Wine Quality Detection

Machine Learning and Pattern Recognition

Author:

Seyedomid Mahdavi (s299837)

Introduction

The purpose of this project is to discriminate good wine from bad wine. The dataset is taken from the UCI repository. The original dataset contains wines with 10 levels of quality. However, for this project, it has been binarized, collecting all wines with low quality (lower than 6) into class 0, and good quality (greater than 6) into class 1. Wines with quality 6 have been discarded. Furthermore, red and white wines have been merged for the project.

For each sample, 11 features have been measured each one representing physical properties of the wine. Dataset involves both training and test set. The training set contains 1839 samples which 613 (one third of them) are labeled 1 (high-quality). On the other hand, the test set includes 1822 samples that have to be classified.

Through this detailed report, the problem will be discussed and analyzed, different modeling approaches will be evaluated on the test set and it will be concluded which models are more suitable to be adopted and are able to provide good predictions on unseen data.

Classification Process

Dimensionality reduction

The feature space is often very large and contains a large amount of unwanted and potentially harmful information. Dimensionality reduction techniques compute a mapping from the n-dimensional feature space to a m-dimensional space, having $m << n$.

Dimensionality reduction has several advantages. Firstly, dropping the dimensionality down to two or three, makes it possible to visualize the data on a

plot, which means important insights can be achieved by analyzing the patterns in terms of clusters and more. Secondly and more importantly, dimensionality reduction also simplifies the classification. It reduces curse of dimensionality and prevents our model from being overfit. This feature is highly desirable when it comes to assessing our model's performance on real data.

PCA and LDA are two linear techniques typically used to reduce dimensionality and therefore pick the two most reliable and discriminative features describing the samples. Here we describe each of them briefly.

PCA provides us a subspace that allows preserving most of the information described by the data. It focuses on finding the direction of maximum variation in the dataset. PCA is usually applied directly to centered data. Then, we sort them and pick m largest eighenvalues each of which corresponds to the variance along the corresponding axis. Selection of optimal m can be done by cross–validation using a validation set. We can also select m as to retain a given percentage t (e.g. 95%) of the variance of the data.

The other one, which is LDA, finds a direction that maximizes the between-class variability while minimizing within-class variability for the transformed samples. Additionally, LDA has one superiority over PCA that it also achieves classification of the data simultaneously. For LDA, the steps to be taken are somehow different. First of all, we compute the between and within class covariance matrices. Then, we can solve the eighenvalue problem either by solving the generalized eighenvalue problem or joint diagonalization of S_B and S_W. From the definition of S_B, the number of non–zero eigenvalues is at most C – 1. Therefore, LDA allows estimating at most C – 1 directions which is not suitable for binary classification. Hence, LDA is not going to be used in this project.

Model Evaluation

We need a way to perform model selection. It can be done through a single split or using K-fold cross-validation approach.

We can build a validation set by extracting some data from the training set. Part of the training set will be used for estimating our models. The remaining part

(validation set) will simulate the evaluation data, and will be used to infer hyper-parameters and for model selection.

When training data is scarce, cross-validation is usually employed. First, we divide the training data into K folds. Then, use K-1 folds as training data and the remaining fold as the validation set. Combine the validation results on the K folds and select optimal values for hyper-parameters. Finally, retrain the model using the selected hyper-parameters but using all data. In our project, we evaluated different models using the K-Fold approach with k=3.

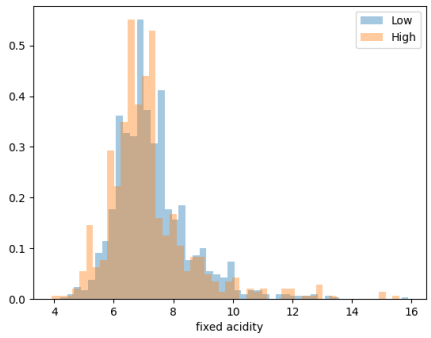
Evaluation of a model is based on different criteria. We measure the error rate as the number of incorrect classified samples over the total number of samples. We can also evaluate our model through accuracy that is equal to 1 – error rate.

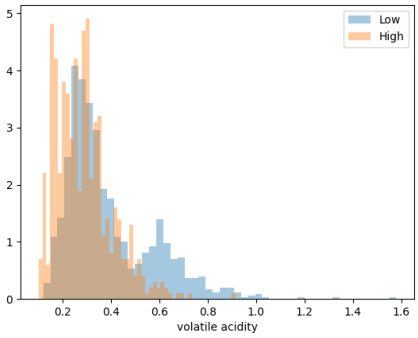
Another approach to analyze the performance of a model is based on Bayes Decisions. It refers to the statistical approach based on tradeoff quantification among various classification decisions based on the concept of Probability and the costs associated with the wrong decisions. The Bayes decision consists in choosing the action that minimizes the expected cost. The cost and class priors used in taking the optimal decision depends on application characteristics.

