Assignment - 3: Report

PCA, LDA and SVM

Group-6

Aaditya Agrawal, 19CS10003
Debanjan Saha, 19CS30014

Dataset

Occupancy Detection Data Set:
https://archive.ics.uci.edu/ml/datasets/Occupancy+Detection+

About the Dataset

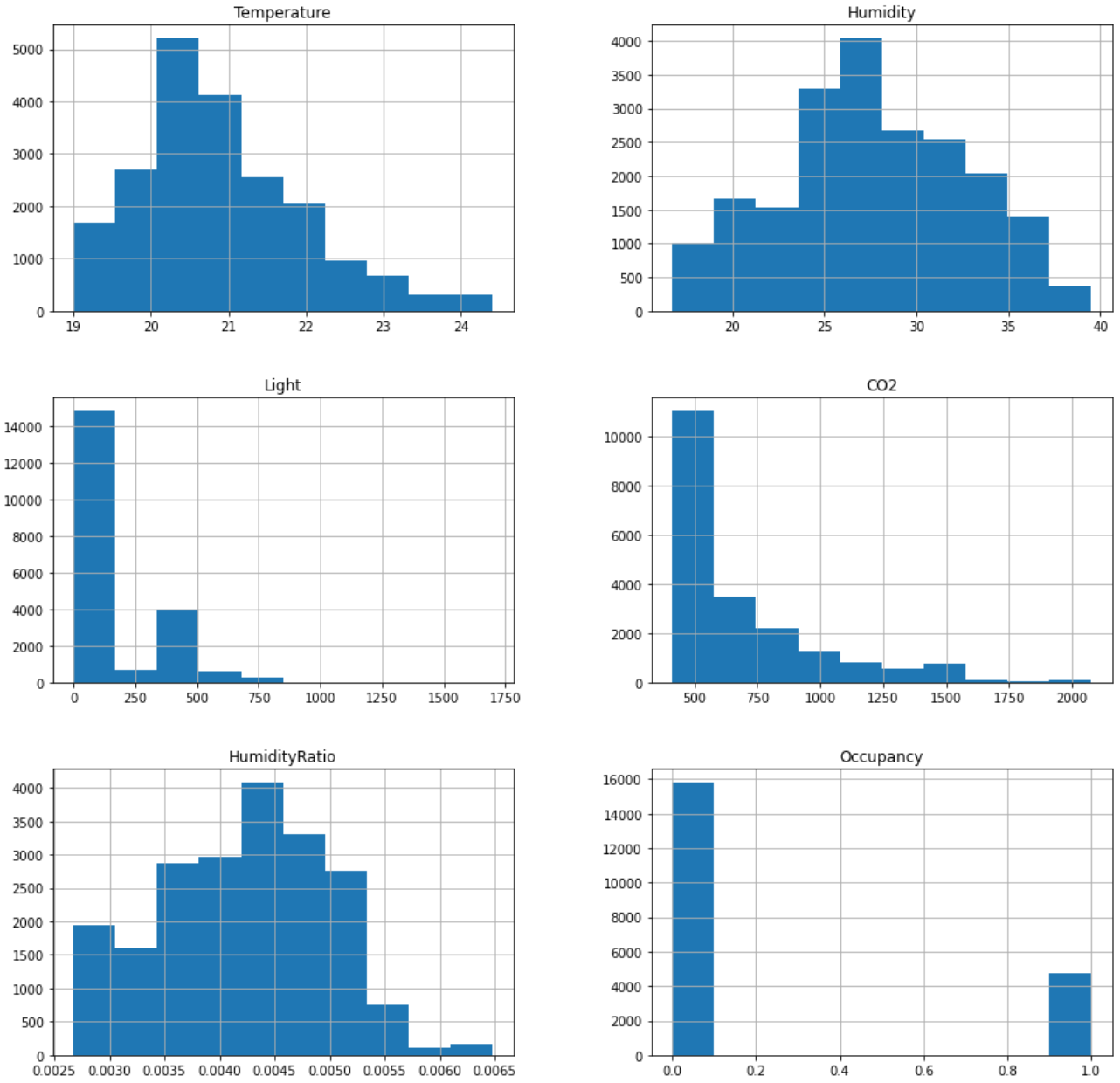
The dataset was obtained from UCI Machine Learning Repository and is prepared from experimental data used for binary classification (room occupancy) from Temperature, Humidity, Light and CO_2. Ground-truth occupancy was obtained from time stamped pictures that were taken every minute.

