Model Selection on Breast Cancer Recurrence

line 1: Rocky Saxena
line 2: *CSC 7825*
line 3: *Wayne State University*

Abstract**—** The aim of this project is to classify if patients breast cancer had recurred based on using a breadth of probabilistic classifiers. Additionally, this project showed that relying on accuracy alone may not be the best method, therefore generating receiver operating characteristic (roc auc) and area under the curve graph is crucial. The roc auc provided optimal thresholds to decrease the FNR for each classifier, the results for each algorithm were Naïve Bayes had FNR of 27% and FPR of 62%, logisitic regression had FNR of 27% and FPR of 31%, support vector machines FNR was 27% and 51% FPR, MLP provided FNR of 18% and a FPR of 20%, and finally k nearest neighbors showed a FNR of 27% and FPR of 20%. Based on the data MLP was the best model, followed by KNN, logisitic regression, support vector machine, and finally Naïve Bayes.

I. Background Introduction

Breast cancer is a disease that affects many women around the world, and it is the second most common cancer that affects women [1]. Therefore, roughly 1 in 8 women will be diagnosed with breast cancer and 42,170 women in the U.S. alone will die from breast cancer [2]. However, the death rate has steadily declined from 2013 through 2017 by 1.3%. This decline in death rate is due to early detection of the disease so proper treatments can be administered before the cancer metastasizes. The importance of early detection of breast cancer influenced the idea of this project which is to identify those patients who were diagnosed with breast cancer, treated, and if their cancer had returned. The objective is to try a breadth of machine learning models that include Naïve Bayes, logisitic regression, support vector machines, multi-layer perceptron, and k nearest neighbors on a supervised dataset obtained from UCI machine learning repository. Furthermore, the models will be tuned based on threshold receiver operating characteristics area under the curve scores (roc auc) and ranked in terms of effectiveness in classifying recurrence of cancer on this dataset.

II. Methods

A. Data Details

The dataset was obtained from UCI Machine learning repository and was originally gathered by Dr. William H. Wolberg from University of Wisconsin [3]. The dataset is composed of 198 samples (patients), 34 features, and binary target variables (1 cancer recurred, and 0 cancer did not return). The features are all continuous and include ID number, time cancer recurrence or time of being healthy since seen by physician, tumor size, lymph node status, and 13 more features that were imaged by fine needle aspirate (FNA). FNA provided feature information of the cell that include the radius, texture, perimeter, area, smoothness, compactness, concavity, concave points, symmetry, and fractal dimensions. The dataset was imbalanced with the majority class 0 (healthy patients) obtaining 151 patients and class 1 (cancer recurred) at 47. Finally, there were 4 missing values in the lymph node status. The data was further visualized based on two features, time (time of being healthy or time of having cancer since last visit) and tumor size. **Figure 1** shows the visualization of these two features and how it related to the class label (outcome).

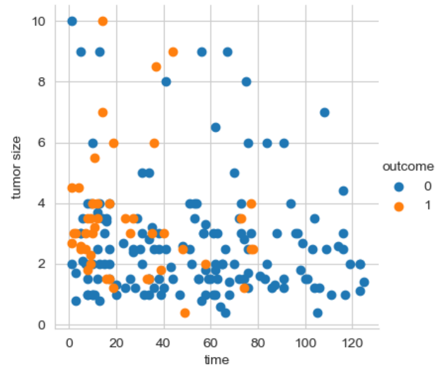

Figure 1: Data visualization based on tumor size and time.

B. Data Pre-Processing

The dataset was pre-processed in the following way, ID features were removed, values in the dataset stored 'R' as recurred and 'N' as no cancer were replaced with 1 and 0. Missing values were counted and replaced with the mean of the column and each feature was standardized to have a mean of 0 and variance of 1. Additionally, cross validation was used with a split of 138 samples of training, 20 samples of validation, and 40 sample size for testing. Next dimensionality reductions were utilized to project the data to explain 95% of the variance via principal component analysis (PCA) [4]. The last step was to perform Synthetic minority oversampling technique (SMOTE) which oversamples the minority class. This process is done by using linear interpolation and applying k nearest neighbors to find optimal new cases of a minority sample [5].

C. Gaussian Naïve Bayes

Naive Bayes functions by assuming that each feature is independent of one another, this assumption means that there is no correlation between each feature. This is a strong assumption and in reality it is generally not true, however, in practice it performs wel. **Figure 2** below compares Naïve Bayes while applying Synthetic Minority Oversampling Technique (SMOTE) against no SMOTE applied, this is

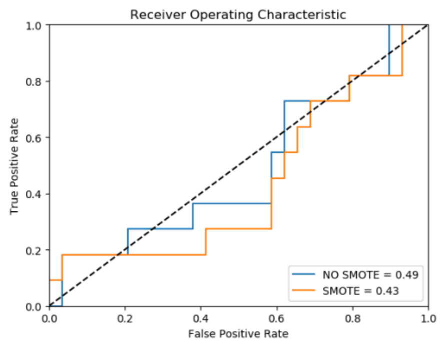

Figure 2: SMOTE performance.

