CS60050
MACHINE LEARNING

Assignment 3

Animesh Jha - 19CS10070

Nisarg Upadhyaya - 19CS30031

1. Code

1. main.py

This file contains code to run the experiments. It has the following methods:

- read_data(): this function reads “datatraining.txt”, “datatest.txt” and “datatest2.txt” and stores them in a dataframe
- train_test_split(): this generates stratified 70:10:20 train test validation splits.
- pca_runs(X_train, X_val, X_test, y_train, y_val, y_test): Runs steps 2 and 3 over given dataset split
- lda_runs(X_train, X_val, X_test, y_train, y_val, y_test): Runs steps 4 and 5 over given dataset split

2. occupancy_data

This folder holds the given dataset and the files

1. “datatraining.txt”
2. “datatest.txt”
3. “datatest2.txt”

Procedure followed

We combine all three data files and then generate 70:10:20 dataset splits. We use the standard PCA, SVC and LDA functions of sklearn library. We iterate over a set of hyperparameters and find the model which performs the best on the validation split. Then we take the best performing model and calculate its accuracy over the test split. We do this for both PCA and LDA. We drop the date feature.

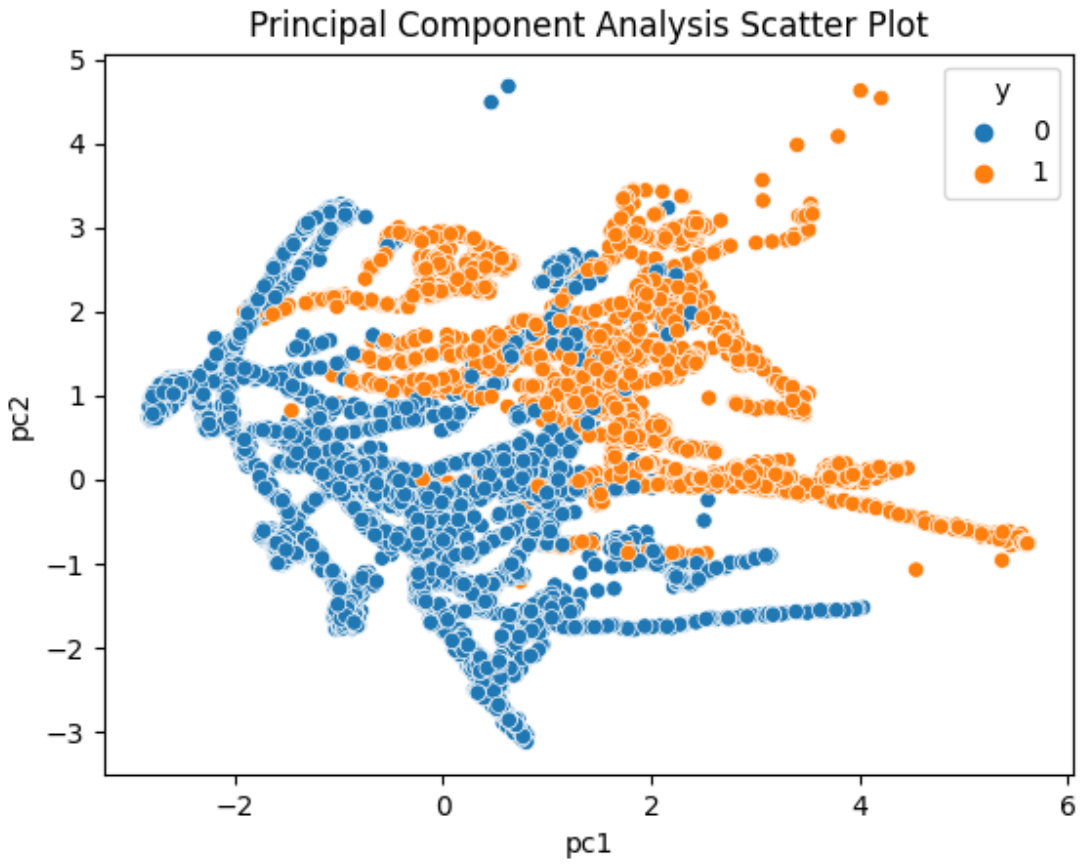
Features used: ['Temperature', 'Humidity', 'Light', 'CO2', 'HumidityRatio']
Label: ‘Occupancy’
X_train shape (14391, 5)
y_train shape (14391,)
X_val shape (2056, 5)
y_val shape (2056,)
X_test shape (4113, 5)
y_test shape (4113,)

How to run the code?

After installing all dependencies just run `python main.py` and enjoy! README has more details.

