Index

[Introduction 2](#_Toc119663783)

[HTRU2 Data Set 2](#_Toc119663784)

[Analysis of the features 4](#_Toc119663785)

[Histograms 4](#_Toc119663786)

[Correlation 8](#_Toc119663787)

[Validation approach 10](#_Toc119663788)

[Multivariate Gaussian Classifiers (MVG) 12](#_Toc119663789)

[Logistic Regression 15](#_Toc119663790)

[Linear Logistic Regression 15](#_Toc119663791)

[Quadratic Logistic Regression 17](#_Toc119663792)

[Support Vector Machine (SVM) 18](#_Toc119663793)

[Linear SVM 18](#_Toc119663794)

[Quadratic SVM 20](#_Toc119663795)

[RBF SVM 22](#_Toc119663796)

[Gaussian Mixture Models 24](#_Toc119663797)

[Score calibration 25](#_Toc119663798)

[Combining classifiers(fusion approach) 29](#_Toc119663799)

[Experimental results on evaluation set 30](#_Toc119663800)

[References 31](#_Toc119663801)

# Introduction

This report is to analyzes the results of different machine learning algorithms applied to the HTRU2 Data Set [[1]](../../../../../UCI%20Machine%20Learning%20Repository_%20HTRU2%20Data%20Set.html). The final goal is to individuate models with best performance.

Initially an analysis of the feature is employed. Then different classifiers are applied and evaluated. In particularly: Gaussian models, Logistic Regression, Support Vector Machine (linear, RBF and quadratic) and Gaussian Mixture Models. First the results of classifiers are measured in terms of minimum Detection Cost Function (min DCF), then also actual DCF is taken in account with a discussion about score calibration.  
To select best hyperparameters for each model a validation dataset is extracted from training data set and K-fold cross-validation is employed.

In the end the best models with the selected hyperparameters are tested on the evaluation test to verify if the models works well on unseen (evaluation) data.

## HTRU2 Data Set

HTRU2 is a data set which describes a sample of pulsar candidates collected during the High Time Resolution Universe Survey (South) [2].

**Pulsars** are a rare type of Neutron star that produce radio emission detectable here on Earth.

As pulsars rotate, their emission beam sweeps across the sky, and when this crosses our line of sight, produces a detectable pattern of broadband radio emission. As pulsars rotate rapidly, this pattern repeats periodically. Thus pulsar search involves looking for periodic radio signals with large radio telescopes.

Each sample in the HTRU2 Data set is a **'candidate'**, a potential signal detection known averaged over many rotations of the pulsar, as determined by the length of an observation. In the absence of additional info, each candidate could potentially describe a real pulsar. However in practice almost all detections are caused by radio frequency interference (RFI) and noise, making legitimate signals (actual pulsar) hard to find.

HTRU2 contains a set of pulsar candidates, the actual pulsar are labeled with 1, while the others samples are labeled with 0. More in details the HTRU2 data set contains 16,259 spurious examples caused by RFI/noise, and 1,639 real pulsar examples. These examples have all been checked by human annotators. The HTRU2 is thus **highly imbalanced**: the legitimate pulsar examples are a minority positive class, and spurious examples the majority negative class.

**Classification systems** which treat the candidate data sets as **binary classification problems** can be employed to automatically label pulsar candidates to facilitate rapid analysis. Results of these classifiers are reported in the next sections of this document.

Each candidate is described by **8 continuous variables**. The first four are simple statistics obtained from the integrated pulse profile (folded profile). This is an array of continuous variables that describe a longitude-resolved version of the signal that has been averaged in both time and frequency (see [3] for more details). The remaining four variables are similarly obtained from the DM-SNR curve (again see [3] for more details). Each samples has the features (where the last one is the label) summarized below:

0. Mean of the integrated profile.

1. Standard deviation of the integrated profile.

2. Excess kurtosis of the integrated profile.

3. Skewness of the integrated profile.

4. Mean of the DM-SNR curve.

5. Standard deviation of the DM-SNR curve.

6. Excess kurtosis of the DM-SNR curve.

7. Skewness of the DM-SNR curve.

8. Class

The HTRU2 contains 17,898 total examples: 1,639 positive examples (labeled with 1) and 16,259 negative examples (labeled with 0).

The dataset has been split into **Train** and Evaluation (**Test**) data.

The training set contains 8108 negative examples and 821 positive examples.

The test set contains 8151 negative examples and 818 positive examples

# Analysis of the features

HTRU2 contains 8 continuous variables. The mean (µ) and standard deviation (σ) of each features for the **training dataset** are:

It can be observed that features of the data set have different scales, they have large differences between their ranges. So, in this case, **Z-normalization** on the data-set to bring all the features on the same scale could be useful. Z-normalization centers the feature columns at mean 0 with standard deviation 1. Thus before applying any operation each sample of the training set has been transformed through the expression:

Where is the sample after the Z-score normalization, while is the original sample in the original data set.

## Histograms

It is useful to analyze the distribution of the features, since some of the classifiers applied in the following make assumptions on the distribution of data. First the histogram of features of different classes for the entire training data set are plotted:



The 8 features result easy separable. The histograms shows that in the most of the cases features of class 0 are concentrated around a certain value. The shape of the histograms of features of class 0 is expected to be good for some classifiers that will be implemented (Gaussian classifiers). On the other hand the features of class 1 results more spread and irregular. So, it could be worth to apply **Gaussianization** to the features. This is a pre-processing technique which helps to deal with the fact that some distributions are not Gaussian like. Since one of the technique we will apply is the Gaussian classifier, we will apply Gaussianization pre-processing stage which transforms our features sample as if they behave more likely as a Gaussian distribution.

Gaussianization allows to transform features in such a way that the empirical cumulative distribution of the transformed features is the same as that of a Gaussian distribution. In particularly we will consider as a target a Gaussian distribution with unit variance and zero mean. To perform Gaussianization we will compute empirical c.d.f on the features, we will map this c.d.f. to a uniform distribution and then we will transform it to Gaussian c.d.f.

