Machine Learning (CS60050) - Assignment 3 Report

Support Vector Machine for Occupancy Detection

Group 3

Vanshita Garg - 19CS10064

Ashutosh Kumar Singh - 19CS30008

Introduction and Tasks

In this report, we analyse dimensionality reduction techniques like Principal Component Analysis and Linear Discriminant Analysis and a Support Vector Machine trained on the Occupancy Detection Dataset. The tasks performed with regard to this are:

1. Randomly split the feature matrix into train, validation and test sets in the ratio 70:10:20.
2. Reduce the feature dimension of the above data into a two dimensional feature space using Principle Component Analysis (PCA).
3. Train an SVM classifier on the reduced dimensional data generated from the step 2 by trying different kernel types by varying the appropriate hyperparameters of the classifier.
4. Reduce the feature dimension of the above data into a one dimensional feature space using Linear Discriminant Analysis (LDA).
5. Repeat step 3 on the data obtained from step 4.

Dataset

The Occupancy Detection Dataset contains experimental data used for binary classification of room occupancy. Ground-truth occupancy was obtained from time stamped pictures that were taken every minute. The dataset has the following data fields:

- **Date** - The date and time when the data sample was taken.
- **Temperature** - The temperature of the room (in Celsius).
- **Humidity** - The relative humidity (in %).
- **Light** - The amount of light in the room (in Lux).
- **CO2** - The amount of carbon dioxide in the room (in ppm).
- **Humidity Ratio** - A derived quantity from temperature and relative humidity (in kgwater-vapor/kg-air).
- **Occupancy** - The binary label, 0 for not occupied, 1 for occupied.

The dataset in total has 20,560 training examples. The number of entries for each label (0 and 1) is as follows:

Label	Count	Percentage (%)
0	15810	76.90
1	4750	23.10

Some Important Terms and Definitions

Abbreviations: TP - True Positive, FP - False Positive, TN - True Negative, FN - False Negative.

- $Accuracy = \frac{No.\ of\ examples\ correctly\ classified}{Total\ no.\ of\ examples}$
- $Precision\,(Positive\,Predictive\,Value) = \frac{TP}{TP + FP}$
- $Recall\,(Sensitivity) = \frac{TP}{FN + TP}$
- $Specificity = \frac{TN}{FP + TN}$
- $Negative\,Predictive\,Value = \frac{TN}{TN + FN}$
- $F1\,Score = 2 \times \frac{Precision \times Recall}{Precision + Recall}$
- $Prevalence = \frac{TP + FN}{Total\ no.\ of\ examples}$
- $Detection\,Rate = \frac{TP}{Total\ no.\ of\ examples}$
- $Detection\,Prevalence = \frac{TP + FP}{Total\ no.\ of\ examples}$
- $Balanced\,Accuracy = \frac{Sensitivity + Specificity}{2}$

Procedure and Results

1. Data Preprocessing

Procedure :

- The data is provided in three different files - `datatraining.txt`, `datatest.txt` and `datatest2.txt`. So, we read data from all the three files and then concatenate them into a single dataframe.
- We drop the **Date** column from the dataset as it is uninformative and nearly all the data points have very similar date and time values. Also, it was observed that the accuracy increases if we drop this column.
- As a next step, we split the data into 3 parts - train set, validation set and test set in a ratio of 70:10:20.
- We normalize the data to get a mean of 0 and variance of 1. This step is necessary for carrying out Principal Component Analysis and Linear Discriminant Analysis. The important point to note here is that we normalize the validation and test sets using the mean and variance of the training set. So, we first compute the mean and variance of the training set. Then, we normalize all the data sets - training set, validation set and test set using this same mean and variance. This is done to ensure that there is no data leakage from the validation or test set into the training set. Also, this helps in ensuring that during PCA or LDA, the same transformation is applied to the validation and test sets that was learnt from the training set.

Results :

Shape of the Training Set Array	(14392, 5)
Shape of the Validation Set Array	(2056, 5)
Shape of the Test Set Array	(4112, 5)

