INF582: Data Science – Learning from Data

École Polytechnique

Lab 4: Supervised Learning II

Linear Discriminant Analysis and Logistic Regression

Jean-Baptiste Bordes, Fragkiskos Malliaros, Nikolaos Tziortziotis and Michalis Vazirgiannis

January 8, 2016

1 Description

In this lab, we will continue studying supervised learning techniques and we will focus on the following two classification algorithms: (i) *Linear Discriminant Analysis* and (ii) *Logistic Regression*. Initially, we discuss the basic characteristics of each algorithm and then we examine how the algorithms can be applied on real classification problems.

2 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 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 class two 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 data, while PCA does not take into account any difference in class. 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

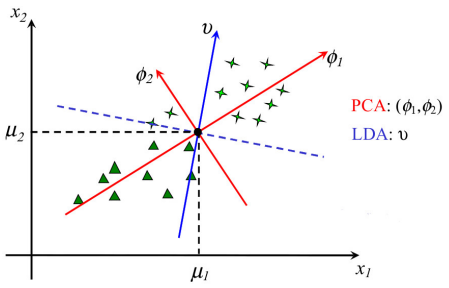

Figure 1: LDA vs. PCA.

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.

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 be readily 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 $\mathbf{m} = \dfrac{1}{n} \sum_{\mathbf{x} \in \mathbf{X}} \mathbf{x}$ 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. To get an intuition of what this means, note that the determinant of the co-variance matrix tells us how much variance a class has. For example, for the co-variance matrix in the PCA (diagonal) projection, the value of the determinant is just the product of the diagonal

elements which are the individual variable variances. The determinant has the same value under any ortho-normal projection. So Fisher's criterion tries to find the projection that maximizes the variance of the class means and minimizes the variance of the individual classes.

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 provides the dimensionality reduction obtained by 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 is projected into the new space defined by matrix $\mathbf{W}$, and then it is assigned to closest class as defined by the Euclidean distance of $\mathbf{x}_{LDA}$ to the centroid vectors (mean) $\mathbf{m}_j^{LDA}$ of each class C_j (as computed at step 5 of Algorithm 1).

2.1 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 in the `LDA/main.py` Python script. Initially we load the data and split the dataset to the training and test parts (after shuffling the data to have instances from all classes to the training and test parts).

```
# 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 = myLDA(trainingData, trainingLabels)
```

