Index

[Introduction 2](#_Toc119344860)

[HTRU2 Data Set 2](#_Toc119344861)

[Analysis of the features 4](#_Toc119344862)

[Histograms 4](#_Toc119344863)

[Correlation 8](#_Toc119344864)

[Validation approach 10](#_Toc119344865)

[Gaussian Classifiers 11](#_Toc119344866)

[Logistic Regression 12](#_Toc119344867)

[Linear Logistic Regression 12](#_Toc119344868)

[Quadratic Logistic Regression 13](#_Toc119344869)

[SVM 14](#_Toc119344870)

[Linear SVM 14](#_Toc119344871)

[RBF SVM 17](#_Toc119344872)

[Gaussian Mixture Models 18](#_Toc119344873)

[Score calibration 19](#_Toc119344874)

[Combining classifiers(fusion approach) 23](#_Toc119344875)

[Experimental results on evaluation set 24](#_Toc119344876)

[References 25](#_Toc119344877)

# Introduction

The goal of this report is to analyze the results of different machine learning algorithms applied to the HTRU2 Data Set [1].

Initially an analysis of the feature will be employed. Then will be applied gaussian, logistic regression, SVM and GMM models….

## 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.

This 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:

1. Mean of the integrated profile.

2. Standard deviation of the integrated profile.

3. Excess kurtosis of the integrated profile.

4. Skewness of the integrated profile.

5. Mean of the DM-SNR curve.

6. Standard deviation of the DM-SNR curve.

7. Excess kurtosis of the DM-SNR curve.

8. Skewness of the DM-SNR curve.

9. 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 **x’** is the sample after the Z-score normalization, while **x** is the original sample in the original data set.

## Histograms

The first thing we can do is plotting the different features of different classes for the entire training set. Below are shown the histograms of the features:



The 8 features result easy separable. The histograms shows that in the most of cases the features of the class 0 are concentrated around certain values. 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 the 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(x) basically counts the number of samples of the features that have a lower value than x and divide it by the number of samples N. The rank add 1 to the numerator and 2 to denominator for numerical reasons: rank(x)=0 should be avoided because it would lead to be mapped to minus infinity.

Rank(x) is basically the empirical cumulative distribution for the data. We replace each sample x 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. We want to find a value y such that the cumulative distribution of y is equal to the rank of x.

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 are used in the code [4]

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

Let’ see the histograms of each feature after Gaussianization.



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 vey 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. 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.  
We can compute the **Pearson correlation coefficient** as:

Actually we will consider the absolute value of Pearson correlation 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 features, while lighter color means low correlation between 2 features.  
We plot the correlation considering: the whole data samples (grey); samples belonging to class 0 (orange); 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 7-8, 6-7, 1-2, 3-4. 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; 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.



# 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. We want to identify which kind of classifier performs better on HTRU2 data set and which is the best combination of hyperparameters.

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 we can extract a validation set from training set. We can use 2 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 and at the end of the operation……How to explain K-fold with few words??

The final trainer 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 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 we will consider K-fold cross-validation 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 = 5/6/7 will be considered. Previous attempts with higher value of K have been tried but they required too much computation time especially for the SVM 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 we will consider different applications:

* 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)

# Gaussian Classifiers

First models considered are Gaussian classifiers. We will consider full, diagonal, tied full an tied diagonal covariance matrices. There is significant correlation between features, so the Gaussian classifier with diagonal hypothesis is expected to have poorer results than full covariance Gaussian classifier. Since full covariance Gaussian models require to estimate more parameters than diagonal covariance models, and this could lead to less robust estimates. So, it worth checking the results of the diagonal covariance Gaussian classifiers even if they are expected to be less powerful.  
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

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
|  | 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 |

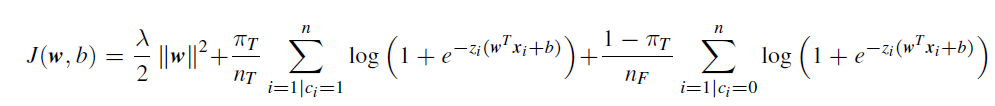
# Logistic Regression

The attention is now turned to discriminative models.

We can expect that PCA has limited effects on this models, as well as Gaussianization, since Logistic Regression is a discriminative model and does not require specific assumptions on data distribution.

