Breast Cancer Tumor Detection using Machine Learning Techniques

Donya Ashtiani Haghighi

University of Victoria

1. Objective

Breast Cancer is the most common cancer among all women around the world and according to the statistics, it is a significant public health problem in today's society. The number of cases is increasing particularly in developing countries where the majority of cases are diagnosed in late stages. Early detection results in not only less severe treatments but also a greater likelihood of survival. Precise classification in different types of tumors can prevent patients suffering from unnecessary treatments. So, the correct diagnosis of Breast Cancer and classification of patients into malignant or benign groups is an important topic for research. Machine learning has gained increasing popularity for discriminating benign and malignant tumors. In this project, we evaluated the efficiency of different classification algorithms on the Wisconsin Breast Cancer dataset and trying to improve its accuracy using different optimization techniques.

2. Introduction

Today, the use of Machine Learning approaches in the medical field may be considered to be a great assistance in medical decision making. These techniques have increasingly expanded for distinguishing malignant and benign tumors and facilitating early cancer tumors detection. In this project, 3 different algorithms Logistic Regression, Linear Regression and Support-Vector Machines for modeling the breast cancer dataset are applied. The results were compared based on several performance measurements including accuracy rate, recall and precision rate. After that, we applied some other techniques including removing noise and non-informative features by using PCA feature extraction and using some quadratic classification models for being able to separate the data into more accurate way.

3. Features of Data

The dataset was created by Dr. William H. Wolberg, physician at the University of Wisconsin Hospital at Madison, Wisconsin, USA. To create the dataset Dr. Wolberg used fluid samples, taken from patients with solid breast masses and an easy-to-use graphical computer program called Xcyt, which analyse cytological features based on a digital scan. The dataset is widely used for the classification analysis. It has 569 cases virtually noise free. The dataset consists of 212 malignant (M) cases and 357 benign (B) cases. The original data has some missing data with 699 cases. In this project, the formatted dataset with no missing data is used, unless otherwise stated.
The program uses a curve-fitting algorithm, to compute ten features from each one of the cells in the sample, then it calculates the mean value, extreme value and standard error of each feature for the image. Then, it returns a 30 real-valuated vector. The mean, standard error and "worst" or largest (mean of the three largest values) of these features were computed for each image, resulting in 30 features, "radius (mean of distances from center to points on the perimeter), texture (standard deviation of gray-scale values), perimeter, area, smoothness (local variation in radius lengths),

compactness (perimeter / area — 1.0), concavity (severity of concave portions of the contour), concave points (number of concave portions of the contour), symmetry, fractal dimension ("coastline approximation" — 1)". The range of values of raw data varies widely.

There are some features heavily skewed right as shown in Figure 2 and Table 1. It means those features have some bigger value outliers. For example, ranges of the Area Mean are from 143.56 to 2501. It is possible to transform the skewed features by taking the logarithm or any power-transformation or removing extreme outliers as many machine learning practitioners/statisticians do. We examined the results of removing 2 some features using the feature selection (shown in figure 2) and using PCA values (dimensionality reduction) for comparative analysis in the first phase of our data analysis.

Feature	Range	Feature	Range
Radius Mean	6.981-28.11	**Compactness Se**	0.002252-0.1354
Texture Mean	9.71-39.28	**Concavity Se**	0-0.396
Perimeter Mean	43.79-188.5	**Concave Point Se**	0-0.05279
Area Mean	143.5-2501	**Symmetry Se**	0.007882-0.07895
Smoothness Mean	0.05263-0.1634	**Fractal Dimension Se**	0.000895-0.02984
Compactness Mean	0.01938-0.3454	**Radius Worst**	185.2-4254
Concavity Mean	0-0.4268	**Texture Worst**	12.02-49.54
Concave Point Mean	0-0.2012	**Perimeter Worst**	50.41-251.2
Symmetry Mean	0.106-0.304	**Area Worst**	7.93-36.04
Fractal Dimension Mean	0.04996-0.09744	**Smoothness Worst**	0.07117-0.2226
Radius Se	0.1115-2.873	**Compactness Worst**	0.027929-1.058
Texture Se	0.3603-4.885	**Concavity Worst**	0-1.252
Perimeter Se	0.757-21.98	**Concave Point Worst**	0-0.291
Area Se	6.802-542.2	**Symmetry Worst**	0.1565-0.6638
Smoothness Se	0.001713-0.03113	**Fractal Dimension Worst**	0.05504-0.2075

Table1: Ranges of Numeric Feature Value

Correlation Analysis is useful to understand the relationship among variables. In this project, the Pearson Correlation Coefficient that measures the strength and direction of a linear relationship between two variables was used. There are several pairs of features highly correlated (Figure 3). For instance, consider radius-mean and radius-worst. Because they are both the measurement of radius from the perimeter, it is natural for two features to be correlated. However, the pairs may also be strongly correlated to the target. The feature importance analysis in the next section demonstrates that removing features does not always result in a better outcome. Training time increases exponentially with number of features. Models have increasing risks of overfitting as the dimensionality increases. However, it is also possible to lose important information by removing feature. It is necessary to examine tradeoffs between accuracy and memory/computation saving.