The figure shows that when not applying SMOTE the model performed better based on the area under the curve score (auc). Therefore, picking a threshold that will minimize the FNR rate and trying to keep the FPR rate low was chosen and provided in the **table 1** below.

	SMOTE	NO SMOTE
TN	12	11
FP	17	18
FN	7	3
TP	4	8
FNR	63.40%	27.20%
FPR	58.60%	62.00%
Training Accuracy	70%	60%
Testing Accuracy	42.50%	47.50%

Table 1: Performance of 2 Naïve Models Based on sampling.

The objective in classification targeted to diagnosing diseases puts a greater importance on increasing the true positive rate (Sensitivity or TPR) and decreasing the false negative rates (FNR). It would be catastrophic if a patient had breast cancer and was sent home as being healthy because it increases the likelihood of death. On the other hand, if the false positive rate (FPR) is higher due to trade off, it would not be as harsh. Therefore, obtaining low FNR is a better method than accuracy.

The table shows that the ideal Navies Bayes model would be the one without applying SMOTE due to a lower FNR of 27.2% compared to the 63.4%. Although SMOTE performed better with the FPR (diagnosing healthy people with cancer) being 58.6% compared to the 62%, there is more weight put on minimizing the FNR. Choosing the no SMOTE Naïve Bayes model is still not considered a good classifier for this dataset due to the FNR and FPR ratio.

D. Logisitic Regression

The discriminative algorithm logisitic regression was implemented with trying 3 different fundamental parameters. Parameter 1 included L1 (LASSO regularization) and solver liblinear. Note LASSO is considered feature selection based on zeroing out the weights. Parameter 2 included L2 (Ridge regression) and solver lbfgs. L2 is not considered for feature selection but rather model selection based on its ability to shrink the weights but not zero them out. The final parameter chosen was no regularization and solver of newton-cg. **Figure 3** shows the roc auc curve of all 3 models.

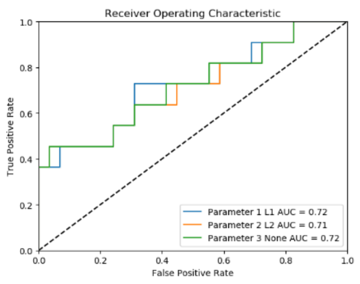

Figure 3: Regularization on logisitic regression model.

Based on figure 2 it can be observed that all three models perform similar in their prediction strength, but further analysis was done to pick the threshold to optimize the ratio of FNR to FPR. **Table 2** shows the optimal threshold and confusion matrix applied to each model.

	L1	L2	None
TN	20	16	13
FP	9	13	16
FN	3	3	3
TP	8	8	8
FNR	27.30%	27.30%	27.30%
FPR	31.00%	44.80%	55.20%
Training Accuracy	75%	70%	75%
Testing Accuracy	70.00%	60.00%	52.50%
Threshold	0.202807	0.151156	0.113439

Table 2: Confusion matrix based on optimal threshold.

Table 2 provides data on choosing the model that minimized the FNR while trying to also keep the FPR low, throughout this project it is observed that there is an inverse relationship between FPR and FNR rate so measure must be taken by choosing the model with the optimal threshold that minimizes both of these but puts more emphasis in keeping the FNR lower. Based on table 2, parameter utilizing L2 regularization with a threshold of 0.202807 performed the best out of the 3 models. The FNR and FPR ratio of 27.30% and 31% make logisitic regression a favorable model for this dataset.

E. Support Vector Machine

Support Vector Machines are considered a discriminative classifier and they try to find the decision boundary between the binary classes by projecting from 2 dimensions to a higher dimension. 3 SVM models were applied with 3 different types of parameters. Parameter 1involved a rbf kernel and L2 regularization of 50, parameter 2 consisted of a sigmoid kernel and L2 value of 2, and parameter 3 has a linear kernel with a regularization of 0.0001. **Figure 4** shows the roc auc curve of three parameters plotted against one another.

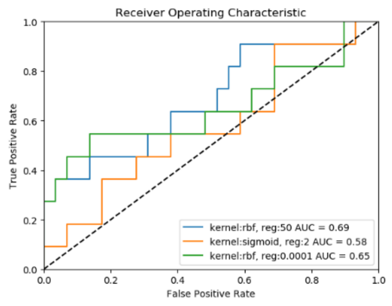

Figure 4: Displays different kernel and regularization values of SVM classifier.