Then, we use the projection matrix $\mathbf{W}$ and the centroid vectors of each class to perform classification of the test data. The process that is followed is the one described above and implemented in the `predict()` function in the `LDA/predict.py` file that 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
```

2.2 Tasks to be Performed

The goal of this part 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.

3 Logistic Regression

Logistic regression is a discriminative probabilistic statistical classification model that can be used to predict the probability of occurrence of an event. It is a supervised learning algorithm that can be applied to binary or multinomial classification problems. Similar to the case of linear regression, each feature has a coefficient θ (i.e., a weight), that captures the contribution of the feature to the variable y (recall that in linear regression with only one feature x, we want to predict the value y of a new instance according to $y = \theta_1 x + \theta_0$; the goal is to learn parameters θ_0 and θ_1 from the training data). In the case of two-class logistic regression, variable y does not take continues values (as in linear regression), but actually corresponds to the class values, namely 0 or 1.
That way, the probability that a new instance belongs to class 0 is given by: $p(Y = 0|x) = \theta_1 x + \theta_0$. In other words, the goal of logistic regression is to predict the probability that a new instance belongs to class 0 or 1. This is the main reason why the probability $p(x)$ cannot be considered as the variable y in a linear regression problem (the value should be in the $[0, 1]$ range).

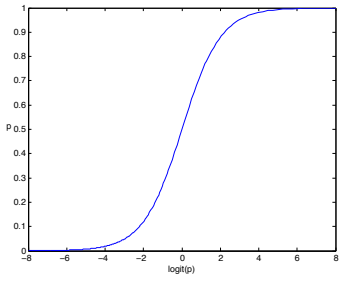

Figure 2: Logistic function.

To overcome this problem, we apply the *logistic transformation* of probability $p(x)$. In other words, we replace probability p with $\text{logit}(p) = \log\left(\frac{p}{1-p}\right)$[1]. While probability p can take values from 0 to 1, $\text{logit}(p)$ ranges

[1]Wikipedias lemma for *Logit*: `http://en.wikipedia.org/wiki/Logit`.

from $-\infty$ to $+\infty$. Figure 2 shows an example of the logistic function. Observe that the function is symmetric around 0.5, where the function takes value 0. In our case, the logistic function is useful because it can take an input with any value from negative to positive infinity, whereas the output always takes values between zero and one and hence is interpretable as a probability. That way, the logistic regression model can be expressed as:

$$\log \frac{p(x)}{1-p(x)} = \theta_0 + \theta_1 x. \tag{7}$$

Solving for p we have:

$$p(x) = \frac{e^{\theta_0+\theta_1 x}}{1+e^{\theta_0+\theta_1 x}} = \frac{1}{1+e^{-(\theta_0+\theta_1 x)}}. \tag{8}$$

In the case of a binary classification problem (i.e., two classes), we should predict class $Y = 1$ with probability $p \geq 0.5$ and class $Y = 0$ with probability $p < 0.5$. This means guessing class 1 whenever $\theta_1 x + \theta_0$ is non-negative, and 0 otherwise. So logistic regression gives us a *linear classifier*. The decision boundary separating the two predicted classes is the solution of $\theta_1 x + \theta_0 = 0$, which is a point if x is one dimensional, a line if it two dimensional, a plane in the case of three features, etc. Logistic regression not only says where the boundary between the classes is, but also says (via Eq. (8)) that the class probabilities depend on the distance from the boundary, in a particular way, and that they go towards the extremes (0 and 1) more rapidly when $\|\theta\|$ is larger.

In the case where we have more than one features, the logistic regression model can be expressed as follows:

$$\log \frac{p(x)}{1-p(x)} = \theta_0 + \theta_1 x_1 + \theta_2 x_2 + \ldots + \theta_n x_n. \tag{9}$$

Solving for p we have:

$$p(x) = \frac{e^{\theta_0+\theta_1 x_1+\theta_2 x_2+\ldots+\theta_n x_n}}{1+e^{\theta_0+\theta_1 x_1+\theta_2 x_2+\ldots+\theta_n x_n}} = \frac{1}{1+e^{-(\theta_0+\theta_1 x_1+\theta_2 x_2+\ldots+\theta_n x_n)}}. \tag{10}$$

Logistic regression learns the weights θ_i so as to maximize the likelihood of the data. The likelihood function can be expressed in terms of the Bernoulli distribution:

$$p(Y|\theta) = \prod_{i=1}^{m} p(x_i)^{y_i}(1-p(x_i))^{1-y_i}, \tag{11}$$

where $p(x_i)$ the predicted probability that x_i belongs to the first class (as shown in Eq. 10), y_i the class label of instance i and m the number of instances. Based on this, we can define an error (or cost) function by taking the negative logarithm of the likelihood and applying the product rule for logarithms:

$$J(\theta) = -\frac{1}{m}\sum_{i=1}^{m}\big[y_i \log\big(\frac{1}{1+e^{-(\theta_0^i+\theta_1^i x_1+\theta_2^i x_2+\ldots+\theta_n^i x_n)}}\big) + \big(1-y_i\big)\log\big(1-\frac{1}{1+e^{-(\theta_0^i+\theta_1^i x_1+\theta_2^i x_2+\ldots+\theta_n^i x_n)}}\big)\big], \tag{12}$$

Thus, logistic regression aims to find parameters θ in order to minimize the above error (or cost) function. This can be achieved taking the gradient of the cost function with respect to parameters $\theta_0, \theta_1, \ldots, \theta_n$ and applying the gradient descent method:

$$\nabla J(\theta) = \frac{\partial J(\theta)}{\partial \theta_j} = \frac{1}{m}\sum_{i=1}^{m}\big(\frac{1}{1+e^{-(\theta_0^i+\theta_1^i x_1+\theta_2^i x_2+\ldots+\theta_n^i x_n)}} - y_i\big)x_j^i. \tag{13}$$

3.1 Pipeline of the Task

The goal of this part is to implement the logistic regression method and use it to find the decision boundary in a classification problem. The problem is to decide the probability that a student will be admitted to the Master's program of a university based on the grades of two courses.

The idea is that we already have available the grades for students of previous years, as well as the decision (accepted or not). This will be the training data for the logistic regression classifier. Figure 3 depicts the training data. The axes correspond to the two courses. The blue dots show the grades of the students that were admitted to the Master's program, while the red $\times$ to the students that were rejected. Next, we will apply logistic regression to design a model that predicts the acceptance probability for a student.

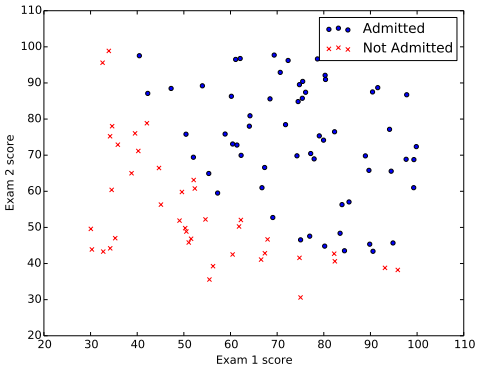

Figure 3: Training data.

The pipeline of the task is in the `logistic/main.py` script file. Initially, we load the data contained in the `data1.txt` file. The first two columns correspond to the two features of the dataset (grades on two courses), while the third one to the class label (admitted or not). We can also plot the data, as shown in Fig. 3.

