# Wine Project Report

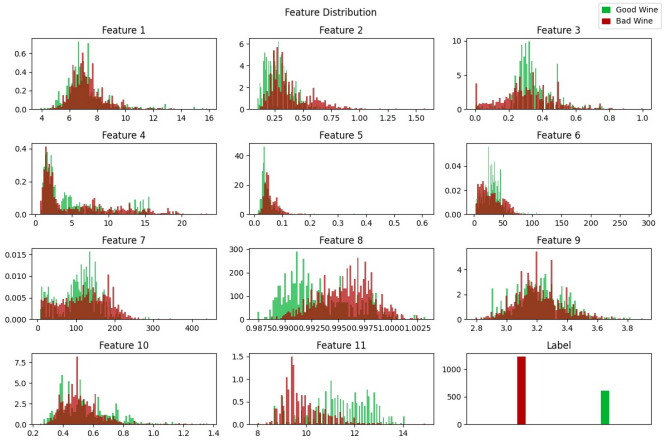
Ruggero Nocera (SXXXXX1)
Quarta Matteo (SXXXXXX)

Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Ut purus elit, vestibulum ut, placerat ac, adipiscing vitae, felis. Curabitur dictum gravida mauris. Nam arcu libero, nonummy eget, consectetuer id, vulputate a, magna. Donec vehicula augue eu neque. Pellentesque habitant morbi tristique senectus et netus et malesuada fames ac turpis egestas. Mauris ut leo. Cras viverra metus rhoncus sem. Nulla et lectus vestibulum urna fringilla ultrices. Phasellus eu tellus sit amet tortor gravida placerat. Integer sapien est, iaculis in, pretium quis, viverra ac, nunc. Praesent eget sem vel leo ultrices bibendum. Aenean faucibus. Morbi dolor nulla, malesuada eu, pulvinar at, mollis ac, nulla. Curabitur auctor semper nulla. Donec varius orci eget risus. Duis nibh mi, congue eu, accumsan eleifend, sagittis quis, diam. Duis eget orci sit amet orci dignissim rutrum.

## Contents



Figure 1: Histogram of Class' Features




# 1 Preliminary Data Analysis

## 1.1 Feature Distribution

Before discussing the varius models and techniques that can give us robust result, we shortly discuss how the feature of our dataset are distributed.

We start by noticing that the order of magnitude of some features can differ much from one another: for example, if we look at feature 5, it seems to span from 0.0 to as much as 0.6, but feature 7 instead spans two whole orders, occasionally being over 200, and with some outliers even double as much. An interesting observation is that some features do look gaussian distributed or regular enough to be described by a combination of gaussian distributions.

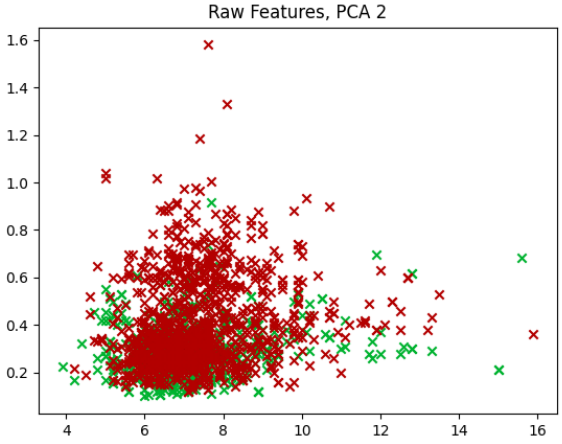
This may hint that some dimensionality reduction will be strongly biased towards features like

7, and that normalization can be taken as a valid pre-processing technique.

We also notice that our dataset is unbalanced: the quantity of bad wine is roughly twice the quantity of good one. This has to be taken into account when developing models, but we will be trating them as if the dataset is balanced, that is considering and equal probability of good wine and bad wine, occasionally rebalancing scores.