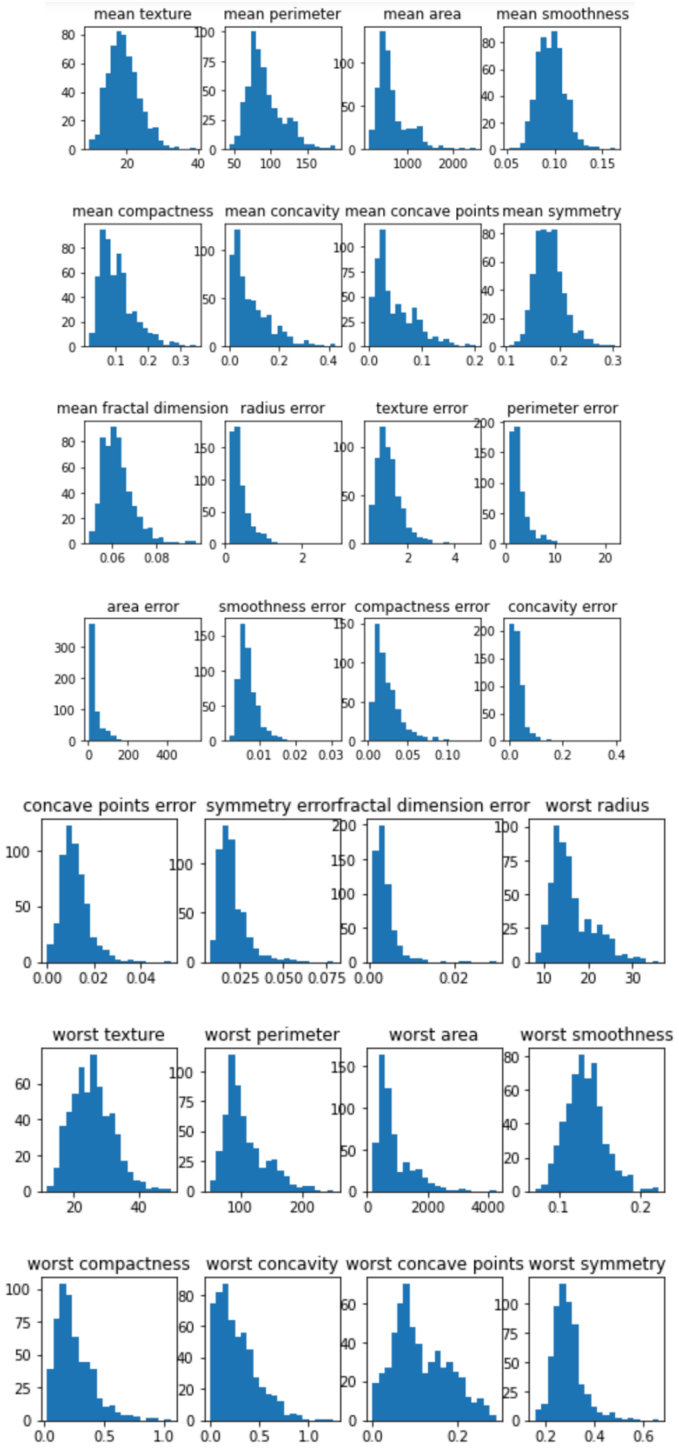

Figure 1: Feature Histogram

Figure 2: Mutual information of each feature and class

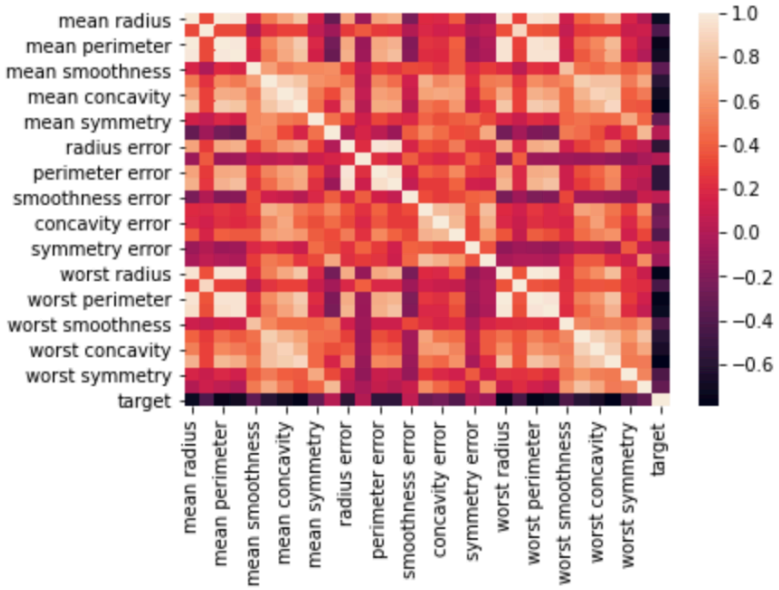

Figure 3: Correlation Matrix

A great tool to visualize the data distribution is Principal Component Analysis (PCA) which is a technique to maximize variation and disclose strong pattern in a dataset and it is often used for data visualization.

4. Methodology

The goal is to classify the breast cancer benign or malignant tumors. To achieve this, we used machine learning classification methods to fit a function that can predict the discrete class of new input.