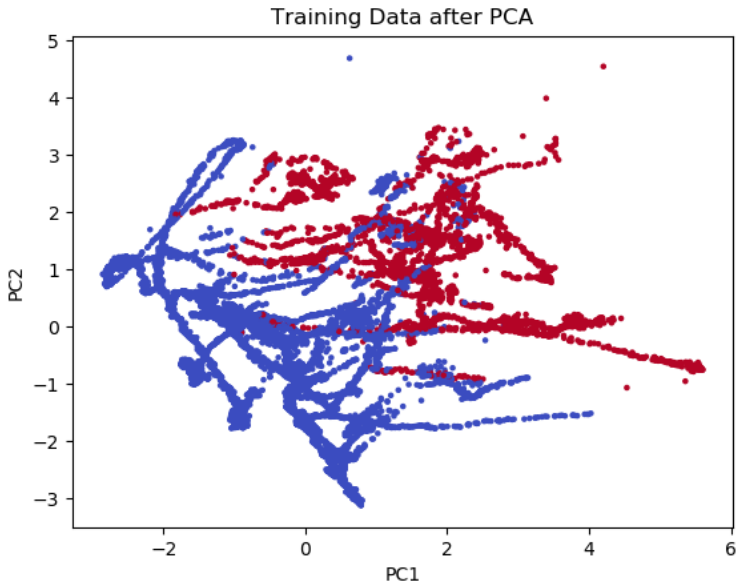
2. Principal Component Analysis

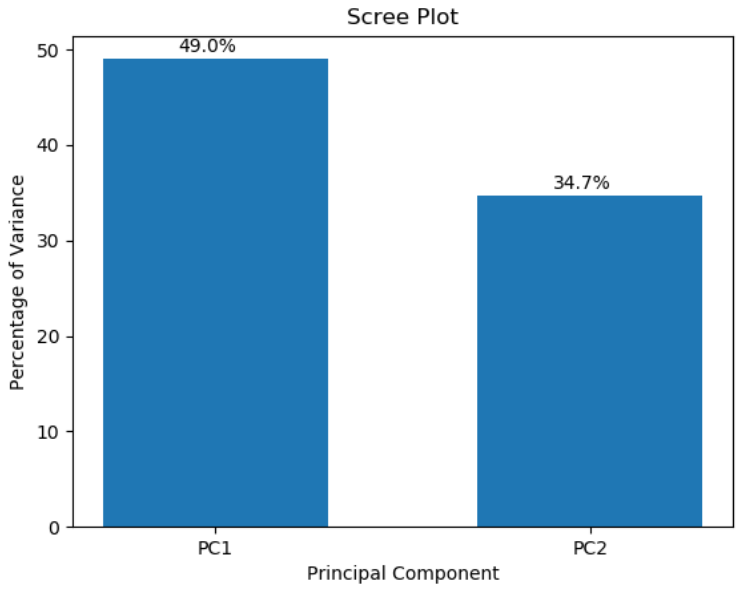
Procedure :

- Our aim is to reduce the feature dimension of the data into a two dimensional feature space using Principal Component Analysis. To achieve this we use `sklearn.decomposition.PCA`
- We learn the projection matrix using only the training set. We then use that matrix to reduce the dimension of the training, validation and test sets.
- Then we plot the reduced dimensional data of the training set in a 2-D plane, and also plot the scree plot for the explained variance percentage.

Results :

Shape of the Training Set after PCA	(14392, 2)
Shape of the Validation Set after PCA	(2056, 2)
Shape of the Test Set after PCA	(4112, 2)





We can see that the two principal components preserve most of the information as they preserve 49.0% + 34.7% = 83.7% of the variance from the original 5 dimensions.

3. Training a Support Vector Machine after PCA

Procedure :

- We want to train a support vector machine on the reduced 2-D data. We use `sklearn.svm.SVC`
- We try different types of kernels and vary their hyperparameters and compute their accuracy on the validation set. Out of these, we choose the SVM with the highest accuracy on the validation set and use that to compute the accuracy on the test set. The kernels and their hyperparameters are as follows:
  - Linear Kernel (`linear`): $\langle x, x' \rangle$, no hyperparameter
  - Radial Basis Function Kernel (`rbf`): $exp(-\gamma \|x - x'\|^2)$, hyperparameter is gamma(γ)
  - Sigmoid Kernel (`sigmoid`): $tanh(\gamma \langle x, x' \rangle)$, hyperparameter is gamma(γ)
  - Polynomia Kernel (`poly`): $(\gamma \langle x, x' \rangle)^d$, hyperparameters are gamma(γ) and degree(d)

Results :

Accuracy on Validation Set for Different Kernels and Hyperparameters

kernel	gamma	degree	accuracy (in %)
linear	-	-	92.4611
rbf	0.001	-	92.3638
rbf	0.01	-	93.0447
rbf	0.1	-	94.8930
rbf	1	-	96.3521
sigmoid	0.001	-	92.3152
sigmoid	0.01	-	92.4611
sigmoid	0.1	-	86.4786
sigmoid	1	-	79.7179
poly	0.001	2	76.3619
poly	0.001	3	76.3619
poly	0.001	4	76.3619
poly	0.01	2	85.0681
poly	0.01	3	82.0039
poly	0.01	4	76.9455
poly	0.1	2	87.0623
poly	0.1	3	92.1206
poly	0.1	4	88.3268
poly	1	2	87.0623
poly	1	3	92.2179
poly	1	4	88.6673

The kernel with highest validation set accuracy comes out to be the `rbf` kernel with gamma = 1.
Test Set Accuracy obtained = 97.0088%

We now show the classification report for the best SVM (rbf kernel with gamma = 1) on the test set.