Figure 3 shows that the preferred classifier is parameter 1 with kernel rbf and regularization of 50, followed by parameter 2, and finally 3 based on their respective auc scores, to further choose the model optimal threshold values based off the figure were chosen to minimize FNR. **Table 3** shows the confusion matrix based on the threshold value that minimized FNR the most.

Regularization	50	2	0.0001
Kernel	rbf	sigmoid	linear
TN	14	12	11
FP	15	17	18
FN	3	4	3
TP	8	7	8
FNR	27.27%	36.36%	27.27%
FPR	51.72%	58.62%	62.07%
Traning Accuracy	70%	55.00%	70%
Testing Accuracy	55.00%	47.50%	47.50%
Threshold	0.150286	0.182704	0.186207

Table 3: Confusion matrix of kernel and regularization values applied to SVM.

Table 1 shows the best parameter to use is a rbf kernel with a ridge regression regularization value of 50, which yielded a FNR of 27.27% and a FPR value of 51.72%. Unfortunately, this would not be a strong model due to how high the FPR is even though the FNR rate is adequate. The FPR rate shows that over 51% of patients who are healthy will be diagnosed as having cancer which is a poor outcome.

F. Multi-Layer Perceptron

Multi-Layer perceptron is another type of discriminative classifier that tries to form a decision boundary; however, it differs from linear or logisitic model because it allows for nonlinear functions. **Figure 5** displays the roc auc curve for all parameters chosen.

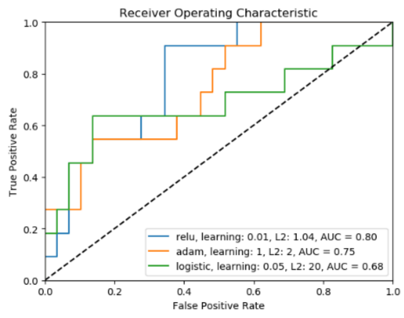

Figure 5: Parameter performance on MLP.

Based on the figure, model with the largest area under the curve uses an activation function called relu, regularization of 1.04, batch size 4, and 100 hidden layers (shown in table below). Choosing the best model to improve the sensitivity is shown in **Table 4**, where the confusion matrix of each model based on the optimal threshold is displayed.

Regularization L2	1.04	2	20
Activation	relu	tanh	logistic
learning rate	0.01	1	0.05
batch size	4	6	1
hidden layers	100	6	25
max iterations	100000	100000	100000
TN	19	15	16
FP	10	14	13
FN	2	2	4
TP	9	9	7
FNR	18.18%	18.18%	36.36%
FPR	34.48%	48.28%	44.83%
Traning Accuracy	70%	70.00%	55%
Testing Accuracy	70.00%	60.00%	57.50%
Threshold	0.22431	0.277603	0.25399

Table 1: Confusion matrix based on thresholds.

Table 1 shows that the best parameter set is indeed model 1 as shown with figure 1's auc. Further analysis shows the optimal threshold is 0.22431 providing a FNR of 18.18% and FPR of 34.48% making this model the best predictor throughout the whole experiment.

G. K Nearest Neighbors

K nearest neighbors (KNN) is a generative classifier that aims to find the joint probability distribution. The models chosen are shown in **figure 6** where the roc auc will be used to find the best model with sufficient parameters.

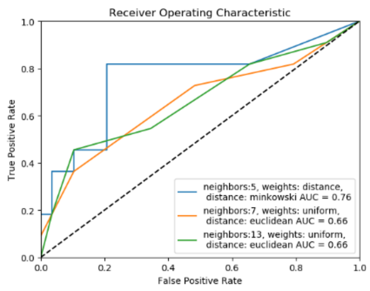

Figure 6: Model selection based on parameter choice

The figure shows that the parameter using 5 neighbors, calculating distance by minkowski, and weights set to distance (closer points have more weight than further points) yields the highest auc score. The other two models with similar parameters from one another (differing only by neighbors) produced the same score. **Table 5** provides the optimal threshold to minimize the FNR.

Neighbors	5	7	13
weights	distance	uniform	uniform
distance	minkowski	euclidean	euclidean
TN	23	15	10
FP	6	14	19
FN	3	3	2
TP	8	8	9
FNR	27.27%	27.27%	18.20%
FPR	20.70%	48.30%	65.50%
training accuracy	60%	50%	45%
testing accuracy	77.50%	57.50%	47.50%
threshold	0.227311	0.285714	23.08%

Table 5: Threshold confusion matrix.

Based on the figure and further proved by the table shows the best model is KNN with its FNR accuracy of 27.7% and exceptionally low FPR of 20.7%. Experimenting with different parameters shows that KNN is a powerful model for this dataset, provided that parameter set one is used.

