**CPSC6114: Fundamentals of Machine Learning**

**Assignment 2: K-nearest Neighbor Algorithm**

Model

The goal of this assignment is to build a **K-nearest neighbor model** to classify breast cells as either cancerous or benign based on the observations provided in **GSE58606\_data.csv** file.

K-nearest neighbor algorithm measures the distances from a point that needs to be classified to already classified points using either Euclidean distance formula or some other distance measure (e.g. Manhattan distance formula). Once measured, the results are sorted by shortest distance. Depending on how many Ks (or classified points that will be used for the analysis) are chosen, the algorithm chooses the classification of the majority (e.g. if K = 5 and we have 2 closest distances of Class A and 3 closest distances of Class B, the point is classified as Class B since that class has a majority). It is advisable to choose odd number of Ks in order to avoid a tie situation.

The model effectiveness is measured by scores obtained from confusion matrix, precision, recall, F1 score and accuracy:

* **Confusion matrix** shows: how many true positive/negative and false positive/negative predictions our model made
* **Precision** shows: out of all positive predicted, what percentage is truly positive (Precision = TP/(TP + FP)
* **Recall** shows: out of the total positive, what percentage are predicted positive (Recall = TP/(TP+FN)
* **F1 score** combines recall and precision and shows its harmonic mean. When F1 score equals to 1, it means all classes were correctly predicted (usually F1 score is rarely equals to 1) (F1 = 2\*(Precision\*Recall)/(Precision + Recall)
* **Accuracy**: compares actual test set of values with predicted values (Accuracy = (TP+TN)/(TP+TN+FP+FN))

Model Building

We start by importing all the relevant libraries into Python, including sklearn, numpy, pandas, and os. We then import the data from insurance.csv into a data frame.

*data1 = pd.read\_csv('GSE5806\_data.csv')*

Once the file is loaded into a dataframe, we need to define our X and Y (target) values and normalize the data.

*X = dataset[dataset.columns[0:1924]]*

*Y = dataset.iloc[:,-2]*

In general when building a KNN model, d**ata normalization** is necessary to make sure that the scale of all variables are comparable. For data normalization we use **MinMax** approach to convert the original range of values into a new range between 0 and 1. However, scaling must be done on a set of well balanced data. In case where data is unbalanced, data scaling will not be as effective.

For data normalization we use **MinMax** approach to convert the original range of values into a new range between 0 and 1.

*X = (X - np.min(X))/(np.max(X) - np.min(X))*

Once normalized we can split the data into training (**75%**) and testing (**25%**).

*X\_train,X\_test,Y\_train,Y\_test=train\_test\_split(X,Y,test\_size=0.25,random\_state=1)*

We are now ready to implement our model. Note, that our choice of K is usually a trial and error process. We can start with **K= 2** and then re-run the model with different values of K to maximize its accuracy

*n\_neighbors=2*

*knn = KNeighborsClassifier(n\_neighbors)*

*y\_pred = knn.fit(X\_train, Y\_train).predict(X\_test)*

Upon execution, we can check model’s performance by running the following functions:

*print("Confusion matrix is:")*

*print(ConfusionMatrix)*

*print("")*

*print("Recall is:")*

*print(Recall)*

*print("")*

*print("Precision is:")*

*print(Precision)*

*print("")*

*print("Accuracy is:")*

*print(Accuracy)*

*print("")*

*print("F1 score is:")*

*print(F1Score)*

First Run: K = 2

The first run produces the below. The model accuracy is fairly good but we can still attempt to improve it by increasing K to a higher value

**Text, letter

Description automatically generated**

Second Run: K = 3

K=3 gives us better results across all scores. We can try to attempt to improve it further.

