PROJECT – WINE QUALITY DETECTION

REPORT

Leonardo Rolandi, Flavio Patti

S301216@studenti.polito.it, s301104@studenti.polito.it

July 13, 2022

• INTRODUCTION

The project requires discriminating between good and bad quality wines.
We have available two main files: the training set that is stored in the file “Train.txt” and contains 1226 samples of class 0 and 613 samples of class 1 and the evaluation set that is stored in the file “Test.txt” and contains 1158 samples of class 0 and 613 of class 1. The class labels used are 0 (bad wines) and 1 (good wines). Our goals will be to perform classification for the proposed task by employing different models, to analyze results and to motivate our choices.

There are 11 features, that represent physical properties of the wine.

Input variables (based on physicochemical tests):
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

Output variable (based on sensory data):
12 - quality (score between 0 and 10)

For the project the classes have been binarized. Feature 12 is simply 0 (low quality, <= 5) or 1 (high quality, >= 7). To simplify the problem, wines with quality 6 have been removed.

• FEATURE ANALYSIS AND PRE-PROCESSING

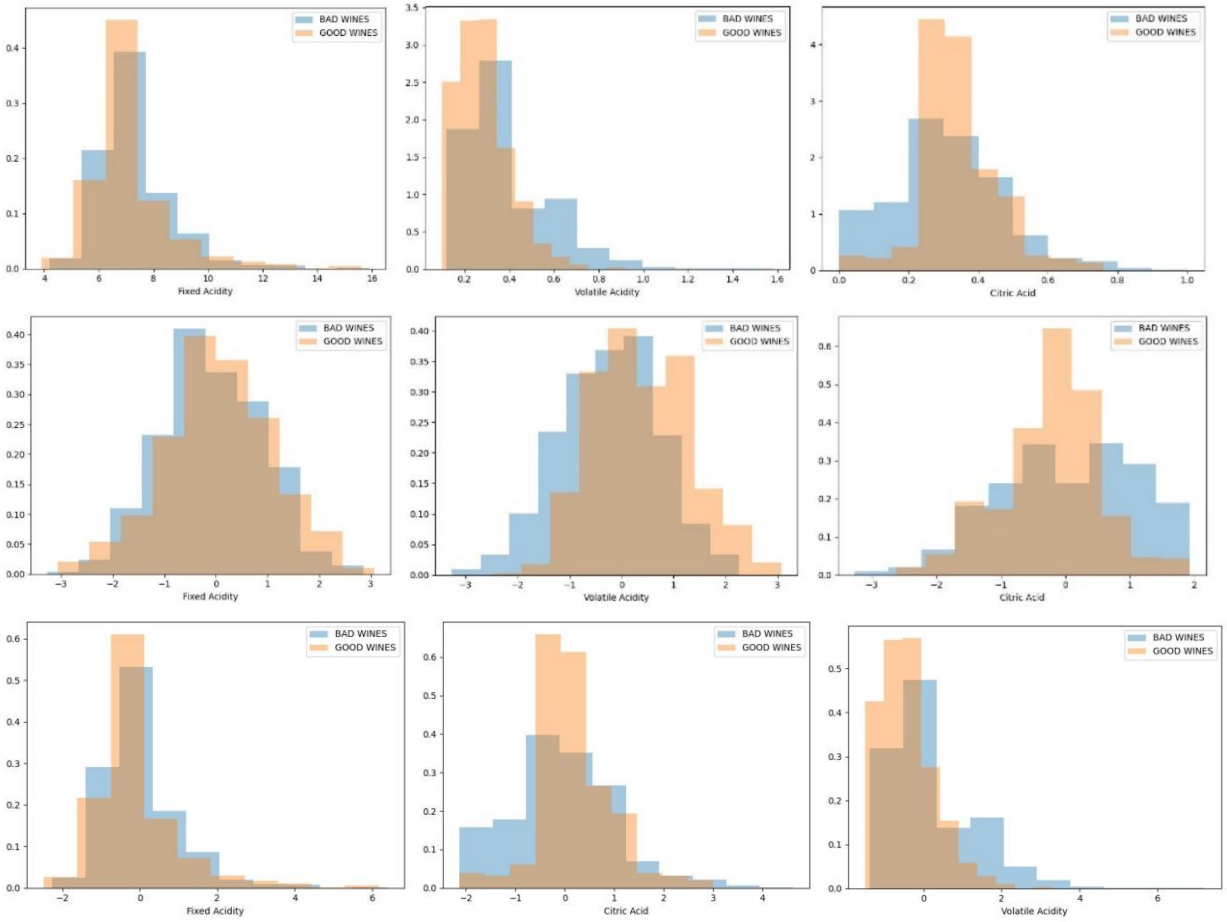
To begin our analyses we can plot histograms of the training set features. We can see that in many cases the graphs have outliers and in general do not have a good Gaussian shape, for this reason it is possible to adopt some forms of data pre-processing such as Gaussianization to make features more regular. Gaussianization is a procedure that allows mapping a set of features to values whose empirical cumulative distribution function is well approximated by a Gaussian c.d.f.
The process consists in mapping the features to a uniform distribution and then transforming the mapped features through the inverse of Gaussian cumulative distribution function.

Furthermore, since dataset features values turned out to be high, to simplify computations and to avoid overflow problems that may arise (especially when working with exponentials) data has been pre-processed by means of Z-normalization:

$$z_i = \frac{x_i - \mu}{\sigma}, \ \forall i \in \{1, ..., n\}$$

where xi is the observed value for the i-th sample, μ is the mean vector of the samples computed along columns and σ is the standard deviation vector of the samples computed along columns. This procedure allows centering and scaling data so that the Z-normalized R.V. Z will have a standard distribution with zero-mean and one-variance.

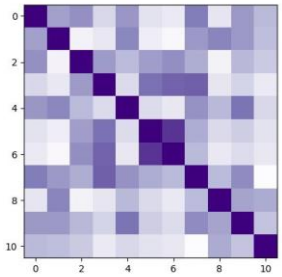
In the following there are three histograms of the first three attributes of the wines in the first row, the corresponding mapping in Gaussianization in the second row and Z-normalization in the third row.



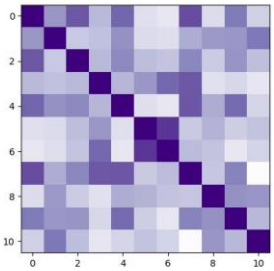
All the other histograms and scatters are reported in the Graphics folder of the project.

Furthermore, heatmaps of training set features have been plotted to analyze features correlation. The heatmaps show the Pearson correlation coefficient (darker color implies larger value for the coefficient)

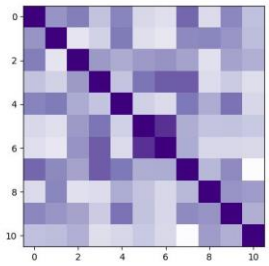
$$\frac{Cov(X,Y)}{\sqrt{Var(X)}\sqrt{Var(Y)}}$$



Bad wines correlation



Good wines correlation



Global wines correlation

We can see that some features are quite correlated, above all the 6th with the 7th, and the 8th with the 1st, so we can suppose that using PCA will give us some benefits.