III. Results

The individual results have been provided and models were selected based off certain thresholds, this section aims to pick one model that will be the classifier used for this dataset. **Figure 7** below shows the roc auc curve of all the classifiers plotted together (each classifier was chosen with its optimal parameter list).

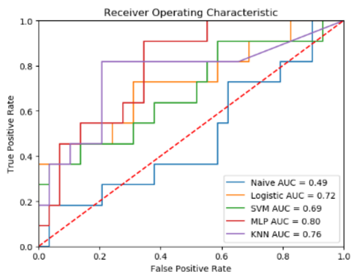

Figure 7: roc auc for all classifiers.

Based off figure 1, MLP and KNN were the strongest classifiers in increasing the ratio of TPR/FPR due to the auc score. Logistic regression and SVM performed well, while Naïve Bayes performance being unfavorable. The final decision to rank the classifiers will be based off the threshold value obtained by the roc auc curve. **Table 6** shows each classifiers FNR, FPR, and accuracies based on the best threshold value.

Classifier	FNR	FPR	Accuracy
Naïve Bayes	27%	62%	47.50%
Logisitic Regression	27%	31%	70%
Support Vector Machines	27%	51%	55%
Multi-Layer Peceptron	18%	34%	70%
K Nearest Neighbors	27%	20%	77.50%

Table 6: Model threshold rates.

Table 1 shows that the number one classifier for this dataset is MLP, followed by KNN, logisitic regression, and lastly Naïve Bayes.

IV. Conclusion

The project showed a breadth of probabilistic classifier in diagnosing if patients breast cancer returned. The experiments ran showed that accuracy alone is not sufficient enough in choosing a model, there were several cases in which a high testing accuracy produced a FNR of 81%, meaning cancer patients were classified as healthy. This is an issue because patients are not getting the treatment they need and increasing the risk of death. This problem can be prevented by utilizing a roc auc curve and evaluating the confusion matrix based on the desired threshold from the curve. Therefore, employing this method provided the best model (MLP) that can reduce the risk of FNR and provide good classification measures.

Reference

1. Breast Cancer Statistics. (2019, May 28). Retrieved April 20, 2020, from https://www.cdc.gov/cancer/breast/statistics/index.htm

2. U.S. Breast Cancer Statistics. (2020, January 27). Retrieved April 20, 2020, from https://www.breastcancer.org/symptoms/understand_bc/statistics

3. Wolberg, W. H. (n.d.). Breast Cancer Wisconsin (Prognostic) Data Set. Retrieved April 20, 2020, from http://archive.ics.uci.edu/ml/datasets/Breast Cancer Wisconsin (Prognostic)

4. Powell, V. (n.d.). Principal Component Analysis explained visually. Retrieved April 20, 2020, from https://setosa.io/ev/principal-component-analysis/

5. Brownlee, J. (2020, April 7). SMOTE for Imbalanced Classification with Python. Retrieved April 20, 2020, from https://machinelearningmastery.com/smote-oversampling-for-imbalanced-classification/