Analysing the Data

- Before starting the implementation, we analysed the dataset to get an idea about the values and also to check for missing data. Here is a description of our dataset

	Temperature	Humidity	Light	CO2	HumidityRatio	Occupancy
count	20560.000000	20560.000000	20560.000000	20560.000000	20560.000000	20560.000000
mean	20.906212	27.655925	130.756622	690.553276	0.004228	0.231031
std	1.055315	4.982154	210.430875	311.201281	0.000768	0.421503
min	19.000000	16.745000	0.000000	412.750000	0.002674	0.000000
25%	20.200000	24.500000	0.000000	460.000000	0.003719	0.000000
50%	20.700000	27.290000	0.000000	565.416667	0.004292	0.000000
75%	21.525000	31.290000	301.000000	804.666667	0.004832	0.000000
max	24.408333	39.500000	1697.250000	2076.500000	0.006476	1.000000

- The overall dataset has 20560 rows and 7 columns of attributes denoting - date, temperature, humidity, light, CO_2, humidityRatio and the corresponding label class 'Occupancy; to denote whether the room is occupied or not.



- The above two figures give us an idea about the range and distribution of values in the data set. We see that there is an even distribution of data for Temperature, Humidity and HumidityRate. The distribution for Light and CO_2 is skewed to lower values.
- We can also observe the ratio of the room being not-occupied : occupied is also high giving a skewed distribution of the outcome classes
- We then find out the number of unique values that the features take and the number of null values in the dataset. The left figure below shows the unique

entries in the dataset and the figure in the right shows the number of null entries for each attribute.

```
Unique Entries Count:

date             20560
Temperature        485
Humidity          2480
Light             1905
CO2               5167
HumidityRatio     9686
Occupancy            2
dtype: int64
```

```
Null Entries Count:

date             0
Temperature      0
Humidity         0
Light            0
CO2              0
HumidityRatio    0
Occupancy        0
dtype: int64
```

- We observe that Occupancy takes 2 distinct values and so the problem can be thought of as a binary classification problem and therefore we don't need to apply One Hot encoding. Also, we don't have to handle missing data as there are no null values in the columns.

Implementation:

Information about Dependency:

- We relied mostly on the scikit-learn(sklearn) package to implement our assignment and used various methods and modules present in this package to solve different problems assigned to us.

Merging Datasets:

- From the given source, we obtained 3 files containing data divided as datatrain.csv, datatest.csv, datatest2.csv. We merged all the 3 different datasets as discussed in our doubt clearing session and prepared a single dataframe which we used to generate our own split for testing, training and validation sets.

Step 1: Random Splitting into Test, Train, Validation Set:

- We used the method **sklearn.model_selection.train_test_split** which helps us to randomly generate two subsets from a given set and a given set of percentage splitting.

- First, we split the overall dataset **randomly** into the training set and test+validation set following a **70:30** split ratio. Then we further **randomly** split the test+validation set using the split ratio of **20:10** into the test and validation set respectively.

Step 2: Principle Component Analysis and Plot:

- For the principal component analysis part, we have used the class **sklearn.decomposition.PCA** to instantiate an object with **number of components = 2** and fit the projection matrix using the train split.

- Once the training had been done, we generated the reduced dimensions for the test and the validation splits using the projection matrix trained before following as instructed in our assignment specification.