In order to perform our analysis, we will try to adopt two approaches, at least in the raw data processing. In the first one we will split the training set into training and validation subsets (this approach is called single-fold in the following), but we will also exploit K-fold cross-validation approach, that consists in partitioning training data in k subsets, using k-1 subsets for training and the remaining one for validation, and then iteratively change this division to make every subset being used for validation once. The single-fold consists of 66.6% training data and 33.3% validation data. The K-fold is implemented with K=5. In both cases data has been shuffled before splitting (but in the k-fold, data in different classifiers remain consistent since the seed is always equal to 0).

In the following we will see that min DCF using single-fold are often slightly better than the ones using K-fold, even if using a k-fold approach provides in general more robust and versatile results, since with our PCs it really takes too much time, in the following we will take our decisions based on a single-fold approach.

We will consider three different applications: a uniform prior application and two unbalanced applications where the prior is biased towards one of the two classes:

$$(\widetilde{\pi},\, C_{fp},\, C_{fp}) = (\ 0.5,\ 1,\ 1\ )$$
$$(\widetilde{\pi},\, C_{fp},\, C_{fp}) = (\ 0.1,\ 1,\ 1\ )$$
$$(\widetilde{\pi},\, C_{fp},\, C_{fp}) = (\ 0.9,\ 1,\ 1\ )$$

Our target application will be mainly the balanced one. We want to find the most promising approach, so we can measure performances through the metric of the normalized minimum Detection Cost Function (min DCF), which measures the cost that we would pay if we made optimal decisions using the recognizer scores.

Now we analyze preprocessing techniques such Gaussianization, Z-Normalization and PCA for some of the classifiers, since LDA doesn't make much sense given that we are training a binary classificatory and LDA collapses the dimensions into 1.

PCA is a dimensionality reduction technique that computes a linear mapping from the n-dimensional feature space to a m-dimensional space, with m << n, while preserving the directions with highest variance. We'll exploit PCA to reduce the training set dimensionality up to 10-9 and see how models behave. We expect that dimensionality reduction will not cause major losses of information and at the same time it will allow us to reduce the number of parameters to estimate.

GAUSSIANIZATION

Train	Single Fold		
minDCF	Prior = 0.5	Prior = 0.1	Prior = 0.9
MVG	0.269	0.802	0.752
Naïve	0.451	0.844	0.805
Tied MVG	0.350	0.861	0.817
Tied Naïve	0.436	0.835	0.906
Linear LR, l= 0	0.340	0.858	0.754
Balanced Linear LR, l=0	0.341	0.840	0.783
Quadratic LR, l=0	0.273	0.645	0.646
Balanced LR, l=0	0.274	0.642	0.600
Linear LR, l= 1e-06	0.340	0.858	0.754
Balanced Linear LR, l=1e-06	0.341	0.840	0.783
Quadratic LR, l=1e-06	0.273	0.645	0.646
Balanced LR, l=1e-06	0.274	0.642	0.603
Linear LR, l= 1e-03	0.345	0.830	0.757
Balanced Linear LR, l=1e0-3	0.341	0.823	0.793
Quadratic LR, l=1e-03	0.275	0.650	0.629
Balanced LR, l=1e-03	0.280	0.638	0.600
Linear LR, l= 1	0.426	0.826	0.887
Balanced Linear LR, l=1	0.423	0.823	0.862
Quadratic LR, l=1	0.314	0.785	0.867
Balanced LR, l=1	0.305	0.816	0.867
SVM Linear, K=1, C=0.1	0.334	0.836	0.870
Balanced SVM Linear, K=1, C = 0.1	0.351	0.878	0.766
SVM Linear, K=1, C=1	0.341	0.840	0.889
Balanced SVM Linear, K=1, C = 1	0.351	0.878	0.887
SVM Linear, K=1, C=10	0.349	0.859	0.887
Balanced SVM Linear, K=1, C = 10	0.353	0.884	0.805
SVM Linear, K=10, C=0.1	0.337	0.831	0.862
Balanced SVM Linear, K=10, C = 0.1	0.351	0.878	0.766

SVM Linear, K=10, C=1	0.341	0.840	0.889
Balanced SVM Linear, K=10, C = 1	0.353	0.878	0.786
SVM Linear, K=10, C=10	0.349	0.859	0.887
Balanced SVM Linear, K=10, C = 10	0.351	0.884	0.805
SVM Polynomial Kernel, K=1, C=0.1,c=0	0.844	0.934	0.923
Balanced SVM Polynomial Kernel, K=1,C=0.1,c=0	0.431	0.798	0.925
SVM Polynomial Kernel, K=1, C=0.1,c=1	0.802	0.608	0.680
Balanced SVM Polynomial Kernel, K=1,C=0.1,c=1	0.250	0.708	0.591
SVM Polynomial Kernel, K=1, C=1,c=0	0.840	0.791	0.961
Balanced SVM Polynomial Kernel, K=1,C=1,c=0	0.838	0.807	0.889
SVM Polynomial Kernel, K=1, C=1,c=1	0.803	0.859	0.588
Balanced SVM Polynomial Kernel, K=1,C=1,c=1	0.265	0.743	0.615
SVM Polynomial Kernel, K=1, C=10,c=0	0.840	0.802	0.935
Balanced SVM Polynomial Kernel, K=1,C=10,c=0	0.407	0.793	0.891
SVM Polynomial Kernel, K=1, C=10,c=1	0.263	0.854	0.579
Balanced SVM Polynomial Kernel, K=1,C=10,c=1	0.799	0.911	0.622
SVM RBF Kernel, K=1, C=0.1, g = 1	0.775	0.864	0.543
Balanced SVM RBF Kernel, K=1, C=0.1, g = 1	0.253	0.695	0.687
SVM RBF Kernel, K = 1, C = 0.1, g=10	0.800	0.878	0.608
Balanced SVM RBF Kernel, K = 1, C = 0.1, g=10	0.795	0.878	0.608
SVM RBF Kernel, K = 1, C = 1, g=1	0.777	0.588	0.512
Balanced SVM RBF Kernel, K=1, C=1, g = 1	0.775	0.852	0.531
SVM RBF Kernel, K=1, C=1, g = 10	0.790	0.878	0.608
Balanced SVM RBF Kernel, K=1, C=1, g = 10	0.800	0.878	0.620
SVM RBF Kernel, K = 1, C = 10, g=1	0.232	0.553	0.480
Balanced SVM RBF Kernel, K=1, C=10, g = 1	0.777	0.842	0.480
SVM RBF Kernel, K=1, C=10, g = 10	0.800	0.878	0.620
Balanced SVM RBF Kernel, K=1, C=10, g = 10	0.800	0.878	0.620
GMM version=full, M=2, psi=0.01	0.320	0.769	0.817
GMM version=full, M=4, psi=0.01	0.259	0.698	0.528
GMM version=full, M=8, psi=0.01	0.230	0.642	0.617
GMM version=diagonal, M=2, psi=0.01	0.446	0.844	0.762
GMM version=diagonal, M=4, psi=0.01	0.291	0.673	0.682
GMM version=diagonal, M=8, psi=0.01	0.302	0.722	0.680
GMM version=tied, M=2, psi=0.01	0.320	0.769	0.817
GMM version=tied, M=4, psi=0.01	0.259	0.698	0.528
GMM version=tied, M=8, psi=0.01	0.230	0.642	0.617