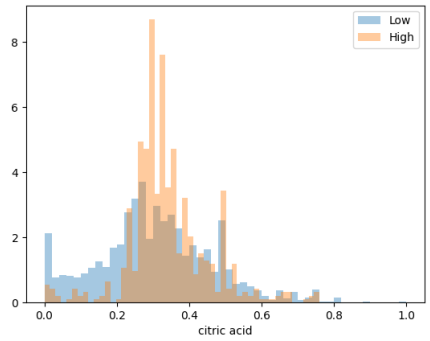
Generally, using Bayes Decisions is more reliable than accuracy. Moreover, we have seen that in case of having an equal cost for each wrong decision and the same class priors, Bayes Decision and accuracy are equivalent. In other words, accuracy will be equal to DCF (Detection Cost Function) for an application with (π=0.5, Cfn=1, Cfp = 1. In the following, we use min DCF (the optimum DCF calculated with the best threshold) to evaluate different models and after choosing the most appropriate model, we determine DCF on training data with theoretical threshold. Afterwards, it will be examined if further steps have to be made to achieve a better DCF.

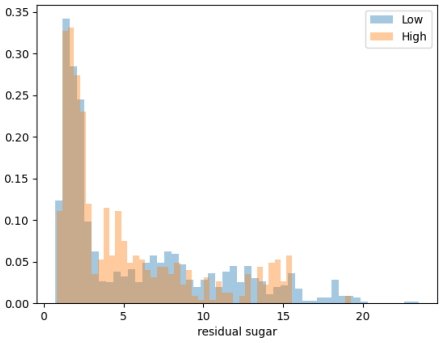
Features

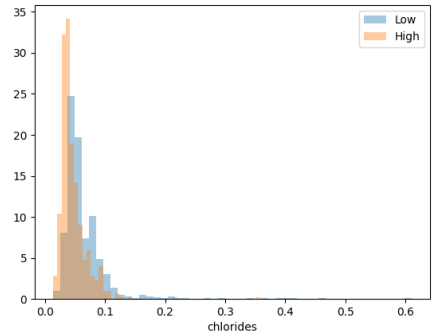
We decided to study each feature separately at first in order to figure out how helpful each feature is as a discriminant and to draw some conclusions before beginning to build the models. Our studies on each feature, at the end, resulted in the following histograms.

