

     nb = GaussianNB()
     nb.fit(X_train_scaled, y_train)
```

```
GaussianNB()
```

```
ypred_gnb = nb.predict(X_test_scaled)
print("In-box prediction accuracy of test data:")
print(accuracy_score(y_test,ypred_gnb))

print("In-box confution matrix of test data:")
plot_confusion_matrix(cm=confusion_matrix(y_test,ypred_gnb), classes=cm_plot_labels, title='Confusion Matrix')
```

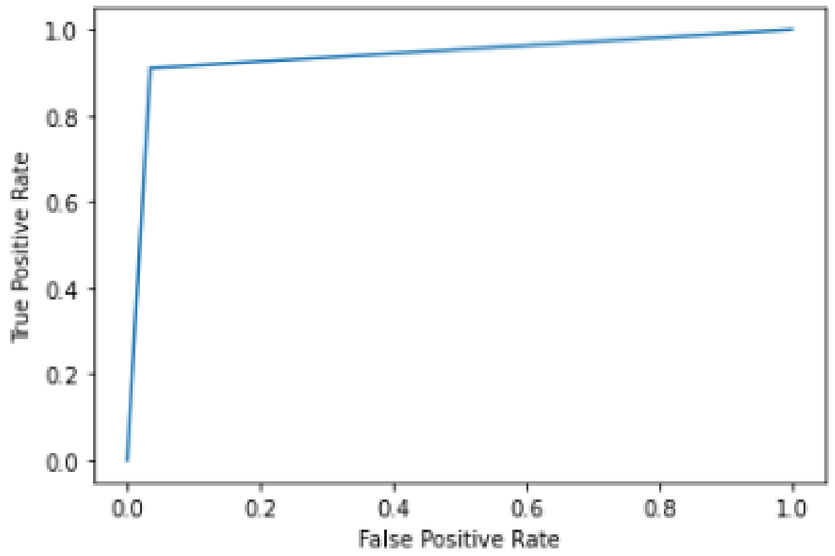
```
In-box prediction accuracy of test data:
0.9440559440559441
In-box confution matrix of test data:
```



ROC curve

As we can see from the plot, that model does better job of classifying the data into categories then ours.

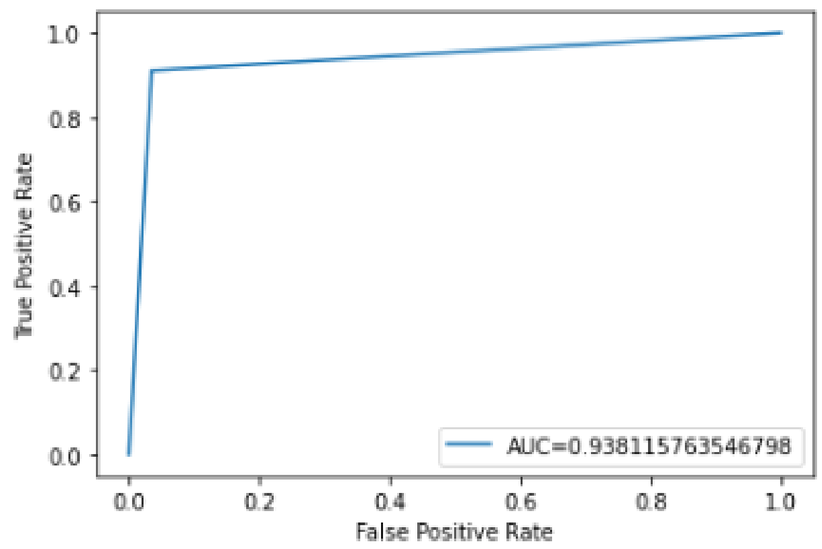
```
fpr, tpr, _ = metrics.roc_curve(y_test,  ypred_gnb)

#create ROC curve
plt.plot(fpr,tpr)
plt.ylabel('True Positive Rate')
plt.xlabel('False Positive Rate')
plt.show()
```



AUC

The AUC for this model turns out to be 0.938. Since this is close to 1, this confirms that the model does a great job of classifying data.

```
auc = metrics.roc_auc_score(y_test, ypred_gnb)

#create ROC curve
plt.plot(fpr,tpr,label="AUC="+str(auc))
plt.ylabel('True Positive Rate')
plt.xlabel('False Positive Rate')
plt.legend(loc=4)
plt.show()
```



Results

The classifier in-box showed a result of 0.944, while out-of-box showed only 0.874.

The ROC and AUC metrics showed the corresponding result.
Which means that our realization is worse than the in package.

As we can see out-of-box classifier results are worse, but still good enough (especcially for benign tumors case) to use.

Thanks for attention!