Z-NORMALIZATION

Train	Single Fold		
minDCF	Prior = 0.5	Prior = 0.1	Prior = 0.9
MVG	0.304	0.777	0.812
Naïve	0.437	0.818	0.875
Tied MVG	0.334	0.779	0.733
Tied Naïve	0.412	0.832	0.901
Linear LR, l= 0	0.361	0.811	0.764
Balanced Linear LR, l=0	0.368	0.790	0.766
Quadratic LR, l=0	0.264	0.713	0.783
Balanced LR, l=0	0.276	0.701	0.783
Linear LR, l= 1e-06	0.361	0.811	0.764
Balanced Linear LR, l=1e-06	0.368	0.790	0.766
Quadratic LR, l=1e-06	0.264	0.718	0.783
Balanced LR, l=1e-06	0.276	0.706	0.783
Linear LR, l= 1e-03	0.358	0.806	0.762
Balanced Linear LR, l=1e0-3	0.368	0.774	0.769
Quadratic LR, l=1e-03	0.262	0.691	0.714
Balanced LR, l=1e-03	0.270	0.691	0.728
Linear LR, l= 1	0.392	0.849	0.867
Balanced Linear LR, l=1	0.392	0.849	0.843
Quadratic LR, l=1	0.375	0.777	0.906
Balanced LR, l=1	0.372	0.833	0.896
SVM Linear, K=1, C=0.1	0.342	0.785	0.750
Balanced SVM Linear, K=1, C = 0.1	0.342	0.832	0.733
SVM Linear, K=1, C=1	0.342	0.802	0.745
Balanced SVM Linear, K=1, C = 1	0.356	0.831	0.766
SVM Linear, K=1, C=10	0.344	0.789	0.737
Balanced SVM Linear, K=1, C = 10	0.354	0.846	0.747
SVM Linear, K=10, C=0.1	0.342	0.785	0.754
Balanced SVM Linear, K=10, C = 0.1	0.344	0.831	0.747
SVM Linear, K=10, C=1	0.342	0.794	0.747
Balanced SVM Linear, K=10, C = 1	0.356	0.831	0.766
SVM Linear, K=10, C=10	0.342	0.789	0.745
Balanced SVM Linear, K=10, C = 10	0.354	0.846	0.747
SVM Polynomial Kernel, K=1, C=0.1,c=0	0.464	0.832	0.947

Balanced SVM Polynomial Kernel, K=1,C=0.1,c=0	0.830	0.904	0.932
SVM Polynomial Kernel, K=1, C=0.1,c=1	0.258	0.939	0.692
Balanced SVM Polynomial Kernel, K=1,C=0.1,c=1	0.792	0.941	0.735
SVM Polynomial Kernel, K=1, C=1,c=0	0.442	0.941	0.894
Balanced SVM Polynomial Kernel, K=1,C=1,c=0	0.827	0.973	0.927
SVM Polynomial Kernel, K=1, C=1,c=1	0.255	0.695	0.764
Balanced SVM Polynomial Kernel, K=1,C=1,c=1	0.254	0.773	0.740
SVM Polynomial Kernel, K=1, C=10,c=0	0.825	0.835	0.894
Balanced SVM Polynomial Kernel, K=1,C=10,c=0	0.444	0.927	0.935
SVM Polynomial Kernel, K=1, C=10,c=1	0.254	0.703	0.790
Balanced SVM Polynomial Kernel, K=1,C=10,c=1	0.264	0.949	0.786
SVM RBF Kernel, K=1, C=0.1, g = 1	0.798	0.762	0.560
Balanced SVM RBF Kernel, K=1, C=0.1, g = 1	0.799	0.994	0.620
SVM RBF Kernel, K = 1, C = 0.1, g=10	0.821	0.994	0.581
Balanced SVM RBF Kernel, K = 1, C = 0.1, g=10	0.868	0.878	0.581
SVM RBF Kernel, K = 1, C = 1, g=1	0.786	0.994	0.569
Balanced SVM RBF Kernel, K=1, C=1, g = 1	0.257	0.994	0.507
SVM RBF Kernel, K=1, C=1, g = 10	0.409	0.878	0.581
Balanced SVM RBF Kernel, K=1, C=1, g = 10	0.868	0.604	0.581
SVM RBF Kernel, K = 1, C = 10, g=1	0.868	0.859	0.536
Balanced SVM RBF Kernel, K=1, C=10, g = 1	0.250	0.994	0.536
SVM RBF Kernel, K=1, C=10, g = 10	0.818	0.604	0.581
Balanced SVM RBF Kernel, K=1, C=10, g = 10	0.818	0.604	0.581
GMM version=full, M=2, psi=0.01	0.292	0.694	0.762
GMM version=full, M=4, psi=0.01	0.295	0.635	0.617
GMM version=full, M=8, psi=0.01	0.262	0.661	0.798
GMM version=diagonal, M=2, psi=0.01	0.430	0.818	0.853
GMM version=diagonal, M=4, psi=0.01	0.354	0.774	0.807
GMM version=diagonal, M=8, psi=0.01	0.334	0.758	0.788
GMM version=tied, M=2, psi=0.01	0.292	0.694	0.762
GMM version=tied, M=4, psi=0.01	0.295	0.635	0.617
GMM version=tied, M=8, psi=0.01	0.262	0.661	0.798

PCA, M = 9

Train	Single Fold			K-Fold(K=5)		
minDCF	Prior = 0.5	Prior = 0.1	Prior = 0.9	Prior = 0.5	Prior = 0.1	Prior = 0.9
MVG	0.326	0.803	0.824	0.323	0.832	0.817
Naïve	0.371	0.856	0.846	0.385	0.866	0.871
Tied MVG	0.333	0.809	0.668	0.338	0..840	0.733
Tied Naïve	0.337	0.784	0.661	0.345	0.815	0.724
Linear LR, l= 0	0.360	0.803	0.754	0.348	0.826	0.654
Balanced Linear LR, l=0	0.365	0.798	0.745	0.355	0.832	0.648
Quadratic LR, l=0	0.393	0.792	0.774	0.388	0.845	0.819
Balanced LR, l=0	0.396	0.787	0.762	0.375	0.854	0.752
Linear LR, l= 1e-06	0.366	0.803	0.754	0.347	0.823	0.656
Balanced Linear LR, l=1e-06	0.363	0.798	0.745	0.356	0.832	0.645
Quadratic LR, l=1e-06	0.392	0.827	0.764	0.388	0.844	0.887
Balanced LR, l=1e-06	0.391	0.800	0.706	0.389	0.840	0.744
Linear LR, l= 1e-03	0.355	0.784	0.699	0.347	0.821	0.700
Balanced Linear LR, l=1e0-3	0.365	0.789	0.627	0.351	0.834	0.668
Quadratic LR, l=1e-03	0.416	0.809	0.752	0.377	0.840	0.815
Balanced LR, l=1e-03	0.415	0.794	0.759	0.376	0.838	0.760
Linear LR, l= 1	0.391	0.874	0.968	0.465	0.961	0.993
Balanced Linear LR, l=1	0.390	0.949	0.961	0.459	1	0.997
Quadratic LR, l=1	0.384	0.826	0.805	0.396	0.828	0.874
Balanced LR, l=1	0.402	0.800	0.790	0.369	0.840	0.765

