**Report of the project for the exam of the *Machine learning and pattern recognition* course.**

Candidate 1: Diego Gasco s296762

Candidate 2: Giovanni Genna s304684

**Abstract**

We have chosen to work on the *Wine quality detection* task. The request of the problem is to discriminate between good and bad quality wines. The first goal of our work is to study and analyse the provided problem, in particular the kind of features, their ranges and their distribution. The second part consists in developing the most appropriate classification algorithms and discarding models that are not proper for the considered task, by means of the training data. Finally, the different approaches chosen are evaluated on the test set.

**The dataset**

The dataset is taken from the UCI repository. The original dataset consists of 10 classes (quality 1 to 10). For this project, the dataset has been binarized, collecting all wines with low quality (score lower than 6) into class 0, and good quality (score greater than 6) into class 1. Wines with quality 6 have been discarded to simplify the task. The dataset contains both red and white wines (originally separated, they have been merged). There are 11 features, that represent physical properties of the wine:

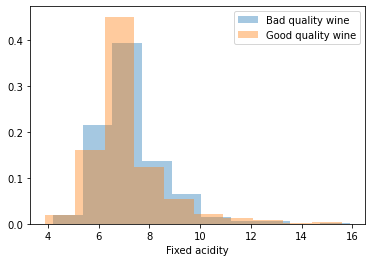
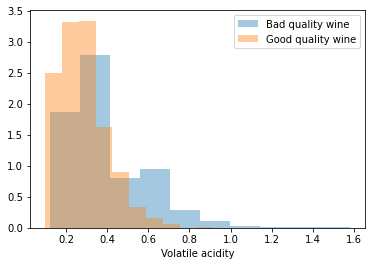
1. Fixed acidity
2. Volatile acidity
3. Citric acid
4. Residual sugar
5. Chlorides
6. Free sulfur dioxide
7. Total sulfur dioxide
8. Density
9. pH
10. Sulphates
11. Alcohol

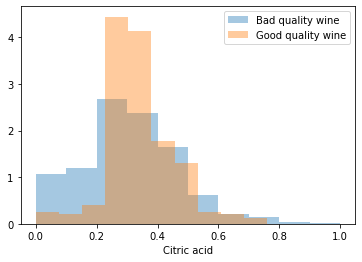
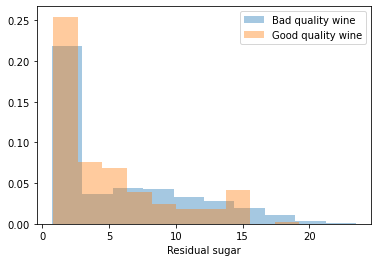
All the features can be considered continuous.

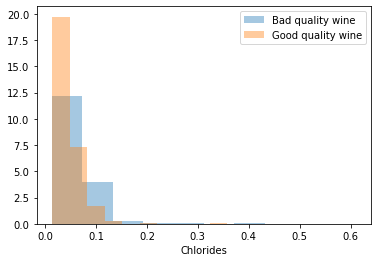
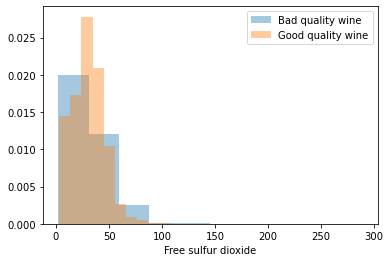
In the training set, there are 1839 samples, with their own features and another field with the class to which they belong (0 or 1). In particular, in this set, 1226 samples belong to the bad quality wine class and 613 to the good quality wine class. We modified the initial ‘Train.txt’ file to extract a matrix with on each column a sample (the numerical values of the eleven features: 11 rows, 1839 columns) and an array whose each element is the class label of the considered sample (we did these operations also for the ‘Text.txt’ file). In the test set, there are 1822 samples, 1158 belonging to the bad quality wine class and 664 belonging to the good quality wine class.

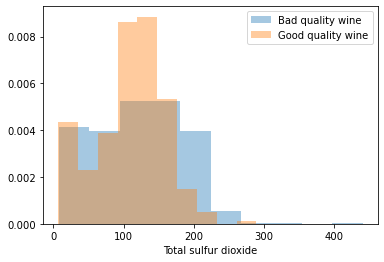
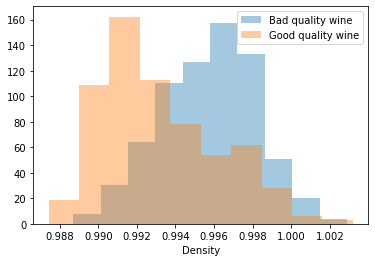
From this point, we will consider only the data of the training set. Later, when we will start working also on the test set (evaluation set), we will specify it.

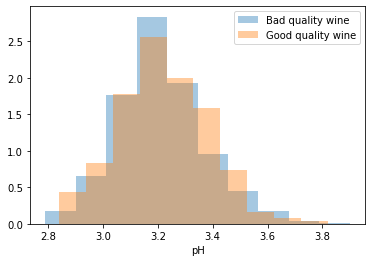
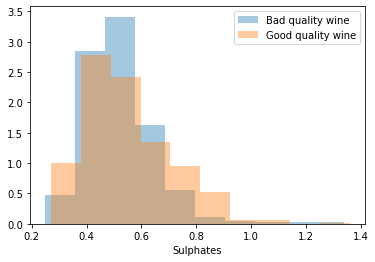
To understand how the several features of the two classes are distributed, for each of them, it is possible to plot the corresponding histogram:

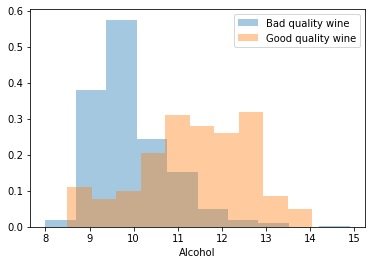
 



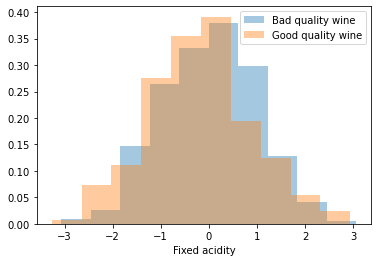
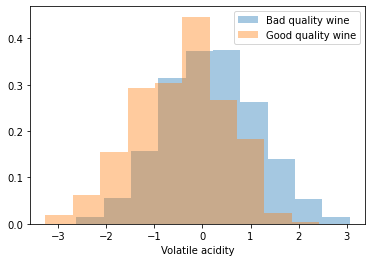
We can observe that, in general, for all features, there is no very well separation, vice versa, these is a large overlap and, sometimes, as in the case of *fixed acidity, residual sugar, pH* and *sulphates* the data is almost totally overlapping. For *density* and *alcohol*, it is possible to notice quite clearly that they are the features more separate. We can note that the data of the two classes for most of the features are distributed, more or less, like a Gaussian distribution, particularly for *fixed acidity, citric acid, density, pH, sulphates* and *alcohol*. In some classes, there is also the presence of outliers, especially for *chlorides* and *free sulfur dioxide.*

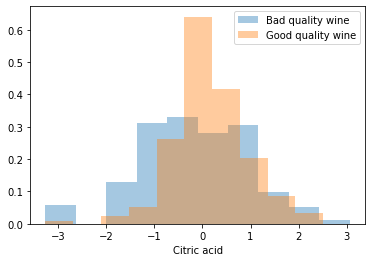
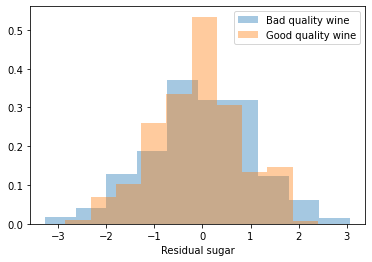
Even if most of the features have distribution like a Gaussian, given the presence of some features distributions less similar to a Gaussian and the presence of outliers, we try “Gaussianizing” the features. Gaussianization is a procedure that maps set of features to values whose empirical cumulative distribution function is well approximated by a Gaussian cumulative distribution function. The processing consists in mapping the features to a uniform distribution and then transforming the mapped features through the inverse of Gaussian cumulative distribution function. Let *x* be the feature to transform: the first step is to compute its rank *r(x)* over the training dataset:

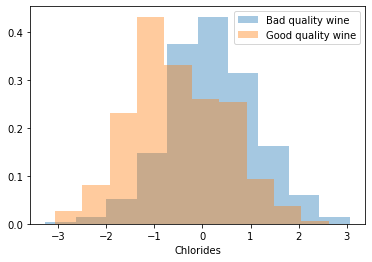
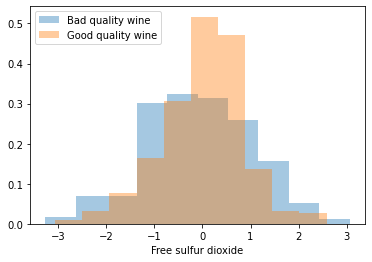
where xi is the value of the considered feature for the ith training sample. 1 to the numerator and 2 to the denominator are added to avoid numerical issues in the next stage (i.e., to assume the existance of a feature smaller than all the others and a feature larger than all the others). Finally, we compute the transformed feature as *y = Φ-1 (r(x))*, where *Φ* is the inverse of the cumulative distribution function of the standard normal distribution.

