# Machine Learning

## Lab. 3. Dimensionality reduction. Linear discriminant analysis and principal component analysis

Many datasets contain rich information about the objects they concern. Let's take as an example the problem of interpreting medical images. The task is to label areas of the image where a tumour is present. This is, of course, a binary classification problem, where image voxels should be classified as either 0 - healthy or 1 - cancerous. For each voxel, we can determine tens of texture features, calculated from small neighbourhoods around the voxel, for example, with sizes of 5x5x5. Each of the calculated features describes one voxel of the image, the whole image consists of many voxels, and the entire dataset may consist of many images.

For many reasons, we may want to get rid of some of the calculated features. For example, if we have too many features compared to the number of samples, we are at risk of **overfitting** - excessive fitting of model parameters to training data, resulting in poor ability to **generalize**, i.e., to deal with new examples to be classified. High-dimensional data are also hard to visualize.

It is also often the case that we do not know *a priori* whether the determined features are useful - whether they have the ability to distinguish classes. In the previous exercises, we saw that some features of iris flowers cause probability distributions of data on flowers to overlap, so the classes are not separable using that feature. Features with weak discriminatory power can be rejected in the **feature selection** process.

**Dimensionality reduction** is something different. Sometimes discarding a certain feature entirely is not optimal. An appropriate linear transformation of the feature space can improve the situation, resulting in the calculation of new features that are combinations (linear) of the original features. Then we can obtain information about the significance of the features and reduce their number by discarding those that contribute the least information to the problem. By removing features, we reduce the number of dimensions of the feature space, hence the name "dimensionality reduction".

In this exercise, we will focus on two algorithms used for transforming the original feature space and reducing dimensionality - linear discriminant analysis (LDA) and principal component analysis (PCA). These methods are quite similar, but they have one fundamental difference - LDA is a supervised method, and PCA is an unsupervised one. This means that in the first case, the class labels are necessary, while in the second case, they are not used. LDA is also a type of simple classifier that does not require learning. PCA is a general-purpose technique for reducing the dimensionality of a dataset, while LDA is a classification-specific technique for finding a linear combination of features that can best separate the classes in the dataset. Both techniques have their own strengths and limitations and can be used in different scenarios depending on the problem at hand.

### Linear discriminant analysis

Linear Discriminant Analysis is essentially a type of linear classifier, but it can also be used for dimensionality reduction. In this case, the dataset represented by the matrix X is projected into a lower-dimensional space. During the transformation, we lose a certain number of dimensions, but we process the input features in such a way that the remaining new features contain most of the original

information contained in the dataset. It is important to note the assumptions underlying this method: the probability of each class's features has a normal (Gaussian) distribution, and their covariance matrices are equal. If you are not familiar with the Gaussian distribution, refer to the literature. The covariance matrix measures the co-variance of each pair of features in the dataset. The assumption of equal covariance matrices means that the co-variance of each pair of features in each class is the same.

We will begin the analysis of the LDA method by examining a ready-made example from the scikit-learn module's documentation. It has been copied into the file analysis_LDA_i_QDA.py. The example includes a comparison of LDA to another type of discriminant analysis: quadratic discriminant analysis (QDA). This is a similar method, but it does not require the assumption of equal covariances.

Task 1: Open the analysis_LDA_i_QDA.py program. Examine the code, in particular lines 155-156 and 162-163. Run the program and view the results, read the comment below.

After running the program, plots appear: the left column shows results for LDA, and the right column shows results for QDA. The plots display samples from two classes (red and blue colours, crosses indicate misclassified samples) that were pseudorandomly generated in a controlled manner. In the first row of the plot, we have data with equal class covariance (in accordance with LDA assumption), and in the second row, covariances are different. Since LDA and QDA are primarily classification methods, decision boundaries (white lines) have been drawn. The covariance ellipses represent the probability distribution of data in each class, where the directions of the ellipses' axes correspond to the eigenvectors of the covariance matrix. It can be observed that LDA performs much better when the assumption of equal covariance is satisfied. On the other hand, QDA gives the same result as LDA in case of equal covariance, but performs much better in case of different covariances.