Code

```
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.preprocessing import StandardScaler
import sklearn.model_selection as model_selection
from sklearn.naive_bayes import GaussianNB
from sklearn.linear_model import LogisticRegression
from sklearn.neural_network import MLPClassifier
from imblearn.over_sampling import SMOTE
from sklearn.decomposition import PCA
from sklearn.metrics import accuracy_score
from sklearn.metrics import confusion_matrix
from sklearn.metrics import roc_curve, auc
import numpy as np
from sklearn.svm import SVC
from sklearn.neighbors import KNeighborsClassifier
import seaborn as sns
# pycharm code below to print out how many features to
display

pd.set_option('display.width', 400)
pd.set_option('display.max_columns', 36)

#=====================================================data
pre-
proccessing==============================================
=
# load the dataset
dataset = pd.read_csv("breast_cancer.csv",sep='\s*,\s*',
engine='python',encoding='utf-8-sig') # sep used to
ignore spaces in column name when calling them

print(dataset.head())
print(dataset.shape)# 198 by 35

# replace R with 1 and N with 0
dataset.replace(to_replace = ['R','N'], value=[1,0],
inplace=True)
print(dataset)

# visualize the data
sns.set_style("whitegrid")
sns.FacetGrid(dataset, hue="outcome", height=4) \
   .map(plt.scatter, "time", "tumor size") \
   .add_legend()
plt.show()

print(dataset['outcome'].value_counts()) # count how many
numbers are in each class of the target variable
#The dataset is really unbalanced at class 0: 151 and
class 1: 47

#sns.countplot(x = 'outcome', data=dataset) # counts how
many different class and how many of each there are
#plt.show()

print(dataset.groupby('outcome').mean()) # displays the
average value of each column that gives that particular
class

########################################Data-processing
#########################################################
######
# this is to check if there are any missing values in our
dataset in which case there are 4 in lymph node
print('Nan before impute\n ',dataset.isna().sum()) #
count the number of NaN values (missing feature values)

# replace Nan values with average value of the feature
dataset.fillna(dataset.mean(), inplace=True)

# check if nan have been changed with average value
print('Nan before impute\n ',dataset.isna().sum()) #
count the number of NaN values (missing feature values)

# drop the ID column because it is not useful
dataset.drop('ID', axis=1, inplace=True)
print(dataset)

# Separate the features and target variables
X = dataset.iloc[:,0:33]
y = dataset.iloc[:,-1]
print(X.head())
# The next step is to standardize our features we will
use the z score which gives mean 0 and variance 1
sc = StandardScaler() # create object of class
xScaled = sc.fit_transform(X)


#split the scaled data into train (70%), test (20%), and
validation (10%)
#This is done by running train test split twice
X_train, X_test, y_train, y_test =
model_selection.train_test_split(xScaled, y,
test_size=0.2, random_state=1)
X_train, X_val, y_train, y_val =
model_selection.train_test_split(X_train,
y_train,test_size=0.125, random_state=1)

print('X_train: ', X_train.shape)
print('X_test: ', X_test.shape)
print('X_val shape: ',X_val.shape)
# Apply PCA for dimensionality reduction
pca = PCA(.95) # 95% variance retained
pca.fit(X_train) # fit only on the training set
print('95% of variance amounts to ', pca.n_components_,'
components')
# transform on everything
X_train = pca.transform(X_train)
X_val = pca.transform(X_val)
X_test = pca.transform(X_test)
# we want to over sample the minority class to help
balance the dataset and generate "new" samples of
patients whose breast cancer reoccured
smote = SMOTE(random_state=0)
xSmote,ySmote=smote.fit_sample(X_train, y_train)

# compare class counts before and after
print(y_train.value_counts()) # before 0: 107, 1: 31
print(ySmote.value_counts()) # after 0: 107, 1:107


def naives():
    THRESHOLD = 0.031
    classifier = GaussianNB()
    classifier.fit(X_train, y_train)
    fpr, tpr, threshold = roc_curve(y_test,