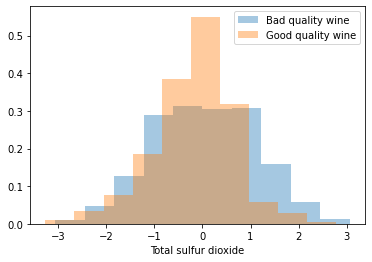
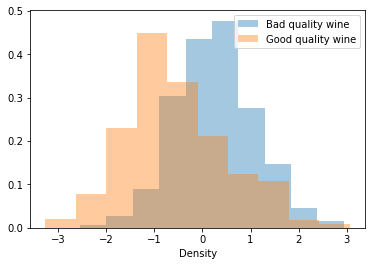
When we will work on the test set (evaluation set), we will apply the transformation both to training and evaluation samples, but the ranking should always be computed using the training data.

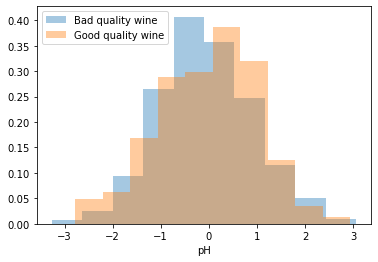
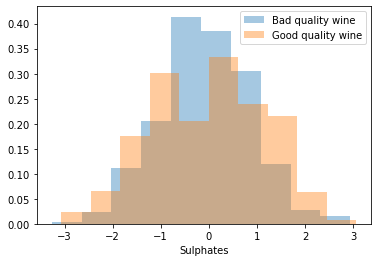
After the “Gaussianization”, the several features of the two classes are distributed as represented in the following histograms:

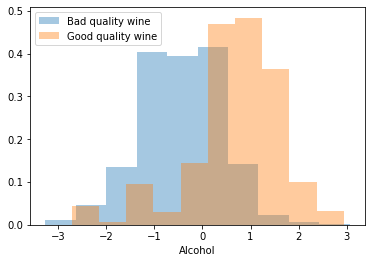
 

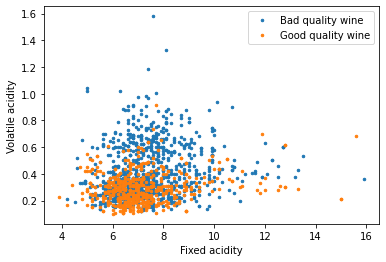
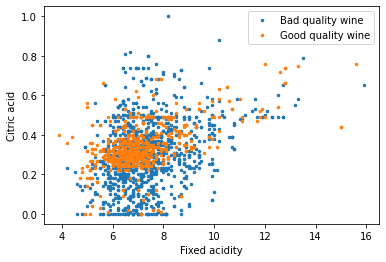
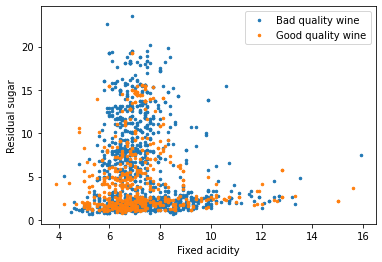
 

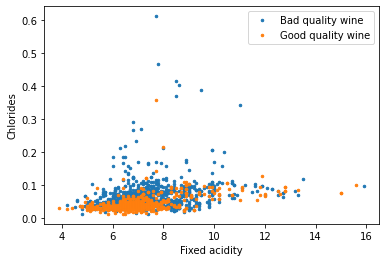
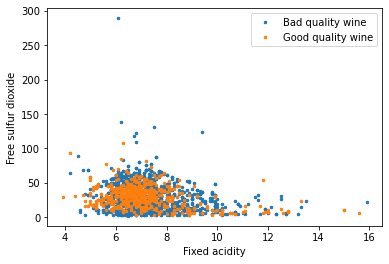
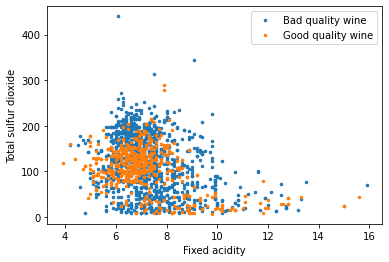
 

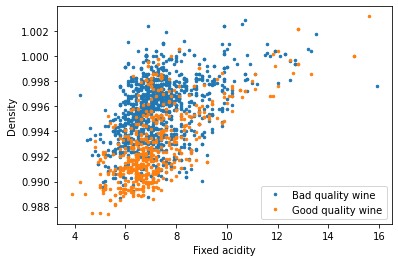
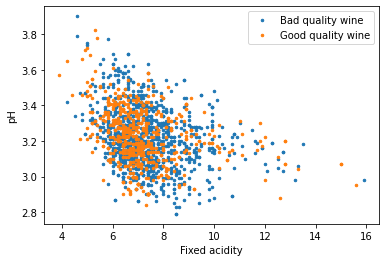
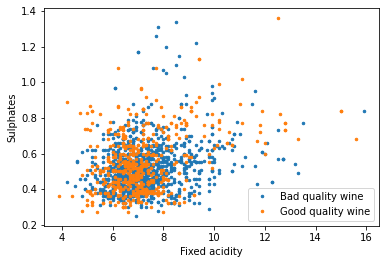


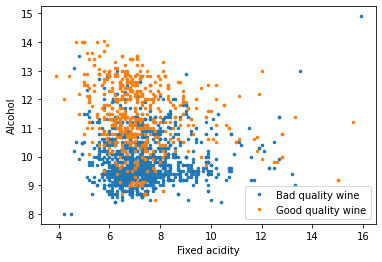
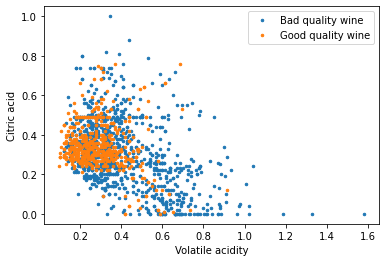
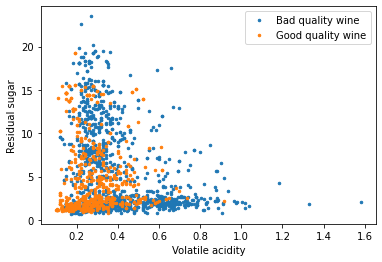
“Gaussianization” could be useful mainly for Gaussian classifier.

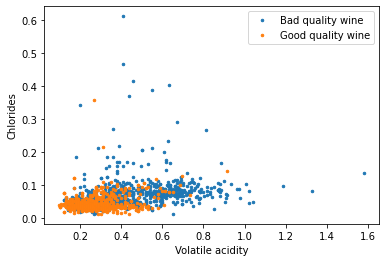
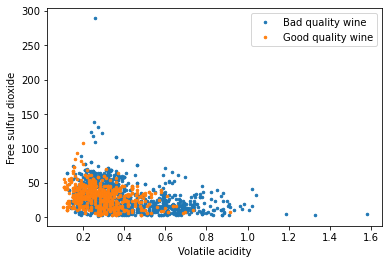
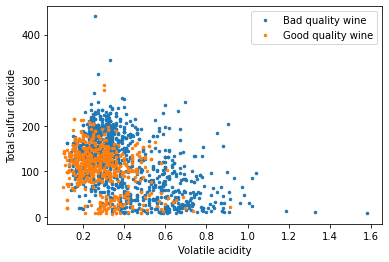
To have a view of the relationship between the different features for the different classes, we can visualize the scatter plots of the different features pairs for each class:

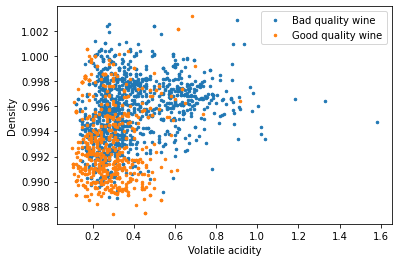
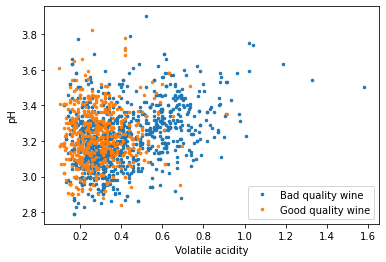
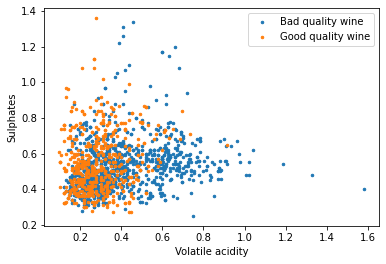
  

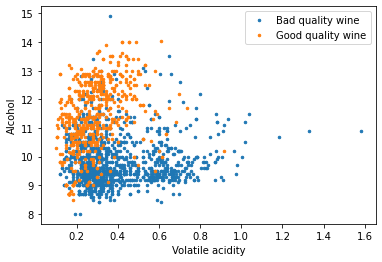
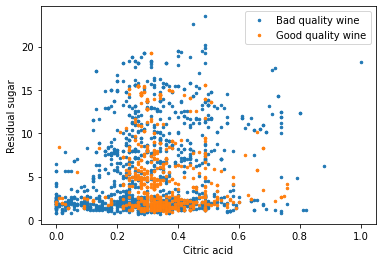
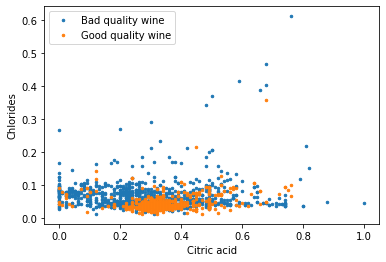
  

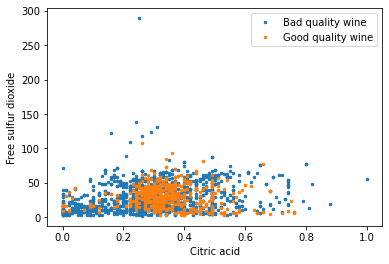
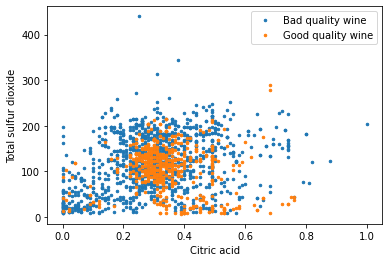
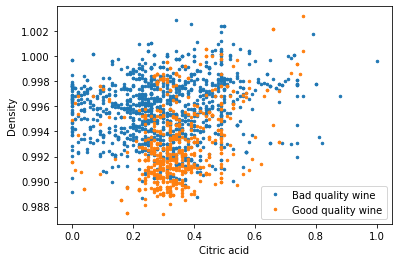
  

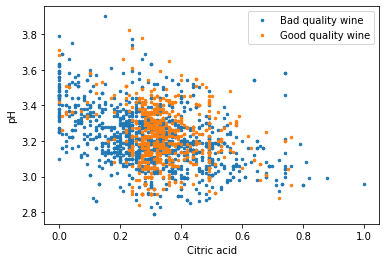
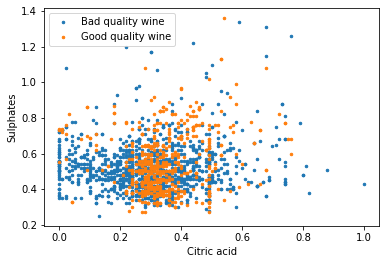
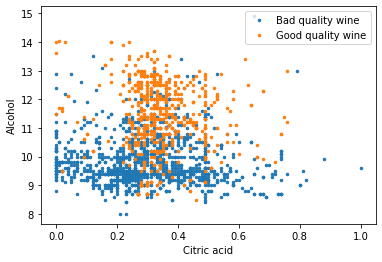
  

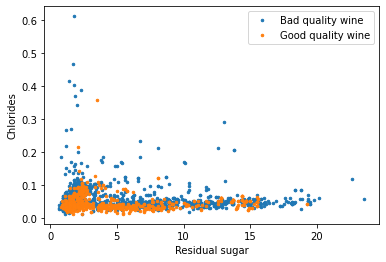
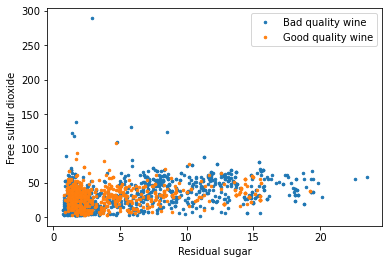
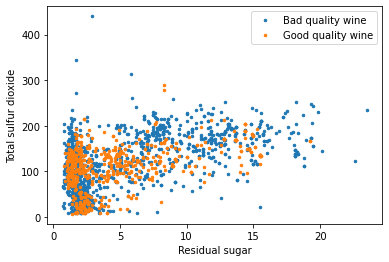
  

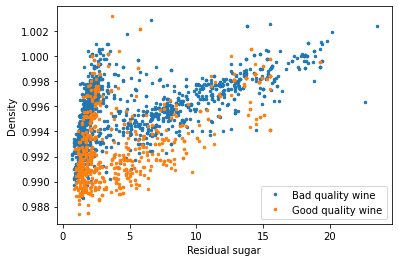
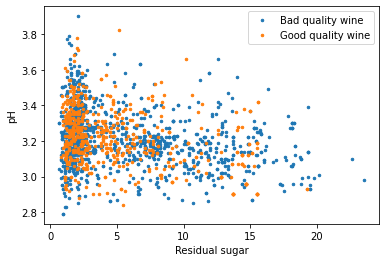
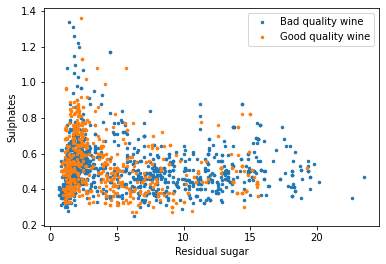
  

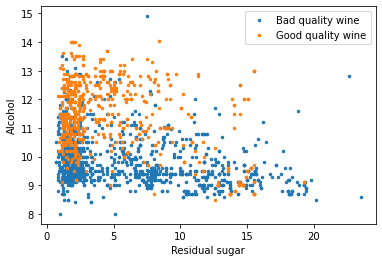
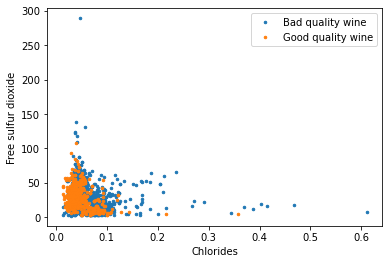
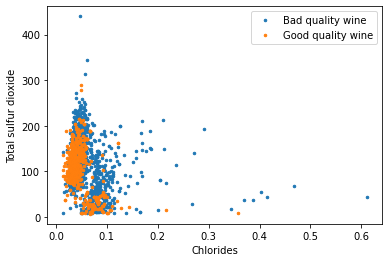
  

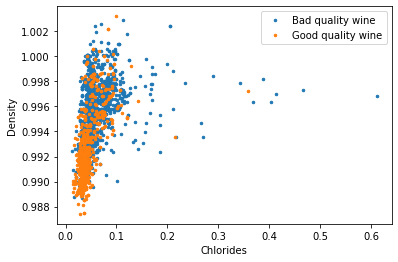
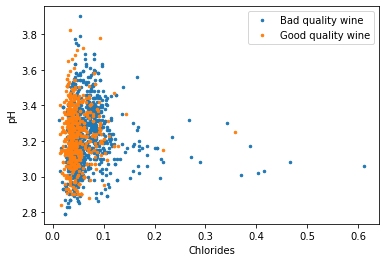
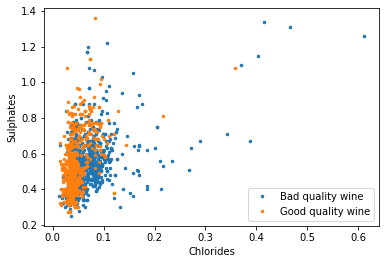
  