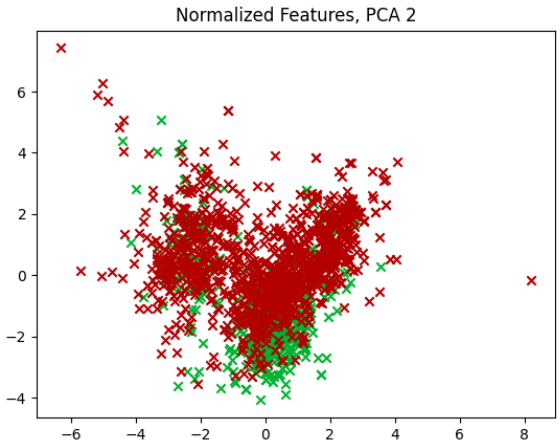
Let's apply a 2D projection of our data to further discuss it.

Figure 2: 2D-PCA Projection




The features look highly correlated and fairly regularly distributed: this may be a hint that some gassuain models may perform well enough, like Full-Covariang MVG Classifiers or Tied-Covariance ones, but some other with linear decision boundaries, like linear SVMs, may perform poorly.

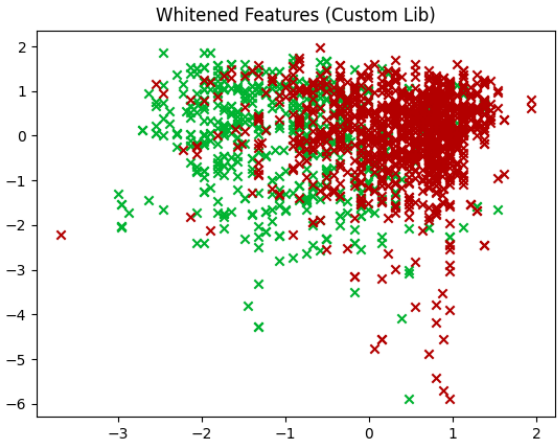
Figure 3: 2D-PCA Projection, Normalized Data




After normalization (Z-Normalization, mean $\mu = 0$ and St.D. $\sigma = 1$ ) our points do get apart but not significantly: we could expect linear models to still perform poorly and gaussian classifiers to get slightly worse. On the other hand, Gaussian Mixture Models look like a more interesting choice.

Lastly we take a look at normalized and whitened data.

Figure 4: 2D-PCA Projection, Whitened Data




Here the results are a bit more interesting: points do get separeted just like after normalization, but same class points stay close to each other while different class get more apart. In this scenario, it looks like even linear model might have

a chance to achieve a decent performance.

# 2 Pre-Processing Analysis

## 2.1 Pre-Processing for Gaussian Models

We will shortly decide, by running some non-optimized models, which pre-processing techniques are useful and which are not.

We will be reporting the best results for each classifiers, extracted from a 3 or 5 Fold Validation approach. The decision rule, since this is a binary problem, will be set to $t = \log(\frac{\pi_T}{1-\pi_T}) \lessgtr 0$, where $\pi_T$ is assumed $= 0.5$ and it is the prior probability of class Good.

Let's start with Gaussian Classifiers: Full-Covariance, Tied-Covariance, and Naive-Bayes. We find out that a small number of features can be removed, with modest gains.

Let's combine these results with normalization and whitening.

| Type | PCA | DCF | minDCF |
|---|---|---|---|
| Raw | 9 | 0.401 | **0.362** |
| Normalized | 7 | 0.464 | 0.420 |
| Whitened | 2 | 0.430 | 0.410 |

Table 1: Full-Covariance MVG - Best Results

| Type | PCA | DCF | minDCF |
|---|---|---|---|
| Raw | 7 | **0.375** | 0.366 |
| Normalized | 9 | **0.375** | 0.371 |
| Whitened | 11 | 0.402 | 0.401 |

Table 2: Tied-Covariance MVG - Best Results

| Type | PCA | DCF | minDCF |
|---|---|---|---|
| Raw | 7 | 0.383 | 0.366 |
| Normalized | 10 | 0.429 | 0.391 |
| Whitened | 5 | 0.402 | 0.386 |

Table 3: Naive-Bayes MVG - Best Results

Now we can finally say something on MVG Classifiers:

- As expected, the Tied Covariance model works fairly well
- Unexpectedly, the Naive-Bayes model outperforms the Full-Covariance one, even without whitening
- Normalization and whitening seem to hurt performance
- PCA can be used, but gains are not particularly high

## 2.2 Pre-Processing for GM Models

Staying of the subject of Gaussian Classifiers, we make similar operations for GMMs. This is a dummy 4-Component model.