We tried to use Scikit learn package to implement machine learning algorithms and matplotlib and seaborn libraries to visualize data. The experimental framework consists of two phases. In the first phase of the experiment four different scenarios were experimented; running on original data without normalization, running on normalized data, running on selected features using mutual information between each feature and class (shown in Figure 2) and running on two extracted features using PCA. Figure 4 shows the transformation of data into the new coordinator system.

These scenarios will enable us to understand which algorithms require data normalization. In the second phase of our experiment, we selected some algorithms, Logistic Regression, and Support Vector Machine for detection of tumors.

Feature Selection and PCA are used for the first phase of the data analysis. In the second phase Grid Search was employed for hyper-parameter optimization.

4.1.1 Feature Selection

Feature selection is a widely used technique which reduce or shrink the number of features to avoid overfitting by increasing the generality of data and also to enhance the performance of algorithm. There are several approaches for choosing the most important features among all possible features. mutual information is one of the most well-known approaches. In this project, mutual information measures the dependencies between a feature and the class and is defined by Equation 1. Mutual information value is between 0 to 1, higher level of mutual information means that a given feature is more informative. Figure 2 shows the mutual information values of each feature. In this scenario we chose features in which their mutual information is greater than 0.3.

$$I(F;C) = \sum_{f_i \in F} \sum_{c_i \in C} P(F = f_i, C = c_i) log_2 \frac{P(F = f_i, C = c_i)}{P(F = f_i)\, P(C = c_i)} \qquad (1)$$

4.1.2. Principal Component Analysis (PCA):

Principal Component analysis is an unsupervised method which aims to reduce the dimensions of a data by finding a new set of variables and maintains most of the sample's information. It is defined by linear transformation of data to new uncorrelated vectors which are ordered by the fraction of information they keep. Principal components are achieved from covariance matrix.

$$S = \frac{1}{n}\sum_{i=1}^{n}(x_i - \bar{x_i})(x_i - \bar{x_i})' = \frac{1}{n}XX' \qquad (2)$$

The first principal component is the first eigenvector of the covariance matrix, $PC_1 = \lambda_1$ and the *i*th PC corresponds to the *i*th eigenvector $PC_i = \lambda_i S$.

4.2. Machine Learning Algorithms

Three machine learning algorithms are used in this project.

4.2.1. Logistic Regression (with Regularization)

Logistic Regression is used to predict a binary class given a set of independent feature variables. It generates the probability of occurrence of an event by fitting training data to a sigmoid function. The goal is to maximize

$$\max ln\, L(\omega) = \sum_{k=1}^{n} ln\left(\frac{1}{1 + e^{-y^k \omega' x^k}}\right)$$

or to minimize

$$\min E(\omega) = \frac{1}{2n}\sum_{k=1}^{n} ln\left(1 + e^{-y^k \omega' x^k}\right)$$

By adding the regularization term

$$\min E(\omega) = \frac{1}{2n}\sum_{k=1}^{n}(y - h_\omega(x_k))^2 + \lambda\sum_{i=1}^{m}\omega_i^2$$

4.2.2. Support Vector Machines

SVM is a discriminative classifier defined by a separating hyperplane. The SVM algorithm finds an optimal hyperplane which separates classes. The distance of a point to a line (margin) is

$$\frac{[\omega x_k + b]}{\|\omega\|} = \frac{1}{\|\omega\|}$$

The goal is to maximize the margin, which is the same as minimizing

$$\min \frac{1}{2}\omega.\omega'$$

Adding the soft margin term accommodate points between two boundary lines,

$$\min_{\omega,b,\xi} \frac{1}{2}\omega.\omega' + C\sum_{k=1}^{n}\xi_k$$

Subject to

$$y^k(\omega_k + b) \geq 1 - \xi_k, \qquad k \in [1,n], \quad \xi_k \geq 0$$

4.2.3. Linear Regression

Linear Regression is another tool for making the decision boundary between samples. It works quite similar to logistic regression. Its unique difference is that this classifier does not apply the Sigmoid function to make values bounded in (0,1) interval. It linearly products the value of each feature to its weight and reports the final predicted value. Finally, we need to apply sign function to make the predicted values into {0,1} labels.

4.3. Performance Evaluation

The metrics to evaluate the performance of the algorithms are classification accuracy, recall, precision, harmonic and confusion matrix. Recall is equivalent to the true positive rate, and precision is equivalent to positive predictive value. F1 score is the harmonic mean of recall and precision. A confusion matrix is a table that describes the performance of a classification model. It contains four numbers, TP, FP, TN, and FN.

5. Data Analysis and Results

The dataset was shuffled and partitioned by 80% for training and 20% for testing in every run of each algorithm. The results described in the following sections are one of example runs. Their metrics are stable with no significant divergence for each run.

5.1. Normalization vs No Normalization

In the first phase of analysis, all the methods have been run without normalization.

Algorithm	Accuracy No Normalization	Accuracy with Normalization
Logistic Regression	Failed	0.94
Support Vector Machine	0.58	0.92
Linear Regression	0.9	0.94

Table 2: Accuracy without normalization and with normalization

As shown in Table 2, Linear Regression classifier works quite well against non-normalized data and other classifiers like Logistic Regression and SVM are heavily dependent to normalized data. On the other hand, normalization process would make these classifiers improve their performances. Furthermore, we evaluated the results of Feature Selection and PCA adapted model.