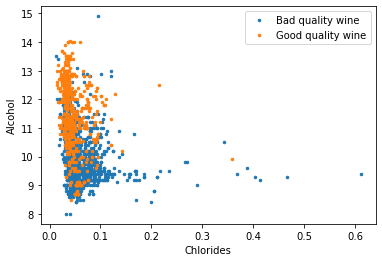
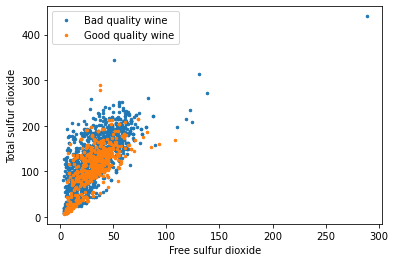
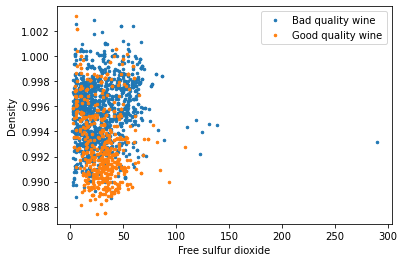
  

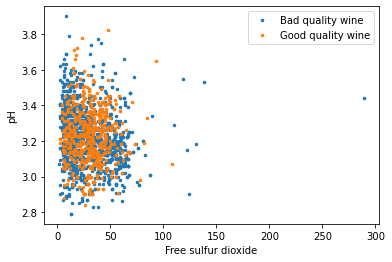
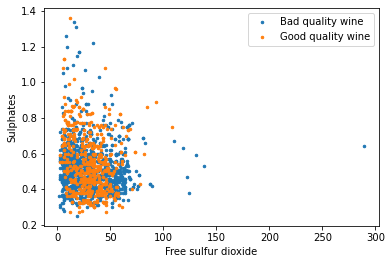
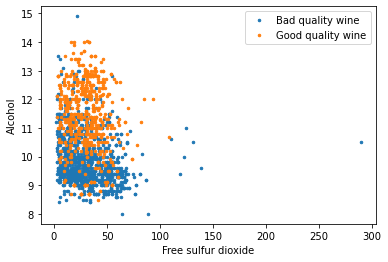
  

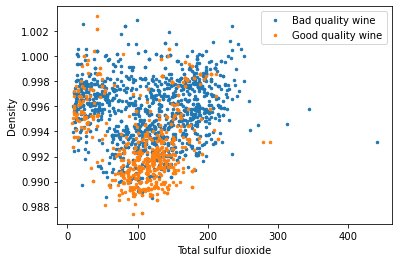
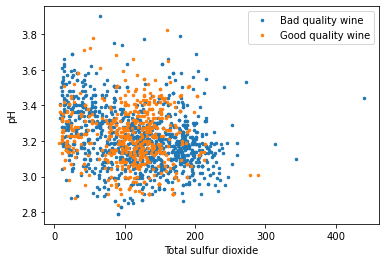
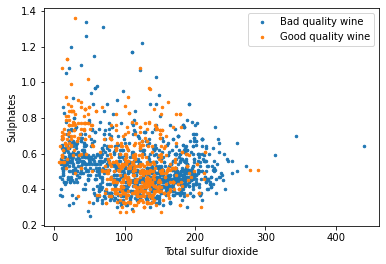
  

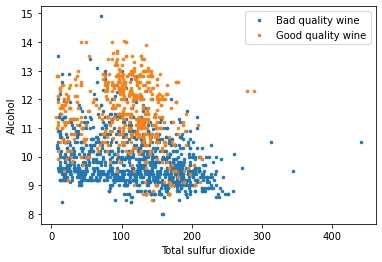
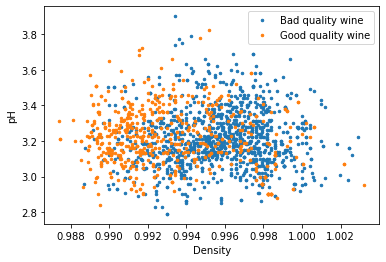
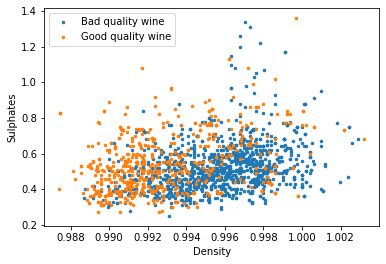
  

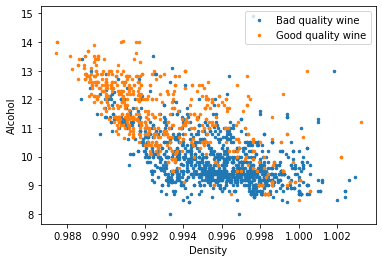
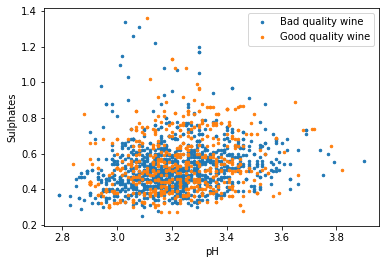
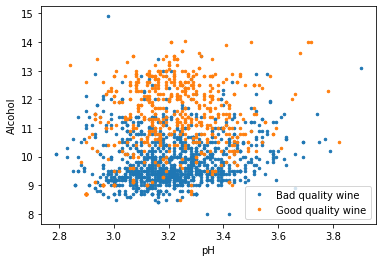
  

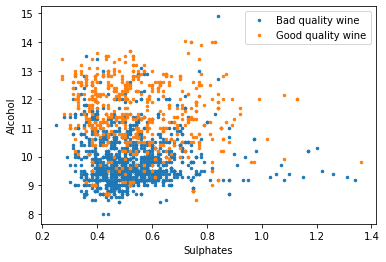
  



**Dimensionality Reduction**

Dimensionality reduction techniques compute a mapping from the n-dimensional original features space (in this case, n = 11), to a m-dimensional space, with m<n. The goal of these methods are several: compress information (also to reduce the computational time), remove unwanted variability, simplify classification, data visualization (in case of m = 2 and m =3).

11 features are not few, so it may be reasonable to apply a dimensionality reduction algorithm, to compress the information, preserving the most useful for the classification. The method we adopted for this problem is the *PCA (Principal Component Analysis)*, a linear unsupervised technique that finds a subspace of ℝn. *PCA* projects the data over the principal components. These can be computed from the eigenvectors of the data covariance matrix *C* corresponding to the largest eigenvalues.

*C =*

Where *N* is the number of samples, *xi*is the *i*-th sample and is the dataset mean:

*C* can be decomposed as:

*C= UΣUT*

Where *U* is a matrix, whose columns are the eigenvectors of *C* and *Σ* is a diagonal matrix, containing the eigenvalues in descending order.

*U = [u1 … um, um+1 … un]*

*P = [u1 … um]*

*P* corresponds to the first m columns of *U*.

Finally, we can apply the projection to the initial n-dimensional matrix of samples, to obtain the dimensionality reduction and have a m-dimensional matrix.

*Y = PTX*

From the scatter plots of the previous paragraph, it is possible to observe that there are some features correlated, particularly:

* *fixed acidity* and *density;*
* *residual sugar* and *density*;
* *alcohol* and *density*;
* *free sulfur dioxide* and *total sulfur dioxide*.

This information suggests that we may benefit from using *PCA* to map data to reduce the number of parameters to estimate. Given that there is the presence of four evident correlations, we will use m in the range [7, 10].

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

It is not possible to use *LDA (Linear Discriminant Analysis)* as a linear dimensionality reduction method, because it allows estimating at most C-1 directions, where C is the number of classes, but in this case, C is equal to 2.

**Classifications**

Classification consists in, given the feature vector representing an object, associate a label to the object based on properties of the representation; so, perform a mapping from the m-dimensional feature space to the space of labels. The mapping is also called decision function.

To evaluate and compare the different models that we will develop, we chosen to adopt the *K-fold cross validation.* This method can be employed to split the dataset in *K*, non-overlapping, subsets. We will then iteratively consider one subset as validation, and the remaining *K − 1* subsets as training set. We will implement the *K-fold cross validation* with *K=5.*

We know that the original dataset has 10 classes, scores from 1 to 10, but wines with quality 6 have been discarded. Therefore, the dataset has been binarized, from the 9 (10 original classes minus the class of wine with quality 6) classes to all wines with low quality (scores from 1 to 5, i.e., the merge of five classes) into class 0, and good quality (scores from 7 to 10, i.e., the merge of four classes) into class 1. For this reason, we chosen a prior probability  *= 4/9*, as the prior probability that a wine belongs to the good quality wine class and *1– = 5/9,* as the prior probability that a wine belongs to the bad quality wine. So, for the main application the prior is slightly biased towards the bad quality wine.

For what concerns the costs of miss-classified samples, we have decided to adopt *Cfp=1* and *Cfn=1*, so our application will be: *(, Cfp, Cfn)= (4/9, 1, 1).*

We will also consider other two unbalanced applications: *(, Cfp, Cfn)= (1/5, 1, 1)*, *(, Cfp, Cfn)= (4/5, 1, 1)* where the prior highly is biased towards one of the two classes.

