Comparison between Kernels for binary classification problems with a Dual Perceptron

Serban Cristian Tudosie

Abstract– Perceptrons made for binary classification use a hyperplane to linearly classify and split the given data in two distinct classes. Real world problems are usually not linearly separable so an alternative to boost the power of a linear classifier is the usage of kernels. This study puts side by side the accuracy of a Dual Perceptron with three different kernels and three different datasets to empirically evaluate efficiency in classification.

1 Introduction

The adopted version of the Dual Perceptron in this study is the one given by Cristianini and Shawe-Taylor [1]. The basic idea stands in the ability to write the hyperplane as a linear combination of the training points as follows:

$$\boldsymbol{w} = \sum_{i=1}^{l} \alpha_i y_i \boldsymbol{x_i} + b$$

The two coefficients alpha and y are respectively: a vector of positive integers α_i that represents how many times $\boldsymbol{x_i}$ has been misclassified and the corresponding correct label y_i. For the decision function $h(\boldsymbol{x})$ the following function in dual coordinates is used:

$$h(\boldsymbol{x}) = sgn(\langle \boldsymbol{w} \cdot \boldsymbol{x} \rangle + b)$$

$$h(\boldsymbol{x}) = sgn(\sum_{i=1}^{l} \alpha_i y_i \langle \boldsymbol{xi} \cdot \boldsymbol{x} \rangle + b)$$

Briefly, the model learns by counting the number of times it misclassifies an example in the input data. Then after a given amount of iterations on the input data it returns the $\boldsymbol{\alpha}$ vector and the scalar parameter b. The expression inside angular brackets is a kernel i.e. any legal definition of a dot product. In this study the kernels used are:

1. Linear Kernel: $K(\boldsymbol{x}, \boldsymbol{z}) = \boldsymbol{x}^T \boldsymbol{z}$
2. Polynomial Kernel: $K(\boldsymbol{x}, \boldsymbol{z}) = (\boldsymbol{x}^T \boldsymbol{z} + 1)^d$
3. Gaussian Kernel: $K(\boldsymbol{x}, \boldsymbol{z}) = exp(-\frac{||\boldsymbol{x}-\boldsymbol{z}||_2^2}{2\sigma^2})$

Applying kernels help us to exploit explicit features that might help the model gain information to perfectly split the data with a hyperplane. This is called "the kernel trick" because we just have to compute the kernel and not the explicit features.

2 First Difficulties & Drawbacks

One weakness of the model is the inability to converge if the given data is not linearly separable. We are forced to introduce a number of maximum epochs. (a "shortcut") The Gaussian kernel has an infinite dimensional space for the explicit features, thus theoretically with this kernel we have convergence, but this is not what we necessarily want because we might encounter overfitting.
A recurrent idea in implementing this model is the use of a Gram Matrix where we pre-compute the kernel so the fitting algorithm along with the prediction on a test set are computationally faster. Let's take a look into one of the Gram Matrices used in the study:

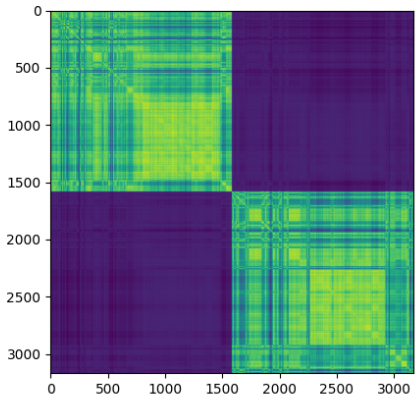

Figure 1: Gram Matrix with Gaussian Kernel of Gender Voice dataset [3]

The disadvantage: the $i-th$ row represents the distance between the $\boldsymbol{x_i}$ example in the dataset and all the other examples, we do that for all the examples in the dataset. So, assume 32 bit float values are used to save our matrix, if a dataset with 50.000 elements is used we end up with a 10GB matrix to store. And as the examples increase the growth of the matrix is quadratic. So for big datasets it wouldn't be practical. [6]

3 The Datasets

Three datasets are taken in consideration in this study, all from mldata.org

1. Bank Marketing [2]
2. Gender Voice [3]
3. Mushroom [4]

Let's plot an insight on one of the datasets to visualize the classes in relation to some of the features.