Algorithm	Accuracy with FS	Accuracy with PCA
Logistic Regression	0.947	0.868
Support Vector Machine	0.91	0.921
Linear Regression	0.89	0.89

Table 3: Accuracy using Feature Selection and PCA

5.2. Cross-Validation with Grid Search

In the first phase of this study, normalization is deliberately opted out to examine how the models behave without normalization. In the second phase of the experiment, cross validation and grid search were added to the selected algorithms that indicated low accuracy in the first phase. For those, ten-fold cross-validation and Sklearn GridSearchCV were applied to the training set for the possible metrics improvement by identifying the best hyper-parameters. We fitted using the best hyperparameters selected to search the optimal parameters, then run on the test set. Table 4 describes the hyper-parameter search setting. As Table 5 indicates, the cross-validation and grid search helped improve performance slightly. The solver is fixed with 'solver' = lbfgs. lbfgs is Limited Memory BFGS (BroydenFletcherGoldfarbShanno).

Algorithm	Grid Search
LG	‘C’: [0.001,0.01,0.1,1,10,100,1000] ‘penalty’: [‘11’,’12’]
SVM	‘gamma’: [0.01,0.03,0.1,0.3,1,3,10,30,100] ‘C’: [0.01,0.03,0.1,0.3,1,3,10,30,100]

Table 4: GridSearchCV Setting

Table 6 shows the metrics of the three algorithms. Recall rate measures how many relevant items are selected while Precision rate measures how many selected items are relevant. F1 harmonic rate is the average of two rates. On the breast cancer dataset, all those rates including the accuracy rate are exceptionally good.

Algorithm	Best Hyper-Parameter
LG	‘C’:10, ‘penalty’: 11
SVM	‘C’:10, ‘gamma’: 0.3

Table 5: Best HyperParameter

Note that the metrics are slightly different for each run because the data was shuffled for each run.

Algorithm	Accuracy	Recall	Precision	F1(Harmonic)
LG	0.967	0.889	0.969	0.928
SVM	0.962	0.917	0.971	0.943

Table 6: Metrics after normalization, cross-validation and grids search

6. Quadratic Discriminant Analysis

We have so far analyzed different linear classifier such as SVM and Logistic Regression using costly Grid Search techniques to find out their best parameters and also using linear regression to get the accuracy as much as we can. But what if we assume a quadratic decision boundary. [1] introduced a quadratic decision boundary which uses Fisher method to find the quadratic decision boundary. Here is the result of using this method. The result shows that this method is able to get the 97 percent accuracy while other methods like SVM and LG require the grid search to find their own best parameters. In other words, we can get the result 97 percent of QDA (Quadratic Decision Boundary) without considering any prior information.

7. Conclusion

This project presented an application of different machine learning algorithms on the breast cancer dataset including several different evaluation metrics. It seems that some algorithms are vulnerable against non-normalized data. Also, using mutual information to select the top informative features or using other techniques like feature extraction such as PCA does not play much for improving the accuracy of result. Finally, we have shown that using quadratic decision boundary is able to improve the accuracy a little bit.

8. References

[1] Sebastian Mika, Gunnar Ratsch, Jason Weston, Bernhard Scholkopf, and Klaus-Robert Mullers. Fisher discriminant analysis with kernels. In Neural networks for signal processing IX: Proceedings of the 1999 IEEE signal processing society workshop (cat. no. 98th8468), pages 41{48. Ieee, 1999.