Our goal, for the moment, is to choose the most promising classification model. It may be useful knowing how good the model would perform if we had selected the best possible threshold, so it is possible to compute the (normalized) DCF (Detection Cost Function) using all possible thresholds and select its minimum value.

To compute the minimum cost, we will consider a set of thresholds corresponding to (−∞, s1 . . . sn, +∞), where s1 . . . sn are the scores, sorted in increasing order. For each threshold t, we will compute the confusion matrix if scores were thresholded at t and the corresponding normalized DCF. The minimum DCF is the minimum of the obtained values.

Min DCF measures the cost we would pay if we made optimal decisions for our case the validation set using the recognizer scores.

The classification algorithm we will consider are:

* Multivariate Gaussian Classifier (MVG)
* Naïve Bayes Gaussian Classifier
* Tied Covariance Gaussian Classifier
* Naïve Tied Gaussian Classifier
* Logistic Regression (LR)
* Linear Support Vector Machines (Linear SVM)
* Polynomial Kernel Support Vector Machines (Poly SVM)
* Gaussian Radial Basis Function Kernel Support Vector Machines (RBF SVM)
* Gaussian Mixture Model (GMM)
* Diagonal Covariance Gaussian Mixture Model (Diag GMM)
* Tied Covariance Gaussian Mixture Model (Tied GMM)

**Multivariate Gaussian Classifier (MVG)**

The first model we will implement is the Multivariate Gaussian Classifier (MVG). This classifier is a generative model and assumes that samples of each class c, in our case ∈ {0, 1}, can be modelled as samples of a multivariate Gaussian distribution with class-dependent mean and covariance matrices.

where *N* is a normal distribution.

The ML solution for the parameters is given by the empirical mean and covariance matrix of each class:

where *xc,i*  is the i-th sample of the class c.

To carry out the computations useful for the calculation of the DCF, we use the log-likelihood ratio (llr), i.e., the logarithm of the ratio between the likelihood of observing the sample given that it belongs to class h1 or to class h0 (we denote the classes sometimes with *h* and sometimes with *c*):

We report in the tables below the obtained computational results:

|  |  |  |  |
| --- | --- | --- | --- |
|  | minimum DCF  *= 4/9)* | minimum DCF  ( *= 1/5)* | minimum DCF  *= 4/5)* |
| MVG, no PCA | 0.445 | 0.685 | 0.811 |
| MVG, with PCA (m=10) | 0.406 | 0.693 | 0.741 |
| MVG, with PCA (m=9) | 0.399 | 0.695 | 0.684 |
| MVG, with PCA (m=8) | 0.398 | 0.684 | 0.716 |
| MVG, with PCA (m=7) | 0.418 | 0.685 | 0.749 |

|  |  |  |  |
| --- | --- | --- | --- |
|  | minimum DCF  *= 4/9)* | minimum DCF  ( *= 1/5)* | minimum DCF  *= 4/5)* |
| MVG, “Gaussianization”, no PCA | 0.465 | 0.726 | 0.782 |
| MVG, “Gaussianization”, with PCA (m=10) | 0.451 | 0.724 | 0.781 |
| MVG, “Gaussianization”, with PCA (m=9) | 0.461 | 0.759 | 0.802 |
| MVG, “Gaussianization”, with PCA (m=8) | 0.462 | 0.752 | 0.785 |
| MVG, “Gaussianization”,  with PCA (m=7) | 0.447 | 0.701 | 0.804 |

The MVG obtains the best performance with PCA (m=8) and  *= 4/9* (min DCF= 0.398). In general, the pre-processing of data through "Gaussianization" always results in worse results than those without.

**Naïve Bayes Gaussian Classifier**

The Naïve Bayes Gaussian Classifier is a “simplification” of the MVG because assumes that, for each class, the different components are independent. In practice, it is simply a Gaussian classifier where the covariance matrices are diagonal. The ML solution for the mean parameters is the same, whereas the ML solution for the covariance matrices is:

i.e., the diagonal of the ML solution for the MVG model

As for the MVG, to carry out the computations useful for the calculation of the DCF, we use the log-likelihood ratio (llr).

We report in the tables below the obtained computational results:

|  |  |  |  |
| --- | --- | --- | --- |
|  | minimum DCF  *= 4/9)* | minimum DCF  *= 1/5)* | minimum DCF  *= 4/5)* |
| Naïve Bayes, no PCA | 0.494 | 0.742 | 0.875 |
| Naïve Bayes, with PCA (m=10) | 0.476 | 0.775 | 0.818 |
| Naïve Bayes, with PCA (m=9) | 0.462 | 0.767 | 0.833 |
| Naïve Bayes, with PCA (m=8) | 0.456 | 0.767 | 0.889 |
| Naïve Bayes, with PCA (m=7) | 0.454 | 0.772 | 0.883 |

|  |  |  |  |
| --- | --- | --- | --- |
|  | minimum DCF  *= 4/9)* | minimum DCF  *= 1/5)* | minimum DCF  *= 4/5)* |
| Naïve Bayes, “Gaussianization”, no PCA | 0.532 | 0.739 | 0.984 |
| Naïve Bayes, “Gaussianization”, with PCA (m=10) | 0.551 | 0.770 | 0.970 |
| Naïve Bayes, “Gaussianization”, with PCA (m=9) | 0.549 | 0.785 | 0.975 |
| Naïve Bayes, “Gaussianization”, with PCA (m=8) | 0.555 | 0.773 | 0.977 |
| Naïve Bayes, “Gaussianization”, with PCA (m=7) | 0.557 | 0.796 | 0.972 |

The Naïve Bayes Gaussian Classifier obtains the best performance with PCA (m=7) and  *= 4/9* (min DCF= 0.454). In general, the pre-processing of data through "Gaussianization" always results in worse results than those without.

The performances are worse than the full-covariance MVG, and this could be related to the fact that the data are highly correlated, and, instead, the Naïve Bayes assumes that, for each class, the different components are independent; in fact, the best performance is obtained for m=7, i.e., when the classification is computed from a dataset of uncorrelated features.

**Tied Covariance Gaussian Classifier**

The Tied Gaussian Classifier is another version of the MVG and assumes that each class has its own mean, but the covariance matrix is the same for all classes. The ML solution for the covariance matrix is given by the empirical within-class covariance matrix:

where N is the number of samples.

As for the MVG, to carry out the computations useful for the calculation of the DCF, we use the log-likelihood ratio (llr).

We report in the tables below the obtained computational results:

|  |  |  |  |
| --- | --- | --- | --- |
|  | minimum DCF  *= 4/9)* | minimum DCF  *= 1/5)* | minimum DCF  *= 4/5)* |
| Tied, no PCA | 0.387 | 0.687 | 0.746 |
| Tied, with PCA (m=10) | 0.397 | 0.672 | 0.692 |
| Tied, with PCA (m=9) | 0.394 | 0.677 | 0.678 |
| Tied, with PCA (m=8) | 0.419 | 0.677 | 0.783 |
| Tied, with PCA (m=7) | 0.422 | 0.669 | 0.789 |

|  |  |  |  |
| --- | --- | --- | --- |
|  | minimum DCF  *= 4/9)* | minimum DCF  *= 1/5)* | minimum DCF  *= 4/5)* |
| Tied, “Gaussianization”, no PCA | 0.440 | 0.703 | 0.708 |
| Tied, “Gaussianization”, with PCA (m=10) | 0.445 | 0.708 | 0.698 |
| Tied, “Gaussianization”, with PCA (m=9) | 0.448 | 0.726 | 0.710 |
| Tied, “Gaussianization”, with PCA (m=8) | 0.470 | 0.729 | 0.724 |
| Tied, “Gaussianization”, with PCA (m=7) | 0.480 | 0.762 | 0.736 |

The Tied Covariance Gaussian Classifier obtains the best performance with no PCA and  *= 4/9* (min DCF=0.387)*.* In general, the pre-processing of data through "Gaussianization" always results in worse results than those without.

The results are better than the previous and this may be linked to the fact that in the dataset there are great correlations and Tied Covariance model can capture these correlations. In general, we have seen that the classes have, more or less, similar distributions, so covariances should be similar: the model should provide a more dependable estimate.

**Naïve Tied Gaussian Classifier**

The Naïve Tied Gaussian Classifier put the two methods explained above together: it assumes that each class has its own mean, but the covariance matrix is the same for all classes and it is diagonal. The ML solution for the covariance matrix is given by:

i.e., the diagonal of the ML solution for the Tied Covariance Gaussian Classifier model.