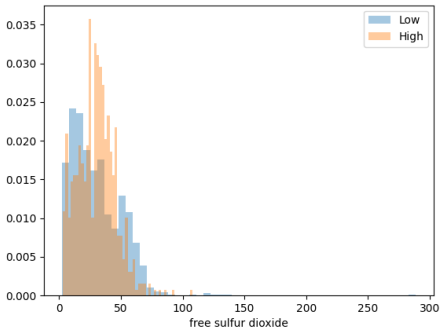

Low
High
fixed acidity

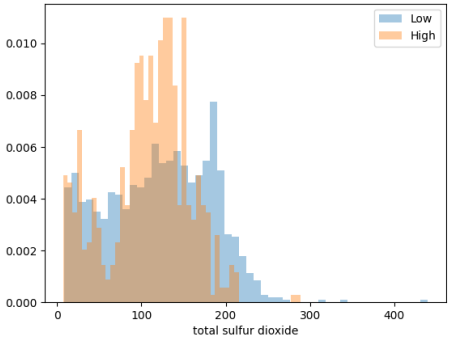

Low
High
volatile acidity

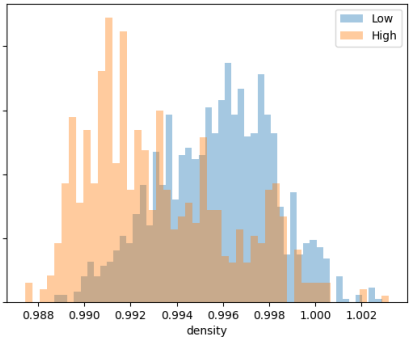

Low
High
citric acid

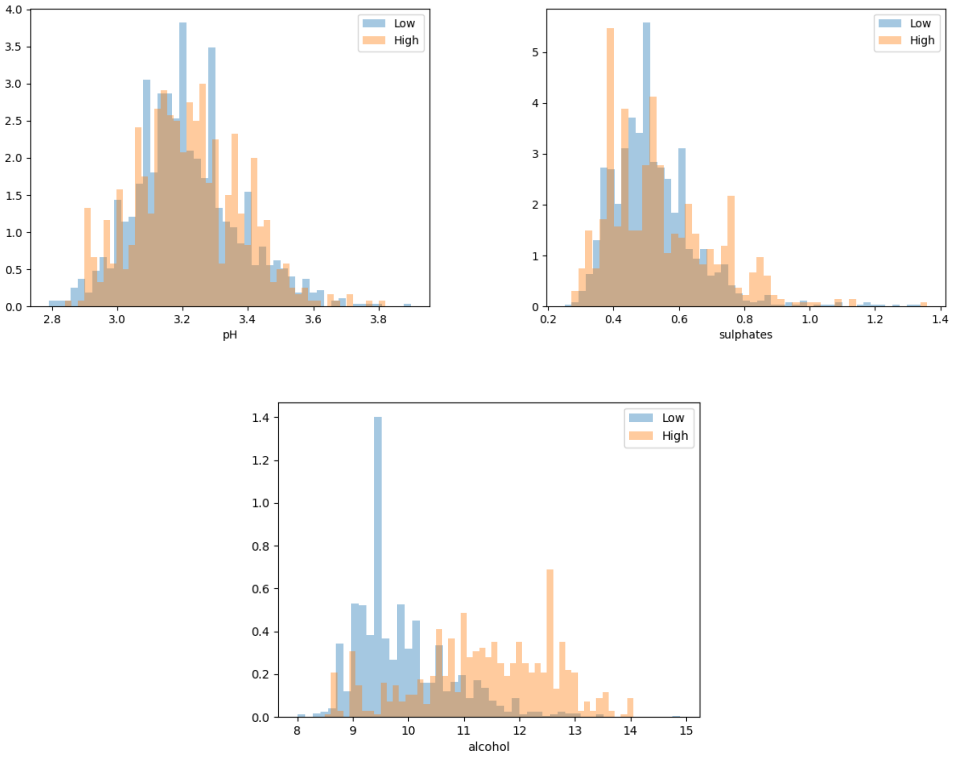

Low
High
residual sugar


Low
High
chlorides


Low
High
free sulfur dioxide


Low
High
total sulfur dioxide


Low
High
density



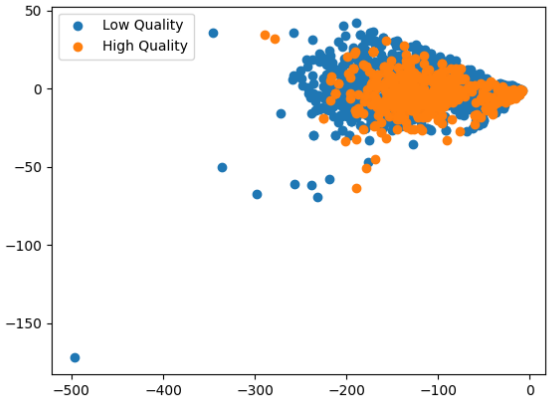
According to the figures derived from the data, we can notice that some features are not normally distributed. Furthermore, it can easily be noticed that in many cases we are dealing with a lot of outliers. It seems that density and alcohol could be the two most discriminant features.

An important fact understood from the data is that features are not ranged similarly. In other words, in order to achieve an acceptable classification model, we need to first normalize all features. Normalization is a pre-process step that can be implemented throw several methods. For this project, we calculate the minimum and the maximum of each feature from the training set. Then, we subtract the minimum from each sample and divide the result by the difference of the maximum and the minimum.

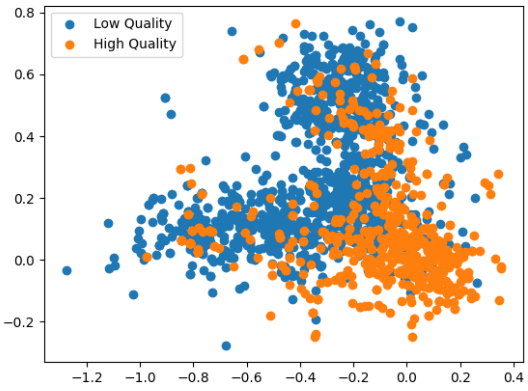
Dimensionality Reduction

Our data is already divided into two parts: training set and test set. The training set contains the observed samples that will be used for the learning stage and the test set that will be used to evaluate our model's performance on unseen data.

For visualization purposes, we decided to apply PCA on the whole training set and pick two most discriminant features for each technique. The results are as shown below:



The plot is not informative enough. As we discussed before, our training set needs a standardization step beforehand. Here the same plot is depicted with the normalized data:



We will also study a variety of feature combinations in order to find out which classification model works best and effectively.