9. Python Code

```
# plot feature Histogram and correlation matrix
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.datasets import load_breast_cancer
# import seaborn as sns

cancer = load_breast_cancer()
```

```
df = pd.DataFrame(np.c_[cancer['data'], cancer['target']], columns = np.append(cancer['feature_names'], ['target']))
df.head()
cols = df.columns.values
data = df.values
gridsize = (7, 4)
fig = plt.figure(figsize=(8, 20))
fig.subplots_adjust(hspace=0.6)
ax1 = plt.subplot2grid(gridsize, (0, 0))
ax2 = plt.subplot2grid(gridsize, (0, 1))
ax3 = plt.subplot2grid(gridsize, (0, 2))
ax4 = plt.subplot2grid(gridsize, (0, 3))
ax5 = plt.subplot2grid(gridsize, (1, 0))
ax6 = plt.subplot2grid(gridsize, (1, 1))
ax7 = plt.subplot2grid(gridsize, (1, 2))
ax8 = plt.subplot2grid(gridsize, (1, 3))
ax9 = plt.subplot2grid(gridsize, (2, 0))
ax10 = plt.subplot2grid(gridsize, (2, 1))
ax11 = plt.subplot2grid(gridsize, (2, 2))
ax12 = plt.subplot2grid(gridsize, (2, 3))
ax13 = plt.subplot2grid(gridsize, (3, 0))
ax14 = plt.subplot2grid(gridsize, (3, 1))
ax15 = plt.subplot2grid(gridsize, (3, 2))
ax16 = plt.subplot2grid(gridsize, (3, 3))
ax17 = plt.subplot2grid(gridsize, (4, 0))
ax18 = plt.subplot2grid(gridsize, (4, 1))
ax19 = plt.subplot2grid(gridsize, (4, 2))
ax20 = plt.subplot2grid(gridsize, (4, 3))
ax21 = plt.subplot2grid(gridsize, (5, 0))
ax22 = plt.subplot2grid(gridsize, (5, 1))
ax23 = plt.subplot2grid(gridsize, (5, 2))
ax24 = plt.subplot2grid(gridsize, (5, 3))
ax25 = plt.subplot2grid(gridsize, (6, 0))
ax26 = plt.subplot2grid(gridsize, (6, 1))
ax27 = plt.subplot2grid(gridsize, (6, 2))
ax28 = plt.subplot2grid(gridsize, (6, 3))
#ax29 = plt.subplot2grid(gridsize, (7, 0))
#ax30 = plt.subplot2grid(gridsize, (7, 1))
#ax31 = plt.subplot2grid(gridsize, (7, 2))
#ax32 = plt.subplot2grid(gridsize, (7, 3))


ax1.set_title(cols[1])
ax1.hist(sorted(data[:,1]), bins=20)
ax2.set_title(cols[2])
ax2.hist(sorted(data[:,2]), bins=20)
```

```
ax3.set_title(cols[3])
ax3.hist(sorted(data[:,3]), bins=20)
ax4.set_title(cols[4])
ax4.hist(sorted(data[:,4]), bins=20)
ax5.set_title(cols[5])
ax5.hist(sorted(data[:,5]), bins=20)
ax6.set_title(cols[6])
ax6.hist(sorted(data[:,6]), bins=20)
ax7.set_title(cols[7])
ax7.hist(sorted(data[:,7]), bins=20)
ax8.set_title(cols[8])
ax8.hist(sorted(data[:,8]), bins=20)
ax9.set_title(cols[9])
ax9.hist(sorted(data[:,9]), bins=20)
ax10.set_title(cols[10])
ax10.hist(sorted(data[:,10]), bins=20)
ax11.set_title(cols[11])
ax11.hist(sorted(data[:,11]), bins=20)
ax12.set_title(cols[12])
ax12.hist(sorted(data[:,12]), bins=20)
ax13.set_title(cols[13])
ax13.hist(sorted(data[:,13]), bins=20)
ax14.set_title(cols[14])
ax14.hist(sorted(data[:,14]), bins=20)
ax15.set_title(cols[15])
ax15.hist(sorted(data[:,15]), bins=20)
ax16.set_title(cols[16])
ax16.hist(sorted(data[:,16]), bins=20)
ax17.set_title(cols[17])
ax17.hist(sorted(data[:,17]), bins=20)
ax18.set_title(cols[18])
ax18.hist(sorted(data[:,18]), bins=20)
ax19.set_title(cols[19])
ax19.hist(sorted(data[:,19]), bins=20)
ax20.set_title(cols[20])
ax20.hist(sorted(data[:,20]), bins=20)
ax21.set_title(cols[21])
ax21.hist(sorted(data[:,21]), bins=20)
ax22.set_title(cols[22])
ax22.hist(sorted(data[:,22]), bins=20)
ax23.set_title(cols[23])
ax23.hist(sorted(data[:,23]), bins=20)
ax24.set_title(cols[24])
ax24.hist(sorted(data[:,24]), bins=20)
ax25.set_title(cols[25])
ax25.hist(sorted(data[:,25]), bins=20)
```

```
ax26.set_title(cols[26])
ax26.hist(sorted(data[:,26]), bins=20)
ax27.set_title(cols[27])
ax27.hist(sorted(data[:,27]), bins=20)
ax28.set_title(cols[28])
ax28.hist(sorted(data[:,28]), bins=20)
#ax29.set_title(cols[29])
#ax29.hist(sorted(data[:,29]), bins=20)
#ax30 .set_title(cols[30])
#ax30.hist(sorted(data[:,30]), bins=20)
#ax31.set_title(cols[31])
#ax31.hist(sorted(data[:,31]), bins=20)
#ax32.set_title(cols[31])
#ax32.hist(sorted(data[:,31]), bins=20)
plt.show()

import seaborn as sn
corrMatrix = df.corr();
sn.heatmap(corrMatrix)
plt.show()

import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import pylab
from random import shuffle
from sklearn.decomposition import PCA
from mpl_toolkits.mplot3d import Axes3D
from sklearn.model_selection import train_test_split
from sklearn.model_selection import KFold
from sklearn.naive_bayes import GaussianNB
from sklearn.metrics import accuracy_score
from sklearn import svm
from sklearn.model_selection import GridSearchCV
from sklearn.ensemble  import RandomForestClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.neural_network import MLPClassifier
from sklearn.model_selection import cross_val_score
from sklearn.model_selection import cross_val_predict