As for the MVG, to carry out the computations useful for the calculation of the DCF, we use the log-likelihood ratio (llr).

We report in the table below the obtained computational results:

|  |  |  |  |
| --- | --- | --- | --- |
|  | minimum DCF  *= 4/9)* | minimum DCF  *= 1/5)* | minimum DCF  *= 4/5)* |
| Naïve Tied, no PCA | 0.459 | 0.739 | 0.922 |
| Naïve Tied, with PCA (m=10) | 0.421 | 0.679 | 0.684 |
| Naïve Tied, with PCA (m=9) | 0.424 | 0.679 | 0.692 |
| Naïve Tied, with PCA (m=8) | 0.448 | 0.684 | 0.735 |
| Naïve Tied, with PCA (m=7) | 0.452 | 0.675 | 0.739 |

|  |  |  |  |
| --- | --- | --- | --- |
|  | minimum DCF  *= 4/9)* | minimum DCF  *= 1/5)* | minimum DCF  *= 4/5)* |
| Naïve Tied, “Gaussianization”, no PCA | 0.521 | 0.749 | 0.940 |
| Naïve Tied, “Gaussianization”, with PCA (m=10) | 0.445 | 0.731 | 0.733 |
| Naïve Tied, “Gaussianization”, with PCA (m=9) | 0.447 | 0.747 | 0.727 |
| Naïve Tied, “Gaussianization”, with PCA (m=8) | 0.475 | 0.721 | 0.749 |
| Naïve Tied, “Gaussianization”, with PCA (m=7) | 0.474 | 0.749 | 0.748 |

The Naïve Tied Gaussian Classifier obtains the best performance with PCA (m=10) and  *= 4/9* (min DCF= 0.421)*.* In general, the pre-processing of data through "Gaussianization" always results in worse results than those without.

This performance is better than the best performance of the Naïve Bayes, but it is worse than the best of the full-covariance MVG and than the best performance of the Tied Covariance.

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

The best performance between all the generative models considered is given by the Tied Covariance with no PCA and  *= 4/*9: minimum DCF= 0.387; the second-best result is given by the Tied Covariance with PCA (m=9) and  *= 4/*9: minimum DCF= 0.394; ; the third-best is given by the Tied Covariance with PCA (m=10) and  *= 4/*9: minimum DCF= 0.397; the fourth-best result is given by the MVG with PCA (m=8) and  *= 4/*9: minimum DCF= 0.398.

As already mentioned, the pre-processing of data through "Gaussianization" always results in worse results than those without. This could be related to the fact that most of the features are already more or less distributed as Gaussian and few classes have outliers. Given that the “Gaussianization” did not give the desired results for the classifiers that should benefit most from it, this pre-processing step will not be used later for other classifiers that should not be too affected by the type of initial distribution of the data.

**Logistic Regression**

The Logistic regression is a discriminative approach for classification.

The regularized Logistic Regression objective can be written as follows:

,

The ML solution is the solution that minimize *J(w,b)*.

This is a prior-weighted version of the model; in fact, it considers the empirical class prior of the training data πT. We will consider πT = 4/9, πT = 1/5 and πT = 4/5.

For Logistic Regression models closed for expressions are not available for the ML solution. Nevertheless, we turn to numerical optimization, particularly, we will use the L-BFGS algorithm.

The scores *s* obtained computing *w\** and *b\** are equal to posterior-llr:

But, for the calculation of the actual DCF, we will use an optimal threshold for the llr and not for the posterior-llr, so we consider score *s’* equal to:

We will consider different values of λ from 1e-5 to until the DCF values worse again.

We report in the table below the obtained computational results:

|  |  |  |  |
| --- | --- | --- | --- |
|  | minimum DCF  *= 4/9)* | minimum DCF  *= 1/5)* | minimum DCF  *= 4/5)* |
| LR (*λ=1e-5*, *πT=4/9)*, no PCA | 0.417 | 0.719 | 0.597 |
| LR (*λ=1e-5*, *πT=4/9)*, with PCA (m=10) | 0.417 | 0.715 | 0.604 |
| LR (*λ=1e-5*, *πT=4/9)*, with PCA (m=9) | 0.417 | 0.726 | 0.590 |
| LR (*λ=1e-5*, *πT=4/9)*, with PCA (m=8) | 0.420 | 0.685 | 0.678 |
| LR (*λ=1e-5*, *πT=4/9)*, with PCA (m=7) | 0.437 | 0.682 | 0.702 |
| LR (*λ=1e-4*, *πT=4/9)*, no PCA | 0.410 | 0.719 | 0.607 |
| LR (*λ=1e-4*, *πT=4/9)*, with PCA (m=10) | 0.411 | 0.716 | 0.600 |
| LR (*λ=1e-4*, *πT=4/9)*, with PCA (m=9) | 0.412 | 0.724 | 0.604 |
| LR (*λ=1e-4*, *πT=4/9)*, with PCA (m=8) | 0.416 | 0.682 | 0.684 |
| LR (*λ=1e-4*, *πT=4/9)*, with PCA (m=7) | 0.435 | 0.682 | 0.701 |
| LR (*λ=1e-3*, *πT=4/9)*, no PCA | 0.407 | 0.697 | 0.666 |
| LR (*λ=1e-3*, *πT=4/9)*, with PCA (m=10) | 0.406 | 0.697 | 0.665 |
| LR (*λ=1e-3*, *πT=4/9)*, with PCA (m=9) | 0.406 | 0.697 | 0.665 |
| LR (*λ=1e-3*, *πT=4/9)*, with PCA (m=8) | 0.420 | 0.695 | 0.697 |
| LR (*λ=1e-3*, *πT=4/9)*, with PCA (m=7) | 0.436 | 0.690 | 0.743 |
| LR (*λ=1e-2*, *πT=4/9)*, no PCA | 0.465 | 0.697 | 0.780 |
| LR (*λ=1e-2*, *πT=4/9)*, with PCA (m=10) | 0.465 | 0.700 | 0.780 |
| LR (*λ=1e-2*, *πT=4/9)*, with PCA (m=9) | 0.465 | 0.697 | 0.779 |
| LR (*λ=1e-2*, *πT=4/9)*, with PCA (m=8) | 0.471 | 0.698 | 0.797 |
| LR (*λ=1e-2*, *πT=4/9)*, with PCA (m=7) | 0.475 | 0.682 | 0.805 |

|  |  |  |  |
| --- | --- | --- | --- |
|  | minimum DCF  *= 4/9)* | minimum DCF  *= 1/5)* | minimum DCF  *= 4/5)* |
| LR (*λ=1e-5*, *πT=1/5)*, no PCA | 0.404 | 0.710 | 0.622 |
| LR (*λ=1e-5*, *πT=1/5)*, with PCA (m=10) | 0.407 | 0.718 | 0.613 |
| LR (*λ=1e-5*, *πT=1/5)*, with PCA (m=9) | 0.402 | 0.719 | 0.625 |
| LR (*λ=1e-5*, *πT=1/5)*, with PCA (m=8) | 0.409 | 0.692 | 0.744 |
| LR (*λ=1e-5*, *πT=1/5)*, with PCA (m=7) | 0.432 | 0.695 | 0.738 |
| LR (*λ=1e-4*, *πT=1/5)*, no PCA | 0.393 | 0.723 | 0.625 |
| LR (*λ=1e-4*, *πT=1/5)*, with PCA (m=10) | 0.394 | 0.716 | 0.639 |
| LR (*λ=1e-4*, *πT=1/5)*, with PCA (m=9) | 0.390 | 0.719 | 0.629 |
| LR (*λ=1e-4*, *πT=1/5)*, with PCA (m=8) | 0.414 | 0.692 | 0.759 |
| LR (*λ=1e-4*, *πT=1/5)*, with PCA (m=7) | 0.433 | 0.692 | 0.744 |
| LR (*λ=1e-3*, *πT=1/5)*, no PCA | 0.407 | 0.685 | 0.719 |
| LR (*λ=1e-3*, *πT=1/5)*, with PCA (m=10) | 0.405 | 0.688 | 0.718 |
| LR (*λ=1e-3*, *πT=1/5)*, with PCA (m=9) | 0.402 | 0.687 | 0.719 |
| LR (*λ=1e-3*, *πT=1/5)*, with PCA (m=8) | 0.424 | 0.684 | 0.782 |
| LR (*λ=1e-3*, *πT=1/5)*, with PCA (m=7) | 0.426 | 0.674 | 0.792 |