For each feature x we have to compute the **rank** as:

the rank basically counts the number of samples of the features that have a lower value than and divide it by the number of samples . The rank expression adds 1 to the counts the number of samples of the features that have a lower value than and adds 2 to number of samples for numerical reasons: rank(x)=0 should be avoided because it would lead to a gaussianized sample mapped to minus infinity.

is basically the empirical cumulative distribution for the data. Each sample is replaced with a sample that has the same value for the cumulative distribution but the new sample will have a cumulative distribution assumed to be taken from a Gaussian density. The purpose of Gaussianization is to find a value such that the cumulative distribution of is equal to the rank of .

We assume that is the normal gaussian cumulative distribution. Solving with respect to y we obtain:

Where is called **percent point function (p.p.f)** and is the inverse of the cumulative distribution function of the standard normal distribution.

Scipy library’s implementation of p.p.f is used in the code [4]

Gaussianization should be applied to both training and evaluation . The ranking is computed using only the training data; evaluation data are single samples, ranking of the evaluation samples is computed comparing them to the training samples.

Below histograms of each feature after Gaussianization are showed.



Gaussianization process ignore the classes, it works considering the whole dataset, it makes the **whole dataset** more Gaussian like, so samples of one class could be quite different from a Gaussian distribution.

It can be observed that HTRU2 features of class 0 after Gaussianization have a shape very similar to a Gaussian distribution. Features of class 1 after Gaussianization look more likely to a Gaussian distribution rather than before Gaussianization, though they are still quite asymmetrical.

In the following we will compare the results of classifiers applied to the samples Gaussianized and not Gaussianized (**raw features**). It is expected that some classifiers will be heavily affected by the Gaussianization while other will be less affected.

## Correlation

To analyze the interaction of features we can compute the **correlation** of features. This is useful because it suggest if PCA could be useful and how many features can be discarded.  
**Pearson correlation coefficient** is considered to measure correlation, it can be computed as:

Actually absolute value of Pearson correlation is considered because we are only interested to understand if there is correlation or not

Which basically is the covariance scaled by the square root of the variances. The abs Pearson correlation coefficient can take value between 0 and 1. If 0 it means that the 2 features are uncorrelated, while 1 means that the features are completely correlated: actually one feature is the scaled version of the other.  
To plot the correlation between features we use heatmap where darker color means high correlation between 2 features, while lighter color means low correlation between 2 features.  
We plot the correlation considering: the whole data training samples (grey); training samples belonging to class 0 (orange); training samples belonging to class 1 (green).

Correlation of features without Gaussianization. On the left all the data set is considered; in the center only samples of class 1 are considered; on the right only the samples of class 0 are considered.



The strongest correlation can be observed on couples of features 6-7, 5-6, 0-1, 2-3. It may be worth to map the features to a space with lower dimensionality than 8. In the following will be considered result of the classifiers after applying PCA up to m = 5.  
Furthermore these heatmaps suggest that Gaussian classifier with diagonal covariance matrix won’t perform better than full covariance Gaussian classifier since there is high correlations between features; anyway we will evaluate also the Gaussian classifiers with diagonal covariance matrix to compare them with other Gaussian classifiers.   
We can also plot the correlation of Gaussianized features:

The observation made for the non-Gaussianized features are still valid and even strengthened for the Gaussianized features because the correlation of the previous quoted couple of features seems to be increased.

Correlation of features after Gaussianization. On the left all the data set is considered; in the center only samples of class 0 are considered; on the right only the samples of class 1 are considered.



In particular, after Gaussianization the correlation between features that were strongly correlated before Gaussianization has increased, while the correlation between features that were weakly correlated before Gaussianization has decreased.

# Validation approach

In the following different models for classification will be considered. For each of them training data is used to first train and then validate the models. The goal of this job is to identify which kind of classifier performs better on HTRU2 data set and which is the best combination of hyperparameters. Costs of different classifier type are compared. For the binary problem the **cost function** can be separated in 2 components: the goodness of classifier itself and the goodness of the threshold chosen. At the beginning the threshold problem is not considered, we just need to know which classifier better discriminate the scores, for this reason **minimum detection cost (min DCF)** will be used to measure the performance.

The problem of selecting an optimal threshold will be considered afterward.

*Explain more what min DCF is and how is it computed?*

To validate the models a validation set can be extracted from training set. Models are trained using training set (without validation set samples) then scores and min DCF is computed on validation samples. The model can be trained with different combinations of hyperparameters, results (min DCF computed on the scores of validation samples) of the same model with different hyperparameters are compared to find the optimal ones.

this can be done with2 different methods:

* Split the training set in 2 different set: one used for training the models actually; the other set used for validation (**single split**)
* Employ **K-fold cross-validation**

With the single split approach the final classifier will be the same evaluated on the validation set. In this case not all the samples available on of training data set will be used to build the final classifier. The big advantage of single split is that only one model need to be trained, but on the other hand fewer data for training and evaluation are used and this could lead to less robust results.  
The K-fold cross validation consists of splitting the training data set in K folds: K-1 folds are used for training the model, while the last one is used for validation. This process is repeated K times, at each iteration is computed the score for the validation samples. At the end of K iterations, scores has been computed for each sample of training set and min DCF can be computed starting from this set of scores.

After hyperparameters have been selected, the final classifier will be obtained re-training over the whole training data set. So the K-fold cross validation has the advantage of having more data available for training and evaluation (at the end all the samples of training set have been used as training and validation samples). On the other hand decisions are made over the validation set for the models trained using folds and for this reason they may not be optimal for the mode learned from all training data. Furthermore K-fold cross-validation has a big con: require to train and validate models K+1 time and this could require lot of computational time.

Generally if feasible the K-fold cross-validation leads to more robust results rather than single split, for this reason in the following K-fold cross-validation is considered rather than single split. It is desirable to have a high value of K to have access to big amount of training data at each iteration. Thought the amount of time required to estimate model parameters increase with higher value of K, so in the following K= 5 will be considered. Previous attempts with higher value of K have been tried but they required too much computation time especially for the Support Vector Machine and Gaussian Mixture Models.

