**Hamza Oudrhiri**

**K1910115**

**Machine Learning and Artificial Intelligence**

**Assignment 1: Report**

**Part I:**

The dataset we used for our assignment was part of the scikit-learn library.

It is a study conducted by Dr.William H. Wolberg, W.Nick Street and Olvi L. Mangasarian in November 1995

We stored the dataset in a variable by calling the function load\_breast\_cancer()

It returns a Bunch object, which is similar to a dictionary object.

The bunch contains 6 key-value pairs, which are defined below:

* Data: this is where our measurement data are stored
* Target: An array of integers which state the class of each instance
* Target\_names: In this case, it is a binary classification, so these values are the 2 class names: benign and malignant
* DESCR: A description of the dataset
* Feature\_names: The names of each features measured
* Filename: Where the library file is stored on the computer

The data gathered here is of 569 instances, where 10 unique features were measured:

- 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^2 / 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 radius, texture, perimeter and area are float64 numbers that can be greater than 1. The smoothness, compactness, concavity, concave points, symmetry and fractal dimension are float64 variables whose values are between 0 and 1.

These features are separated into 3 attributes: mean, error and worst. This gives us a dataset that is an array of 569 rows by 30 columns. The data was divided into a training and testing set using the *train\_test\_split()* function with the testing set being 30% of the original dataset.

Each instance of data is classified into one of the two classes: benign (no cancer is present) or malignant (the person has breast cancer). The distribution of the classes is as follow: 212 are malignant and 357 benign.

**Part II: Clustering**

Clustering is a machine learning method that aims to split the data into *n* clusters (defined by the user), where every point in a cluster are similar and points in different clusters are far apart. We can then use the trained model to predict which cluster a new point belongs in. In our case, since the data had 2 classes, we decided to assign the number of clusters of our model to 2.

1st method: K-means clustering

K-means clustering is a protocol that arranges the data into a given number of clusters by randomly initializing cluster centers and assigning the data points to the appropriate cluster, then recalculating the cluster center as the mean of the data points belonging to that cluster, these steps are done multiple times until the cluster centers do not change.

For our application, since we wanted to visualize the data we first had to pre-process the data by applying the Principal Component Analysis (PCA). PCA reduces the number of features of our data and “rotates the dataset such that the rotated features are statistically uncorrelated” (1).

We gave the data 2 principal components to simplify the visualization and plotting. This was done by calling the *PCA()* method with an argument *n\_components* = 2, training the model and transforming the data using the *transform()* method. Once the data was processed, we could move on to applying the K-means protocol.

The dataset contained only 2 classes, therefore it was logical to create a model that computes 2 clusters, so our model could be evaluated effortlessly.

*Kmeans()* function is part of the scikit-learn cluster library, calling it initializes and returns the untrained model, so we stored it in our application in a variable named *kmean*. By default, the function would create a model of 8 clusters, so we had to modify the *n\_clusters* parameter in the function call and assign it to 2.

Once the model created, a handy method *fit\_predict(X)* can be called in order to train the model with the data *X* and apply the model to predict the outcome on *X*. The function returned a one-dimensional array the same size as X.

The training took 7 iterations to come up with a satisfying clustering model.

**Evaluation:**

|  |  |
| --- | --- |
| Adjusted rand index | 0.659 |
| Homogeneity | 0.531 |
| Completeness | 0.55 |
| V-Measure | 0.54 |

2nd method: Agglomerative Clustering

Agglomerative clustering is a protocol that starts by declaring each data point as its own cluster, and then merges similar data points in clusters until the number of clusters defined in the parameters is met. Clusters are combined by a linkage criterion, in our case ‘*ward’* which gave us the best results. “*Ward* picks the two clusters to merge such that the variance within all clusters increases the least. This often leads to clusters that are relatively equally sized” (2).

*AgglomerativeClustering()* method is part of the scikit-learn cluster library, we stored the untrained model in a variable named *agg*. By default, the function would create a model of 2 clusters and used the ‘*ward’* linking criterion, so there was no need to modify the parameters.