Confusion Matrix

True \ Pred	0	1
0	3109	94
1	29	880

Classification Statistics

Precision (Positive Predictive Value)	0.9035
Recall (Sensitivity)	0.9681
Specificity	0.9707
Negative Predictive Value	0.9908
F1-Score	0.9347
Prevalence	0.2211
Detection Rate	0.2140
Detection Prevalence	0.2369
Balanced Accuracy	0.9694

Overall Statistics

Accuracy	97.0088%
95% Confidence Interval	(96.4881%, 97.5294%)

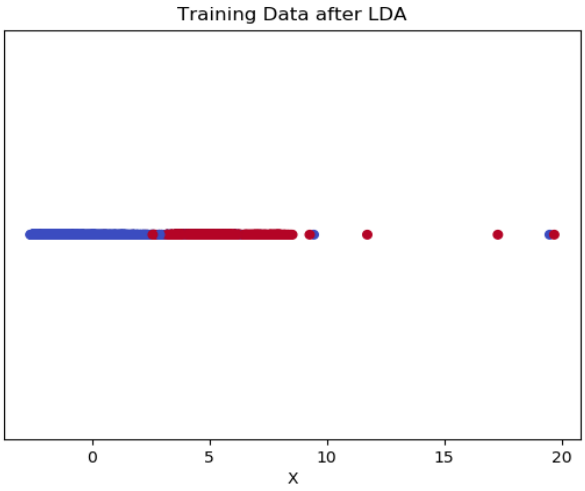
4. Linear Discriminant Analysis

Procedure :

- Now we want to reduce the feature dimension of the data into a one dimensional feature space using Linear Discriminant Analysis.
  For this we use `sklearn.discriminant_analysis.LinearDiscriminantAnalysis`
- We learn the projection matrix using only the training set. We then use that matrix to reduce the dimension of the training, validation and test sets.
- Then we plot the reduced dimensional data of the training set.

Results :

Shape of the Training Set after LDA	(14392, 1)
Shape of the Validation Set after LDA	(2056, 1)
Shape of the Test Set after LDA	(4112, 1)



3. Training a Support Vector Machine after LDA

Procedure :

- We want to train a support vector machine on the reduced 1-D data. We use `sklearn.svm.SVC`
- We again try different types of kernels and vary their hyperparameters and compute their accuracy on the validation set as done in part 3. Out of these, we choose the SVM with the highest accuracy on the validation set and use that to compute the accuracy on the test set.

Results :

Accuracy on Validation Set for Different Kernels and Hyperparameters

kernel	gamma	degree	accuracy (in %)
linear	-	-	98.7354
rbf	0.001	-	98.6868
rbf	0.01	-	98.6868
rbf	0.1	-	98.4922
rbf	1	-	98.4922
sigmoid	0.001	-	98.5895
sigmoid	0.01	-	98.4922
sigmoid	0.1	-	94.3093
sigmoid	1	-	98.4922
poly	0.001	2	84.6790
poly	0.001	3	76.3619
poly	0.001	4	76.3619
poly	0.01	2	98.4922
poly	0.01	3	98.4922
poly	0.01	4	94.0661
poly	0.1	2	98.6381
poly	0.1	3	98.6868
poly	0.1	4	98.1031
poly	1	2	98.6381
poly	1	3	98.6868
poly	1	4	97.4142

The kernel with highest validation set accuracy comes out to be the `linear` kernel.
Test Set Accuracy obtained = 98.8327%

We now show the classification report for the best SVM (linear kernel) on the test set.