The results with the single split have been computed just to verify that they are pretty consistent with the results of K-fold cross-validation, the results with single split are not reported in the following because it doesn’t make sense to compare the result of one validation approach with the other.

For each iteration of K-fold cross validation the scores relative to the evaluation set are computed. At the end these scores are put all together and a single min DCF is computed.

Furthermore data are shuffled before splitting, so that the data of different folds are homogeneous.

It has not been specified which application should be used and which conditions the models will be operating. In the next sections of this document different applications are considered, results of validation and evaluation with different prior (*π*) are reported, in particularly the applications considered are:

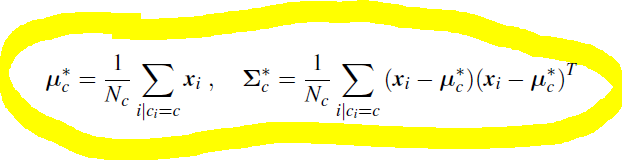
* Uniform prior application with (*π, Cfp, Cfn*) = (0.5, 1, 1)
* Unbalanced application with (*π, Cfp, Cfn*) = (0.1, 1, 1)
* Unbalanced application with (*π, Cfp, Cfn*) = (0.9, 1, 1)

# Multivariate Gaussian Classifiers (MVG)

First models considered are Gaussian classifiers. These are **generative models** (they try to model the class distribution of observed samples). Gaussian classifier assumes that both the training set and evaluation samples are **independent and identically distributed (i.i.d.)** given a set of parameters . In particular Gaussian distribution for a samples given the class is assumed:



The model parameters are estimated using Maximum Likelihood estimation approach. This lead to find:



Where for the HTRU2 dataset there are only 2 classes and . is the number of samples belonging to class .

Explain how to compute score and why for gaussian classifiers

We will consider different models with different assumptions to compute covariance matrix:

* **Full covariance matrix**(Full-cov): compute covariance matrix without any simplifications (with formula?). This model is expected to be robust if the number of training samples is far bigger than number of dimensions of samples. For the HTRU2 dataset good results are foreseen.
* **Diagonal covariance matrix** (Diag-cov): Multivariate Gaussian classifier with **diagonal covariance matrices** where the diagonal element of row-*i* are the variances of the

feature-*i* of the training samples. Diag-cov is a diagonal version of the original full covariance matrix (same elements on the diagonal, 0 in the other elements).This model work well in scenario where features are more or less **independent** (not correlated). In HTRU2 There is significant correlation between features, so the Gaussian classifier with diagonal hypothesis is expected to have poorer results than full covariance Gaussian classifier.

* Immagine che contiene orologio

  Descrizione generata automaticamente**Tied covariance matrix**: assume that Gaussian parameters of different classes are related in one to each other. In particularly this model assume that covariance matrices of different classes are the same, there is only one covariance matrix. This single covariance matrix will somehow corresponds to a weighted average empirical covariance matrix for each class.  
  Both full tied-covariance (**Full Tied-cov**) matrix and diagonal tied-covariance (**Diag Tied-cov**)

This model works well when there are classes with few samples. For HTRU2 The tied hypothesis is expected to have good performance because the training data set is highly unbalanced and the class 1 of pulsar samples have fewer samples.

Since computation of tied-covariance try to average the empirical covariance matrix of each class good results are obtained when the covariance matrices of the classes are pretty similar. In the case of HTRU2 the covariances of class 0 and class 1 are:

**Sigma0** = array([[ 3.01695102e+02, 4.47467614e+01, -4.91914658e+00,

-1.04100586e+01, 5.18711514e+01, 3.57404026e+01,

-8.72826407e+00, -1.45002432e+02],

[ 4.47467614e+01, 3.79643442e+01, -8.34824664e-01,

-4.78776317e+00, 3.40806244e+01, 2.06992618e+01,

-4.19387224e+00, -5.99792293e+01],

[-4.91914658e+00, -8.34824664e-01, 1.11728324e-01,

2.28599653e-01, -9.07479327e-01, -4.83949358e-01,

1.23449247e-01, 1.90352730e+00],

[-1.04100586e+01, -4.78776317e+00, 2.28599653e-01,

1.10907534e+00, -1.25563917e+00, -8.50331704e-01,

2.45684093e-01, 3.92280716e+00],

[ 5.18711514e+01, 3.40806244e+01, -9.07479327e-01,

-1.25563917e+00, 5.99265507e+02, 3.14631711e+02,

-5.76687513e+01, -7.86408122e+02],

[ 3.57404026e+01, 2.06992618e+01, -4.83949358e-01,

-8.50331704e-01, 3.14631711e+02, 2.76925837e+02,

-5.43337138e+01, -9.58728197e+02],

[-8.72826407e+00, -4.19387224e+00, 1.23449247e-01,

2.45684093e-01, -5.76687513e+01, -5.43337138e+01,

1.77746220e+01, 4.10277342e+02],

[-1.45002432e+02, -5.99792293e+01, 1.90352730e+00,

3.92280716e+00, -7.86408122e+02, -9.58728197e+02,

4.10277342e+02, 1.09097417e+04]])

**Sigma1=** array([[ 906.24754263, 143.53278507, -53.41992963, -360.56823239,

-754.55216452, -281.40568834, 53.31643615, 580.08601988],

[ 143.53278507, 61.30877511, -10.64875778, -83.94451429,

-27.40806933, -25.69966253, 5.00234955, 74.06596916],

[ -53.41992963, -10.64875778, 3.59370445, 26.17378747,

43.41362902, 17.62625441, -3.15788023, -34.19336742],

[-360.56823239, -83.94451429, 26.17378747, 203.60189672,

277.14049597, 111.22547442, -19.70554703, -209.48151579],

[-754.55216452, -27.40806933, 43.41362902, 277.14049597,

2047.21521891, 641.48416118, -99.95589218, -837.8513041 ],

[-281.40568834, -25.69966253, 17.62625441, 111.22547442,

641.48416118, 397.00076262, -53.70259914, -612.22604233],

[ 53.31643615, 5.00234955, -3.15788023, -19.70554703,

-99.95589218, -53.70259914, 9.57649728, 130.31928408],

[ 580.08601988, 74.06596916, -34.19336742, -209.48151579,

-837.8513041 , -612.22604233, 130.31928408, 2261.14205048]])