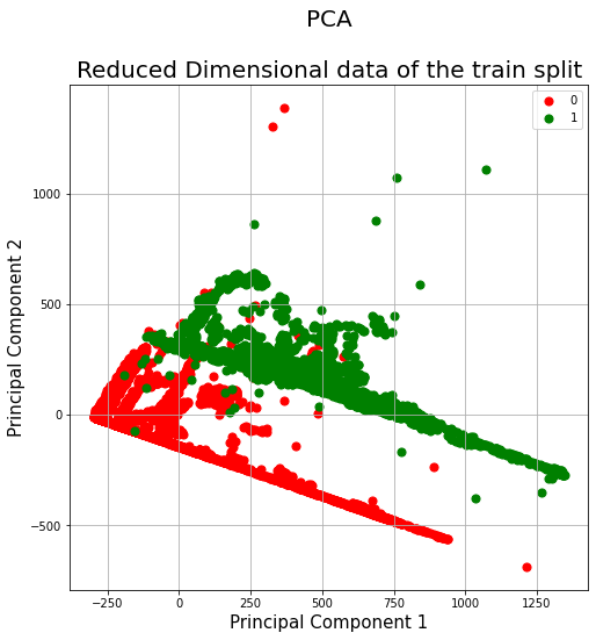
- **Results**

```
# Information Retention
print(f"Explained variation per principal component: {pca.explained_variance_ratio_}")
```

```
Explained variation per principal component: [0.77910549 0.22073638]
```

- After the training, the **explained_variance_ratio** provides us with the amount of information or variance each principal component holds after projecting the data to a lower-dimensional subspace. Here, we can observe that the principal component 1 holds **77.9%** of the information while the principal component 2 holds only **22%** of the information.

- Then we used **matplotlib.pyplot** library to plot the reduced dimension of the train split. Class 0 (Occupancy = False) and Class 1 (Occupancy = True) were plotted using red colour and green colour respectively. The plot is as follows:



Step 3: Training and Evaluating SVM Classifier:

We have defined a few utility functions for this part of the assignment. Those are as follows:

- ### Utility Functions
  - **get_svm_classifier(hyperparams,X_train,y_train)**: This method takes a bunch of parameters as shown and instantiates an object or a classifier of the class **sklearn.svm.SVC** using the **kernel** defined in the **hyperparameters.** Then it trains the classifier using the training data and returns the classifier.

  - **get_accuracy(clf, X_actual, y_actual):** This method takes a **pretrained-classifier** and predicts the possible outcomes using the actual input values(X_actual) and generates the accuracy by comparing it with the actual outcomes and returns the accuracy.

  - **explore_kernels(hyperparams,X_valid,y_valid):** This method takes the data from the validation split and the hyperparameters as its parameters. We define a list of possible kernels inside this method and start iterating through that list. While iterating, we tune the hyperparameters according to current kernel and call the **get_svm_classifier** method to get a trained-classifier corresponding to it. Then, we generate accuracy of each kernel type using the **get_accuracy** method in a tabular manner and finally return the classifier having **best accuracy** in the **validation set**

- ### Results
- We called the method **explore_kernels** once **using the validation set** generated earlier and got all the statistics in a tabular manner as follows:

```
+-----------------------------------------------+
|         Kernel        |        Accuracy       |
+-----------------------------------------------+
|         linear        |   0.9891891891891892  |
+-----------------------------------------------+
|          poly         |   0.9891891891891892  |
+-----------------------------------------------+
|          rbf          |   0.9891891891891892  |
+-----------------------------------------------+
|        sigmoid        |   0.8511056511056511  |
+-----------------------------------------------+
```

- We got the best classifier from the above method to apply on the test set that we generated earlier. Upon evaluation using **get_accuracy** method, the following results were found **(Test Accuracy = 98.83%)** :

```
test_accuracy = get_accuracy(best_clf, X_test_pca, y_test)
print(f"Test Accuracy: {test_accuracy}")
```

```
Test Accuracy: 0.9883861601742076
```

