2EL1730: MACHINE LEARNING

CENTRALESUPÉLEC

Lab 2: Naive Bayes classifier and Linear Discriminant Analysis

Instructors: Nora Ouzir and Maria Vakalopoulou
TAs: Theodore Aouad, Hakim Benkirane, Aaron Mamann, Lily Monnier

December 6, 2022

1 Description

In this lab, we will continue studying classification techniques, and in particular we will focus on the *Linear Discriminant Analysis* algorithm. Initially, we will discuss a question on *Naive Bayes classifier*, and then we discuss the basic characteristics of the LDA algorithm and how it can be applied to a real classification problem.

2 Naive Bayes classifier

Consider the problem of binary classification using the Naive Bayes classifier. You are given two dimensional features (X_1, X_2) and the categorical class conditional distributions in the tables below. The entries in the tables correspond to $P(X_1 = x_1|C_i)$ and $P(X_2 = x_2|C_i)$ respectively. The two classes are *equally likely*.

$X_1 =$ \ Class	C_1	C_2
-1	0.2	0.3
0	0.4	0.6
1	0.4	0.1

$X_2 =$ \ Class	C_1	C_2
-1	0.4	0.1
0	0.5	0.3
1	0.1	0.6

Given a data point $(-1, 1)$, calculate the following posterior probabilities:

(a) $P(C_1|X_1 = -1, X_2 = 1)$

(b) $P(C_2|X_1 = -1, X_2 = 1)$

3 Linear Discriminant Analysis

Linear Discriminant Analysis (LDA) is a method used in statistics, pattern recognition and machine learning to find a linear combination of features which characterizes or separates two or more classes of objects or events. The resulting combination can be used as a linear classifier or as a dimensionality

reduction method. More precisely, one way to view a linear classification model is in terms of dimensionality reduction. Let's consider the case of two classes, $K = 2$, and suppose that we take the d-dimensional input vector $\mathbf{x}$ and project it down to one dimension using $y(\mathbf{x}) = \mathbf{w}^T\mathbf{x} + w_0$. Then, setting a threshold on $y(\mathbf{x})$, we can classify $\mathbf{x}$ to the first class if $y(\mathbf{x}) \geq 0$, and to the second class otherwise. In general, the projection onto one dimension leads to a considerable loss of information, and classes that are well separated in the original d-dimensional space may become strongly overlapping in one dimension. However, by adjusting the components of the weight vector $\mathbf{w}$, we can select a projection that maximizes the class separation.

LDA is also closely related to Principal Component Analysis (PCA). Both methods look for linear combinations of variables which best explain the data. LDA explicitly attempts to model the difference between the classes of the data, while PCA does not take into account any difference in class labels. Figure 1 illustrates the difference between PCA and LDA. In PCA we process the data as a whole and do not consider any division into classes. The axes are optimal for representing the data as they indicate where the maximum variation actually lies. The LDA axis is optimal for distinguishing between the different classes. In general, the number of axes that can be computed by the LDA method is one less than the number of classes in the problem. In the figure, the ϕ_1 axis is less effective for separating the classes than the LDA axis v. The ϕ_2 axis is clearly useless for classification.

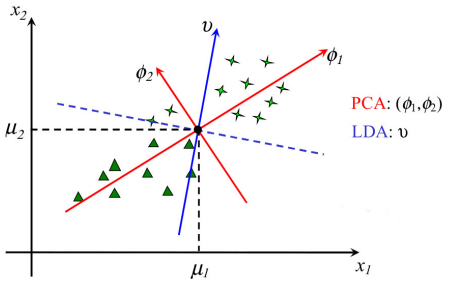

Figure 1: LDA vs. PCA.

LDA is a classical statistical approach for supervised dimensionality reduction and classification. LDA computes an optimal transformation (projection) by minimizing the within-class distance and maximizing the between-class distance simultaneously – thus achieving maximum class discrimination. The optimal transformation in LDA can readily be computed by applying an eigendecomposition on the so-called scatter matrices.