|  |  |  |  |
| --- | --- | --- | --- |
|  | minimum DCF  *= 4/9)* | minimum DCF  *= 1/5)* | minimum DCF  *= 4/5)* |
| LR (*λ=1e-5*, *πT=4/5)*, no PCA | 0.432 | 0.729 | 0.563 |
| LR (*λ=1e-5*, *πT=4/5)*, with PCA (m=10) | 0.433 | 0.734 | 0.569 |
| LR (*λ=1e-5*, *πT=4/5)*, with PCA (m=9) | 0.432 | 0.739 | 0.545 |
| LR (*λ=1e-5*, *πT=4/5)*, with PCA (m=8) | 0.440 | 0.711 | 0.639 |
| LR (*λ=1e-5*, *πT=4/5)*, with PCA (m=7) | 0.456 | 0.690 | 0.664 |
| LR (*λ=1e-4*, *πT=4/5)*, no PCA | 0.433 | 0.734 | 0.560 |
| LR (*λ=1e-4*, *πT=4/5)*, with PCA (m=10) | 0.431 | 0.728 | 0.554 |
| LR (*λ=1e-4*, *πT=4/5)*, with PCA (m=9) | 0.433 | 0.729 | 0.547 |
| LR (*λ=1e-4*, *πT=4/5)*, with PCA (m=8) | 0.438 | 0.710 | 0.644 |
| LR (*λ=1e-4*, *πT=4/5)*, with PCA (m=7) | 0.456 | 0.688 | 0.669 |
| LR (*λ=1e-3*, *πT=4/5)*, no PCA | 0.428 | 0.700 | 0.605 |
| LR (*λ=1e-3*, *πT=4/5)*, with PCA (m=10) | 0.430 | 0.698 | 0.604 |
| LR (*λ=1e-3*, *πT=4/5)*, with PCA (m=9) | 0.426 | 0.695 | 0.603 |
| LR (*λ=1e-3*, *πT=4/5)*, with PCA (m=8) | 0.439 | 0.706 | 0.695 |
| LR (*λ=1e-3*, *πT=4/5)*, with PCA (m=7) | 0.449 | 0.701 | 0.697 |
| LR (*λ=1e-2*, *πT=4/5)*, no PCA | 0.485 | 0.728 | 0.785 |
| LR (*λ=1e-2*, *πT=4/5)*, with PCA (m=10) | 0.485 | 0.728 | 0.785 |
| LR (*λ=1e-2*, *πT=4/5)*, with PCA (m=9) | 0.483 | 0.728 | 0.786 |
| LR (*λ=1e-2*, *πT=4/5)*, with PCA (m=8) | 0.493 | 0.732 | 0.805 |
| LR (*λ=1e-2*, *πT=4/5)*, with PCA (m=7) | 0.489 | 0.731 | 0.806 |

The best performance is obtained for *λ=1e-4*, with PCA (m=9), *πT=1/5*,  *= 4/9*: minimum DCF= 0.390. The second-best performance is obtained for *λ=1e-4*, with no PCA, *πT=1/5*,  *= 4/9*: minimum DCF= 0.393. The third-best performance is obtained for *λ=1e-4*, with PCA (m=10), *πT=1/5*,  *= 4/9*: minimum DCF= 0.394.

**Linear Support Vector Machines**

Linear Support Vector Machines are linear classifiers that look for maximum margin separation hyperplanes. The (primal) SVM objective consists in minimizing:

where n is the number of training samples, C is a hyper-parameter and zi is the class label for the ith sample, encoded as:

To solve the SVM problem, it is possible to consider the dual formulation:

where **1** is a n-dimensional vector of ones, and H is a matrix whose elements are:

The SVM dual solution is the maximizer of . The dual and primal solutions α\* and w\* are related through:

and the optimal bias b\* can be computed considering a sample xi that lies on the margin:

*) = 1*

and then solving for b\*.

It is possible to slightly modify the SVM problem to employ L-BFGS-B to solve the dual problem:

where

where k is a constant that contrasts the effect of regularizing b

The dual objective of the modified primal SVM becomes the maximization of:

where is a matrix whose elements are:

To carry out the computations useful for the calculation of the DCF, we use the scores:

For our computations, we will fix the value of k to one and we will test the model with different value of C (0.001, 0.01, 0.1, 1, 10).

We report in the table below the obtained computational results:

|  |  |  |  |
| --- | --- | --- | --- |
|  | minimum DCF  *= 4/9)* | minimum DCF  *= 1/5)* | minimum DCF  *= 4/5)* |
| SVM (C= 0.001, k=1), no PCA | 0.799 | 0.997 | 1.013 |
| SVM (C= 0.001, k=1), with PCA (m=10) | 0.809 | 0.997 | 1.013 |
| SVM (C= 0.001, k=1), with PCA (m=9) | 0.798 | 0.997 | 1.013 |
| SVM (C= 0.001, k=1), with PCA (m=8) | 0.809 | 0.997 | 1.013 |
| SVM (C= 0.001, k=1), with PCA (m=7) | 0.836 | 0.997 | 0.999 |
| SVM (C= 0.01, k=1), no PCA | 0.519 | 0.763 | 0.954 |
| SVM (C= 0.01, k=1), with PCA (m=10) | 0.524 | 0.775 | 0.962 |
| SVM (C= 0.01, k=1), with PCA (m=9) | 0.520 | 0.770 | 0.964 |
| SVM (C= 0.01, k=1), with PCA (m=8) | 0.529 | 0.778 | 0.959 |
| SVM (C= 0.01, k=1), with PCA (m=7) | 0.545 | 0.811 | 0.961 |
| SVM (C= 0.1, k=1), no PCA | 0.460 | 0.746 | 0.763 |
| SVM (C= 0.1, k=1), with PCA (m=10) | 0.512 | 0.768 | 0.909 |
| SVM (C= 0.1, k=1), with PCA (m=9) | 0.476 | 0.742 | 0.749 |
| SVM (C= 0.1, k=1), with PCA (m=8) | 0.481 | 0.773 | 0.834 |
| SVM (C= 1, k=1), with PCA (m=7) | 0.488 | 0.790 | 0.874 |
| SVM (C= 1, k=1), no PCA | 0.602 | 0.834 | 0.914 |
| SVM (C= 1, k=1), with PCA (m=10) | 0.630 | 0.837 | 0.966 |
| SVM (C= 1, k=1), with PCA (m=9) | 0.538 | 0.824 | 0.928 |
| SVM (C= 1, k=1), with PCA (m=8) | 0.499 | 0.757 | 0.783 |
| SVM (C= 1, k=1), with PCA (m=7) | 0.647 | 0.992 | 0.931 |
| SVM (C= 10, k=1), no PCA | 0.847 | 0.982 | 0.977 |
| SVM (C= 10, k=1), with PCA (m=10) | 0.920 | 0.976 | 0.966 |
| SVM (C= 10, k=1), with PCA (m=9) | 1 | 1 | 0.99 |
| SVM (C= 10, k=1), with PCA (m=8) | 0.943 | 1 | 0.972 |
| SVM (C= 10, k=1), with PCA (m=7) | 0.878 | 1 | 0.968 |

The best performance for the Linear SVM, for the considered hyperparameter and constant, is obtained for C= 0.1, k=1 and no PCA.

**Kernel Support Vector Machines**

A kernel functions allows training a SVM in a large dimensional Hilbert space, without requiring to explicitly compute the mapping. The complexity of the primal problem may be too large, but the complexity of the dual problem depends only on the number of training points. In practice, it is possible to compute a linear separation in the expanded space, which corresponds to a non-linear separation surface in the original feature space.

Practically, for Kernel SVM, is a matrix whose elements are:

As stated above, we cannot compute the primal solution, so, we cannot compute the scores in the same way as for the Linear SVM. However, it is possible to compute the score in the following way:

We will use these scores to carry out the computations useful for the calculation of the DCF.