from sklearn import metrics
from sklearn.metrics import roc_curve, auc
from sklearn.tree import export_graphviz
import pydot
import os
import csv
```

```
os.environ["PATH"] += os.pathsep + 'C:/Program Files (x86)/Graphviz2.38/bin/'
from sklearn.metrics import precision_recall_curve
import warnings

# from mlxtend.preprocessing import shuffle_arrays_unison

def loadfile(file):
        df = pd.read_csv(file)
        cols = df.columns.values
        data = df.values
        return cols, data

        # plot histogram - to save code, each graph contains 8 subplot
        # and repeated 4 times to generate all feature graphs
def plotFeatures1(cols, data):
        gridsize = (6, 2)
        fig = plt.figure(figsize=(8, 20))
        fig.subplots_adjust(hspace=0.6)
        ax1 = plt.subplot2grid(gridsize, (1, 0))
        ax2 = plt.subplot2grid(gridsize, (1, 1))
        ax3 = plt.subplot2grid(gridsize, (2, 0))
        ax4 = plt.subplot2grid(gridsize, (2, 1))
        ax5 = plt.subplot2grid(gridsize, (3, 0))
        ax6 = plt.subplot2grid(gridsize, (3, 1))
        ax7 = plt.subplot2grid(gridsize, (4, 0))
        ax8 = plt.subplot2grid(gridsize, (4, 1))

        ax1.set_title(cols[1])
        ax1.hist(sorted(data[:, 1]), bins=20)
        ax2.set_title(cols[2])
        ax2.hist(sorted(data[:, 2]), bins=20)
        ax3.set_title(cols[3])
        ax3.hist(sorted(data[:, 3]), bins=20)
        ax4.set_title(cols[4])
        ax4.hist(sorted(data[:, 4]), bins=20)
        ax5.set_title(cols[5])
        ax5.hist(sorted(data[:, 5]), bins=20)
        ax6.set_title(cols[6])
        ax6.hist(sorted(data[:, 6]), bins=20)
        ax7.set_title(cols[7])
        ax7.hist(sorted(data[:, 7]), bins=20)
        ax8.set_title(cols[8])
        ax8.hist(sorted(data[:, 8]), bins=20)
        pylab.savefig('Chart1.png', bbox_inches='tight')

        plt.clf()
```

```
        # plot some boxcharts to see the variance, but they were not used for the report
def plotFeatures2(cols, data):
        df = pd.DataFrame(data, columns=cols)
        df_num = df.convert_objects(convert_numeric=True)
        plt.figure(figsize=(8, 15))
        f, axes = plt.subplots(2, 1)
        ax1 = sns.boxplot( data=df_num[['radius_mean', 'texture_mean', 'perimeter_mean',
'texture_worst', 'perimeter_worst']], orient='v', ax=axes[0])
        ax2 = sns.boxplot( data=df_num[['smoothness_mean', 'compactness_mean',
'fractal_dimension_mean', 'smoothness_se', 'smoothness_worst']], orient='v', ax=axes[1])
        ax1.set_xticklabels(ax1.get_xticklabels(), rotation=40, ha="right")
        ax2.set_xticklabels(ax2.get_xticklabels(), rotation=40, ha="right")

        plt.tight_layout()
        pylab.savefig('Boxchart.png', bbox_inches='tight')

        # calculate pca
def pcaFeatures(data):
        pca = PCA(n_components=3)
        pcaData = pca.fit_transform(data)
        print('Shape of PCA data: ', pcaData.shape)
        return pcaData

        # plot PCA 3D chart
def plotPCA(labels, data):
        fig = plt.figure()
        ax = fig.add_subplot(111, projection='3d')
        xs = data[:, 0]
        ys = data[:, 1]
        zs = data[:, 2]

        ax.scatter(xs, ys, zs, c=labels)
        ax.set_title('PCA Class Distribution')
        ax.set_xlabel('PCA 1')
        ax.set_ylabel('PCA 2')
        ax.set_zlabel('PCA 3')
        pylab.savefig('PCA_Class_Distribution.png', bbox_inches='tight')

        # plt.show()

        # plot correlation heatmap
def plotCorr(cols, data):
        df = pd.DataFrame(data, columns=cols)
        dff = df.drop(['id', 'diagnosis'], axis=1)
        dff =dff.convert_objects(convert_numeric=True)
```

```
	corr = dff.corr(method='pearson')
	# hide the upper triangle
	mask = np.zeros_like(corr, dtype=np.bool)
	mask[np.triu_indices_from(mask)] = True
	f, ax = plt.subplots(figsize=(11, 9))

	# Generate a custom diverging colormap
	cmap = sns.diverging_palette(220, 10, as_cmap=True)

	# Draw the heatmap with the mask and correct aspect ratio
	sns.heatmap(corr, mask=mask, cmap=cmap, center=0,
		square=True, linewidths=.5, cbar_kws={"shrink": .5})
	plt.savefig('Correlation_Heatmap.png', bbox_inche='tight')
	# plt.show()

	# process data
def prepData(data):
	m, n = data.shape

	# normalize data
	id_class = data[:, 0:2]
	features = data[:, 2:]
	dataNorm = (features - np.mean(features, axis=0))/(np.max(features,axis=0)-
np.min(features, axis=0))
	print('Shape of normalized feature data: ', dataNorm.shape)
	# print(dataNorm[0])
	# find pca
	pcaData = pcaFeatures(dataNorm)
	# assign 1 for Malignant and 0 for Beneighn
	labTemp = []
	for i in id_class[:, 1]:
		if i=='M':
			labTemp.append(1)
		else:
			labTemp.append(-1)