Results seem on par, not significantly lower than MVG Classifiers, but GMM actually performs best when keeping as much dimensions as possible.

| Type | PCA | DCF | minDCF |
|---|---|---|---|
| Raw | No | 0.410 | **0.394** |
| Normalized | 9 | **0.407** | 0.402 |
| Whitened | 4 | 0.429 | 0.420 |

Table 4: GMM - Best Results

Also here, normalization and whitening seem to do more bad than good. Results do not get much worse, sometimes even get better but not significantly.

Our best option is to work with raw features on GMMs. Weirdly enough, Whitened GMM can discard many components and still perform fairly well compared to the raw counterpart.

In the next two subsections we will discuss pre-processing on two types of kernel SVMs: Polynomial and RBF ones. Given the non-probabilistic nature of the scores, using a threshold of 0 is somewhat an arbitrary choice: to account for that, we will be giving a greater importance to minimum DCFs rather than plain DCFs.

## 2.3 Pre-Processing for Polynomial Kernel SVMs

As usual we start by looking at the effect of PCA on a dummy Polynomial Kernel. The polynomial kernel used is the following:

$$\phi(\mathbf{x}_1)^T\phi(\mathbf{x}_2) = k(\mathbf{x_1}, \mathbf{x_2}) =$$
$$= (\mathbf{x}_1^T\mathbf{x}_2 + c)^d + b = (\mathbf{x}_1^T\mathbf{x}_2 + 1)^2 + 0.5$$

With $\phi(\cdot)$ being:

$$\phi(\mathbf{x}) = \begin{bmatrix} vec(\mathbf{x}\mathbf{x}^T) \\ \sqrt{2}\mathbf{x} \\ 1 \end{bmatrix} \quad (1)$$

[1]

| Type | PCA | DCF | minDCF |
|---|---|---|---|
| Raw | 7 | 0.554 | 0.545 |
| Normalized | 6 | **0.393** | **0.381** |
| Whitened | 11 | 0.399 | 0.393 |

Table 5: Polynomail Kernel SVM - Best Results

The results are promising: data whitening actually harms performance compared to just normalization, while normalization gives us good results, on par with other models. Many features look like they can be safely removed, sometimes even giving a gain in performance.

## 2.4 Pre-Processing for RBF Kernel SVMs

Just like we did for the polynomial one, here we try again a dummy kernel to see how we should pre-process our data. The dummy kernel function used is:

$$k(\mathbf{x_1}, \mathbf{x_2}) = e^{-\gamma||\mathbf{x}_1-\mathbf{x}_2||^2} + b = e^{-0.05||\mathbf{x}_1-\mathbf{x}_2||^2} + 0.05$$

Small values of $\gamma$ and $b$ were picked to avoid numerical issues when using non-normalized models.

[1] Above we freely set $c$ as it is the final derivation of this mapping, but the original kernel as described here would require it to be 1

The RBF seems to be robust while working with all features or almost, and start behaving worse when the number of dimensions gets much lower.

Given the non-optimed hyperparameters, trying various combination of hyperparameters with high dimensionalities is needed to draw any conclusions.

| Type | PCA | DCF | minDCF |
|---|---|---|---|
| Raw | 10 | 0.588 | 0.579 |
| Normalized | 11 | **0.356** | **0.352** |
| Whitened | 11 | **0.356** | **0.352** |

Table 6: RBF Kernel SVM - Best Results

Just like polynomial kernel SVMs, here normalization has no effect whatsoever whilte normalization greatly improves our performance. The RBF Kernel works significantly better when working with all or most components.

We are not yet discussing perfomance in detail but so far, this is our most promising model.

# 3 Optimizing Models

## 3.1 Optimizing MVG Classifiers