PCA, M = 10

Train	Single Fold			K-Fold(K=5)		
minDCF	Prior = 0.5	Prior = 0.1	Prior = 0.9	Prior = 0.5	Prior = 0.1	Prior = 0.9
MVG	0.325	0.793	0.812	0.323	0.832	0.817
Naïve	0.366	0.850	0.860	0.385	0.866	0.871
Tied MVG	0.343	0.814	0.658	0.338	0.840	0.733
Tied Naïve	0.341	0.779	0.661	0.345	0.815	0.724
Linear LR, l= 0	0.364	0.808	0.757	0.348	0.826	0.654
Balanced Linear LR, l=0	0.370	0.794	0.747	0.355	0.832	0.648
Quadratic LR, l=0	0.424	0.810	0.908	0.388	0.845	0.819
Balanced LR, l=0	0.405	0.824	0.781	0.375	0.854	0.752
Linear LR, l= 1e-06	0.364	0.808	0.757	0.347	0.823	0.656
Balanced Linear LR, l=1e-06	0.370	0.794	0.747	0.356	0.832	0.645
Quadratic LR, l=1e-06	0.426	0.800	0.908	0.388	0.844	0.887
Balanced LR, l=1e-06	0.370	0.834	0.762	0.389	0.840	0.744
Linear LR, l= 1e-03	0.350	0.784	0.694	0.347	0.821	0.700
Balanced Linear LR, l=1e0-3	0.365	0.789	0.634	0.351	0.834	0.668
Quadratic LR, l=1e-03	0.388	0.826	0.800	0.377	0.840	0.815
Balanced LR, l=1e-03	0.407	0.830	0.747	0.376	0.838	0.760
Linear LR, l= 1	0.391	0.874	0.968	0.465	0.961	0.993
Balanced Linear LR, l=1	0.390	0.949	0.961	0.459	1	0.997
Quadratic LR, l=1	0.419	0.782	0.810	0.396	0.828	0.874
Balanced LR, l=1	0.376	0.825	0.740	0.369	0.840	0.765

- TEST DIFFERENT CLASSIFIERS

GAUSSIAN CLASSIFIERS

We start considering generative models and in particular Gaussian classifiers (MVG classifier, MVG classifier with Naive Bayes assumption, MVG classifier with tied covariance). All of them assume gaussian distributed data, given the class:

$$\boldsymbol{X}|C=c \sim \mathcal{N}(\boldsymbol{\mu}_c, \boldsymbol{\Sigma}_c)$$

The MVG classifier with tied covariance matrices assumes that each class has its own mean μc, but the covariance matrix Σ is the same for all classes. The Naive Bayes Gaussian classifier corresponds to a MVG classifier with diagonal covariance matrices for the different classes. The Naive Bayes assumption supposes that features are independently distributed and, if this is the case, then the features are also uncorrelated and will have zero-covariances, which are the elements off-diagonal on the covariance matrices and this justify the diagonalization. Since some of our features are highly correlated, we expect covariances off-diagonal not to be approximately zero, so we have no guarantee that the Naive Bayes assumption will actually work out well, we may get not-optimal results. Instead, the MVG classifier with full covariance matrices and the MVG classifier with tied covariance matrices are able to capture correlations and may output better results, also considering the sufficient number of samples at our disposal.

Train	Single Fold			K-Fold(K=5)		
minDCF	Prior = 0.5	Prior = 0.1	Prior = 0.9	Prior = 0.5	Prior = 0.1	Prior = 0.9
MVG	0.304	0.777	0.812	0.312	0.779	0.842
Naïve	0.437	0.818	0.875	0.420	0.846	0.921
Tied MVG	0.334	0.779	0.733	0.333	0.811	0.747
Tied Naïve	0.412	0.832	0.901	0.403	0.866	0.932

As we can expect we have the best result with the MVG classifier when the priority is uniform at 0.5 for both single fold and k-fold cases.
Comparisons can be made also with MVG after Gaussianization and Z-normalization where in the first case minDCF = 0.269 so we obtain a relevant improvement while in the second case the minDCF is practically the same.
More in general we can see that using Gaussianization for Naïve, Tied MVG, Tied Naïve classifiers, results are slightly worse in all the applications while with Z-normalization the results are identical.

LOGISTIC REGRESSION

Now we turn our attention to discriminative models. We start considering Linear Logistic Regression, the function that we need to minimize is:

$$J(\boldsymbol{w}, b) = \frac{\lambda}{2}\|\boldsymbol{w}\|^2 + \frac{1}{n}\sum_{i=1}^{n} \log\left(1 + e^{-z_i(\boldsymbol{w}^T \boldsymbol{x}_i + b)}\right)$$

We consider also regularized Linear Logistic Regression. Since classes are unbalanced, we change the objective function that we need to minimize so that costs of different classes are re-balanced, and we have chosen to use only a balanced prior of 0.5:

$$J(\boldsymbol{w}, b) = \frac{\lambda}{2}\|\boldsymbol{w}\|^2 + \frac{\pi_T}{n_T}\sum_{i=1|c_i=1}^{n} \log\left(1 + e^{-z_i(\boldsymbol{w}^T \boldsymbol{x}_i + b)}\right) + \frac{1-\pi_T}{n_F}\sum_{i=1|c_i=0}^{n} \log\left(1 + e^{-z_i(\boldsymbol{w}^T \boldsymbol{x}_i + b)}\right)$$

The model parameters are (w, b). The model assumes that decision rules are linear hyperplanes orthogonal to w. The regularization term λ/2 *||w||^2 helps obtaining simpler solutions in terms of lower norm of w. This will lead to not so certain models, helping in reducing over-fitting of the training data. The hyper-parameter λ allows for a tradeoff between:

1. poor separation of classes and simpler solutions with small norm of w, when λ >>0

2. good separation of classes but poor generalization on unseen data, when λ ≃ 0

• LINEAR LOGISTIC REGRESSION

Train	Single Fold			K-Fold(K=5)		
minDCF	Prior = 0.5	Prior = 0.1	Prior = 0.9	Prior = 0.5	Prior = 0.1	Prior = 0.9
Lambda = 0	0.365	0.808	0.757	0.348	0.824	0.651
Lambda = 1e-6	0.365	0.808	0.757	0.345	0.824	0.653
Lambda =1e-3	0.350	0.779	0.697	0.346	0.823	0.702
Lambda = 1	0.391	0.874	0.968	0.465	0.961	0.993

• BALANCED LINEAR LOGISTIC REGRESSION

Train	Single Fold			K-Fold(K=5)		
minDCF	Prior = 0.5	Prior = 0.1	Prior = 0.9	Prior = 0.5	Prior = 0.1	Prior = 0.9
Lambda = 0	0.365	0.794	0.752	0.357	0.824	0.638
Lambda = 1e-6	0.368	0.794	0.752	0.357	0.823	0.639
Lambda =1e-3	0.365	0.789	0.627	0.352	0.837	0.668
Lambda = 1	0.390	0.949	0.961	0.459	1	0.997

In this case we notice that Linear Logistic Regression gives in general better results with respect to the balanced version so we can say that in this case balancing data don't give us advantages.