Min DCF results for MVG (with different assumptions on covariance matrix) with K-fold cross validation (K=5)

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
|  | Raw features – no PCA | | |  | Gaussianized features – no PCA | | |
| **π** | **π= 0.5** | **π= 0.1** | **π=0.9** |  | **π= 0.5** | **π= 0.1** | **π=0.9** |
| Full-Cov | 0.141 | 0.284 | 0.669 | 0.153 | 0.246 | 0.697 |
| Diag-Cov | 0.193 | 0.314 | 0.746 | 0.153 | 0.276 | 0.606 |
| Tied Full-Cov | 0.112 | 0.223 | 0.574 | 0.131 | 0.232 | 0.534 |
| Tied Diag-Cov | 0.161 | 0.265 | 0.580 | 0.163 | 0.291 | 0.614 |
|  | Raw features – PCA m = 7 | | | Gaussianized features – PCA m = 7 | | |
| **π** | **π= 0.5** | **π= 0.1** | **π=0.9** | **π= 0.5** | **π= 0.1** | **π=0.9** |
| Full-Cov | 0.139 | 0.301 | 0.643 | 0.153 | 0. 245 | 0.690 |
| Diag-Cov | 0.214 | 0.505 | 0.724 | 0.163 | 0.246 | 0.660 |
| Tied Full-Cov | 0.112 | 0.222 | 0.573 | 0.134 | 0.243 | 0.533 |
| Tied Diag-Cov | 0.138 | 0.270 | 0.599 | 0.137 | 0.253 | 0.562 |
|  | Raw features – PCA m = 6 | | | Gaussianized features – PCA m = 6 | | |
| **π** | **π= 0.5** | **π= 0.1** | **π=0.9** | **π= 0.5** | **π= 0.1** | **π=0.9** |
| Full-Cov | 0.153 | 0.287 | 0.648 | 0.154 | 0.241 | 0.695 |
| Diag-Cov | 0.223 | 0.525 | 0.721 | 0.156 | 0.239 | 0.643 |
| Tied Full-Cov | 0.140 | 0.258 | 0.582 | 0.136 | 0.247 | 0.545 |
| Tied Diag-Cov | 0.164 | 0.297 | 0.590 | 0.141 | 0.255 | 0.588 |
|  | Raw features – PCA m = 5 | | | Gaussianized features – PCA m = 5 | | |
| **π** | **π= 0.5** | **π= 0.1** | **π=0.9** | **π= 0.5** | **π= 0.1** | **π=0.9** |
| Full-Cov | 0.150 | 0.250 | 0.642 | 0.152 | 0.246 | 0.698 |
| Diag-Cov | 0.220 | 0.453 | 0.733 | 0.156 | 0.240 | 0.633 |
| Tied Full-Cov | 0.150 | 0.261 | 0.574 | 0.136 | 0.247 | 0.542 |
| Tied Diag-Cov | 0.171 | 0.311 | 0.604 | 0.141 | 0.256 | 0.594 |

Tied Full covariance MVG obtains the best results both in raw and Gaussianized features with and without PCA; this is due to the fact that class 1 samples are fewer than class 0 samples. Neither Gaussianization nor PCA improved the performance at all, in fact the best results are obtained with raw features without PCA. All the models obtain poorer result when PCA is employed.

Diagonal covariance matrix MVG performs worse than full covariance matrix since, as shown before, there is strong correlation between features and the diagonal hypothesis is not accurate.

Finally, the results related to unbalanced application with are far from results of other applications since the HTRU2 dataset actually has much more samples belonging to class 0 rather than samples belonging to class 1. Bad outcomes for unbalanced application with are expected also for the other classifier models.

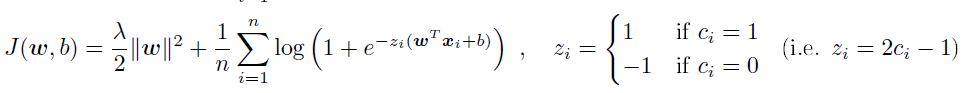
# Logistic Regression

The attention is now turned to **discriminative models** starting from Logistic Regression. Discriminative models try to directly model the class posterior distribution, rather than modelling the distribution of observed samples (generative models).

It can be expected that PCA has limited effects on Logistic Regression models, as well as Gaussianization, since discriminative models does not require specific assumptions on data distribution.

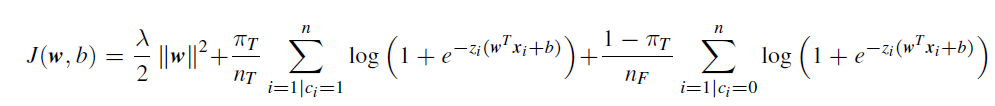
## Linear Logistic Regression

First we consider a **regularized linear Logistic Regression.** This model has 2 parameters that can be found minimizing the expression reported here:



expression correspond to the average **cross-entropy** between the distribution of observed and predicted labels plus a **regularization term**  .is a hyper-parameter that allows specifying the relative weight of the regularization term.

Since classes on training set are not balanced, the expression has been modified as:

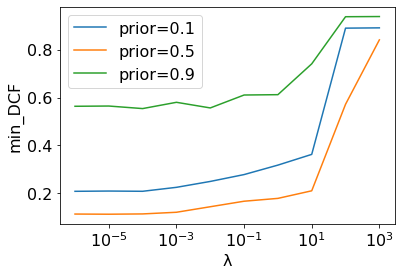


Initially is considered the main application, thus πT is set to 0.5. Then results with πT = 0.1 and

πT =0.9 will be explored.