Similarly to K-means, we called the *fit\_predict()* method on the model to train it using the data and predict the class outcomes for the same dataset.

**Evaluation:**

|  |  |
| --- | --- |
| Adjusted Rand index | 0.659 |
| Homogeneity | 0.531 |
| Completeness | 0.545 |
| V-measure | 0.538 |

As we can see, both the K-means and agglomerative clustering protocols return quite similar external clustering performance evaluation scores. However, the k-means surpasses the agglomerative clustering in completeness and v-measure scores. Completeness is defined as “A clustering result satisfies completeness if all the data points that are members of a given class are elements of the same cluster” (3), which means the K-means protocol performed better in clustering datasets of the same class.

**Part III: Classification**

Classification is a supervised machine learning method. This method aims to label new data as one of the classes. It is supervised as during the training and testing of the model, each data point is labeled the real label, called ground truth. In our case for the breast cancer dataset, only 2 classes exist, which is called a binary classification task.

As the assignment requested, a k-fold cross-validation was applied to the data. In our case, the dataset was partitioned into 10 parts of equal size, and the models were trained 10 times, where each fold once became the test set and 9 times the training set. This is done so the model generalizes better to new data, and doesn’t overfit on the training set.

1st method: Gradient boosted regression trees (GBM)

Gradient boosted regression trees are a method that combines multiple decision trees into one. “The main idea behind gradient boosting is to combine many simple models (in this context known as weak learners), like shallow trees. Each tree can only provide good predictions of part of the data, and so more and more trees are added to iteratively improve performance.” (4)

The GBM has 2 notable parameters that can be changed to modify the outcome: the maximum depth and the learning rate. Lowering the maximum depth of the trees will lead to a simpler model, but it is less likely to overfit. The learning rate controls the degree to which each tree is allowed to correct the mistakes of the previous trees.

To find the best values of the parameter, we did what is called a grid-search, combining different parameter pairs. In our case, the optimal learning rates was 0.1 and the maximum tree depth was 1. The parameter were assessed by using the score, thanks to the *cross\_val\_score()* function. As this method returns an array of 10 scores, we had to compute the mean, and compare the means of each unique combination of parameter values.

**Evaluation:**

|  |  |
| --- | --- |
| Balanced Accuracy | 0.946 |
| F1-score | 0.963 |
| ROC AUC score | 0.996 |

Confusion matrix for GBRT:

[[ 58 5]

[ 3 105]]

2nd method: Logistic regression

Logistic regression is a linear model used for classification such as the formula can be simplified as:



In our case, since there are 30 features then *n* = 30.

The model tries to predict the class by looking at where the point stands compared to the plane. A binary linear classifier separates two classes using a lane, a plane, or a hyperplane.

Logistic regression has one important parameter to tune, called C which determines the strength of the regularization. Higher values of C try to fit the training set as best as possible, but can lead to overfitting.

Similarly to the GBM, we searched through an array of values of C to find the optimal one, in our case it was 10, and the maximum number of iterations for the solvers to converge was 10000.

|  |  |
| --- | --- |
| Balanced Accuracy | 0.954 |
| F1-score | 0.968 |
| ROC AUC score | 0.998 |

Confusion matrix for LR:

[[ 59 4]

[ 3 105]]

Once again, the two methods gave similar evaluation scores. However, from the balanced accuracy, f1-score and ROCAUC score, we can see that the Logistic Regression performed better on the test set.

**References**

(1). A.Mueller (2018), *Introduction to Machine Learning with Python*, p.156

(2). A.Mueller (2018), *Introduction to Machine Learning with Python*, p.198

(3).<https://docs.w3cub.com/scikit_learn/modules/generated/sklearn.metrics.completeness_score/>

(4) A.Mueller (2018), *Introduction to Machine Learning with Python*, p.105

**Appendix**

Part I:

****

Part II:

import numpy as np  
import pandas as pd  
import matplotlib.pyplot as plt  
  
from sklearn.datasets import load\_breast\_cancer  
from sklearn.model\_selection import train\_test\_split  
from sklearn.model\_selection import ShuffleSplit  
from sklearn.metrics import homogeneity\_score  
from sklearn.metrics import completeness\_score  
from sklearn.metrics import adjusted\_rand\_score  
from sklearn.metrics import v\_measure\_score  
from sklearn.decomposition import PCA  
from sklearn.cluster import KMeans  
from sklearn.cluster import AgglomerativeClustering  
from sklearn.preprocessing import StandardScaler  
  
#Loading data and partition it  
  
  
cancer = load\_breast\_cancer()  
  
X = cancer.data  
y = cancer.target  
  
X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, random\_state=0, test\_size=0.3)  
  
shuffle\_split = ShuffleSplit(n\_splits=10, test\_size=0.3, train\_size=0.7)  
  
##############################################################################  
#PART II : Clustering  
print("PART II: Clustering")  
  
#PCA dimension reductionality  
scaler = StandardScaler()  
scaler.fit(X)  
X\_scaled = scaler.transform(X)  
  
pca = PCA(n\_components=2)  
pca.fit(X\_scaled)  
X\_pca = pca.transform(X\_scaled)  
  
plt.scatter(X\_pca[:,0], X\_pca[:,1], c=y)  
#plt.show()  
  
#Method 1 : Kmeans  
kmean = KMeans(n\_clusters=2)  
y\_predKmean = kmean.fit\_predict(X\_pca)  
print("number of iteration: ",kmean.n\_iter\_)  
#Evaluation  
adj1 = adjusted\_rand\_score(y,y\_predKmean)  
print("Adjusted rand index for Kmeans Clustering: {:.3f}".format(adj1))  
  
hom1 = homogeneity\_score(y, y\_predKmean)  
print("Homogeneity score for Kmeans Clustering: {:.3f}".format(hom1))  
  
com1 = completeness\_score(y, y\_predKmean)  
print("Completeness score for Kmeans Clustering: {:.3f}".format(com1))  
  
v1 = v\_measure\_score(y, y\_predKmean)  
print("V-measure for Kmeans Clustering: {:.3f}".format(v1))  
  
#Method 2 : Agglomerative Clustering  
agg = AgglomerativeClustering()  
y\_predAgg = agg.fit\_predict(X\_pca)  
  
#Evaluation  
adj2 = adjusted\_rand\_score(y, y\_predAgg)  
print("Adjusted rand index for Agglomerative Clustering: {:.3f}".format(adj2))  
  
hom2 = homogeneity\_score(y, y\_predAgg)  
print("Homogeneity score for Agglomerative Clustering: {:.3f}".format(hom2))  
  
com2 = completeness\_score(y, y\_predAgg)  
print("Completeness score for Agglomerative Clustering: {:.3f}".format(com2))  
  
v2 = v\_measure\_score(y, y\_predAgg)  
print("V-measure for Agglomerative Clustering: {:.3f}".format(v2))

Part III:

import numpy as np  
import pandas as pd  
import matplotlib.pyplot as plt  
  
from sklearn.datasets import load\_breast\_cancer  
from sklearn.model\_selection import train\_test\_split  
from sklearn.model\_selection import ShuffleSplit  
from sklearn.model\_selection import cross\_val\_predict  
from sklearn.model\_selection import cross\_val\_score  
from sklearn.linear\_model import LogisticRegression  
from sklearn.metrics import confusion\_matrix  
from sklearn.metrics import f1\_score  
from sklearn.ensemble import GradientBoostingClassifier  
from sklearn.metrics import balanced\_accuracy\_score  
from sklearn.metrics import roc\_auc\_score  
from sklearn.metrics import roc\_curve  
  
##############################################################################  
#PART III : Classification  
print("PART III: Classification")  
cancer = load\_breast\_cancer()  
  
X = cancer.data  
y = cancer.target  
  
X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, random\_state=0, test\_size=0.3)  
  
shuffle\_split = ShuffleSplit(n\_splits=10, test\_size=0.3, train\_size=0.7)  
  