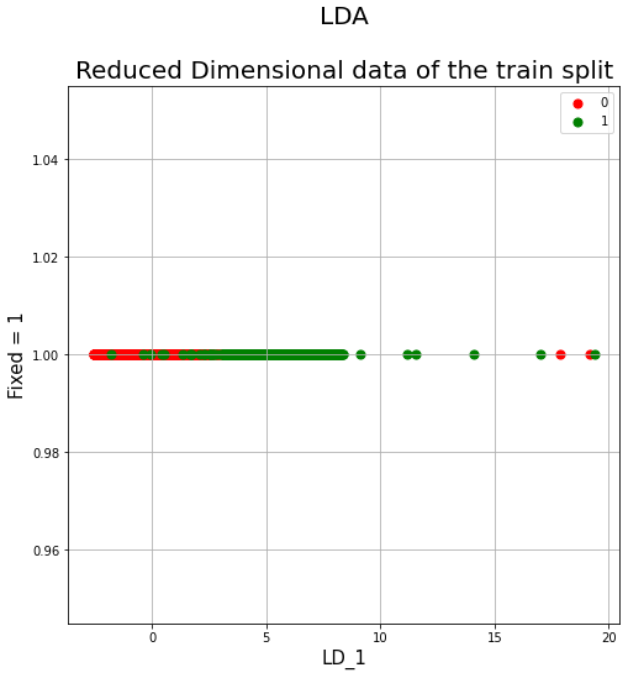
Step 4: Reducing feature dimension to 1-dimension by applying LDA:

- For the linear discriminant analysis part, we have used the class **sklearn.discriminant_analysis.LinearDiscriminantAnalysis** to instantiate an object with **number of components = 1** and fit the projection matrix using the same train split as used in PCA.

- Once the training had been done, we generated the reduced dimensions for the test and the validation splits using the trained projection matrix.

Results

- Then we used **matplotlib.pyplot** library to plot the reduced dimension of the train split. Class 0 (Occupancy = False) and Class 1 (Occupancy = True) were plotted using red colour and green colour respectively. The plot is as follows:



Step-5: Re-applying Step-3(Training SVM) on data obtained from Step-4 (LDA) :

- We called the method **explore_kernels** once again **using the validation set** generated with reduced dimensions obtained from LDA and got all the statistics in a tabular manner as follows:

- **Results**

Kernel	Accuracy
linear	0.9891891891891892
poly	0.9891891891891892
rbf	0.9896805896805897
sigmoid	0.9533169533169533

- We **got the best classifier** from the above method to **apply on the test set** with reduced dimension that we generated earlier using the trained projection matrix of LDA. Upon evaluation using **get_accuracy** method, the following results were found **(Test Accuracy = 98.91%)** :

```
test_accuracy = get_accuracy(best_clf, X_test_lda, y_test)
print(f"Test Accuracy: {test_accuracy}")
```

executed in 18ms, finished 18:25:24 2021-11-14

```
Test Accuracy: 0.9891120251633196
```

Observation after Reapplying SVM on data from LDA

- When we compare the test accuracy obtained in Step-3 and Step-5 , we see that after applying PCA, we obtained a 98.83% test accuracy after evaluation on the SVM classifier. However, after applying LDA, we achieved a test accuracy of 98.91%.

- Although this is not a significant improvement but even **this slight improvement is expected.** Normally, PCA performs better in case where number of samples per class is less whereas LDA works better with large dataset having multiple classes.

- However, if we take a step back and look into the validation set accuracies, we can see that
    - For the “rbf” kernel, the validation set accuracy increased from 98.91% to 98.96%.
    - For the “**sigmoid**” kernel, the validation set accuracy increased from **85.11% to 95.33%.**

- The above improvements (whether slight or significant) can be related to the fact that when we apply PCA on a dataset, it reduces the dimensions of the dataset without considering the outcome or resulting classes. However, LDA is trained using both features and the outcome of those features due to which it can classify with somewhat better accuracy in comparison to PCA.