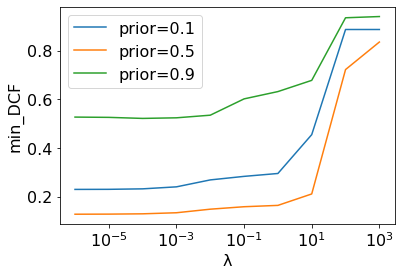
*scipy.optimize.fmin\_l\_bfgs\_b*[7] is employed to find minimum of . Then **score** of a sample can be computed as:

To understand which is a good value for , min DCF for logistic regression with different values of are computed. The results are reported for raw (left) and Gaussianized (right) features:



Linear Logistic Regression (πT=0.5) : Min DCF with respect to ––

Raw features – no PCA



Linear Logistic Regression (πT=0.5) : Min DCF with respect to –– Gaussianized features – no PCA

Values of ranging from 10-6 to 103 . The x-axis is reported in logarithm scale.

The Gaussianization does not affect the results, the plots remain quite similar both for raw and Gaussianized features. So, in the following only the results for raw features are considered.

From these plot we can observe that terms does not help in reducing the min\_DCF. Considering the plot of the application with prior π=0.5 the lowest value of min\_DCF corresponds to the lowest value of (10-6). So, best results are expected for with very low values. is considered.

Min DCF results for Linear Logistic Regression (with different values of πT) with K-fold cross validation (K=5)

|  |  |  |  |
| --- | --- | --- | --- |
| Raw features – no PCA | | | |
| **π** | **π= 0.5** | **π= 0.1** | **π=0.9** |
| LogReg(λ=10-6, πT=0.5 ) | 0.117 | 0.216 | 0.526 |
| LogReg(λ=10-6, πT=0.1 ) | 0.113 | 0.211 | 0.555 |
| LogReg(λ=10-6, πT=0.9 ) | 0.119 | 0.220 | 0.511 |
| Raw features –PCA m = 7 | | | |
| **π** | **π= 0.5** | **π= 0.1** | **π=0.9** |
| LogReg(λ=10-6, πT=0.5 ) | 0.114 | 0.215 | 0.542 |
| LogReg(λ=10-6, πT=0.1 ) | 0.112 | 0.209 | 0.561 |
| LogReg(λ=10-6, πT=0.9 ) | 0.116 | 0.217 | 0.528 |
| Raw features –PCA m= 6 | | | |
| **π** | **π= 0.5** | **π= 0.1** | **π=0.9** |
| LogReg(λ=10-6, πT=0.5 ) | 0.126 | 0.242 | 0.578 |
| LogReg(λ=10-6, πT=0.1 ) | 0.126 | 0.236 | 0.565 |
| LogReg(λ=10-6, πT=0.9 ) | 0.134 | 0.240 | 0.523 |
| Raw features –PCA m = 5 | | | |
| **π** | **π= 0.5** | **π= 0.1** | **π=0.9** |
| LogReg(λ=10-6, πT=0.5 ) | 0.136 | 0.230 | 0.602 |
| LogReg(λ=10-6, πT=0.1 ) | 0.134 | 0.236 | 0.598 |
| LogReg(λ=10-6, πT=0.9 ) | 0.143 | 0.242 | 0.555 |

On the main application () and on application with prior all the 3 models obtain similar results, Logistic Regression performs slightly better in this 2 case.

On the unbalanced application with prior the outcomes are quite worse as expected, in this case Logistic Regression would be the best choose.

PCA doesn’t improve the performance at all, results on data with PCA m=7 are quite similar to results on data without PCA. While outcomes on PCA with m<7 are even worse than outcomes on data without PCA.

Min DCF of Logistic Regression has similar values to min DCF of full tied-covariance MVG model.

## Quadratic Logistic Regression

# Support Vector Machine (SVM)

Support Vector Machine (SVM) is a supervised discriminative model for classification. Given a set of training examples, each marked as belonging to one of two categories, an SVM training algorithm builds a model that assigns new examples to one category or the other. Training samples are mapped to points in space so as to maximize the **margin**: width of the gap between the two categories.

## Linear SVM

Linear SVM looks for separation hyperplane which maximizes the margin. The linear SVM objective consists in minimizing the expression (**primal formulation**):

Where actually include also the bias term b and is extended with 1:

The **dual formulation** of the linear SVM is used. This can be obtained using the Lagrange multiplier and become the maximization of the expression:

Where

For solving the maximization of the dual formulation of SVM *scipy.optimize.fmin\_l\_bfgs\_b* [6] is used. This implementation compute minimizer of a function, so actually minimization of is considered rather than maximization of .

Once solution with respect to is computed, the primal solution can be retrieved as:

At this point the **score** for a sample is computed as:

To make the classes balanced different values of hyperparameter C for the 2 different classes are used:

* For samples of class 1 the hyperparameter considered is
* For samples of class 1 the hyperparameter considered is

Where and are the empirical priors, i.e. number of samples of class 1 and 0 over the total number of samples in training set. In the following are considered linear SVM with ; , as well as the linear SVM without re-balance.

For linear SVM we need to tune the hyperparameter C. As we did for selecting λ in Logistic Regression section, min\_DCF with respect to different values of C is plotted and the optimal value of C is chosen from this plot. Linear SVM with is considered for this plot.



Linear SVM (): Min\_DCF with respect to C - Gaussianized features – no PCA

Linear SVM (): Min\_DCF with respect to C –

raw features – no PCA



C

In the range between 10-2 ­and 102 the min DCF is quite steady both for Gaussianized and raw data. Outside this range the min DCF has higher values.

C=1 as optimal value for the hyperparameter is selected.

Since there is no any improvement with the Gaussianized features, only the results on raw features are considered in the following.