#1st method : Gradiant Boosting Regression Trees  
print("1st method selected: Gradient boosting regression")  
max\_score = 0  
for max\_depth in [1, 2, 3, 4, 5, 6, 7, 8]:  
 for learning\_rate in [0.001, 0.01, 0.1, 1, 10, 100]:  
 gradient = GradientBoostingClassifier(random\_state=0, learning\_rate = learning\_rate, max\_depth=max\_depth)  
 gradient.fit(X\_train, y\_train)  
  
 score = cross\_val\_score(gradient, X\_test,y\_test).mean()  
  
 if score > max\_score:  
 max\_score = score  
 rateOptimal = learning\_rate  
 depthOptimal = max\_depth  
  
print("Parameters for GBRT: learning\_rate={} , max\_depth={}".format(rateOptimal, depthOptimal))  
gbrt = GradientBoostingClassifier(random\_state=0, n\_estimators=100, learning\_rate=rateOptimal,max\_depth=depthOptimal)  
gbrt.fit(X\_train,y\_train)  
  
scoregbrt = cross\_val\_score(gbrt, X, y, cv=shuffle\_split)  
y\_predGBRT = cross\_val\_predict(gbrt, X\_test, y\_test, cv = 10)  
print("GBRT score:{:.3f}".format(scoregbrt.mean()))  
  
#Evaluation:  
#Balanced accuracy  
print("Balanced accuracy for GBRT: {:.3f} ".format(balanced\_accuracy\_score(y\_test, y\_predGBRT)))  
  
#f1Score  
print("f1Score for GBRT: {:.3f}".format(f1\_score(y\_test, y\_predGBRT)))  
  
#ROC AUC  
roc\_aucGBRT = roc\_auc\_score(y\_test, gbrt.predict\_proba(X\_test)[:, 1])  
print("ROC AUC score: {:.3f}".format(roc\_aucGBRT))  
  
#ROC curve  
fpr,tpr, thresholds = roc\_curve(y\_test, gbrt.decision\_function(X\_test))  
  
#plt.plot(fpr, tpr, label="ROC curve")  
plt.xlabel("FPR")  
plt.ylabel("TPR")  
#plt.show()  
  
#Confusion matrix  
confusionGBRT = confusion\_matrix(y\_test, y\_predGBRT)  
print("Confusion matrix for GBRT:\n",confusionGBRT)  
  
#2nd method : Logistic Regression  
print("2nd method selected : Logistic Regression")  
  
max\_score = 0  
for C in [0.001, 0.01, 0.1, 1, 10, 100]:  
 for max\_iter in [100, 1000, 10000, 100000]:  
 lr = LogisticRegression(C=C, max\_iter= max\_iter)  
 lr.fit(X\_train, y\_train)  
  
 score = cross\_val\_score(lr, X\_test, y\_test).mean()  
  
 if score > max\_score:  
 max\_score = score  
 Coptimal = C  
 max\_iterOptimal = max\_iter  
  
print("Parameters : C ={} , max\_iter={}".format(Coptimal, max\_iterOptimal))  
logreg = LogisticRegression(C = Coptimal, max\_iter=max\_iterOptimal)  
logreg.fit(X\_train, y\_train)  
y\_predLR = cross\_val\_predict(logreg, X\_test, y\_test, cv = 10)  
  
  
#Evaluation:  
#Balanced accuracy  
print("Balanced accuracy for LogReg: {:.3f} ".format(balanced\_accuracy\_score(y\_test, y\_predLR)))  
  
#f1Score  
print("f1Score for LogReg: {:.3f}".format(f1\_score(y\_test, y\_predLR)))  
  
#ROC AUC  
roc\_aucLR = roc\_auc\_score(y\_test, logreg.predict\_proba(X\_test)[:, 1])  
print("ROC AUC score for LogReg: {:.3f}".format(roc\_aucLR))  
  
#ROC curve  
fpr,tpr, thresholds = roc\_curve(y\_test, gbrt.decision\_function(X\_test))  
  
#plt.plot(fpr, tpr, label="ROC curve")  
#plt.show()  
  
#Confusion matrix  
confusionLR = confusion\_matrix(y\_test, y\_predLR)  
print("Confusion matrix for LR:\n",confusionLR)