Furthermore if we define

$$\phi(\boldsymbol{x}) = \begin{bmatrix} \mathrm{vec}(\boldsymbol{x}\boldsymbol{x}^T) \\ \boldsymbol{x} \end{bmatrix}$$

and

$$\boldsymbol{w} = \begin{bmatrix} \mathrm{vec}(\boldsymbol{A}) \\ \boldsymbol{b} \end{bmatrix}$$

where vec(M) is the operator that stacks the columns of matrix M, then the posterior log-likelihood ratio can be expressed as

$$s(\boldsymbol{x}, \boldsymbol{w}, c) = \boldsymbol{w}^T \phi(\boldsymbol{x}) + c \quad (1)$$

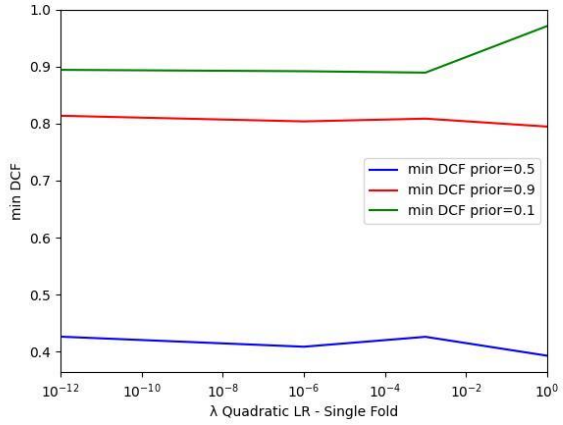
We can thus train a LR model using feature vectors ϕ(x) rather than x. We will obtain a model that has linear separation surface in the space defined by the mapping ϕ. This space is also called expanded feature space. The LR model (both binary and multiclass) allows computing linear separation rules for the transformed features ϕ(x). Since expressions (1) correspond to quadratic forms in the original feature space, we are actually estimating quadratic separation surfaces in the original space.

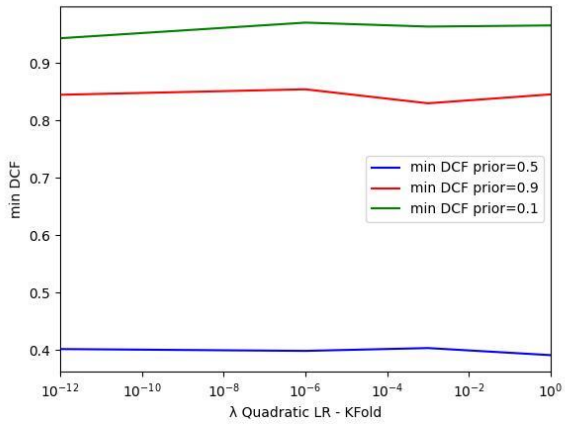
- QUADRATIC LOGISTIC REGRESSION

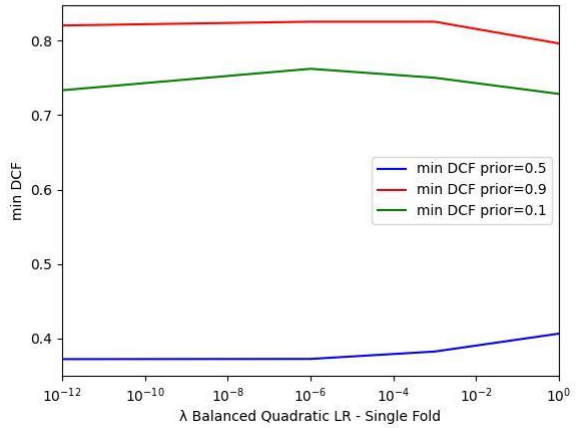
Train	Single Fold			K-Fold(K=5)		
minDCF	Prior = 0.5	Prior = 0.1	Prior = 0.9	Prior = 0.5	Prior = 0.1	Prior = 0.9
Lambda = 0	0.426	0.813	0.894	0.401	0.844	0.943
Lambda = 1e-6	0.408	0.803	0.891	0.398	0.854	0.970
Lambda =1e-3	0.426	0.808	0.889	0.403	0.829	0.963
Lambda = 1	0.392	0.794	0.971	0.391	0.845	0.965

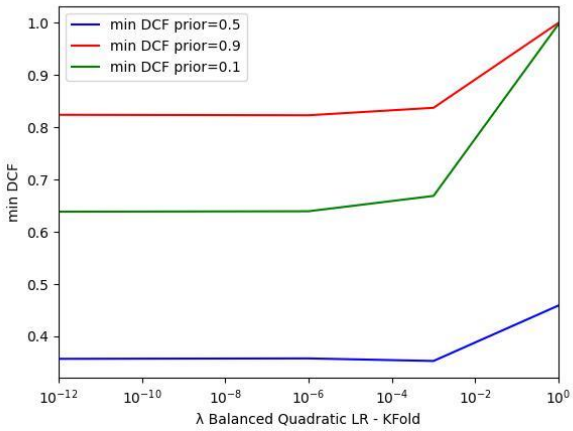
- BALANCED QUADRATIC LOGISTIC REGRESSION

Train	Single Fold			K-Fold(K=5)		
minDCF	Prior = 0.5	Prior = 0.1	Prior = 0.9	Prior = 0.5	Prior = 0.1	Prior = 0.9
Lambda = 0	0.372	0.820	0.733	0.386	0.841	0.852
Lambda = 1e-6	0.372	0.825	0.762	0.375	0.860	0.888
Lambda =1e-3	0.382	0.825	0.750	0.389	0.846	0.861
Lambda = 1	0.406	0.796	0.728	0.395	0.836	0.845


min DCF prior=0.5
min DCF prior=0.9
min DCF prior=0.1
min DCF
λ Quadratic LR - Single Fold


min DCF prior=0.5
min DCF prior=0.9
min DCF prior=0.1
min DCF
λ Quadratic LR - KFold


min DCF prior=0.5
min DCF prior=0.9
min DCF prior=0.1
min DCF
λ Balanced Quadratic LR - Single Fold


min DCF prior=0.5
min DCF prior=0.9
min DCF prior=0.1
min DCF
λ Balanced Quadratic LR - KFold