Min DCF results for Linear SVM (with different values of πT) with K-fold cross validation (K=5)

|  |  |  |  |
| --- | --- | --- | --- |
| Raw features – no PCA | | | |
| **π** | **π= 0.5** | **π= 0.1** | **π=0.9** |
| Linear SVM(C=1, πT =0.5 ) | 0.112 | 0.221 | 0.553 |
| Linear SVM(C=1, πT =0.1 ) | 0.113 | 0.213 | 0.543 |
| Linear SVM(C=1, πT =09 ) | 0.123 | 0.227 | 0.539 |
| Linear SVM(C=1, no-rebalancing) | 0.112 | 0.213 | 0.551 |
| Raw features –PCA m=7 | | | |
| **π** | **π= 0.5** | **π= 0.1** | **π=0.9** |
| Linear SVM(C=1, πT =0.5 ) | 0.112 | 0.220 | 0.559 |
| Linear SVM(C=1, πT =0.1 ) | 0.112 | 0.213 | 0.554 |
| Linear SVM(C=1, πT =09 ) | 0.120 | 0.225 | 0.537 |
| Linear SVM(C=1, no-rebalancing) | 0.113 | 0.212 | 0.554 |
| Raw features – PCA m=6 | | | |
| **π** | **π= 0.5** | **π= 0.1** | **π=0.9** |
| Linear SVM(C=1, πT =0.5 ) | 0.125 | 0.234 | 0.579 |
| Linear SVM(C=1, πT =0.1 ) | 0.121 | 0.236 | 0.608 |
| Linear SVM(C=1, πT =09 ) | 0.138 | 0.247 | 0.528 |
| Linear SVM(C=1, no-rebalancing) | 0.122 | 0.237 | 0.608 |
| Raw features –PCA m=5 | | | |
| **π** | **π= 0.5** | **π= 0.1** | **π=0.9** |
| Linear SVM(C=1, πT =0.5 ) | 0.130 | 0.229 | 0.593 |
| Linear SVM(C=1, πT =0.1 ) | 0.123 | 0.231 | 0.628 |
| Linear SVM(C=1, πT =09 ) | 0.147 | 0.253 | 0.553 |
| Linear SVM(C=1, no-rebalancing) | 0.121 | 0.233 | 0.625 |

It can be observed that Linear SVM with πT =0.1, Linear SVM with πT =0.5 and SVM without rebalancing obtain similar results, while SVM with πT = 0.9 performs slightly worse. So, re-balance doesn’t improve results, but it doesn't even downgrade them.

As for Logistic Regression and MVG models PCA doesn’t improve the performance: results on data with PCA m=7 are quite similar to results on data without PCA. While outcomes on PCA with m<7 are even worse than outcomes on data without PCA.

On the unbalanced application with prior the outcomes are quite worse than outcomes on application with prior and with prior as expected.

Linear SVM with πT =0.1, Linear SVM with πT =0.5 and Linear SVM without rebalancing on raw data without PCA obtain similar results to Linear Logistic Regression and Full Tied-Covariance MVG classifiers.

## Quadratic SVM

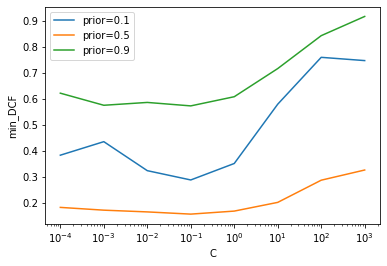
Considering the dual formulation of SVM problem, embedding a non–linear transformation only requires to perform dot–products in the expanded space. It is possible to compute a linear separation surface in the expanded space, which corresponds to a non–linear separation surface in the original feature space.

The **kernel function *k***computes the dot product in the expanded space:

The matrix H becomes:

The quadratic (non-linear) SVM provides quadratic separation surfaces, it has a kernel computed as:

As for linear SVM the min DCF for different values of C is plotted to choose the best value of hyperparameter C



Quadratic SVM (): Min\_DCF with respect to C - gaussianized features – no PCA

observing the plot C=0.1 is selected

Min DCF results for Quadratic SVM (with different values of πT) with K-fold cross validation (K=5)

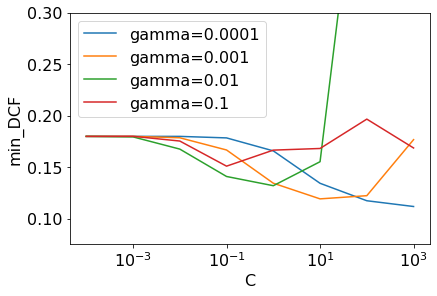
|  |  |  |  |
| --- | --- | --- | --- |
| Raw features – no PCA | | | |
| π | π= 0.5 | π= 0.1 | π=0.9 |
| Quadratic SVM(C=0.1, πT=0.5 ) | 0.156 | 0.287 | 0.573 |
| Quadratic SVM(C=0.1, πT=0.1 ) | 0.191 | 0.352 | 0.632 |
| Quadratic SVM(C=0.1, πT=09 ) | 0.140 | 0.255 | 0.542 |
| Quadratic SVM(C=0.1, no-rebalancing) | 0.173 | 0.271 | 0.616 |
| Raw features – PCA m =7 | | | |
| π | π= 0.5 | π= 0.1 | π=0.9 |
| Quadratic SVM(C=0.1,γ πT=0.5 ) |  |  |  |
| Quadratic SVM(C=0.1, πT=0.1 ) |  |  |  |
| Quadratic SVM(C=0.1, πT=09 ) |  |  |  |
| Quadratic SVM(C=0.1, no-rebalancing) |  |  |  |
| Raw features – PCA m =6 | | | |
| π | π= 0.5 | π= 0.1 | π=0.9 |
| Quadratic SVM(C=0.1,γ πT=0.5 ) | 0.160 | 0.274 | 0.607 |
| Quadratic SVM(C=0.1, πT=0.1 ) | 0.188 | 0.317 | 0.640 |
| Quadratic SVM(C=0.1, πT=09 ) | 0.159 | 0.319 | 0.636 |
| Quadratic SVM(C=0.1, no-rebalancing) | 0.191 | 0.319 | 0.636 |
| Raw features – PCA m =5 | | | |
| π | π= 0.5 | π= 0.1 | π=0.9 |
| Quadratic SVM(C=0.1,γ πT=0.5 ) |  |  |  |
| Quadratic SVM(C=0.1, πT=0.1 ) |  |  |  |
| Quadratic SVM(C=0.1, πT=09 ) |  |  |  |
| Quadratic SVM(C=0.1, no-rebalancing) |  |  |  |

dds

## RBF SVM

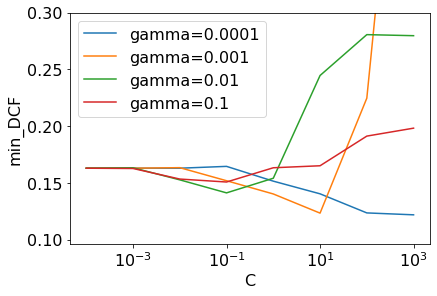
Another type of non-linear SVM can be obtained applying **Radial Basis Function (RBF)** kernel:

RBF introduces an additional hyperparameter which set width of the kernel. So, for this model 2 hyperparameters need to be selected (e.g. and ). To perform this choice min DCF with different with respect to for different values of is plotted:



RBF SVM (): Min\_DCF with respect to C - gaussianized features – no PCA

RBF SVM (): Min\_DCF with respect to C - gaussianized features – no PCA



We consider gamma =0.1 and C=1

|  |  |  |  |
| --- | --- | --- | --- |
| Raw features – no PCA | | | |
| π | π= 0.5 | π= 0.1 | π=0.9 |
| RBF SVM(C=1,γ=0.1, πT=0.5 ) | 0.168 | 0.267 | 0.591 |
| RBF SVM(C=1, γ=0.1, πT=0.1 ) | 0.172 | 0.274 | 0.620 |
| RBF SVM(C=1, γ=0.1, πT=09 ) |  |  |  |
| RBF SVM(C=1, γ=0.1, no-rebalancing) | 0.173 | 0.271 | 0.616 |
| Raw features – PCA m =7 | | | |
| π | π= 0.5 | π= 0.1 | π=0.9 |
| RBF SVM(C=1, γ=0.1, πT=0.5 ) |  |  |  |
| RBF SVM(C=1, γ=0.1, πT=0.1 ) |  |  |  |
| RBF SVM(C=1, γ=0.1, πT=09 ) | 0.160 | 0.270 | 0.560 |
| RBF SVM(C=1, γ=0.1, no-rebalancing) |  |  |  |
| Raw features – PCA m =6 | | | |
| π | π= 0.5 | π= 0.1 | π=0.9 |
| RBF SVM(C=1, γ=0.1, πT=0.5 ) | 0.168 | 0.267 | 0.596 |
| RBF SVM(C=1, γ=0.1, πT=0.1 ) | 0.171 | 0.273 | 0.614 |
| RBF SVM(C=1, γ=0.1, πT=09 ) | 0.162 | 0.275 | 0.576 |
| RBF SVM(C=1, γ=0.1, no-rebalancing) |  |  |  |
| Raw features – PCA m =5 | | | |
| π | π= 0.5 | π= 0.1 | π=0.9 |
| RBF SVM(C=1, γ=0.1, πT=0.5 ) |  |  |  |
| RBF SVM(C=1, γ=0.1, πT=0.1 ) |  |  |  |
| RBF SVM(C=1, γ=0.1, πT=09 ) |  |  |  |
| RBF SVM(C=1, γ=0.1, no-rebalancing) |  |  |  |

# Gaussian Mixture Models

Last model considered is Gaussian Mixture Model (GMM). This a generative approach which can approximate generic distribution, there is no supposition on distribution of data like in the Multivariate Gaussian Models (MVG). So, GMM is expected to obtain better results than Gaussian models.  
Full covariance, diagonal covariance, full tied covariance and diagonal covariance models are considered. Fix width of covariance matrix is considered to avoid singular models and numerical issues.

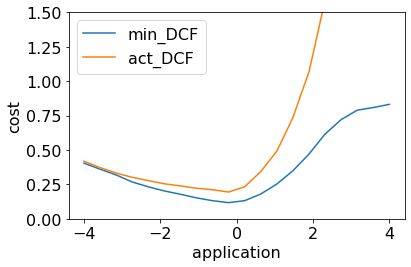
# Score calibration

Up to now the metric min\_DCF has been used to compare the models. This is the cost if the optimal threshold is known. In other words, min\_DCF measures the cost to pay if optimal decisions for the evaluation set are made using the recognizer scores. However what it is paid in practice is not the minimum cost, but it is the **acutal cost**. In practice it is not possible to know the optimal threshold for evaluation data, this would require to know the evaluation labels, but this is impossible.  
If scores are well calibrated the optimal threshold is a threshold that optimize the Bayes risk and is given by the expression of the **theoretical threshold:**

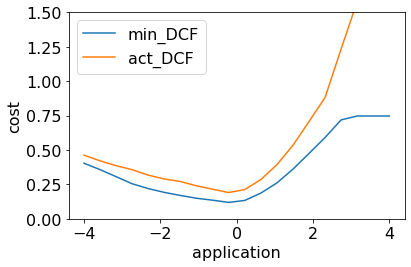
where π is the effective prior.  
To perform decisions starting from scores there are 2 options:

1. **Well calibrated**: assuming scores are we calibrated (because recognizer has given well calibrated scores or because re-calibration on recognizer output has been applied) and we use the theoretical threshold
2. **Not well calibrated**: estimating a good threshold for the target application (using validation set). This approach assumes that scores are not well calibrated, thus for each application looks for a specific good threshold

It can be evaluated if the scores are well calibrated comparing the min\_DCF with the actual cost obtained using the theoretical threshold for each application. This comparison is applied to all the models that have been considered so far to allow to judge if the score calibration need to be applied or not.  
For each model type the best one has been selected to compare the minimum and actual cost:



Min\_DCF vs actual DCF for full tied-covariance Gaussian model raw features no PCA



Min vs act dcf for logistic linear regression lambda=0, pit=0.1, raw features no PCA