Classification Models

There are a number of classification algorithms used to identify new observations based on training data, i.e., to learn from a given dataset or observations and then

label them with a number of classes or groups. Employing each model separately, we can analyze the data and extract some useful information on how precisely the models can make predictions and label any input data inserted. Our aim is to design a suitable decision function that can provide such predictions on unseen data. In fact, comparing the minimum DCF of each model, we can opt for the model which can identify relationships and patterns between variables in a more accurate way.

It is worth to mention that one third of the training samples are labeled as high-quality wine. Therefore, it is reasonable to consider the prior probability as 0.33. However, since the prior depends completely on the application, we will evaluate the min DCF just for these three applications: (0.5, 1, 1), (0.1, 1, 1) and (0.9, 1, 1). Our main focus is the application with prior equal to 0.5. Hence, choosing the hyper-parameters are in the way that decreases the min DCF related to this application. For other applications further evaluations might be necessary.

The first approach we adopted to train our model was **Multivariate Gaussian**. The whole process is mainly based on the computation of Multivariate Gaussian Density - we fit a Gaussian density over the samples of each class. Throughout our experiments with various numbers of features, almost similar results were obtained.

Multivariate Gaussian	**no PCA**	**m=10**	**m=9**	**m=8**	**m=7**	**m=6**
(0.5, 1, 1)	0.324	0.341	0.335	0.320	0.334	0.378
(0.1, 1, 1)	0.778	0.813	0.831	0.810	0.839	0.812
(0.9, 1, 1)	0.878	0.862	0.887	0.804	0.791	0.917

Considering all the possible values for the number of features, applying PCA did not affect much on the results. Considering the main application (prior = 0.5), the best min DCF achieved was 0.320 with 8 features. It was slightly better than the case without applying dimensionality reduction.

Next, we tried **Naïve Bayes Gaussian** assuming that, for each class, the different components are approximately independent so that we could simplify the estimate. To put it in another way, the Naive Bayes Gaussian classifier actually corresponds to a Multivariate Gaussian classifier with diagonal covariance matrices. In comparison with MVG, the performance decreased dramatically. The optimal dimensionality for this model was 7 and the corresponding min DCF was equal to 0.386.

Naïve Bayes	**no PCA**	**m=10**	**m=9**	**m=8**	**m=7**	**m=6**
(0.5, 1, 1)	0.427	0.405	0.403	0.392	0.386	0.406
(0.1, 1, 1)	0.857	0.831	0.831	0.841	0.876	0.840
(0.9, 1, 1)	0.896	0.962	0.966	0.855	0.918	0.949

The two aforementioned models each has a variant which assumes the covariance matrices of different classes are tied, that is, each class has its own mean, but the covariance matrix is the same for all classes. The tied Covariance Matrix would become useful if we had large dimensional data or a small number of samples.

Tied Gaussian	**no PCA**	**m=10**	**m=9**	**m=8**	**m=7**	**m=6**
(0.5, 1, 1)	0.337	0.336	0.336	0.344	0.367	0.388
(0.1, 1, 1)	0.817	0.826	0.825	0.835	0.852	0.829
(0.9, 1, 1)	0.743	0.753	0.759	0.765	0.827	0.888

Results obtained from Tied Gaussian model were better than the ones we achieved from Naïve Bayes Gaussian model but they were still least accurate comparing to the Multivariate Gaussian. Although the least DCF was 0.336 with both m = 10 and m = 9, the difference with the model built on original space is negligible.

The last Gaussian model we are going to analyze is Tied Naïve Bayes Gaussian which is the combination of the last two models. In this case, PCA improved the results significantly. Building the model by 9 features was the ideal option. Yet, it does not excel the MVG model.

Tied Naïve Gaussian	**no PCA**	**m=10**	**m=9**	**m=8**	**m=7**	**m=6**
(0.5, 1, 1)	0.405	0.343	0.342	0.345	0.359	0.386
(0.1, 1, 1)	0.863	0.830	0.813	0.833	0.854	0.829
(0.9, 1, 1)	0.922	0.775	0.788	0.782	0.822	0.891

Thus far, we have seen that if we assume the classes are equally distributed (π is 0.5), the best model is the unconstrained MVG with 8 dimensions and the best min DCF is 0.320. the most accurate model for the (0.1, 1, 1) application is the same. On the other hand, if we consider the prior probability as 0.9, the best performance is attained by the Tied Gaussian.

Our results from **Logistic Regression** modeling approach were telling a quite different story. In brief, Logistic regression, despite its name, is a discriminative approach for classification. Rather than modeling the distribution of observed samples, we directly model the class posterior distribution. In other words, the Logistic Regression model is obtained by minimizing the average cross-entropy between the model predictions and the observed labels.