SVM

Let's see now SVM. We will consider the linear SVM, the polynomial quadratic kernel and the Radial Basis Function kernel formulations. We start considering a linear model that does not balance the two classes. To solve the SVM problem, we can consider the dual formulation:

$$J^D(\boldsymbol{\alpha}) = -\frac{1}{2}\boldsymbol{\alpha}^T \boldsymbol{H}\boldsymbol{\alpha} + \boldsymbol{\alpha}^T \mathbf{1}$$

$$subject\ to\ 0 \leq \alpha_i \leq C, \forall i \in \{1, ..., n\}, \sum_{i=1}^{n} \alpha_i z_i = 0$$

where n is the number of training samples, C is a hyper-parameter, 1 is a n-dimensional vector of ones, zi is the class label for the i-th sample encoded as:

$$z_i = \begin{cases} +1, & if\ x_i\ belongs\ to\ class\ 1 \\ -1, & if\ x_i\ belongs\ to\ class\ 0 \end{cases}$$

and H is a matrix whose elements are:

$$\boldsymbol{H}_{i,j} = z_i z_j \boldsymbol{x}_i^T \boldsymbol{x}_j$$

From the dual formulation we then derive the primal solution. Due to the fact that we use the L-BFGS-B algorithm and it is only able to handle box constraints, we have to modify the formulation to already include the second constraint. We will have that the modified dual formulation corresponds to

$$\widehat{J}^D(\boldsymbol{\alpha}) = -\frac{1}{2}\boldsymbol{\alpha}^T \widehat{\boldsymbol{H}}\boldsymbol{\alpha} + \boldsymbol{\alpha}^T \mathbf{1}$$

$$subject\ to\ 0 \leq \alpha_i \leq C, \forall i \in \{1, ..., n\}$$

and, since we use the mapping

$$\widehat{\boldsymbol{x}}_i = \begin{bmatrix} \boldsymbol{x}_i \\ K \end{bmatrix}$$

with K=1 or 10, the matrix H is modified accordingly:

$$\widehat{\boldsymbol{H}}_{i,j} = z_i z_j (\boldsymbol{x}_i^T \boldsymbol{x}_j + 1) \quad (2)$$

We can also consider the balanced version, also in this case only with prior=0.5. Balancing is done by considering a different value of C for the different classes in the box constraint of the dual formulation:

$$0 \leq \alpha_i \leq C_i, \forall i \in \{1, ..., n\}$$

where Ci = CT for samples of the True class and Ci = CF for samples of the False class.

We select CT = C * (πT/πT emp) and CF = C * (πF/πF emp) where πT emp and πF emp are the empirical priors.

- LINEAR SVM

Train	Single Fold			K-Fold(K=5)		
minDCF	Prior = 0.5	Prior = 0.1	Prior = 0.9	Prior = 0.5	Prior = 0.1	Prior = 0.9
K=1, C = 0.1	0.411	0.858	0.855	0.401	0.862	0.884
K=1, C = 1	0.377	0.811	0.790	0.610	0.931	0.984
K=1, C=10	0.704	0.974	0.872	0.844	0.991	0.993
K=10, C=0.1	0.354	0.816	0.735	0.478	0.978	0.980
K=10, C=1	0.763	1	0.915	0.559	0.998	0.977
K=10, C=10	0.765	1	0.959	0.870	1	0.986

- BALANCED LINEAR SVM

Train	Single Fold			K-Fold(K=5)		
minDCF	Prior = 0.5	Prior = 0.1	Prior = 0.9	Prior = 0.5	Prior = 0.1	Prior = 0.9
K=1, C = 0.1	0.406	0.839	0.836	0.412	0.870	0.943
K=1, C = 1	0.384	0.816	0.814	0.567	0.995	0.993
K=1, C=10	0.978	1	0.990	0.883	1	0.998
K=10, C=0.1	0.346	0.806	0.747	0.411	0.873	0.928
K=10, C=1	0.351	0.821	0.694	0.644	0.926	0.990
K=10, C=10	0.670	0.969	0.882	0.872	1	0.990

In this case we can see that in contrast to what happens in the logistic regression, in the linear SVM balancing the data generally helps to have better results.

We now analyze the non-linear formulations. In SVM, the non-linearity is obtained through an implicit expansion of the features in a higher dimensional space. The dual SVM formulation, as seen in (2), depends on the training samples through the dot product and it's possible to compute scores through scalar products between training and evaluation samples. For this reason, it's not required to explicitly compute the feature expansion, it's enough to be able to compute the scalar product between the expanded features, the so called kernel function k:

$$k(\boldsymbol{x}_i, \boldsymbol{x}_j) = \boldsymbol{\phi}(\boldsymbol{x}_i)^T \boldsymbol{\phi}(\boldsymbol{x}_j)$$

We will employ two different kernels:

1. Polynomial kernel of degree d = 2:

$$k(\boldsymbol{x}_i, \boldsymbol{x}_j) = \quad (\boldsymbol{x}_i^T \boldsymbol{x}_j + c)^d$$

2. Radial Basis Function (RBF) kernel:

$$k(\boldsymbol{x}_i, \boldsymbol{x}_j) = \quad e^{-\gamma \|\boldsymbol{x}_i - \boldsymbol{x}_j\|^2}$$

- QUADRATIC SVM – POLYNOMIAL KERNEL

No balancing

Train	Single Fold		
minDCF	Prior = 0.5	Prior = 0.1	Prior = 0.9
K=1,C=0.1,D=2,c=0	0.957	1	0.998
K=1,C=0.1D=2,c=1	0.929	1	0.998
K=1,C=1,D=2,c=0	0.963	0.975	0.983
K=1, C=1,D=2, c=1	0.780	0.954	0.947
K=1,C=10,D=2, c=0	0.936	1	0.937
K=1,C=10,D=2,c=1	0.815	1	0.993
K=10,C=0.1,D=2,c=0	0.952	0.975	0.990
K=10,C=0.1,D=2,c=1	0.800	1	0.987
K=10,C=1,D=2,c=0	0.993	1	1
K=10, C=1,D=2,c=1	0.955	0.980	0.983
K=10, C=10,D=2,c=0	0.840	0.980	0.976
K=10,C=10,D=2,c=1	0.940	0.990	0.986

Balancing

Train	Single Fold		
minDCF	Prior = 0.5	Prior = 0.1	Prior = 0.9
K=1,C=0.1,D=2,c=0	0.912	1	0.993
K=1,C=0.1,D=2,c=1	0.740	1	0.989
K=1,C=1,D=2,c=0	0.814	1	0.984
K=1, C=1,D=2, c=1	0.987	1	0.998
K=1,C=10,D=2, c=0	0.955	0.985	0.964
K=1,C=10,D=2,c=1	0.949	0.976	0.990
K=10,C=0.1,D=2,c=0	0.923	1	0.930
K=10,C=0.1,D=2,c=1	0.774	0.990	0.954
K=10,C=1,D=2,c=0	0.805	0.985	0.988
K=10, C=1,D=2,c=1	0.772	1	0.942
K=10, C=10,D=2,c=0	0.782	0.990	0.942
K=10,C=10,D=2,c=1	0.920	0.988	0.998