Next we describe the process of computing the LDA axes in the general case, where the number of classes is $K > 2$. The classes are denoted as $C_1, C_2, \ldots, C_K$. Let $\mathbf{x} \in \mathbb{R}^d$ be an input vector (i.e., instance of our dataset), and let n be the total number of instances (i.e., the input matrix $\mathbf{X}$ has dimensions $n \times d$). We introduce $d' > 1$ linear features $y_k = \mathbf{w}^T\mathbf{x}$, where $k = 1, 2, \ldots, d'$. These features can be grouped to form a vector $\mathbf{y}$. In a similar way, the weight vectors $\mathbf{w}_k$ can be considered to be the columns of a matrix $\mathbf{W}$:

$$\mathbf{y} = \mathbf{W}^T\mathbf{x}. \tag{1}$$

The goal is to find matrix $\mathbf{W}$. The first step of LDA is to find two scatter matrices, referred to as *between class* and *within class* scatter matrices. The within class matrix $\mathbf{S}_w$ of size $d \times d$ is defined as the within-class *covariance matrix* and is considered for all the K classes in the dataset as follows:

$$\mathbf{S}_w = \sum_{j=1}^{K} \sum_{\mathbf{x} \in C_j} (\mathbf{x} - \mathbf{m}_j)(\mathbf{x} - \mathbf{m}_j)^T, \tag{2}$$

where $\mathbf{m}_j = \dfrac{1}{n_j} \sum_{\mathbf{x} \in C_j} \mathbf{x}, j = 1, 2, \ldots, K$ is the mean vector (centroid) of the j-th class (n_j is the number of instances in class j). Note that the outer sum is for the different classes, while the inner sum is for the instances of each class and is equal to the covariance matrix per class.

The between class scatter matrix $\mathbf{S}_b$ of size $d \times d$ is defined as follows:

$$\mathbf{S}_b = \sum_{j=1}^{K} n_j (\mathbf{m}_j - \mathbf{m})(\mathbf{m}_j - \mathbf{m})^T, \tag{3}$$

where $\mathbf{m} = \frac{1}{n} \sum_{\mathbf{x} \in \mathbf{X}} \mathbf{x}$ is the mean of the total data. A different way to see the between class scatter matrix is by considering the total covariance matrix of the data, defined as:

$$\mathbf{S}_t = \sum_{\mathbf{x} \in \mathbf{X}} (\mathbf{x} - \mathbf{m})(\mathbf{x} - \mathbf{m})^T, \tag{4}$$

where we sum up over all data instances. The total covariance $\mathbf{S}_t$ can be decomposed into the sum of the within class covariance matrix, plus the between class covariance matrix as $\mathbf{S}_t = \mathbf{S}_w + \mathbf{S}_b$.

The main objective of LDA is to find a projection matrix $\mathbf{W}$ that minimizes the within class separability while simultaneously maximizes the between class separability. This forms the following optimization problem:

$$\mathbf{W}_{LDA} = \arg\max_{W} \frac{|\mathbf{W}^T \mathbf{S}_b \mathbf{W}|}{|\mathbf{W}^T \mathbf{S}_w \mathbf{W}|}. \tag{5}$$

This ratio is known as Fisher's criterion. It is known that the solution to the optimization problem in Eq. (5) can be obtained by solving the following generalized eigenvalue problem:

$$\mathbf{S}_b \mathbf{w}_j = \lambda \mathbf{S}_w \mathbf{w}_j \Leftrightarrow \mathbf{S}_w^{-1} \mathbf{S}_b \mathbf{w}_j = \lambda \mathbf{w}_j, \quad j = 1, \ldots, K-1, \tag{6}$$

where $\mathbf{w}_j$ are vectors that correspond to columns of the projection matrix $\mathbf{W}$, i.e., $\mathbf{W} = [\mathbf{w}_1 | \mathbf{w}_2 | \ldots | \mathbf{w}_{K-1}]$. Therefore, the projection matrix $\mathbf{W}$ can be obtained by the eigenvectors that correspond to the $K-1$ largest eigenvalues of the $\mathbf{S}_w^{-1} \mathbf{S}_b$ matrix. Additionally, the new representation of the data $\mathbf{X}_{LDA}$ can be obtained by projecting the data $\mathbf{X}$ to the new space defined by $\mathbf{W}$ (i.e., dot product of $\mathbf{X}$ and $\mathbf{W}$).