Confusion Matrix

True \ Pred	0	1
0	3156	47
1	1	908

Classification Statistics

Precision (Positive Predictive Value)	0.9508
Recall (Sensitivity)	0.9989
Specificity	0.9853
Negative Predictive Value	0.9997
F1-Score	0.9742
Prevalence	0.2211
Detection Rate	0.2208
Detection Prevalence	0.2322
Balanced Accuracy	0.9921

Overall Statistics

Accuracy	98.8327%
95% Confidence Interval	(98.5044%, 99.1610%)

Analysis of Results

- After Principal Component Analysis, using the best SVM, we get a test accuracy of 97.0088%, and after Linear Discriminant Analysis, using the best SVM, we get a test accuracy of 98.8327%. Although the difference in accuracy is not very high, yet we have an increase of nearly 2% in the accuracy on using LDA as compared to PCA. This can be attributed to the fact that although PCA captures the direction of maximum variance for a dataset, but it does not capture the direction of maximum separation between the groups of data points of differing labels. However, LDA projects the data points along a line of maximum separation between the two labels, and this happens because of the fact that PCA is an unsupervised technique whereas LDA is a supervised technique.
- The data points in the 2-D plane after PCA are separated into 2 regions, but they have some overlap, hence a linear or polynomial kernel does not provide the best results. Instead, we get the best SVM using the radial (rbf) kernel.
- We can see from the plot of the data points after LDA that they are almost entirely linearly separable, and this explains the fact that we get the best kernel for the SVM to be the linear kernel.

Major Functions Used in the Code

1. Data Processing

- `read_data` : Reads the data from the file and returns the data as numpy arrays.
- `split_data` : Splits the data into training, validation and test sets.
- `normalize` : Normalizes the data to have zero mean and unit variance.

2. Dimensionality Reduction

- `perform_PCA` : Performs PCA on the training set and then applies the same transformation vector on the validation and test set data.
- `perform_LDA` : Performs LDA on the training set and then applies the same transformation vector on the validation and test set data.
- `plot_PCA` : Plots the two principal components of the training data after PCA.
- `scree_plot_pca` : Plots the scree plot of the PCA object.
- `plot_LDA` : Plots the training data after LDA.

3. SVM

- `choose_best_SVM` : Chooses the best SVM after comparing SVMs with differnt kernels and hyper-parameters.

4. Metric Functions

- `accuracy` : Calculates the accuracy given the predictions and true labels.
- `precision` : Calculates the precision (positive predictive value) value based on predictions and true labels.
- `recall` : Calculates the recall (sensitivity) value based on predictions and true labels.
- `specificity` : Calculates the specificity value based on predictions and true labels.
- `negative_predictive_value` : Calculates the negative predictive value value based on predictions and true labels.
- `f1_score` : Calculates the F1-score based on predictions and true labels. It is the harmonic mean of the precision and recall.
- `prevalence` : Calculates the prevalence value based on predictions and true labels. Prevalence is the proportion of all positives in the total number of observations.
- `detection_rate` : Calculates the detection rate value based on predictions and true labels. Detection rate is the proportion of true positives in the total number of observations.
- `detection_prevalence` : Calculates the detection prevalence value based on predictions and true labels. Detection prevalence is the number of positive class predictions made as a proportion of all predictions.
- `balanced_accuracy` : Calculates the balanced accuracy value based on predictions and true labels. Balanced accuracy is the average of the sensitivity and the specificity.
- `confusion_matrix` : Calculates and returns the confusion matrix.

- `ci_95` : Calculates the 95% confidence interval of accuracy.
- `display_metrics` : Displays all the required metrics in a formatted manner.

Files Present

- `src` directory:
  - `requirements.txt` - Contains all the necessary dependencies with their versions.
  - `data_processing.py` - Contains necessary functions to process the data.
  - `feature_extraction.py` - Contains the funstions for principal component analysis and linear discriminant analysis.
  - `svm.py` - Contains the code to choose the best SVM.
  - `metrics.py` - Contains the functions to evaluate the performance of our classification.
  - `main.py` - Main file for completing all tasks required.
- `occupancy_data` directory:
  - `datatraining.txt` - Dataset file 1.
  - `datatest.txt` - Dataset file 2.
  - `datatest_2.txt` - Dataset file 3.
- `plots` directory:
  - `pca.png` - Reduced 2-D data for the train split after PCA.
  - `lda.png` - Reduced 1-D data for the train split after LDA.
  - `pca_scree.png` - Scree plot for PCA.
- `output.txt` - The output for a run of the entire code.
- `ML_Assn_3_Report.pdf` - A report containing the step-wise description of the implementation and analysis of results.
- `README.md` - A README file describing the files present and the instructions to execute the code.

Instructions to Execute the Code

- Navigate to the `src` directory.
- Ensure you are using a latest version of Python3, and install all dependencies.
  `pip install -r requirements.txt`
- Execute the file `main.py`
  `python main.py`
- The relevant output will be displayed on the terminal or console.
- The plots will be created and saved in the `plots` directory.