2. Experiments

1. Principal Component Analysis



	kernel	gamma	class_weight	degree	val_accuracy
0	poly	auto	-	2	0.874513618677043
1	poly	auto	balanced	2	0.828793774319066
2	poly	auto	-	3	0.925583657587549
3	poly	auto	balanced	3	0.92704280155642
4	poly	auto	-	4	0.881322957198444
5	poly	auto	balanced	4	0.875
6	poly	scale	balanced	2	0.828793774319066

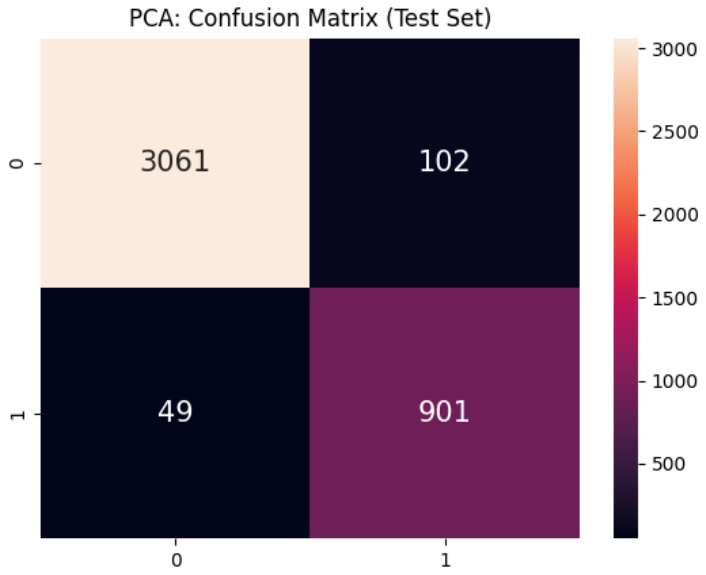
7	poly	scale	-	2	0.874513618677043
8	poly	scale	-	3	0.925583657587549
9	poly	scale	balanced	3	0.92704280155642
10	poly	scale	balanced	4	0.875
11	poly	scale	-	4	0.882295719844358
12	**rbf**	**auto**	**-**	**-**	**0.962062256809338**
13	rbf	auto	balanced	-	0.954280155642023
14	rbf	scale	-	-	0.956225680933852
15	rbf	scale	balanced	-	0.953793774319066
16	sigmoid	auto	balanced	-	0.768482490272374
17	sigmoid	auto	-	-	0.81079766536965
18	sigmoid	scale	balanced	-	0.808852140077821
19	sigmoid	scale	-	-	0.834143968871595
20	linear	-	-	-	0.935311284046693

HyperParameter Tuning Table

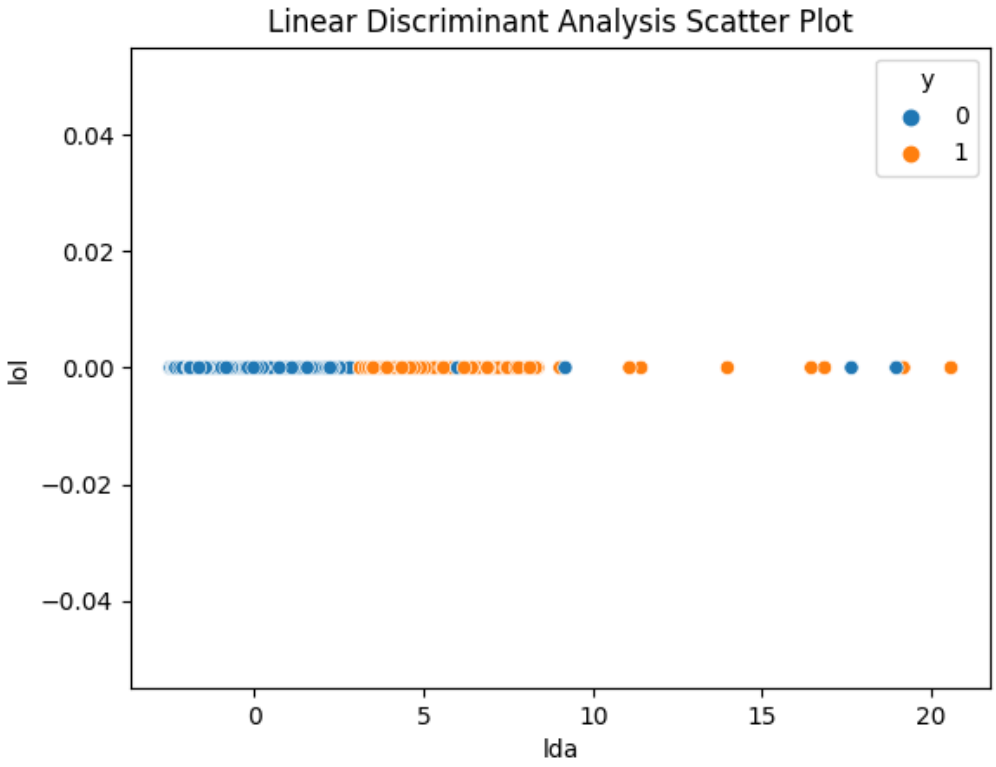
```
Best params on validation set: {'kernel': 'rbf', 'gamma': 'auto', 'class_weight': None}
Classification Report on Test Set for the best model on validation set
              precision    recall  f1-score   support

           0       0.98      0.97      0.98      3163
           1       0.90      0.95      0.92       950

    accuracy                           0.96      4113
   macro avg       0.94      0.96      0.95      4113
weighted avg       0.96      0.96      0.96      4113
```