classifier.predict_proba(X_test)[:,1])

    # Parameters 2
    clf = GaussianNB()
    clf.fit(xSmote,ySmote)
    fpr2, tpr2, threshold2 = roc_curve(y_test,
clf.predict_proba(X_test)[:, 1])
    naive_roc_auc2 = auc(fpr2,tpr2)

    naive_roc_auc = auc(fpr, tpr)
    plt.figure()
    plt.title('Receiver Operating Characteristic')
    plt.plot(fpr, tpr, label='NO SMOTE = %0.2f' %
naive_roc_auc)
    plt.plot(fpr2, tpr2, label='SMOTE = %0.2f' %
naive_roc_auc2)
    plt.legend(loc='lower right')
    plt.plot([0, 1], [0, 1], 'k--')
    plt.xlim([0, 1])
    plt.ylim([0, 1])
    plt.ylabel('True Positive Rate')
    plt.xlabel('False Positive Rate')

    # # create the axis of thresholds (scores)
    # ax2 = plt.gca().twinx()
    # ax2.plot(fpr, threshold, markeredgecolor='r',
linestyle='dashed', color='r')
    # ax2.set_ylabel('Threshold', color='r')
    # ax2.set_ylim([threshold[-1], threshold[0]])
    # ax2.set_xlim([fpr[0], fpr[-1]])
    plt.show()

    # Predicting the model
    preds_threshold =
np.where(clf.predict_proba(X_val)[:, 1] > THRESHOLD, 1,
0)
    tn, fp, fn, tp = confusion_matrix(y_val,
preds_threshold).ravel()
    print('tn: {} fp: {} fn: {} tp: {}' .format(tn, fp,
fn, tp))
    print("Accuracy score:
{}".format(accuracy_score(y_val, preds_threshold)*100))
    data = {'FPR':fpr2, 'TPR':tpr2}
    table = pd.DataFrame(data, index = threshold2)
    print(table)
    return fpr, tpr, threshold


def logRegres():
    THRESHOLD = 0.113439
    classifier =
LogisticRegression(penalty='l1',solver='liblinear',
max_iter=100000)  # create object of the class
LogisticRegression
    classifier.fit(X_train, y_train)
    fpr, tpr, threshold = roc_curve(y_test,
classifier.predict_proba(X_test)[:,1])

    # Parameters 2
    clf2 = LogisticRegression(penalty='l2',
solver='lbfgs', max_iter=100000)
    clf2.fit(X_train, y_train)
    fpr2, tpr2, threshold2 = roc_curve(y_test,
clf2.predict_proba(X_test)[:, 1])
    logs_roc_auc2 = auc(fpr2, tpr2)

    #parameter 3
    clf3 = LogisticRegression(penalty='none',
solver='newton-cg', max_iter=100000)
    clf3.fit(X_train, y_train)
    fpr3, tpr3, threshold3 = roc_curve(y_test,
clf3.predict_proba(X_test)[:, 1])
    logs_roc_auc3 = auc(fpr3, tpr3)

    logs_roc_auc = auc(fpr, tpr)
    plt.figure()
    plt.title('Receiver Operating Characteristic')
    plt.plot(fpr, tpr, label='Parameter 1 L1 AUC = %0.2f'
% logs_roc_auc)
    plt.plot(fpr2, tpr2, label='Parameter 2 L2 AUC =
%0.2f' % logs_roc_auc2)
    plt.plot(fpr3, tpr3, label='Parameter 3 None AUC =
%0.2f' % logs_roc_auc3)
    plt.legend(loc='lower right')
    plt.plot([0, 1], [0, 1], 'k--')
    plt.xlim([0, 1])
    plt.ylim([0, 1])
    plt.ylabel('True Positive Rate')
    plt.xlabel('False Positive Rate')
    #
    # # create the axis of thresholds (scores)
    # ax2 = plt.gca().twinx()
    # ax2.plot(fpr, threshold, markeredgecolor='r',
linestyle='dashed', color='r')
    # ax2.set_ylabel('Threshold', color='r')
    # ax2.set_ylim([threshold[-1], threshold[0]])
    # ax2.set_xlim([fpr[0], fpr[-1]])
    plt.show()

    preds_threshold =
np.where(clf3.predict_proba(X_val)[:, 1] > THRESHOLD, 1,
0)
    tn, fp, fn, tp = confusion_matrix(y_val,
preds_threshold).ravel()
    print('tn: {} fp: {} fn: {} tp: {}'.format(tn, fp,
fn, tp))
    print("Accuracy score:
{}".format(accuracy_score(y_val, preds_threshold) * 100))
    data = {'FPR': fpr3, 'TPR': tpr3}
    table = pd.DataFrame(data, index=threshold3)
    print(table)
    return fpr, tpr, threshold


def svm():
    THRESHOLD = 0.186207
    classifier = SVC(kernel= 'rbf', random_state=0,
probability=True, tol=1e-5,max_iter=100000,C= 50)
    classifier.fit(X_train, y_train)
    fpr, tpr, threshold = roc_curve(y_test,
classifier.predict_proba(X_test)[:,1])

    # Parameters 2
    clf2 = SVC(kernel='sigmoid', C=2, probability=True,
max_iter=100000, random_state=0)
    clf2.fit(X_train, y_train)
    fpr2, tpr2, threshold2 = roc_curve(y_test,
clf2.predict_proba(X_test)[:, 1])
    svm_roc_auc2 = auc(fpr2, tpr2)

    # parameter 3
    clf3 = SVC(kernel='linear', C= 0.0001,
probability=True, max_iter=100000, random_state=0)