In the following, first we built the model by assigning the value 10^{-6} to the regularization term for several dimensionalities. Then, for the m with the best performance, some other values of λ have been tested.

Logistic Regression	**no PCA**				**m=10**	**m=9**	**m=8**	**m=7**
	$\lambda=10^{-6}$	**$\lambda=10^{-3}$**	**$\lambda=10^{-1}$**	**$\lambda=1$**	**$\lambda=10^{-6}$**	**$\lambda=10^{-6}$**	**$\lambda=10^{-6}$**	**$\lambda=10^{-6}$**
(0.5, 1, 1)	0.346	0.349	0.422	0.637	0.347	0.349	0.347	0.377
(0.1, 1, 1)	0.846	0.827	0.875	0.948	0.843	0.851	0.851	0.867
(0.9, 1, 1)	0.695	0.789	0.980	0.982	0.690	0.693	0.689	0.777

According to the results, regularization has aggravated the DCF. It is worth mentioning that dimensionality reduction had just slightly affected the outcome. The best performance was obtained without PCA and $\lambda=0.000001$. Compared to the previous experiments, Logistic Regression did not cause an improvement for the first two applications. For the state with prior equal to 0.9, however, the result acquired for m=8 was the best so far.

Support Vector Machines is a discriminative non-probabilistic classifier. It is a reliable classifier for binary problems and it has a geometrical interpretation. It does not produce a posterior probability, but a score that can be used to assign labels. We can obtain the separation hyperplane by solving either the primal or the dual problem, that is, for linear SVMs, we can optimize either the primal or dual formulation. If we are interested in linear separation, we can directly solve the primal SVM problem. The dual problem also allows for non-linear separation through the kernel trick.

Like the LR approach, in this case, selection of the hyper-parameters is also done step by step. In all test cases, K is equal to 1. First, we check the results for different values of m assuming C=1. Afterwards, the most appropriate C among some specific values will have been found.

Linear SVM	no PCA				m=10	m=9	m=8	m=7
	C=10^{-1}	**C=1**	**C=10**	**C=10^2**	**C=1**	**C=1**	**C=1**	**C=1**
(0.5, 1, 1)	0.365	0.338	0.338	0.334	0.341	0.340	0.347	0.367
(0.1, 1, 1)	0.835	0.816	0.822	0.823	0.821	0.822	0.822	0.836
(0.9, 1, 1)	0.869	0.805	0.760	0.723	0.789	0.790	0.794	0.811

Also, in this case, it was observed that PCA raised the min DCF. The best performance achieved for this model was for the case without PCA and C=100. It is still worse than the result obtained by Multivariate Gaussian.

As we mentioned before, SVMs allow for non-linear classification through an implicit expansion of the features in a higher-dimensional space. The SVM dual objective depends on the training samples only through dot-products, and we can compute a classification score through scalar products between training and evaluation samples. Therefore, SVM does not require that we explicitly compute the feature expansion. A kernel function enables us to compute the scalar product between the expanded features. Two kernel functions are used in this project: Polynomial kernel and Radial Basis Function kernel. The choice of the kernel and of its hyper-parameters can also be made through cross-validation.

Polynomial SVM	no PCA						
	C=10^{-1} c=0, d=2	**C=1** c=0, d=2	**C=10** c=0, d=2	**C=10^2** c=0, d=2	**C=10^2** c=1, d=2	**C=10^2** c=1, d=3	**C=10^2** c=2, d=2
(0.5, 1, 1)	0.359	0.338	0.306	0.286	0.281	0.258	0.285
(0.1, 1, 1)	0.824	0.809	0.772	0.746	0.743	0.639	0.745
(0.9, 1, 1)	0.874	0.759	0.776	0.803	0.785	0.753	0.780

Polynomial SVM	**m=10**	**m=9**	**m=8**	**m=7**
	C=1 c=0, d=2	**C=1** c=0, d=2	**C=1** c=0, d=2	**C=1** c=0, d=2
(0.5, 1, 1)	0.342	0.343	0.344	0.360
(0.1, 1, 1)	0.813	0.816	0.809	0.828
(0.9, 1, 1)	0.753	0.773	0.772	0.812

Radial Basis SVM	**no PCA**					
	$C=10^{-1}$ $\Upsilon=1$	**C=1 $\Upsilon=1$**	**C=10 $\Upsilon=1$**	**$C=10^2$ $\Upsilon=0.1$**	**$C=10^2$ $\Upsilon=1$**	**$C=10^2$ $\Upsilon=10$**
(0.5, 1, 1)	0.365	0.317	0.261	0.311	0.250	0.280
(0.1, 1, 1)	0.815	0.775	0.690	0.773	0.611	0.688
(0.9, 1, 1)	0.758	0.671	0.718	0.705	0.723	0.807

