Lab5: Data classification using K-Nearest Neighbor Classifier

You are given the **Diabetic Retinopathy Debrecen Data Set** as a csv file (`DiabeticRetinipathy.csv`). This dataset contains features extracted from the Messidor image set to predict whether an image contains signs of diabetic retinopathy or not. All features represent either a detected lesion, a descriptive feature of a anatomical part or an image-level descriptor. It consists 1151 tuples each having 20 attributes. The last attribute for every tuple signifies the class label (0 no signs of diabetic retinopathy and 1 signs of diabetic retinopathy). It is a two class problem. Other attributes are input features. For more information refer [1].

Attribute Information:

Attribute1: The binary result of quality assessment. 0 = bad quality 1 = sufficient quality.
Attribute2: The binary result of pre-screening, where 1 indicates severe retinal abnormality and 0 its lack.
Attributes3-8: The results of MA detection. Each feature value stand for the number of MAs found at the confidence levels alpha = 0.5, . . . , 1, respectively.

Attributes9-16: Contain the same information as Attributes 3-8 for exudates. However, as exudates are represented by a set of points rather than the number of pixels constructing the lesions, these features are normalized by dividing the number of lesions with the diameter of the ROI to compensate different image sizes.

Attribute17: The Euclidean distance of the centre of the macula and the centre of the optic disc to provide important information regarding the patient condition. This feature is also normalized with the diameter of the ROI.

Attribute18: The diameter of the optic disc.

Attribute19: The binary result of the AM/FM-based classification.

Attribute20: Class label. 1 = contains signs of diabetic retinopathy, 0 = no signs of diabetic retinopathy.

1. Write a python program to
   a. **Normalize** all the attributes, except class attribute, of `DiabeticRetinipathy.csv` using **min-max** normalization to transform the data in the range [0-1]. Save the file as `DiabeticRetinipathy-Normalised.csv`
   b. **Standardize**, all the attributes, except class attribute, of `DiabeticRetinipathy.csv` using **z-normalization**. Save the file as `DiabeticRetinipathy-Standardised.csv`
2. Split the **data of each class** from `DiabeticRetinipathy.csv` into **train data** and **test data**. Train data contain **70%** of tuples from each of the class and test data contain remaining **30%** of tuples from each class. Save the train data as `DiabeticRetinipathy-train.csv` and save the test data as `DiabeticRetinipathy-test.csv`

a. Classify every test tuple using **K-nearest neighbor (KNN)** method for the different values of K **(1, 3, 5, 7, 9, 11, 13, 15, 17, 21)**. Perform the following analysis :
   i. Find **confusion matrix (**use '*confusion_matrix'***)** for each K.
   ii. Find the **classification accuracy** (You can use '*accuracy_score'*) for each K. Note the value of K for which the accuracy is high.

3. Split the **data of each class** from `DiabeticRetinipathy-Normalised.csv` into **train data** and **test data**. Train data should contain same **70%** of tuples in Question 2 from each of the class and test data contain remaining same **30%** of tuples from each class. Save the train data as `DiabeticRetinipathy-train-normalise.csv` and save the test data as `DiabeticRetinipathy-test-normalise.csv`
   a. Classify every test tuple using **K-nearest neighbor (KNN)** method for the different values of K **(1, 3, 5, 7, 9, 11, 13, 15, 17, 21)**. Perform the following analysis :
      i. Find **confusion matrix (**use '*confusion_matrix'***)** for each K.
      ii. Find the **classification accuracy** (You can use '*accuracy_score'*) for each K. Note the value of K for which the accuracy is high.
4. Split the **data of each class** from `DiabeticRetinipathy-Standardised.csv` into **train data** and **test data**. Train data should contain same **70%** of tuples in Question 2 from each of the class and test data contain remaining same **30%** of tuples from each class. Save the train data as `DiabeticRetinipathy-train-standardise.csv` and save the test data as `DiabeticRetinipathy-test-standardise.csv`
   a. Classify every test tuple using **K-nearest neighbor (KNN)** method for the different values of K **(1, 3, 5, 7, 9, 11, 13, 15, 17, 21)**. Perform the following analysis :
      i. Find **confusion matrix (**use '*confusion_matrix'***)** for each K.
      ii. Find the **classification accuracy** (You can use '*accuracy_score'*) for each K. Note the value of K for which the accuracy is high.

5. Plot and the **classification accuracy vs K**. for each cases (original, normalized and standardized) in a same graph and compare & observe how it is behaving.

6. Why the value of K is considered as **odd** integer?

Note:

1. Note that while splitting the data (original, normalized and standardized) into train and test set, use the same seed value for random split of all 3 cases. This is to ensure that same training and test samples will be there in all 3 cases.

   Sample code:

```
X_train, X_test, X_label_train, X_label_test = train_test_split(X, X_label, test_size=0.3, random_state=42, shuffle=True)
```

Keep the value for `random_state` same for all the cases. This will ensure that for all three cases, same samples will be in train and test set.

2. You can import the `KNeighborsClassifier` class from the `sklearn.neighbors` library
3. You can use the functions `StandardScaler` for standardization and `MinMaxScaler` for min-max normalization in scikit-learn.
4. Refer the slide uploaded in moodle about the performance evaluation to know more about confusion matrix and Accuracy.

Reference:

[1] Balint Antal, Andras Hajdu: An ensemble-based system for automatic screening of diabetic retinopathy, Knowledge-Based Systems 60 (April 2014), 20-27.