	# plot pca fetures in 3D
	plotPCA(labTemp, pcaData)

	# shuffle data
	tempID = id_class[:, 0].reshape(m, 1)
	targets = np.array(labTemp).reshape(m, 1)
	idTarget = np.append(tempID, targets, axis=1)

	ind_list = [i for i in range(m)]
	shuffle(ind_list)
	pcaShuffled = pcaData[ind_list]
```

```
	idTargetShuffled = idTarget[ind_list]

	return idTargetShuffled, pcaShuffled

	# train, test and evaluate
def train(id_class, pcaData, fold, algorithm, features):
	X = pcaData.astype(float)
	y = id_class[:, 1].astype(float)
	X_train, X_test, y_train, y_test = train_test_split(pcaData, y, test_size=0.20, random_state=42)

	# conduct cross validation becaues of the small sample size
	kf = KFold(n_splits=fold, random_state=None, shuffle=False)
	# initialize models
	RFC_clf = RandomForestClassifier(n_estimators=10)
	LG_clf = LogisticRegression()
	SVC_clf = svm.SVC(probability=True)
	MLPC_clf =  MLPClassifier()

	# RFC
	if algorithm == 'RFC':
		warnings.filterwarnings("ignore")
		model = RFC_clf.fit(X_train, y_train)
		# without cross-valdidation
		predict_RFC = model.predict(X_test)
		acc = accuracy_score(predict_RFC, y_test)
		y_prob = model.predict_proba(X_test)[:, 1]
		evalRFC, roc_values_RFC = evaluate(y_test, predict_RFC, y_prob, algorithm)
		# importances = feat_importance(model, features[2:])
		# featuresSelected = ['diagnosis', 'radius_mean', 'texture__mean']
		# createTree(model, featuresSelected)
		eval_rates = evalRate(y_test, predict_RFC)
		print('Confusion Matrix - '+algorithm, evalRFC)
		print('RFC - Recall, Precision, F1: ', eval_rates)

		# with cross-validation
		acc_cv = cross_val_score(RFC_clf, X, y, scoring='accuracy', cv=kf).mean()
		accuracies = [acc, acc_cv]
		return accuracies

	# LG
	if algorithm == "LG":
		model = LG_clf.fit(X_train, y_train)
		predict_LG = model.predict(X_test)
		acc_LG = accuracy_score(predict_LG, y_test)
		y_prob_LG = model.predict_proba(X_test)[:, 1]
```

```
                # with cross-validation and grid search of hyperparameters
                tuned_parameters = {'C': [0.001, 0.01, 0.1, 1, 10, 100, 1000], 'penalty': ['l1', 'l2']}
                LG= GridSearchCV(LG_clf, tuned_parameters, cv=fold)
                model_LG = LG.fit(X_train, y_train)
                print('Logistic Regression Best Parameters: ',  LG.best_params_)
                y_prob_cv_LG = model_LG.predict_proba(X_test)[:, 1]
                y_pred_cv_LG= np.where(y_prob_cv_LG > 0.4, 1, -1)
                # evaluations
                acc_cv_LG = model_LG.best_score_
                # generate confusion matrix and ROC curve
                evalLG,  roc_values_LG  =  evaluate(y_test,  y_pred_cv_LG,  y_prob_cv_LG,
algorithm)
                print('FPR, TPR & Thresholds - '+algorithm, roc_values_LG)
                eval_rates_LG = evalRate(y_test, y_pred_cv_LG)
                analyze(y_test, y_prob_cv_LG, algorithm)
                print('Confusion Matrix - '+algorithm, evalLG)
                print('LG - Recall, Precision, F1: ', eval_rates_LG)
                accuracies = [acc_LG, acc_cv_LG]
                return accuracies

        # SVM
        if algorithm =='SVM':
                model = SVC_clf.fit(X_train, y_train)
                predict_svm = model.predict(X_test)
                acc_SVM = accuracy_score(predict_svm, y_test)

                # with cross-validation and grid search of hyperparameters
                tuned_parameters = {'gamma':[0.01, 0.03, 0.1, 0.3, 1, 3, 10, 30, 100], 'C': [0.01,
0.03, 0.1, 0.3, 1, 3, 10, 30, 100]}
                SVC = GridSearchCV(SVC_clf, tuned_parameters, cv=fold)
                model_SVM = SVC.fit(X_train, y_train)
                print('SVM Best Parameters: ',  SVC.best_params_)
                y_prob_cv_SVM = model_SVM.predict_proba(X_test)[:, 1]
                y_pred_cv_SVM= np.where(y_prob_cv_SVM > 0.4, 1, -1)
                # evaluation
                acc_cv_SVM = model_SVM.best_score_
                # generate confusion matrix and ROC curve
                evalSVM,  roc_values_SVM  =  evaluate(y_test,  y_pred_cv_SVM,
y_prob_cv_SVM, algorithm)
                print('FPR, TPR & Thresholds - '+algorithm, roc_values_SVM)
                eval_rates_SVM = evalRate(y_test, y_pred_cv_SVM)
                analyze(y_test, y_prob_cv_SVM, algorithm)
                print('Best parameters for SVM: ', model_SVM.best_params_)
                print('SVM - Recall, Precision, F1: ', eval_rates_SVM)
                print('Confusion Matrix - '+algorithm, evalSVM)
```

```
                accuracies = [acc_SVM, acc_cv_SVM]
                return accuracies

        # Multi-layer perceptron classifier