Radial Basis SVM	**m=10**	**m=9**	**m=8**	**m=7**
	C=1 $\Upsilon=1$	**C=1 $\Upsilon=1$**	**C=1 $\Upsilon=1$**	**C=1 $\Upsilon=1$**
(0.5, 1, 1)	0.318	0.320	0.320	0.345
(0.1, 1, 1)	0.773	0.789	0.793	0.776
(0.9, 1, 1)	0.662	0.673	0.672	0.719

Both non-linear SVM methods decreased the min DCF noticeably. For all applications, the best results until now are achieved by Radial Basis SVM. Moreover, as other approaches, applying PCA did not modified the DCF considerably. Therefore, a radial basis SVM with all features and ϒ=1 could be the ideal model so far.

The last model we implemented was **Gaussian Mixture Model**. It is a probabilistic model that assumes all the data points are generated from a mixture of a finite number of Gaussian distributions with unknown parameters. However, it is not always completely accurate. We also require a sufficient amount of data to obtain good estimates. The main difficulty in learning Gaussian mixture models from unlabeled data is that we usually do not know which points came from which component. Consequently, it becomes really hard to fit a separate Gaussian distribution to each set of points. Expectation-maximization is a well-founded statistical algorithm to get around this problem by an iterative process. It can be used to estimate the parameters of a GMM that maximize the likelihood for a training set X.

Initialization also plays a vital role in GMM Training. An algorithm that can help us properly take care of initialization is LBG algorithm. Firstly, we split the components of a g-components GMM to obtain an initial 2g-components GMM. Secondly, we run the EM algorithm until convergence for the 2g-components GMM is achieved. At the end, we do some iteration until the desired number of Gaussians is reached.

GMM	**no PCA**				**m=10**	**m=9**	**m=8**	**m=7**
	g=2	**g=4**	**g=8**	**g=16**	**g=2**	**g=2**	**g=2**	**g=2**
(0.5, 1, 1)	0.319	0.334	0.298	0.290	0.320	0.324	0.328	0.345
(0.1, 1, 1)	0.805	0.821	0.774	0.785	0.799	0.800	0.803	0.839
(0.9, 1, 1)	0.838	0.691	0.631	0.638	0.837	0.834	0.837	0.882

The model we trained through GMM performed better than Gaussian models but it is still less accurate compared to SVM. Nevertheless, for the (0.9, 1, 1) application, the min DCF achieved with 8 components was the minimum among all models. Regarding to the main application, the optimal number of GMM components was 16.

We also implemented both diagonal and tied-covariance GMMs for the project. We computed ML solutions for tied models, where each component has a covariance matrix $\Sigma_g = \Sigma$. The results obtained from this solution are as shown in the table below:

Tied GMM	**no PCA**				**m=10**	**m=9**	**m=8**	**m=7**
	g=2	**g=4**	**g=8**	**g=16**	**g=2**	**g=2**	**g=2**	**g=2**
(0.5, 1, 1)	0.339	0.329	0.307	0.315	0.343	0.348	0.349	0.347
(0.1, 1, 1)	0.798	0.780	0.826	0.816	0.799	0.799	0.801	0.827
(0.9, 1, 1)	0.844	0.785	0.786	0.653	0.845	0.843	0.843	0.864

The diagonal variant of the GMM we implemented consists of a model whose components all have diagonal covariance matrices.

Diagonal GMM	**no PCA**	**m=10**				**m=9**	**m=8**	**m=7**
	g=2	**g=2**	**g=4**	**g=8**	**g=16**	**g=2**	**g=2**	**g=2**
(0.5, 1, 1)	0.405	0.370	0.343	0.319	0.325	0.370	0.372	0.380
(0.1, 1, 1)	0.863	0.837	0.790	0.803	0.785	0.839	0.843	0.855
(0.9, 1, 1)	0.908	0.861	0.758	0.642	0.701	0.861	0.860	0.910

As it can be perceived from the tables, the two last models did not obtain a lower DCF comparing to the general GMM model.

Now that our observations are finished, we could analyze the results and draw conclusions. A fact that we noticed was that applying PCA on most of the models did not cause a significant variation for 10, 9 and 8 dimensions. Since we are focusing on the main application with a balanced distribution of classes, we choose the best model among the results for prior equal to 0.5. However, if the model was going to be applied for another application with different distribution, other choices might be adopted. As the result, we build the classifier on the whole training set based on the Radial Basis SVM with full dimensionality, C=100 and ϒ=1.