- QUADRATIC SVM – RBF KERNEL

No balancing

Train	Single Fold		
minDCF	Prior = 0.5	Prior = 0.1	Prior = 0.9
K=1, C = 0.1, g = 1	0.535	0.675	0.728
K=1, C = 0.1,g=10	0.672	0.685	0.726
K=1, C = 1,g=1	0.525	0.904	0.728
K=1, C = 1, g=10	0.672	0.685	0.726
K=1, C = 10,g=1	0.520	1	0.728
K=1, C = 10, g=10	0.914	0.914	0.726
K=10, C = 0.1,g=1	0.567	1	0.728
K=10, C = 0.1,g=10	0.685	1	0.726
K=10, C = 1,g=1	0.874	0.675	0.728
K=10, C = 1,g=10	0.685	0.685	0.726
K=10, C = 10,g=1	0.556	0.678	0.728
K=10, C = 10,g=10	0.685	0.924	0.726

Balancing

Train	Single Fold		
minDCF	Prior = 0.5	Prior = 0.1	Prior = 0.9
K=1, C = 0.1, g = 1	0.876	1	0.733
K=1, C = 0.1,g=10	0.964	0.685	0.726
K=1, C = 1,g=1	0.520	0.888	0.995
K=1, C = 1, g=10	0.674	0.685	0.725
K=1, C = 10,g=1	0.964	0.678	0.728
K=1, C = 10, g=10	0.672	1	0.725
K=10, C = 0.1,g=1	0.876	0.675	0.995
K=10, C = 0.1,g=10	0.685	1	0.726
K=10, C = 1,g=1	0.556	0.888	0.728
K=10, C = 1,g=10	0.685	0.685	0.995
K=10, C = 10,g=1	0.874	0.678	0.728
K=10, C = 10,g=10	0.685	1	0.726

GMM

The last model we analyze is a generative one based on training a Gaussian Mixture Model (GMM) over the data of each of the two classes. A Gaussian Mixture Model is a density model obtained as a weighted combination of Gaussians.

$$f_X(\boldsymbol{x}) = \sum_{c=1}^{K} w_c \mathcal{N}(\boldsymbol{x}; \boldsymbol{\mu}_c, \boldsymbol{\Sigma}_c)$$

We will consider the full covariance model, the diagonal model and the tied one.

We also use the LGB algorithm to generate GMM with M =2,4,8 components.

Train	Single Fold			K-Fold(K=5)		
minDCF	Prior = 0.5	Prior = 0.1	Prior = 0.9	Prior = 0.5	Prior = 0.1	Prior = 0.9
Version=full, M = 2, psi = 0.01	0.307	0.727	0.879	0.309	0.732	0.699
Version=full, M = 4, psi = 0.01	0.327	0.671	0.829	0.303	0.772	0.749
Version=full, M = 8, psi = 0.01	0.297	0.639	0.658	0.289	0.726	0.707
Version= diagional, M = 2, psi = 0.01	0.409	0.832	0.899	0.394	0.834	0.865
Version= diagional, M = 4, psi = 0.01	0.367	0.836	0.906	0.366	0.842	0.789
Version= diagional, M = 8, psi = 0.01	0.367	0.762	0.701	0.306	0.772	0.784
Version= tied, M = 2, psi = 0.01	0.307	0.727	0.879	0.309	0.732	0.699
Version = tied, M = 4, psi = 0.01	0.327	0.671	0.829	0.303	0.772	0.749
Version = tied, M = 8, psi = 0.01	0.297	0.639	0.658	0.289	0.726	0.707

• BEST RESULTS AND SCORES CALIBRATION

The classes are not balanced (e.g. there are many more normal wines than excellent or poor ones), so the balanced algorithms gives in general better results.
And for raw data, linear algorithms seem to perform better than the quadratic ones, and so the distribution of the samples seems quite uniform and can be divided pretty well by a linear hyperplane.
The results using PCA=10 and also more with PCA=9 slightly improves our scores, since some features are quite correlated, so a dimensionality reduction is worth it, but other preprocessing techniques perform better.
The Z-Normalization preprocessing is very effective, mostly in the discriminative classifiers that need normalized data to perform better, such the linear and quadratic logistic regression.
The Gaussianization gives optimal results mostly on the "pure gaussian" algorithms, like MVG and GMM.

Up to now we have only considered the min DCF metric. As already discussed, min DCF measures the cost we would pay if we made optimal decisions for the validation set using the scores of the recognizer. The cost that we actually pay, however, depends on the goodness of the threshold we use to perform class assignment. We therefore start considering also actual DCFs.

Furthermore we focus on calibrating the best results we have obtained(underlined in green):

1) GMM version=full, M=8, psi= 0.01 with Gaussianization and prior = 0.5
   MinDCF = 0.230, actDCF=0.255

2) SVM RBF Kernel, K=1, C=10, g=1 with Gaussianization and prior = 0.5
   minDCF = 0.232, actDCF = 0.394

We try to calibrate these two cases using the linear logistic regression with different values of lambda in order to reach best performance.

For the first case we have:

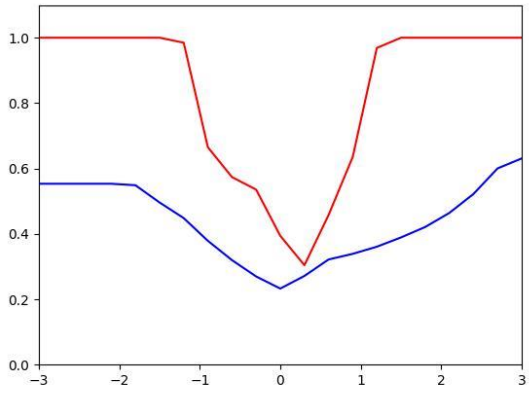
- linear logistic regression with lamb 0: DCF calibrated act = 0.259
- linear logistic regression with lamb 1e-06: DCF calibrated act = 0.259
- linear logistic regression with lamb 0.001: DCF calibrated act = 0.259
- linear logistic regression with lamb 1: DCF calibrated act = 0.271

So we don't see particular improvement.

