Joan Grau Noël - 172578

Practice 3 – Classification Methods

Introduction

In this practical we will attempt to use the pattern recognition methods we used on the last practicals in order to “teach” our program to classify data (this means labeled data) into different classes by using different methods.

To do this practical we focus on two different parts to do, first the dimensionality reduction methods and then the classifying methods.

Dimensionality Reduction

This methods were already implemented on the previous practical, just to name them:

- **PCA**: Projects the data on the maximum number directions of variance that we choose. Then we reproject them with the consequential loss of precision, but we get a much more simplified problem.
- **LDA**: The polar opposite of PCA, in the way that instead of projecting on the maximum direction of variance, it does so on the minimum direction of variance, on the number of directions we choose. That way we should get a projection that separates the classes as much as possible. As LDA alone doesn’t work when using it’s code, we first apply PCA and then LDA, with the same dimensionality, that way we can make it work in our code.
- **Kernel PCA**: This is a new method that we implement in this practical. Based in PCA in the sense of mathematical reduction of dimensions and projecting the data, what it does is use a kernel function to change the space of the data into a more treatable one. By doing this, we can also use PCA on non-linear problems.
- **None**: we can also choose to not to use any DR method, so we can see the differences that we get.

Classification

We will use different methods in order to classify our data. As the data could have a DR applied to it, we need to take it into account and apply the DR both on training and test sets (although we don't need to, but it gives better results). The methods that we implement to classify, first create a training set, that then it's used in order to classify the data. To name them:

- **K-NN**: this method is given at us at the beginning of the practical, what it does is classify the data into clusters by based on it's positions on the plane, and then choosing what cluster is closer to our test samples, based on its mean.
- **SVM**: What this method does is create an hyperplane between the classes in order to separate them. The problem is that in matlab it's only natively implemented as a binary classifier, so what we did is search for a multi-SVM code that helps us classify the test samples into multiple classes. The method that this code uses is one-vs-all. We also modified it to also support kernel-SVM.
- **Mahalanobis distance**: similar to K-NN, but instead of using the euclidean distance to the means, it uses the Mahalanobis distance:

  $$d(x, y) = \sqrt{(x - y)^T S^{-1} (x - y)}$$

  The one closest to our samples will be the class choosen to be classified into.
- **Kernel-SVM**: similar to the kernel-PCA, but in this case it changes the space in order to classify the data. We are asked to use two different types of Kernel-SVM:
  - *Polynomial*, which transforms the space by using a 3rd degree polynomial combination of our space.
  - *Gaussian*: which transforms the space into a gaussian based space.

Results

We will fill the table that was given to us, with both the accuracy and the confusion matrix.

Also, we will use the **'gabor'** filter as a feature for this example, as we have used the code from last practicals where we implemented it. We could also choose a different one from the function *'extractFeaturesFromData'*.

Using emotions 1, 2 and 7:

	Raw data	**PCA** (*30 dims*)	**LDA** (*5 dims*)	**Kernel PCA** (*25 dims*)
K-NN	ACC = 0.6849 CONF = 26 6 12 17 11 8 2 1 63	ACC = 0.7671 CONF = 36 7 8 5 7 6 4 4 69	ACC = 0.5548 CONF = 13 3 7 0 2 10 32 13 66	ACC = 0.8219 CONF = 36 6 4 8 8 3 1 4 76
Mahalanobis *(depending on the dataset it doesn't work)*	*It doesn't work (not-positive covariance matrix)*	ACC = 0.6781 CONF = 24 4 9 19 13 12 2 1 62	*It doesn't work (not-positive covariance matrix)*	*It doesn't work (not-positive covariance matrix)*
SVM	ACC = 0.8904 CONF = 36 2 3 5 15 1 4 1 79	*It doesn't work (no convergence)*	ACC = 0.4726 CONF = 17 5 24 3 0 7 25 13 52	*It doesn't work (no convergence)*

	Raw data	**PCA** (*30 dims*)	**LDA** (*5 dims*)	**Kernel PCA** (*25 dims*)
kernel-SVM (polynomial)	ACC = 0.5685 CONF = 0 0 0 0 0 0 45 18 83	ACC = 0.5685 CONF = 0 0 0 0 0 0 45 18 83	ACC = 0.4247 CONF = 6 3 27 11 7 7 28 8 49	ACC = 0.5685 CONF = 0 0 0 0 0 0 45 18 83
kernel-SVM (gaussian)	ACC = 0.5685 CONF = 0 0 0 0 0 0 45 18 83	ACC = 0.5685 CONF = 0 0 0 0 0 0 45 18 83	ACC = 0.5205 CONF = 9 4 12 0 0 4 36 14 67	ACC = 0.5685 CONF = 0 0 0 0 0 0 45 18 83

- *Cases where Mahalanobis distance doesn't work*:

As the error code gives an output saying that the covariance matrix isn't positive and therefore it can't calculate the result, I added different biases (ex. +10) in the cases that gives that error, in order to at least get some results. The accuracy and confusion matrix are the following:

	Raw Data	**LDA**	**Kernel PCA**
Mahalanobis	ACC = 0.4795 CONF = 13 6 16 8 2 12 24 10 55	ACC = 0.5274 CONF = 8 2 14 0 0 0 37 16 69	ACC = 0.1233 CONF = 0 0 0 45 18 83 0 0 0

- Referencing the cases where SVM doesn't work, I didn't find any solution on why it isn't working.

Conclusions

As we can see by the results, we have many variance on what method it's the best and what we should be using, and it's pretty hard to give a global result on the practical. So we will analyze each result for each classification method:

- **K-NN**: proved to be the best method overall, always giving decent results. Even though it's a "simple" method compared to the others, it's the one that better fits our dataset.
- **Mahalanobis**: We think this method didn't work too well in our dataset, maybe because of an error of implementation. The results were pretty good with the raw data, but we got the worst accuracy when using KernelPCA, so using this in combination of a dimensionality reduction method wouldn't be our choice.
- **SVM**: We don't know why this method didn't work with all the dimensionality reduction methods. First of all we see how the accuracy when combined with the raw data gave us the best accuracy (almost 90%), but when combined with DR methods the accuracy went donwards very easily. Therefore this method would be our fist choice when working with raw data, and probably too when working with DR methods, if we could make it work.
- **Kernel-SVM:** we will treat both polynomial and gaussian at the same time. This is because neither of them gave us good results. We can see how this classificator, on most cases, classified the data into only one class while ignoring the others, we preceive that as a signal that it isn't doing it's job.

The overall conclusion is that, we think that kernel methods in general, don't work well with this dataset, meaning that the data doesn't fit well with this change of basis.

It also proves once again, that the more simpler methods, are a lot of times, the ones that work best.

Bonus question: we are asked what we would do in order to search the best combination of dimensionality reduction method and classifier. What we would do is start with a random number of DR dimensions and a single classifier. Then, once applied to a classifier, and watching the accuracy results, we would change the DR dimensions in one way or another, with some sort of updating weights algorithm, such as gradient descent. Then we would do the same for each of the other different classifiers.