Where does the decision boundary come from in LDA? In LDA, the parameters of the decision boundary can be simply calculated using a specific formula. Firstly, the decision boundary by definition passes through the point exactly between the class centroids (stars in the plot). Secondly, it is perpendicular (orthogonal) to:

$$\boldsymbol{w} = \Sigma^{-1}(m^{(1)} - m^{(2)})$$

where $\Sigma^{-1}$ is the inverse of the covariance matrix (equal for both classes), and $m^{(1)}$ and $m^{(2)}$ are the centroids (means) of the classes. The classification decision is based on the comparison:

$$\boldsymbol{w} \cdot \boldsymbol{x} > \boldsymbol{w} \cdot \frac{1}{2}(m^{(1)} - m^{(2)})$$

where **x** is the data vector (one sample). If the inequality is satisfied, then the sample belongs to class 1. The above inequality means that the classification criterion is a linear combination function of the features of the samples **x**, determined by **w**. This conclusion can also be expressed differently: the classification result depends on the projection of the vector **x** onto the vector **w**.

In the case when there are more classes, the decision boundary is a piecewise linear function and depends on the course of the decision boundaries for each pair of classes. These boundaries intersect at one point. After removing the appropriate half-line starting at the intersection point, we obtain the expected division of the space. Of course, this description applies to the case of two-dimensional feature space. In the case of a larger number of dimensions, lines are replaced by their multidimensional counterparts.

Optional Task 1: using elements of the code from the analysis_LDA_i_QDA.py program, develop a new program visualizing the case of LDA working for three classes and two features.

From the above abstract example, where the data was artificially generated, we will move on to an example on the Iris dataset, taking a closer look at the individual steps of the LDA algorithm. We will also add a dimensionality reduction element that was not used above.

As a reminder: the Iris dataset contains 150 samples of three species of iris: setosa, versicolor, and virginica. Each flower is described by four features, so the feature space is 4-dimensional. The data is loaded into the X matrix with a shape of 150 x 4. The sample labels are contained in the y vector.

Using the LDA algorithm, we will look for the transformation matrix P with a shape of 4 x k. The dot product X·P results in a transformation (projection) of the dataset into a new space of dimension k. The new data matrix will have a shape of 150 x k. The new features will be a linear combination of the old features determined by P.

Task 2: complete the lda.py program at the locations marked with a # !!! comment, in order to implement the LDA algorithm given below. Calculate the covariance matrix using the np.cov() function with the parameter rowvar = False. Use the np.linalg.inv() function to calculate the inverse of the matrix. Use the .dot() method on numpy arrays or the np.dot() function to calculate dot products. Use the np.linalg.eigh() function to calculate eigenvectors and eigenvalues. Use the .T method to transpose matrices.

The LDA algorithm for dimensionality reduction can be presented in five steps:

1. Calculating the mean vectors for each class.

$$m^{(i)} = \frac{1}{N^{(i)}} \sum_{j=1}^{N^{(i)}} x_j^{(i)}$$

2. Calculating the within-class scatter matrix:

$$S_W = \sum_{i=1}^{c} S_i = \sum_{i=1}^{c} \left( \sum_{j=1}^{N^{(i)}} \left( x_j^{(i)} - m^{(i)} \right) \left( x_j^{(i)} - m^{(i)} \right)^T \right)$$

   or alternatively, using the sample covariance matrix:

$$S_W = \sum_{i=1}^{c} (N^{(i)} - 1) \Sigma_i$$

   as well as the between-class scatter matrix:

$$S_B = \sum_{i=1}^{c} N^{(i)} \left( m^{(i)} - m \right) \left( m^{(i)} - m \right)^T$$

3. Calculating the eigenvectors and eigenvalues of the matrix $S_W^{-1} S_B$.

4. Sorting the eigenvectors by decreasing eigenvalue, selecting k eigenvectors with the largest eigenvalues.
5. Transforming the sample vectors using the eigenvector matrix to obtain the new (reduced) k-dimensional feature space.