**Polynomial Kernel Support Vector Machines**

For the Polynomial Kernel Support Vector Machines, the kernel is:

To add a regularized bias, we add a constant value k to the kernel function:

|  |  |  |  |
| --- | --- | --- | --- |
|  | minimum DCF  *= 4/9)* | minimum DCF  *= 1/5)* | minimum DCF  *= 4/5)* |
| Poly (C= 0.001, d=2, c=1, k=1), no PCA | 0.887 | 1 | 0.958 |
| Poly (C= 0.001, d=2, k=1), with PCA (m=10) | 0.939 | 1 | 0.995 |
| Poly (C= 0.001, d=2, k=1), with PCA (m=9) | 0.908 | 0.992 | 0.961 |
| Poly (C= 0.001, d=2, k=1), with PCA (m=8) | 0.859 | 1 | 0.988 |
| Poly (C= 0.001, d=2, c=1, k=1), with PCA (m=7) | 0.826 | 0.993 | 0.992 |
| Poly (C= 0.01, d=2, c=1, k=1), no PCA | 0.928 | 1 | 0.985 |
| Poly (C= 0.01, d=2, c=1, k=1), with PCA (m=10) | 0.997 | 0.997 | 0.995 |
| Poly (C= 0.01, d=2, c=1, k=1), with PCA (m=9) | 0.993 | 1 | 0.985 |
| Poly (C= 0.01, d=2, c=1, k=1), with PCA (m=8) | 0.994 | 1 | 0.965 |
| Poly (C= 0.01, d=2, c=1, k=1), with PCA (m=7) | 1 | 1 | 0.996 |
| Poly (C= 0.1, d=2, c=1, k=1), no PCA | 0.896 | 0.992 | 0.940 |
| Poly (C= 0.1, d=2, c=1, k=1), with PCA (m=10) | 0.955 | 1 | 0.954 |
| Poly (C= 0.1, d=2, c=1, k=1), with PCA (m=9) | 1 | 1 | 0.923 |
| Poly (C= 0.1, d=2, c=1, k=1), with PCA (m=8) | 0.923 | 1 | 0.985 |
| Poly (C= 0.1, d=2, c=1, k=1), with PCA (m=7) | 1 | 1 | 1 |
| Poly (C= 1, d=2, c=1, k=1), no PCA | 0.982 | 0.994 | 1 |
| Poly (C= 1, d=2, c=1, k=1), with PCA (m=10) | 0.927 | 0.887 | 1 |
| Poly (C= 1, d=2, c=1, k=1), with PCA (m=9) | 0.967 | 1 | 1 |
| Poly (C= 1, d=2, c=1, k=1), with PCA (m=8) | 0.980 | 0.978 | 0.982 |
| Poly (C= 1, d=2, c=1, k=1), with PCA (m=7) | 0.875 | 1 | 0.986 |
| Poly (C= 10, d=2, c=1, k=1), no PCA | 0.988 | 1 | 0.989 |
| Poly (C= 10, d=2, c=1, c=1, k=1), with PCA (m=10) | 1 | 1 | 0.963 |
| Poly (C= 10, d=2, c=1, k=1), with PCA (m=9) | 0.892 | 0.959 | 0.982 |
| Poly (C= 10, d=2, c=1, k=1), with PCA (m=8) | 0.996 | 0.997 | 0.983 |
| Poly (C= 10, d=2, c=1, k=1), with PCA (m=7) | 0.866 | 0.985 | 0.982 |

**Gaussian Radial Basis Function Kernel Support Vector Machines**

For the Gaussian Radial Basis Function Kernel Support Vector Machines, the kernel is:

To add a regularized bias, we add a constant value k to the kernel function:

|  |  |  |  |
| --- | --- | --- | --- |
|  | minimum DCF  *= 4/9)* | minimum DCF  *= 1/5)* | minimum DCF  *= 4/5)* |
| RBF (C= 0.001, γ=e-3, k=1), no PCA | 1 | 1 | 0.997 |
| RBF (C= 0.001, γ=e-3, k=1), with PCA (m=10) | 1 | 1 | 0.997 |
| RBF (C= 0.001, γ=e-3, k=1), with PCA (m=9) | 1 | 1 | 0.997 |
| RBF (C= 0.001, γ=e-3, k=1), with PCA (m=8) | 1 | 1 | 0.997 |
| RBF (C= 0.001, γ=e-3, k=1), with PCA (m=7) | 1 | 1 | 0.997 |
| RBF (C= 0.01, γ=e-3, k=1), no PCA | 0.766 | 0.904 | 0.954 |
| RBF (C= 0.1, γ=e-3, k=1), no PCA | 0.759 | 0.896 | 0.978 |
| RBF (C= 1, γ=e-3, k=1), no PCA | 0.730 | 0.878 | 0.958 |
| RBF (C= 10, γ=e-3, k=1), no PCA | 0.704 | 0.915 | 0.984 |
| RBF (C= 0.001, γ=e-2, k=1), no PCA | 0.998 | 0.998 | 0.993 |
| RBF (C= 0.01, γ=e-2, k=1), no PCA | 0.895 | 1 | 0.977 |
| RBF (C= 0.1, γ=e-2, k=1), no PCA | 0.791 | 0.933 | 0.986 |
| RBF (C= 1, γ=e-2, k=1), no PCA | 0.762 | 0.935 | 0.978 |
| RBF (C= 10, γ=e-2, k=1), no PCA | 0.746 | 0.962 | 0.996 |
| RBF (C= 0.001, γ=e-1, k=1), no PCA | 0.998 | 0.998 | 0.983 |
| RBF (C= 0.01, γ=e-1, k=1), no PCA | 0.846 | 0.953 | 0.965 |
| RBF (C= 0.1, γ=e-1, k=1), no PCA | 0.840 | 0.953 | 0.965 |
| RBF (C= 1, γ=e-1, k=1), no PCA | 0.824 | 0.933 | 0.967 |
| RBF (C= 10, γ=e-1, k=1), no PCA | 0.837 | 0.949 | 0.979 |
| RBF (C= 0.001, γ=1, k=1), no PCA | 1 | 1 | 0.987 |
| RBF (C= 0.01, γ=1, k=1), no PCA | 0.932 | 0.984 | 0.971 |
| RBF (C= 0.1, γ=1, k=1), no PCA | 0.935 | 0.985 | 0.974 |
| 0.974RBF (C= 1, γ=1, k=1), no PCA | 0.900 | 0.969 | 0.974 |
| RBF (C= 10, γ=1, k=1), no PCA | 0.919 | 0.974 | 0.970 |
| RBF (C= 0.001, γ=e, k=1), no PCA | 1 | 1 | 0.987 |
| RBF (C= 0.01, γ=e, k=1), no PCA | 0.989 | 0.989 | 0.986 |
| RBF (C= 0.1, γ=e, k=1), no PCA | 0.989 | 0.989 | 0.986 |
| RBF (C= 1, γ=e, k=1), no PCA | 0.984 | 0.989 | 0.986 |
| RBF (C= 10, γ=e, k=1), no PCA | 0.98 | 0.989 | 0.986 |
| RBF (C= 0.001, γ=e2, k=1), no PC | 1 | 1 | 0.988 |
| RBF (C= 0.01, γ=e2, k=1), no PCA | 0.988 | 0.989 | 0.986 |
| RBF (C= 0.1, γ=e2, k=1), no PCA | 0.988 | 0.989 | 0.987 |
| RBF (C= 1, γ=e2, k=1), no PCA | 0.989 | 0.989 | 0.986 |
| RBF (C= 10, γ=e2, k=1), no PCA | 0.976 | 0.987 | 0.986 |

**Gaussian Mixture Model**

Gaussian Mixture Models are models obtained as a weighted combination of Gaussians; in this way, it is possible to approximate the density of a Random Variable X when the density of X is not known:

where

are the components means, covariance matrices and weights, respectively.

GMM can be interpreted as the marginal distribution obtained by marginalizing the joint density:

Gi is a discrete hidden random variable that represents the Gaussian component that was responsible for the generation of xi. The joint density can be expressed as a product of the component (cluster) conditional distribution for Xi and the prior distribution for Gi:

The EM (Expectation-Maximization) algorithm can be used to estimate the parameters of a GMM that maximize the likelihood for a training set X. The EM algorithm consists of two steps:

* E-step: compute the posterior probability for each component of the GMM for each sample, using an estimate (Mt, St, wt) of the model parameters. These quantities are also called responsibilities:
* M-step: Update the model parameters. It is possible to use the statistics:

to obtain the new parameters:

The EM algorithm requires an initial guess for the GMM parameters. The Linde-Buzo-Gray (LBG) algorithm allows to incrementally construct a GMM with 2G components from a GMM with G components. Starting with a single-component GMM (i.e., a Gaussian density), we can build a 2-component GMM and then use the EM algorithm to estimate a ML solution for the 2-components model. We can then split the 2 components to obtain a 4-components GMM, and re-apply the EM algorithm to estimate its parameters, and so on.

We will use the Maximum Likelihood solution for a Gaussian density as starting point: (w, µ, Σ) = (1, µ, Σ), with µ, Σ mean and covariance matrix of the samples of the class in question (obviously, we will apply the model to both classes separately).

The way we will adopt to split the GMM consists in replacing component (wg, µg, Σg) with two components:

We will compute the displacement vector by taking the leading eigenvector of Σg, scaled by the square root of the corresponding eigenvalue, multiplied by some factor α (we will adopt α=1).

**Diagonal Covariance Gaussian Mixture Model**

**Tied Covariance Gaussian Mixture Model**