## Linear Logistic Regression

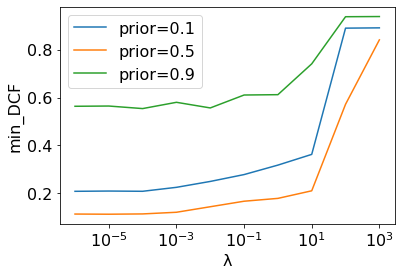
First we consider a **regularized linear Logistic Regression**. Since classes on training set are not balanced, cost of different classes in the logistic regression model is modified with the expression:



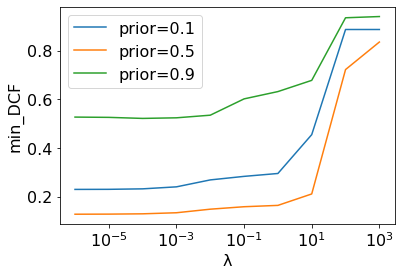
Initially is considered the main application, thus πT is set to 0.5. Then results with πT = 0.1 and

πT =0.9 will be explored.

To understand which is a good value for the regularization term λ, min\_DCF for logistic regression with different values of λ is computed. The results are reported for raw (left) and Gaussianized features:



Raw features



Gaussianized features

Values of λ 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.

From these plot we can observe that λ 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 λ (10-6). So, it can be affirmed that best results are obtained with λ=0.  
Also, the Gaussianization provide no benefit in terms of min\_DCF, so only result with raw features are considered in the following.

|  |  |  |  |
| --- | --- | --- | --- |
| Raw features – no PCA | | | |
| **π** | **π= 0.5** | **π= 0.1** | **π=0.9** |
| LogReg(λ=0, πT=0.5 ) | 0.117 | 0.216 | 0.526 |
| LogReg(λ=0, πT=0.1 ) | 0.113 | 0.211 | 0.555 |
| LogReg(λ=0, πT=09 ) | 0.119 | 0.220 | 0.511 |
| Raw features –PCA m = 7 | | | |
| **π** | **π= 0.5** | **π= 0.1** | **π=0.9** |
| LogReg(λ=0, πT=0.5 ) | 0.114 | 0.215 | 0.542 |
| LogReg(λ=0, πT=0.1 ) | 0.112 | 0.208 | 0.562 |
| LogReg(λ=0, πT=09 ) | 0.116 | 0.217 | 0.528 |
| Raw features –PCA m= 6 | | | |
| **π** | **π= 0.5** | **π= 0.1** | **π=0.9** |
| LogReg(λ=0, πT=0.5 ) | 0.126 | 0.242 | 0.579 |
| LogReg(λ=0, πT=0.1 ) | 0.126 | 0.236 | 0.565 |
| LogReg(λ=0, πT=09 ) | 0.134 | 0.240 | 0.524 |
| Raw features –PCA m = 5 | | | |
| **π** | **π= 0.5** | **π= 0.1** | **π=0.9** |
| LogReg(λ=0, πT=0.5 ) | 0.136 | 0.230 | 0.602 |
| LogReg(λ=0, πT=0.1 ) | 0.134 | 0.235 | 0.598 |
| LogReg(λ=0, πT=09 ) | 0.143 | 0.242 | 0.555 |

## Quadratic Logistic Regression

# 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 look 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 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.

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 are plotted and the optimal value of C is chosen from this plot.



Min\_dcf wrt C for gaussianized features wit piT=0.5

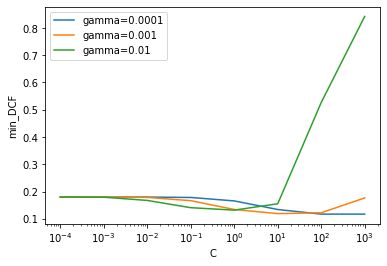
Since there is no any improvement with the Gaussianized features, we’ll consider only the result on the raw features



Min\_dcf wrt C for raw features with piT=0.5

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

## RBF SVM



Min\_DCF wrt to C and different value of gamma, for RBF SVM raw features

# 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.

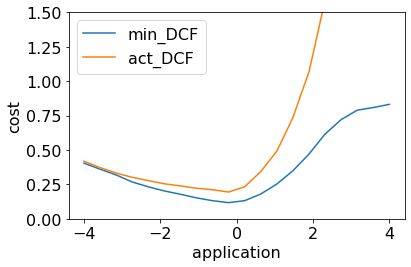
# Score calibration

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:**

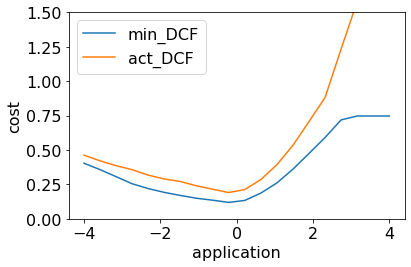
where π is the effective prior.  
To perform decisions starting from scores there are 2 options:

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 – kfold k =5 | | | |
| **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