The notation used above: $i$ is the class index, $m^{(i)}$ is the centroid (mean of samples) of class $i$, $x_j^{(i)}$ is the j-th sample from class $i$, $N^{(i)}$ is the number of samples in class $i$, $m$ is the mean of the entire data set, $c$ is the number of classes, $\Sigma_i$ is the covariance matrix of class $i$.

How does dimensionality reduction using LDA work? As we can see, it is necessary to calculate the between-class scatter matrix (measuring the distance between the means of classes), and then the eigenvectors and eigenvalues of the inverse within-class scatter matrix (measuring the variance within each class) and between-class scatter matrix product. The eigenvectors define specific directions, and the eigenvalues represent the "importance" of these directions in classification. Therefore, transforming the feature space using eigenvectors allows us to find a combination of features that maximizes discriminant ability - maximize inter-class variance and minimize within-class variance.

## Principal Component Analysis

Principal Component Analysis, as mentioned earlier, does not use information about sample class membership. It is an unsupervised method. However, it has significant similarities to LDA because it also seeks directions in which the variability of features in the data set is greatest. An important assumption is made in this regard - consecutive directions must be orthogonal, or perpendicular to each other. The new directions are called principal components. The first principal component explains the most variability in the set, the second explains less, and so on. There are as many principal components as there are features in the original data set.

To calculate the first principal component $w_{(1)}$, we need to find the transformation (linear) vector **w** that maximizes the expression:

$$\frac{\boldsymbol{w}^T X^T X \boldsymbol{w}}{\boldsymbol{w}^T \boldsymbol{w}}$$

There is no need to use optimization methods to maximize this expression. The properties that are omitted here can be used to determine that the maximum value of this expression is the largest eigenvalue of the matrix $X^T X$, which corresponds to the eigenvector $w_{(1)}$ being sought.

The main component w in the transformed feature space is then obtained as

$$x_i w_{(1)}$$

for any sample (data vector) $x_i$. Searching for the second and subsequent components looks a little different. First, subtract the first k-1 principal components from the data matrix:

$$\widehat{X_k} = X - \sum_{n=1}^{k-1} X w_{(n)} w_{(n)}^T$$

Then, we maximize the expression again:

$$\frac{\boldsymbol{w}^T {X_k}^T X_k \boldsymbol{w}}{\boldsymbol{w}^T \boldsymbol{w}}$$

It can be proven that the next eigenvectors $w_{(k)}$ maximizing the above expression are further eigenvalues of the matrix $X^T X$. The decomposition of the data matrix into principal components can be represented as

$$T = XW$$

where W is a matrix whose columns are eigenvectors of the matrix $X^T X$, arranged in decreasing order by the eigenvalues.

Task 3: Load, analyze and run the pca_sklearn.py program. Investigate the differences in LDA and PCA results.

Task 4: Write a program in which you perform classification on the Iris dataset transformed using LDA and PCA (reduce the dimensionality of the feature space to 2 as in the previous task). After reduction, select only samples from the versicolor and virginica classes for classification. Use logistic regression as the classifier. In both cases, count how many samples in the test set are misclassified.

Optional Task 2: Based on the given PCA algorithm, write your own implementation. Use the np.linalg.eigh() function to calculate eigenvectors and eigenvalues. Visualize the results and check if they are the same as those provided by the PCA function from the scikit-learn module.

**This instruction was based on, amongst others:**


https://ml-explained.com/blog/linear-discriminant-analysis-explained

https://stats.stackexchange.com/questions/92157/compute-and-graph-the-lda-decision-boundary

https://scikit-learn.org/stable/auto_examples/decomposition/plot_pca_vs_lda.html#sphx-glr-auto-examples-decomposition-plot-pca-vs-lda-py

https://en.wikipedia.org/wiki/Principal_component_analysis

https://en.wikipedia.org/wiki/Linear_discriminant_analysis

ChatGPT


This instruction was written in Polish by Jakub Jurek, then translated by ChatGPT.