Final Evaluation on the Model

First choice that we want to analyze is the normalization preprocess. In the beginning, we saw that our features' ranges are not similar so we decided to normalize them before building the models. Now that we have chosen our best classifier, we apply the same 3-fold with the same hyper-parameters to evaluate the normalization:

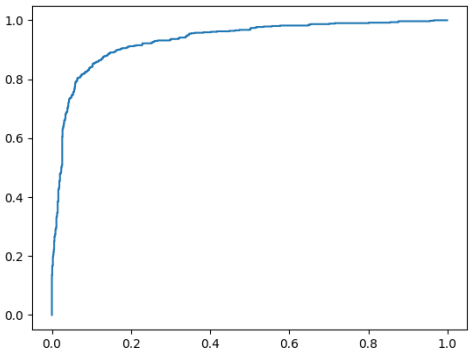
Radial Basis SVM	**Min DCF (with normalization)**	**Min DCF (without normalization)**
(0.5, 1, 1)	0.250	0.582
(0.1, 1, 1)	0.611	0.692
(0.9, 1, 1)	0.723	0.740

According to the data, it is obvious that normalization had a huge effect on the outcome especially for the main application. Hence, we applied a right approach for preprocessing.

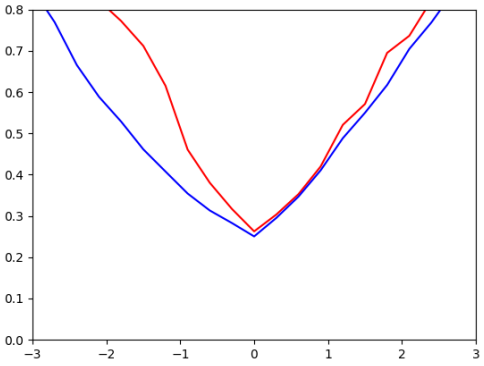
We know that min DCF shows how well a model could perform if we knew the optimal threshold but we do not have the value and we are not sure that the classifier still performs accurately. Now, in order to check whether our data has to

be calibrated, we compute the actual Detection Cost Function (DCF) and set it side by side with min DCF for the main three different applications used during building the models. Here the DCF is computed assuming the theoretical threshold for the scores. We also sketched bias error plot and the ROC curve for the main application.

Application Configuration	**DCF**	**Min DCF**
(0.5, 1, 1)	0.263	0.250
(0.1, 1, 1)	0.833	0.611
(0.9, 1, 1)	0.783	0.723



ROC Curve



Bayes Error

According to the table and to the Bayes Error, the DCF is close to the minimum DCF for the applications with effective prior equal or more than 0.5. Although our target application gives us an acceptable outcome, we are going to find a better solution for the case with a lower prior probability. There are two approaches with which the difference between DCF and min DCF could be reduced. We can find an optimal threshold for decision making or we can further process our score matrix to calibrate them. In this project the first approach is adopted and then the outcome will have been evaluated.

In order to find the best threshold, first we split the score matrix achieved from the return values of our classifier on the train set into train and evaluation sets. Then, we find the threshold with which the DCF had the minimum value among the train

set. In the end, we classify the evaluation set with the calculated threshold and analyze the results. It is worth to mention that using this method, different thresholds must be computed for each application. Even though our observations indicated that the two other applications do not need this step, we are going to find a threshold for them and evaluate the results. The table below, indicates the min DCF, the actual DCF, and the DCF using the optimal threshold (DCF^*) for the evaluation set:

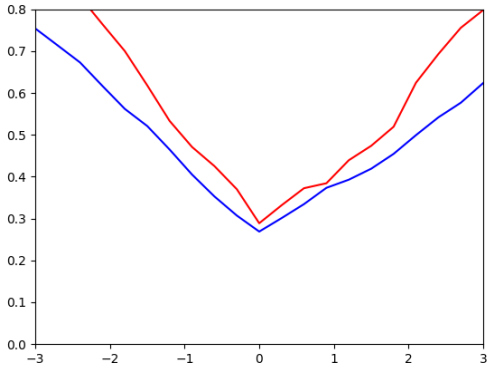
Application Configuration	**Min DCF**	**DCF**	**DCF^***
(0.5, 1, 1)	0.243	0.252	0.272
(0.1, 1, 1)	0.485	0.792	0.507
(0.9, 1, 1)	0.648	0.727	0.795

It could be easily noticed that the threshold improved the DCF considerably. However, the impact was negative on the other two applications.