        if algorithm == "MLPC":
                model = MLPC_clf.fit(X_train, y_train)
                warnings.filterwarnings("ignore")
                predict_MLPC = model.predict(X_test)
                acc_MLPC = accuracy_score(predict_MLPC, y_test)
                y_prob_MLPC = model.predict_proba(X_test)[:, 1]

                # with cross-validation and grid search of hyperparameters
                parameters = {'solver': ['lbfgs'], 'max_iter': [100, 500, 1000], 'alpha': 10.0 ** -
np.arange(1, 3), 'hidden_layer_sizes': np.arange(5, 12), 'random_state': [0, 1, 2, 3, 4, 5, 6, 7, 8, 9]}
                MLPC_clf_grid = GridSearchCV(MLPClassifier(), parameters, n_jobs=-1,
cv=fold)
                warnings.filterwarnings("ignore", category=DeprecationWarning)
                model_MLPC = MLPC_clf_grid.fit(X_train, y_train)
                print("MPLC Best Parameters: ", MLPC_clf_grid.best_params_)
                acc_cv_MLPC = model_MLPC.best_score_
                y_prob_cv_MLPC = model_MLPC.predict_proba(X_test)[:, 1]
                y_pred_cv_MLPC= np.where(y_prob_cv_MLPC > 0.4, 1, -1)
                evalMLPC, roc_values_MLPC =evaluate(y_test, y_pred_cv_MLPC,
y_prob_cv_MLPC, algorithm)
                print('FPR, TPR & Thresholds - '+algorithm, roc_values_MLPC)
                eval_rates_MLPC= evalRate(y_test, y_pred_cv_MLPC)
                analyze(y_test, y_prob_cv_MLPC, algorithm)
                print('MLPC - Recall, Precision, F1: ', eval_rates_MLPC)
                # print('Confusion Matrix - '+algorithm, evalMLPC)
                accuracies = [acc_MLPC, acc_cv_MLPC]
                return accuracies

        # generate confusion matrix and plot ROC curve
def evaluate(y_test, y_pred, y_prob, algorithm):
        # calculate confusion matrix for each algorthm
        confusion_matrix = metrics.confusion_matrix(y_test, y_pred)
        # print('Confusion Matrix - '+algorithm, confusion_matrix)

        # generate ROC curve for each algorithm
        auc_roc = metrics.roc_auc_score(y_test, y_pred)
        false_positive_rate, true_positive_rate, thresholds = roc_curve(y_test, y_prob)
        roc_values = [false_positive_rate, true_positive_rate, thresholds]
        # generate ROC AUC
        roc_auc = auc(false_positive_rate, true_positive_rate)
        plt.figure(figsize=(10, 10))
        plt.title('Receiver Operating Characteristic'+' - '+algorithm)
```

```
        plt.plot(false_positive_rate, true_positive_rate, color='red', label='AUC = %0.2f' %
roc_auc)
        plt.legend(loc='lower right')
        plt.plot([0, 1], [0, 1], linestyle='--')
        plt.axis('tight')
        plt.ylabel('True Positive Rate')
        plt.xlabel('False Positive Rate')
        pylab.savefig(algorithm+' - ROC_Curve.png', bbox_inches='tight')
        # plt.show()
        return confusion_matrix, roc_values

        # calculate recall and precision rate
def evalRate(y_true, y_predict):
        precision = metrics.precision_score(y_true,  y_predict)
        recall = metrics.recall_score(y_true, y_predict)
        f1_score = 2*precision*recall/(precision+recall)
        return [recall, precision, f1_score]

        # export roc value to csv file to check the best threshold against sklearn threshold
def analyze(y_test, y_prob, algorithm):
        y_test = pd.DataFrame(y_test)
        y_prob = pd.DataFrame(y_prob)
        roc_values = pd.concat([y_test, y_prob], axis=1, sort=False)
        roc_values.columns = ['y_true', 'y_prob']
        roc_values.to_csv('roc_values - '+algorithm+'.csv')


if __name__ == '__main__':
        # load data
        warnings.filterwarnings("ignore", category=DeprecationWarning)
        filename = 'Breast_Cancer_data.csv'
        cols, data = loadfile(filename)
        print('Shape of data: ', data.shape)
        print(cols)

        # plot each columns
        plotFeatures2(cols, data)
        plotFeatures1(cols, data)
        plotCorr(cols, data)


        # normalize and PCA transform data
        id_class, pcaData = prepData(data)

        # train, test and evaluate data
        models =['RFC', 'LG', 'SVM', 'MLPC']
```

```
        # results are stored in the 'results' dictionary
        results = {'RFC': {'Accuracy': None, 'Accuracy_cv': None}, 'LG': {'Accuracy': None, 'Accuracy_cv': None}, 'SVM': {'Accuracy': None, 'Accuracy_cv': None}, 'MLPC': {'Accuracy': None, 'Accuracy_cv': None}}
        for model in models:
                acc = train(id_class, pcaData, 10, model, cols)  # cv is set to 10
                results[model]['Accuracy'] = acc[0]
                results[model]['Accuracy_cv'] = acc[1]
        print(results)
```