    clf3.fit(X_train, y_train)
    fpr3, tpr3, threshold3 = roc_curve(y_test,
clf3.predict_proba(X_test)[:, 1])
    svm_roc_auc3 = auc(fpr3, tpr3)

    svm_roc_auc = auc(fpr, tpr)
    plt.figure()
    plt.title('Receiver Operating Characteristic')
    plt.plot(fpr, tpr, label='kernel:rbf, reg:50 AUC =
%0.2f' % svm_roc_auc)
    plt.plot(fpr2, tpr2, label='kernel:sigmoid, reg:2 AUC
= %0.2f' % svm_roc_auc2)
    plt.plot(fpr3, tpr3, label='kernel:rbf, reg:0.0001
AUC = %0.2f' % svm_roc_auc3)
    plt.legend(loc='lower right')
    plt.plot([0, 1], [0, 1], 'k--')
    plt.xlim([0, 1])
    plt.ylim([0, 1])
    plt.ylabel('True Positive Rate')
    plt.xlabel('False Positive Rate')
    #
    # # create the axis of thresholds (scores)
    # ax2 = plt.gca().twinx()
    # ax2.plot(fpr, threshold, markeredgecolor='r',
linestyle='dashed', color='r')
    # ax2.set_ylabel('Threshold', color='r')
    # ax2.set_ylim([threshold[-1], threshold[0]])
    # ax2.set_xlim([fpr[0], fpr[-1]])
    plt.show()

    preds_threshold =
np.where(clf3.predict_proba(X_val)[:, 1] > THRESHOLD, 1,
0)
    tn, fp, fn, tp = confusion_matrix(y_val,
preds_threshold).ravel()
    print('tn: {} fp: {} fn: {} tp: {}'.format(tn, fp,
fn, tp))
    print("Accuracy score:
{}".format(accuracy_score(y_val, preds_threshold) * 100))
    data = {'FPR': fpr3, 'TPR': tpr3}
    table = pd.DataFrame(data, index=threshold3)
    print(table)
    return fpr, tpr, threshold


def mlp():
    THRESHOLD = 0.253990
    classifier =
MLPClassifier(random_state=0,learning_rate_init=0.01,
activation='relu', alpha=1.04,
                        hidden_layer_sizes=100,
max_iter=100000, batch_size= 4)
    classifier.fit(X_train, y_train)
    fpr, tpr, threshold = roc_curve(y_test,
classifier.predict_proba(X_test)[:,1])

    # Parameters 2
    clf2 = MLPClassifier(random_state=0,
learning_rate_init=0.0001, activation='tanh',alpha= 2,
                         hidden_layer_sizes= 6,
batch_size= 6, max_iter=100000  )
    clf2.fit(X_train, y_train)
    fpr2, tpr2, threshold2 = roc_curve(y_test,
clf2.predict_proba(X_test)[:, 1])
    mlp_roc_auc2 = auc(fpr2, tpr2)

    # parameter 3
    clf3 = MLPClassifier(random_state=0,
learning_rate_init=0.05, activation='logistic',alpha= 20,
                         hidden_layer_sizes=25,
batch_size=1, max_iter=100000)
    clf3.fit(X_train, y_train)
    fpr3, tpr3, threshold3 = roc_curve(y_test,
clf3.predict_proba(X_test)[:, 1])
    mlp_roc_auc3 = auc(fpr3, tpr3)

    mlp_roc_auc = auc(fpr, tpr)
    plt.figure()
    plt.title('Receiver Operating Characteristic')
    plt.plot(fpr, tpr, label='relu, learning: 0.01, L2:
1.04, AUC = %0.2f' % mlp_roc_auc)
    plt.plot(fpr2, tpr2, label='adam, learning: 1, L2: 2,
AUC = %0.2f' % mlp_roc_auc2)
    plt.plot(fpr3, tpr3, label='logistic, learning: 0.05,
L2: 20, AUC = %0.2f' % mlp_roc_auc3)
    plt.legend(loc='lower right')
    plt.plot([0, 1], [0, 1], 'k--')
    plt.xlim([0, 1])
    plt.ylim([0, 1])
    plt.ylabel('True Positive Rate')
    plt.xlabel('False Positive Rate')
    #
    # # create the axis of thresholds (scores)
    # ax2 = plt.gca().twinx()
    # ax2.plot(fpr, threshold, markeredgecolor='r',
linestyle='dashed', color='r')
    # ax2.set_ylabel('Threshold', color='r')
    # ax2.set_ylim([threshold[-1], threshold[0]])
    # ax2.set_xlim([fpr[0], fpr[-1]])
    plt.show()

    preds_threshold =
np.where(clf3.predict_proba(X_val)[:, 1] > THRESHOLD, 1,
0)
    tn, fp, fn, tp = confusion_matrix(y_val,
preds_threshold).ravel()
    print('tn: {} fp: {} fn: {} tp: {}'.format(tn, fp,
fn, tp))
    print("Accuracy score:
{}".format(accuracy_score(y_val, preds_threshold) * 100))
    data = {'FPR': fpr3, 'TPR': tpr3}
    table = pd.DataFrame(data, index=threshold3)
    print(table)
    return fpr, tpr, threshold


def KNN():
    THRESHOLD = 0.230769
    classifier = KNeighborsClassifier(n_neighbors=5,
algorithm='auto', weights= 'distance',
metric='minkowski')
    classifier.fit(X_train, y_train)
    fpr, tpr, threshold = roc_curve(y_test,
classifier.predict_proba(X_test)[:, 1])

    # Parameters 2
    clf2 = KNeighborsClassifier(n_neighbors=7,