Algorithm 1 provides the pseudocode of the LDA method.

Algorithm 1 Linear Discriminant Analysis (LDA)

Input: Training data $\mathbf{X} = \{\mathbf{x_1}, \mathbf{x_2}, \ldots, \mathbf{x_n}\}$, where $\mathbf{x_i} = (x_1, x_2, \ldots, x_d)$, $i = 1, \ldots, n$ and their class labels $\mathbf{Y}$

Output: Projected data $\mathbf{X}_{LDA}$ (of dimension $n \times (K-1)$)

1: Compute the within class scatter matrix as shown in Eq. 2
2: Compute the between class scatter matrix as shown in Eq. 3
3: Compute matrix $\mathbf{W}$ solving the eigenvalue problem of Eq. 6 and getting the eigenvectors that correspond to the $K-1$ largest eigenvalues (K is the number of classes)
4: Project the data to the new space defined by $\mathbf{W}$ and get $\mathbf{X}_{LDA}$
5: Project also the mean vectors of each class $\mathbf{m}_j, j = 1, \ldots, K$ to the new space (this will be used later for classification) and get $\mathbf{m}_j^{LDA}, j = 1, \ldots, K$

How to perform classification?

Algorithm 1 performs dimensionality reduction using the LDA method. To perform classification of a new data instance $\mathbf{x}$ to one of the possible classes $C_j, j = 1, \ldots, K$, a similar approach is followed. The new instance $\mathbf{x}$ is projected into the new space defined by matrix $\mathbf{W}$ ($\mathbf{x}_{LDA}$); then, it is assigned to closest class as defined by the Euclidean distance of $\mathbf{x}_{LDA}$ to the centroid vectors (mean) of each class C_j (as computed at step 5 of Algorithm 1).

3.1 Pipeline of the Task

Next, we briefly describe the pipeline that will be followed in the lab. The goal is to implement LDA and apply it on the *Wine* dataset, that corresponds to the chemical analysis of wines grown in the same region in Italy but derived from three different cultivars (three different types of wine; thus, three classes). This dataset is composed by 178 observations, describing the quantities of 13 constituents found in each of it. Thus, the size of the dataset is 178×13.

The pipeline of the task is described in the `LDA/main.py` Python script. Initially, we load the data and split the dataset into training and test part (after shuffling the data in order to have instances from all classes to the training and test parts). (We should note here that the most appropriate way to split the dataset into training and test sets, is by applying cross-validation).

```
# Load data
my_data = np.genfromtxt('wine_data.csv', delimiter=',')
np.random.shuffle(my_data)
trainingData = my_data[:100,1:] # training data
trainingLabels = my_data[:100,0] # class labels of training data

testData = my_data[101:,1:] # training data
testLabels = my_data[101:,0] # class labels of training data
```

After that, we apply LDA to project the data to the new space. This part is implemented by the `my_LDA()` function in the `LDA/my_LDA.py` file, that performs the tasks described in Algorithm 1. The function returns the projection matrix $\mathbf{W}$, the centroid vectors $\mathbf{m}_j^{LDA}, j = 1, \ldots, K$ of each class and the projected data $\mathbf{X}_{LDA}$.

```
# Training the LDA classifier
W, projected_centroid, X_lda = my_LDA(trainingData, trainingLabels)
```

Then, we use the projection matrix $\mathbf{W}$ and the centroid vectors of each class to perform classification of the test dataset. The process that is followed is the one described above and is implemented by the `predict()` function in the `LDA/predict.py` file, which returns the class labels of the new instances (we increment their value by one since the original classes are $1, 2$ and 3, while the returned values are $0, 1, 2$).

```
# Perform predictions for the test data
predictedLabels = predict(testImages, projected_centroid, W)
predictedLabels = predictedLabels+1
```

3.2 Tasks to be Performed

Your task is to implement the functions `my_LDA()` and `predict()`, as described above. Then, the accuracy of the LDA classifier can be computed for the *Wine* dataset.

References

[1] Christopher M. Bishop. "Pattern Recognition and Machine Learning". Springer-Verlag New York, Inc., 2006.

[2] Tom M. Mitchell. "Machine learning". Burr Ridge, IL: McGraw Hill 45, 1997.