How the problem of not well calibrated scores can be solved? We need to estimate an application dependent threshold; given a target application we estimate an optimal threshold. To do this the threshold corresponding to minimum of the DCF on the validation set is taken.   
This approach doesn’t provide scores that are well-calibrated for different applications. For each application we have to re-estimate an optimal threshold on the validation set.  
Before actually using the threshold, it is needed to access whether the threshold is good before using it. Cross validation can be used to select the optimal threshold. Cross validation has an additional problem, we are working on the validation scores themselves, validation scores are treated as if they were a training set for which we want to solve threshold estimating problem. We take all the scores and we need to solve an estimation problem over the score and we want to evaluate how good is the solution.  
Basically the set of scores returned by the kfold function are treated as a training set over which finding optimal threshold. This set of scores will be splitted into training and validation set as well: the training set will be used to estimate the threshold, while the validation set will be used to estimate how good the threshold chosen is. Basically this process can be summarized as:

1. K-fold validation

2) Set of scores

3) shuffle

4) Single/  
kfold validation

HOW TO DO THIS IN CODE?

Alternatively a second approach can be used to estimate optimal threshold. Scores can be transformed in a way that the theoretical threshold can be used. A calibration function Scal = f(s) take as input the scores output of a classifier, gives us new score that then can be compared with the theoretical threshold to obtain optimal decisions. The calibration function f(s) should be **monotone** because the higher non-calibrated scores should favor class 1 while lower non-calibrated scores should favor class 0.   
There are several approaches to do this, in the following **discriminative score model** is applied. This consists on training a new machine learning model directly on the score, a model that has outputs with probabilistic interpretation in terms of log-likelihood ratio. Why a new model is needed and it is not used the original classifier? The original classifier (for example logistic regression model) is trained in a feature space that is very large, small differences in the distribution between validation and evaluation populations may lead to significant errors in term of calibration. The new model for score calibration is trained in a very small space: the space of scores. Each sample will be described by a single score, so the new model will be trained on a one dimensional data. This gives a model with probabilistic interpretation is much stronger because less affected by issues related to overfitting, underfitting or mismatch in distributions of features. Furthermore since the data have one dimension no Z-normalization is needed and so the results score interpretation are no affected by regularization.  
So the idea is to use a transformation function that apply a linear scaling and then sum a byas term:

function f(s) should be interpreted as a **calibrated log-likelihood ratio** for the 2 classes:

If f(s) is a log-likelihood ratio we can use the theoretical threshold to assign the labels. The class posterior probability is obtained as:

So, for this new model we can interpret score as a feature and the log posterior ratio has a very similar form of the log posterior of logistic regression. If we set then log posterior ratio of this new model is actually the same as logistic regression.  
So, we can train a logistic regression model using as prior the application prior that we are interesting in, to estimate a linear transformation of the score that gives us well calibrated score.   
The linear transformation is estimated using the prior-weighted logistic regression model to learn the model parameters . The transformation of the score provided by this mode corresponds to class posterior ratio. The log-likelihood ratio can be obtained starting from the transformation compensating with the prior, the calibrated score f(s) can be retrieved as:

A prior must be specified, the calibration is still optimized for a specific application as for the previous approach. However this kind of approach the model provide good calibration results for different applications. Usually the score calibration solution with this approach works well for a wide range of applications.

# Combining classifiers(fusion approach)

Different classifier based on different assumptions will provide different results; they may agree on some decisions while disagree on others. Combining the decisions of both may result in better predictions labels.   
How can decisions of 2 classifiers be combined? Simple voting scheme approach: each classifiers assign a label and at the end the label assigned more often is selected. The simple voting approach has some issues, if one classifier is almost certain about class 1 and two other classifiers are only slightly in favor of class 0 it not granted that assigning label 0 is a good choice. So, rather than fusing classifiers at decision level, is better to perform a **score-level fusion** voting.   
The idea is to introduce a **fused score** which is a function of the scores of different classifiers. Considering a sample **x**t, if st,A is the score provided by classifier A, while st,B is the score provided by classifier B, the fused score for sample **x**t will be:

The scores of single classifiers should be already calibrated.  
A simple linear form for function f to fuse scores is:

are the parameters to be estimated. The scores of different classifiers are treated as a feature vector. A prior-weighted logistic regression with is used to train the model parameters similar to what has been done for score calibration.

# Experimental results on evaluation set

|  |  |  |  |
| --- | --- | --- | --- |
| Train on all training set - Raw features – no PCA | | | |
| **π** | **π= 0.5** | **π= 0.1** | **π=0.9** |
| Full Cov Gaussian | 0.140 | 0.283 | 0.638 |
| Diag Cov Gaussian |  |  |  |
| Full Tied Cov Gaussian | 0.110 | 0.206 | 0.586 |
| Full Diag Cov Gaussian | 0.150 | 0.260 | 0.549 |
| Linear LogReg(λ=0, πT=0.1 ) | 0.110 | 0.200 | 0.529 |
| Linear SVM (C=1, πT=0.1) | 0.113 | 0.202 | 0.550 |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |

# References

[1] https://archive.ics.uci.edu/ml/datasets/HTRU2

[2] M. J. Keith et al., 'The High Time Resolution Universe Pulsar Survey - I. System Configuration and Initial Discoveries',2010, Monthly Notices of the Royal Astronomical Society, vol. 409, pp. 619-627. DOI: 10.1111/j.1365-2966.2010.17325.x

[3] R. J. Lyon, 'Why Are Pulsars Hard To Find?', PhD Thesis, University of Manchester, 2016.

[4] https://docs.scipy.org/doc/scipy/reference/generated/scipy.stats.norm.html?highlight=norm%20ppf

[5] <https://numpy.org/doc/stable/reference/generated/numpy.corrcoef.html>

[6]<https://docs.scipy.org/doc/scipy/reference/generated/scipy.optimize.fmin_l_bfgs_b.html?highlight=scipy%20optimize%20fmin#scipy.optimize.fmin_l_bfgs_b>

[7] scipy.optimize.fmin\_l\_bfgs\_b