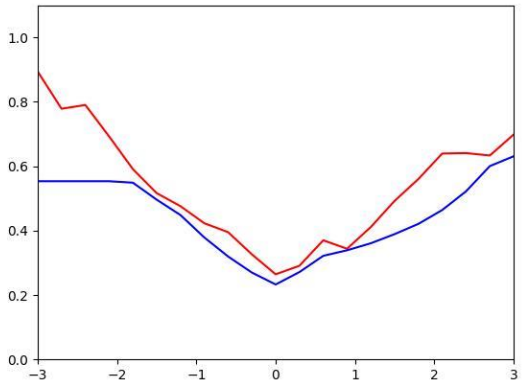
While for the second case we have:

- linear logistic regression with lamb 0: DCF calibrated act = 0.264
- linear logistic regression with lamb 1e-06: DCF calibrated act = 0.264
- linear logistic regression with lamb 0.001: DCF calibrated act = 0.261
- linear logistic regression with lamb 1: DCF calibrated act = 0.349

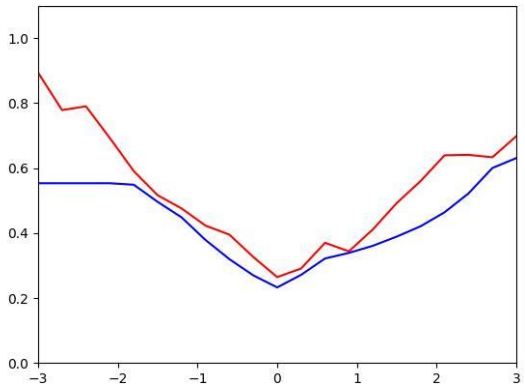
In this case calibration is useful, in fact we notice that with lambda = 0, 1e-06, 1e-03 we obtain an DCF calibrated that is very slower than the actDCF = 0.394 without the calibration.



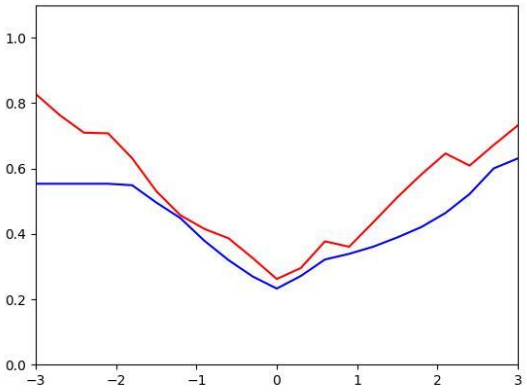
SVM RBF, without calibration



SVM RBF, calibration with lambda = 0



SVM RBF, calibration with lambda = 1e-6



SVM RBF, calibration with lambda = 1e-3

Train	Single Fold		
minDCF	Prior = 0.5	Prior = 0.1	Prior = 0.9
Gaussianizated GMM version=full, M = 8, psi =0.01	0.230	0.642	0.617
Gaussianizated SVM RBF Kernel, K=1, C=10, g=1	0.232	0.553	0.480

Train	Single Fold		
actDCF	Prior = 0.5	Prior = 0.1	Prior = 0.9
Gaussianizated GMM version=full, M = 8, psi =0.01	0.255	0.757	1
Gaussianizated SVM RBF Kernel, K=1, C=10, g=1	0.394	1	1

Train	Single Fold		
actDCF(calibrated, prior=0.5)	Prior = 0.5	Prior = 0.1	Prior = 0.9
Gaussianizated GMM version=full, M = 8, psi =0.01, lambda = 1e-03	0.259	0.721	0.665
Gaussianizated SVM RBF Kernel, K=1, C=10, g=1, lambda = 1e-03	0.261	0.720	0.574

- # FUSION OF THE SCORES

We assume that the fused score is a function of the scores of the different classifiers If st,A and st,B are the scores of classifiers A and B for sample xt , then the fused score for sample xt will be:

$$s_t = f(s_{t,A}, s_{t,B})$$

We will then use the fused score St to perform the decision (i.e. to label the sample xt).

We can assume a simple linear form for f, so we can use again a prior-weighted logistic regression to fuse the scores:

$$f(s_{t,A}, s_{t,B}, s_{t,C}, \dots) = \alpha_A s_{t,A} + \alpha_B s_{t,B} + \alpha_C s_{t,C} + \cdots + \beta$$

Again we focus our attention in the two cases listed previously and try to fuse them.

Train	Single Fold		
minDCF	Prior = 0.5	Prior = 0.1	Prior = 0.9
Fusion between GMM and SVM RBF	0.228	0.569	0.617

Train	Single Fold		
actDCF	Prior = 0.5	Prior = 0.1	Prior = 0.9
Fusion between GMM and SVM RBF	0.237	0.618	0.665

- # TESTING

Up to now we have focused on choosing the best classifier with the best hyperparameters using only the training data splitted also in validation data. Now we try to apply the decisions we have made into the test data, to verify if those work also to other sets of data.

Test	Single Fold		
minDCF	Prior = 0.5	Prior = 0.1	Prior = 0.9
Gaussianizated GMM version=full, M = 8, psi =0.01	0.363	0.759	0.798
Gaussianizated SVM RBF Kernel, K=1, C=10, g=1	0.287	0.809	0.760

Test	Single Fold		
actDCF	Prior = 0.5	Prior = 0.1	Prior = 0.9
Gaussianizated GMM version=full, M = 8, psi =0.01	0.375	0.913	1.559
Gaussianizated SVM RBF Kernel, K=1, C=10, g=1	0.498	1	1

Test	Single Fold		
actDCF(calibrated, prior=0.5)	Prior = 0.5	Prior = 0.1	Prior = 0.9
Gaussianizated GMM version=full, M = 8, psi =0.01, lambda = 1e-03	0.372	0.910	0.805
Gaussianizated SVM RBF Kernel, K=1, C=10, g=1, lambda = 1e-03	0.342	0.844	0.773

Test	Single Fold		
minDCF	Prior = 0.5	Prior = 0.1	Prior = 0.9
Fusion between GMM and SVM RBF	0.306	0.751	0.682

Test	Single Fold		
actDCF	Prior = 0.5	Prior = 0.1	Prior = 0.9
Fusion between GMM and SVM RBF	0.316	0.881	0.745

CONCLUSIONS

Our analysis led us to two different and well performing models, the SVM kernel-RBF , K=1, C=10, g=1 with Gaussianization and the GMM version=full, M=8, psi= 0.01 with Gaussianization. Therefore, gaussianized data appear to be more suited for the wine-quality dataset classification task with respect to raw data. With the fusion of the two models, we are able to achieve a DCF of ≈ 0.3 for the target application $\pi = 0.5$, although also for the other two considered applications with $\pi = 0.1$ and $\pi = 0.9$ results are acceptable, with DCFs of ≈ 0.88 and ≈ 0.75, respectively. The choices we made on our training/validation sets proved to be partially effective for the evaluation set. In effect, the GMM classifier doesn't perform well like in the training stage (minDCF of 0.363 vs 0.230) and in the final testing stage, for the target application the minDCF of the fused system is a little higher than the minDCF of the SVM RBF Kernel(0.306 vs 0.287). This may be due to the choice to train the classifiers with a single-fold split instead of a k-fold one (K=5), and the results are less robust for different evaluating data, but our current technologies only allowed us to do so. We chose equally the fusion option because for the other applications gives an improvement (both in terms of minDCF and DCF)

REFERENCES

1. P. Cortez, A. Cerdeira, F. Almeida, T. Matos and J. Reis. Modeling wine preferences by data mining from physicochemical properties. In Decision Support Systems, Elsevier, 47(4):547-553, 2009.
2. The dataset is taken from the UCI repository (Dua, D. and Graff, C. (2019). UCI Machine Learning Repository [http://archive.ics.uci.edu/ml]. Irvine, CA: University of California, School of Information and Computer Science)