2. Linear Discriminant Analysis



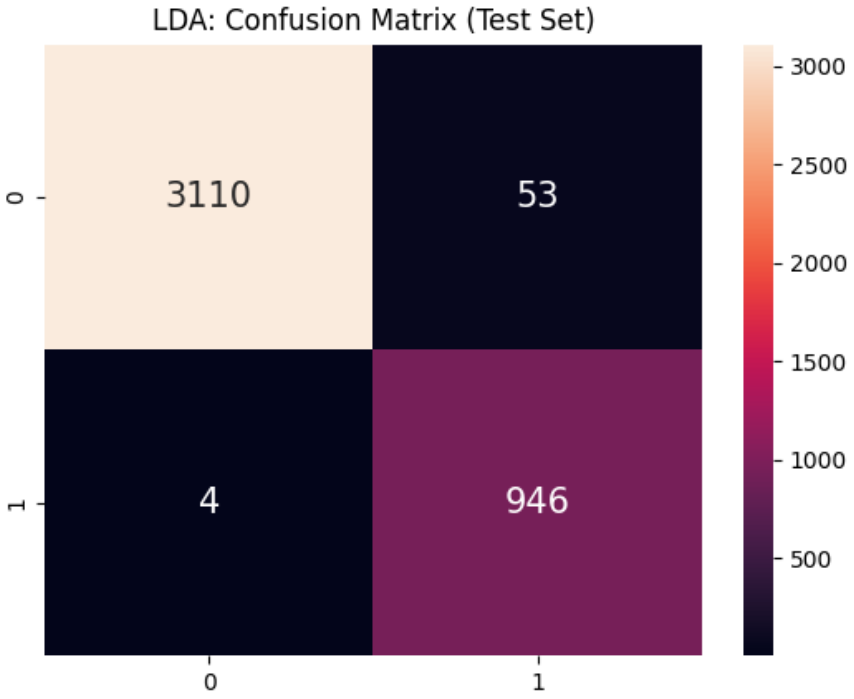
	kernel	gamma	class_weight	degree	val_accuracy
0	poly	auto	balanced	2	0.987354085603113
1	poly	auto	-	2	0.987354085603113
2	poly	auto	-	3	0.987354085603113
3	poly	auto	balanced	3	0.98784046692607
4	poly	scale	balanced	2	0.987354085603113
5	poly	scale	-	2	0.987354085603113
6	poly	scale	-	3	0.987354085603113
7	poly	scale	balanced	3	0.98784046692607
8	rbf	auto	-	-	0.988326848249027
9	rbf	auto	balanced	-	0.988326848249027
10	**rbf**	**scale**	**-**	**-**	**0.988813229571984**
11	rbf	scale	balanced	-	0.988813229571984
12	sigmoid	auto	balanced	-	0.98784046692607
13	sigmoid	auto	-	-	0.987354085603113
14	sigmoid	scale	-	-	0.943579766536965
15	sigmoid	scale	balanced	-	0.957198443579767
16	linear	-	-	-	0.988813229571984

HyperParameter Tuning Table

```
Best params on validation set: {'kernel': 'rbf', 'gamma': 'scale', 'class_weight': None}
Classification Report on Test Set for the best model on validation set
              precision    recall  f1-score   support

           0       1.00      0.98      0.99      3163
           1       0.95      0.99      0.97       950

    accuracy                           0.99      4113
   macro avg       0.97      0.99      0.98      4113
weighted avg       0.99      0.99      0.99      4113
(nlp) animesh@animesh-HP-Pavilion-Notebook-15-bc5xxx:~/Documents/Autumn_2021/ML/CS60050_A
```



3. Analysis

1. Hyperparameters Used

The descriptions are taken from the documentation of SVC from sklearn
We used the following hyperparameters

1. Kernel: Specifies the kernel type to be used in the algorithm. We tried 'linear', 'rbf', 'sigmoid' and 'poly'
2. Gamma: Kernel coefficient for 'rbf', 'poly' and 'sigmoid'.
    a. if 'auto', uses 1 / n_features.
    b. If 'scale' uses 1 / (n_features * X.var())
3. Class_weight: We used this as the dataset is not perfectly balanced
    a. None: all classes are supposed to have weight one
    b. Balanced: weights inversely proportional to class frequencies in the input data as n_samples / (n_classes * np.bincount(y))
4. Degree: Degree of the polynomial kernel function ('poly'). Ignored by all other kernels.

2. Results

We observe that LDA performs better than PCA. LDA is supervised, it uses the values of Y_train while PCA is an unsupervised method and does not use the Y_train values. Further the kernel 'gamma' gives the best performance for both PCA and LDA and surprisingly balanced class_weights does not give better performance. However LDA performs much better on the minority class than PCA, infact much of the difference in the two methods comes from the gain in performance over the minority class. Further for polynomial 3 degree performs much better than other degrees.