metric='minkowski', weights='uniform')
    clf2.fit(X_train, y_train)
    fpr2, tpr2, threshold2 = roc_curve(y_test,
clf2.predict_proba(X_test)[:, 1])
    knn_roc_auc2 = auc(fpr2, tpr2)

    # parameter 3
    clf3 = KNeighborsClassifier(n_neighbors=13,
metric='euclidean', weights='uniform')
    clf3.fit(X_train, y_train)
    fpr3, tpr3, threshold3 = roc_curve(y_test,
clf3.predict_proba(X_test)[:, 1])
    knn_roc_auc3 = auc(fpr3, tpr3)
    KNN_roc_auc = auc(fpr, tpr)

    plt.figure()
    plt.title('Receiver Operating Characteristic')
    plt.plot(fpr, tpr, label='neighbors:5, weights:
distance,\n distance: minkowski AUC = %0.2f' %
KNN_roc_auc)
    plt.plot(fpr2, tpr2, label='neighbors:7, weights:
uniform,\n distance: euclidean AUC = %0.2f' %
knn_roc_auc2)
    plt.plot(fpr3, tpr3, label='neighbors:13, weights:
uniform,\n distance: euclidean AUC = %0.2f' %
knn_roc_auc3)
    plt.legend(loc='lower right')
    plt.plot([0, 1], [0, 1], 'k--')
    plt.xlim([0, 1])
    plt.ylim([0, 1])
    plt.ylabel('True Positive Rate')
    plt.xlabel('False Positive Rate')
    #
    # # create the axis of thresholds (scores)
    # ax2 = plt.gca().twinx()
    # ax2.plot(fpr, threshold, markeredgecolor='r',
linestyle='dashed', color='r')
```

```
    # ax2.set_ylabel('Threshold', color='r')
    # ax2.set_ylim([threshold[-1], threshold[0]])
    # ax2.set_xlim([fpr[0], fpr[-1]])
    plt.show()

    preds_threshold =
np.where(clf3.predict_proba(X_val)[:, 1] > THRESHOLD, 1,
0)
    tn, fp, fn, tp = confusion_matrix(y_val,
preds_threshold).ravel()
    print('tn: {} fp: {} fn: {} tp: {}'.format(tn, fp,
fn, tp))
    print("Accuracy score:
{}".format(accuracy_score(y_val, preds_threshold) * 100))
    data = {'FPR': fpr3, 'TPR': tpr3}
    table = pd.DataFrame(data, index=threshold3)
    print(table)
    return fpr, tpr, threshold

def roc_auc():
    iter = 1000
    naivefpr, naivetpr, naivethreshold = naives()
    logsfpr, logstpr, logsthreshol = logRegres()
    svmfpr, svmtpr, svmthreshold = svm()
    mlpfpr, mlptpr, mlpthreshold = mlp()
    adafpr, adatpr, adathreshold = KNN()

    naive_roc_auc = auc(naivefpr, naivetpr)
    logs_roc_auc = auc(logsfpr, logstpr)
    svm_roc_auc = auc(svmfpr, svmtpr)
    mlp_roc_auc = auc(mlpfpr, mlptpr)
    knn_roc_auc = auc(adafpr, adatpr)

    # image drawing
    plt.figure()
    plt.title('Receiver Operating Characteristic')
    plt.plot(naivefpr, naivetpr, label='Naive AUC =
%0.2f' % naive_roc_auc)
    plt.plot(logsfpr, logstpr, label='Logistic AUC =
%0.2f' % logs_roc_auc)
    plt.plot(svmfpr, svmtpr, label='SVM AUC = %0.2f' %
svm_roc_auc)
    plt.plot(mlpfpr, mlptpr, label='MLP AUC = %0.2f' %
mlp_roc_auc)
    plt.plot(adafpr, adatpr, label='KNN AUC = %0.2f' %
knn_roc_auc)


    plt.legend(loc='lower right')
    plt.plot([0, 1], [0, 1], 'r--')
    plt.xlim([0, 1])
    plt.ylim([0, 1])
    plt.ylabel('True Positive Rate')
    plt.xlabel('False Positive Rate')
    plt.show()
```

```
def classifierSelection(argument):
    switcher = {
        1: naives,
        2: logRegres,
        3: svm,
        4: mlp,
        5: KNN,
        6: roc_auc

    }
    # Get the function from switcher dictionary
    func = switcher.get(argument, lambda: "Invalid
Classifier")
    # Execute the function
    print(func())



# Create a menu system for the user to choose which
classifier to run
def menu():
    print("############################   Menu
############################################")

    print(' Displaying a list of classifier to use\n')

    clf = int(input("""
                  1: Naive Bayes
                  2: Logistic Regression
                  3: Primal Support Vector Machine
                  4: Multi-Layer Perceptron
                  5: K Nearest Neighbors
                  6: roc_auc
                  7: Quit/Log Out


                  Please enter your choice: """))

    if clf == 1 or clf == 2 or clf == 3 or clf == 4 or
clf == 5 or clf == 6:
        classifierSelection(clf)
        menu()
    elif clf == 7:
        print('Quiting program ')
        return None
    else:
        print("Invalid choice, choose again")
        menu()


menu()
```