```
# Load the dataset
# The first two columns contains the exam scores and the third column
# contains the label.
data = loadtxt('data1.txt', delimiter=',')

X = data[:, 0:2]
y = data[:, 2]
```

Then, we proceed with the main task. We initialize parameter θ with zero and call the `minimize()` optimization function of Python[2], that will minimize the value of the `computeCost()` function defined in Eq. (12), with respect to parameter θ. In the `minimize()` function, we also use the `computeGrad()` function, that computes the gradient of the cost function defined in Eq. (13). Also note that the `args` parameter contain the training data $\mathbf{X}$ and the corresponding class labels. The function returns the values of parameter θ.

```
# Initialize fitting parameters
initial_theta = zeros((3,1))

# Run minimize() to obtain the optimal theta
Result = op.minimize(fun = computeCost, x0 = initial_theta, args = (X, y), method = 'TNC',
jac = computeGrad);
theta = Result.x;
```

We can also plot the training data and the decision boundary produced by the logistic regression method.

```
# Plot the decision boundary
plot_x = array([min(X[:, 1]), max(X[:, 1])])
plot_y = (-1.0 / theta[2]) * (theta[1] * plot_x + theta[0])
plt.plot(plot_x, plot_y)
plt.scatter(X[pos, 1], X[pos, 2], marker='o', c='b')
```

[2] `scipy.optimize.minimize` function: `http://goo.gl/HoM6Ib`

```
plt.scatter(X[neg, 1], X[neg, 2], marker='x', c='r')
plt.xlabel('Exam 1 score')
plt.ylabel('Exam 2 score')
plt.legend(['Decision Boundary', 'Not admitted', 'Admitted'])
plt.show()
```

Lastly, we perform the classification of the test data, i.e., the prediction of the class labels. As we have described earlier, this can be done using the logistic function that returns a value in the range $[0, 1]$: if the predicted value of an instance is greater than 0.5 assign it to the first class (otherwise to the second one). This is implemented by the `predict()` function in the `predict.py` file.

```
# Compute accuracy on the training set
p = predict(array(theta), X)
counter = 0
for i in range(y.size):
if p[i] == y[i]:
counter += 1
print 'Train Accuracy: %f' % (counter / float(y.size) * 100.0)
```

3.2 Tasks to be Performed

The goal of the lab is to implement the basic components of the logistic regression classification algorithm:

- Fill in the code of the logistic function in the `sigmoid.py` file. Here we will use a the *sigmoid* function[3] as a case of logistic function:

$$S(z) = \frac{1}{1 + e^{-z}}.$$

- Fill in the code of the `computeCost()` function in the `computeCost.py` file, based on the formula of Eq. (12). Note that, the terms within the logarithms of Eq. (12) correspond to the sigmoid (logistic) function of the dot product between the input data X and parameter θ.

- Fill in the code of the `computeGrad()` function in the `computeGrad.py` file, based on the formula of Eq. (13).

- Finally, fill in the code in the `predict.py` file to implement the `predict()` function. This can be done by applying the logistic (sigmoid) function on the test data (dot product between test data X and parameters θ).

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.

[3] Wikipedia's lemma for *Sigmoid function*: `http://en.wikipedia.org/wiki/Sigmoid_function`.