Conclusions

Now we are ready to build our model with the entire train set and then classify the test set to analyze our model's performance. The first model is our main choice, Radial Basis SVM with normalized data, full dimensionality, C=100 and ϒ=1. In this stage we also compute the error rate for the main application. At the end the error rate was 12.84%.

Application Configuration	**DCF**	**DCF (training phase)**
(0.5, 1, 1)	0.289	0.263
(0.1, 1, 1)	0.784	0.833
(0.9, 1, 1)	0.641	0.783



Bayes Error

It is necessary to emphazise that the data in the above table is the DCF computed with zero threshold and it is not the min DCF. Except for the main application in which the results are aggravated slightly, the other applications demonstrate a better DCF. In additaion to that, considering the Bayes error plot, the diffrence between actual and min DCF is increased for priors greater than 0.5. According to our analyses, it was necessary for the application with prior probability equal to 0.1 to find an optimal threshold. However, we used the threshold for each application from the last stage and calculated the DCFs with it.

Application Configuration	**Min DCF (training phase)**	**Min DCF**	**DCF**	**DCF***
(0.5, 1, 1)	0.250	0.269	0.289	0.272
(0.1, 1, 1)	0.611	0.636	0.784	0.690
(0.9, 1, 1)	0.723	0.515	0.641	0.526

By looking deep into the final numbers we reach interesting results. Except for the third application, the minimum DCF is increased (although the difference is negligible). This means that maybe more training data could have been more effective to build a trust-worthy model. Opposed to the training phase analyses, the optimal threshold was really useful for all three applications.

Another choice that we want to examine, is normalization. The below table describes the min DCF of the model build on non-normalized features. As we expected, the model is much worse than the case with normalization.

Application Configuration	**Min DCF**
(0.5, 1, 1)	0.614
(0.1, 1, 1)	0.984
(0.9, 1, 1)	0.971

Though we consider our model good enough, we are going to evaluate other models to observe whether we could have taken better decisions. the following tables, show the corresponding min DCF for all models analyzed before with the chosen hyper-parameters.

Multivariate Gaussian (m=8)	**Min DCF**	**Min DCF (training phase)**
(0.5, 1, 1)	0.325	0.320
(0.1, 1, 1)	0.694	0.810
(0.9, 1, 1)	0.697	0.804

Naïve Bayes Gaussian (m=7)	**Min DCF**	**Min DCF (training phase)**
(0.5, 1, 1)	0.337	0.386
(0.1, 1, 1)	0.757	0.876
(0.9, 1, 1)	0.696	0.918

Tied Gaussian (m=10)	**Min DCF**	**Min DCF (training phase)**
(0.5, 1, 1)	0.326	0.336
(0.1, 1, 1)	0.711	0.826
(0.9, 1, 1)	0.729	0.753

Tied Naïve Gaussian (m=9)	**Min DCF**	**Min DCF (training phase)**
(0.5, 1, 1)	0.328	0.342
(0.1, 1, 1)	0.729	0.813
(0.9, 1, 1)	0.744	0.788

Logistic Regression (no PCA, $\lambda=10^{-6}$)	**Min DCF**	**Min DCF (training phase)**
(0.5, 1, 1)	0.326	0.346
(0.1, 1, 1)	0.695	0.846
(0.9, 1, 1)	0.654	0.695

Linear SVM (no PCA, C=100)	**Min DCF**	**Min DCF (training phase)**
(0.5, 1, 1)	0.319	0.334
(0.1, 1, 1)	0.667	0.823
(0.9, 1, 1)	0.671	0.723

polynomial SVM (no PCA, C=100, c=1, d=3)	**Min DCF**	**Min DCF (training phase)**
(0.5, 1, 1)	0.267	0.258
(0.1, 1, 1)	0.636	0.639
(0.9, 1, 1)	0.514	0.753

GMM (no PCA, g=16)	**Min DCF**	**Min DCF (training phase)**
(0.5, 1, 1)	0.315	0.290
(0.1, 1, 1)	0.693	0.785
(0.9, 1, 1)	0.606	0.638

Tied GMM (no PCA, g=8)	**Min DCF**	**Min DCF (training phase)**
(0.5, 1, 1)	0.285	0.307
(0.1, 1, 1)	0.770	0.826
(0.9, 1, 1)	0.663	0.786

Diagonal GMM (m=10, g=8)	**Min DCF**	**Min DCF (training phase)**
(0.5, 1, 1)	0.316	0.319
(0.1, 1, 1)	0.772	0.803
(0.9, 1, 1)	0.718	0.642

Considering the main application, the tables did not have any surprises and the results were approximately the same as the training evaluations. However, we would like to mention that the other non-linear SVM model performed almost the same as our chosen model (even very slightly better).