**Text

Description automatically generated**

Third Run: K = 5

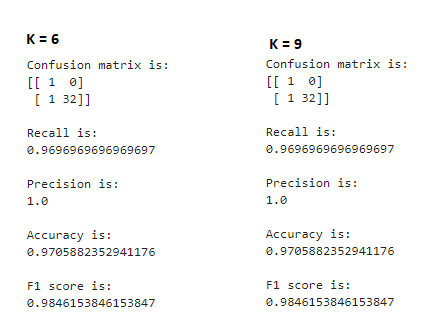
K = 5 gives us the best results. All scores equal to 1 which means our model perfectly predicts the classifications

Text

Description automatically generated

What if we keep increasing K?

If we increase K to 6 or 7 or higher, the model’s performance decreases. Below are the results for K = 6 and K = 9



Conclusion

Based on the current dataset, **K = 5** give us the best predictive results with the current model and current dataset. The algorithm performance worse with values below or greater than 5

Upon performing this analysis, we have to note that the model produces extremely high levels of accuracy which should give us a pause and make us question the dataset itself. After a detailed review we can see that the data provided tin GSE58606\_data.csv file appears to be **UNBALACNED: o**ut of 133 target observations only **11** observations are negative. Unbalanced data set will lead incorrect results, skewing the data in favor of positive results in this case.

Table

Description automatically generated

In case of an unbalance data set, we need to determine if data can be resampled and produce a more balanced set of observations. If obtaining a new data set is not feasible there are other techniques that could attempt to fix the unbalanced data problem. For example we can try the following:

* Under-sampling technique – keep observations from underrepresented class, and randomly select equal number from overrepresented class
* Over-sampling technique – increased the site of underrepresented class by generating new rare samples using Synthetic Minority Oversampling Technique (as one option)

Given that we only have 11 observations in underrepresented class, and will have to reserve at least 70% to training and 30% to testing, it gives us extremely small data set to build our model upon reliably. Therefore I believe that under-sampling technique is not ideal in this case. Most **likely Over-sampling/SMOT technique** would increase the underrepresented class and give us a better sample to work on.

In addition, **data scaling** is an important step for KNN algorithm and should be applied in this case as well. However, data normalization alone **will not be sufficient to get accurate** with unbalanced data set. In this particular case (data set in **GSE58606\_data.csv**), data scaling will not be beneficial to the overall reliability of the model

**Full Python Code:**

*#Import necessary library*

*import os*

*import numpy as np*

*import pandas as pd*

*from sklearn.neighbors import KNeighborsClassifier*

*from sklearn.model\_selection import train\_test\_split*

*from sklearn.metrics import confusion\_matrix, accuracy\_score, precision\_score, recall\_score, f1\_score*

*#Import data*

*os.chdir('C:/Users/denis/OneDrive/Desktop/CPSC6114\_ML')*

*dataset = pd.read\_csv('GSE58606\_data.csv')*

*#Choose and normalize data for our analysis*

*X = dataset[dataset.columns[0:1924]]*

*Y = dataset.iloc[:,-2]*

*#Applying MinMax Scaler formula*

*X = (X - np.min(X))/(np.max(X) - np.min(X))*

*#Split data into train and test as per requirement*

*X\_train,X\_test,Y\_train,Y\_test=train\_test\_split(X,Y,test\_size=0.25,random\_state=1)*

*#Model*

*n\_neighbors=3*

*knn = KNeighborsClassifier(n\_neighbors)*

*y\_pred = knn.fit(X\_train, Y\_train).predict(X\_test)*

*#Model Performance*

*ConfusionMatrix = confusion\_matrix(Y\_test,y\_pred)*

*Accuracy = accuracy\_score(Y\_test,y\_pred)*

*Precision = precision\_score(Y\_test,y\_pred)*

*Recall = recall\_score(Y\_test,y\_pred)*

*F1Score = f1\_score(Y\_test,y\_pred)*

*print("Confusion matrix is:")*

*print(ConfusionMatrix)*

*print("")*

*print("Recall is:")*

*print(Recall)*

*print("")*

*print("Precision is:")*

*print(Precision)*

*print("")*

*print("Accuracy is:")*

*print(Accuracy)*

*print("")*

*print("F1 score is:")*

*print(F1Score)*