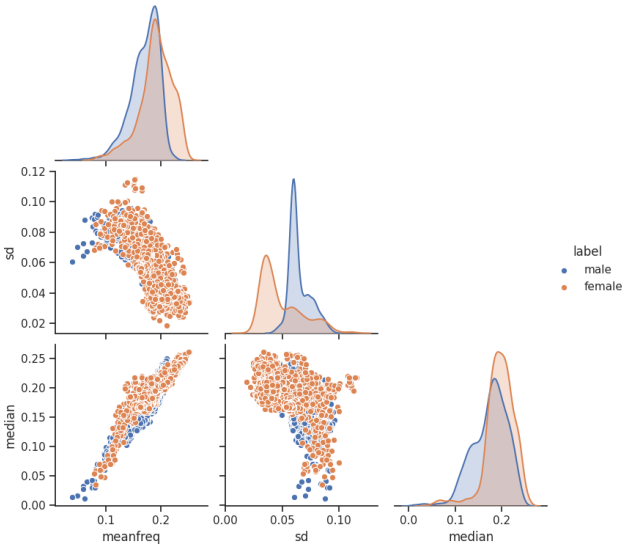

Figure 2: Gender Voice Dataset scatterplots with 3 features using seaborn [5]

This is not enough to determine either the data is linearly separable or not because of the cardinality of the feature space. Thus in the case of N-dimensional feature space (with $N > 3$) we can't visualize the separability by looking at the data itself. Several approaches are possible to do this [13] , the one utilized is solving a linear programming problem. The inspiration is taken from Raffael Vogler [11] and applied in python through scipy.optimize.linprog [12] .
As a consequence, all three datasets in the study are not linearly separable.
The Bank Marketing and Mushroom datasets have string values for some of the features, so a first step in the pre-processing of the data is to map the string values to numerical values. All rows in the datasets are shuffled. Then the dataset can be standardized or normalized. In the study the following procedures have been chosen:

- Bank Marketing: Standardized (some values were numerical and others were strings)
- Gender Voice: Standardized (all the values were floats but had different scales)
- Mushroom: untouched (all values were strings)

4 Implementation & Fitting

To have decent speeds in the evaluation of a given dataset the numba [7] just in time compiler for Python has been used. The Gram Matrix computation and the Prediction on a given test set are done in nopython mode, while the fitting is both in nopython mode and parallel on the cpu.

5 Testing the Perceptron

Let's now explore the performance of the model, a classic train/test split is used, 70% of the dataset is used as the train set and 30% as the test set. A less naive approach would have been the usage of k-fold cross-validation [8].
So, for each dataset and for each kernel a model is created. As a parameter of comparison a Perceptron made with scikit-learn library with default parameters is trained alongside [9] . Each model is tested 10 times, each time the number of epochs is increased by 1, as a result the accuracy of the best epoch is stored. The dimension parameter of the polynomial kernel and the sigma parameter of the gaussian kernel are both set to 5.

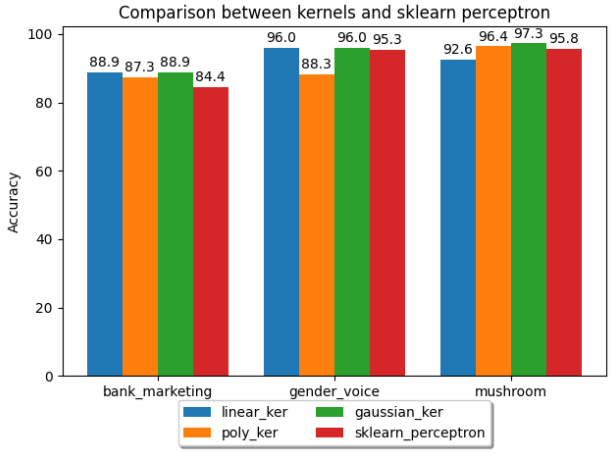

Figure 3: Comparison between kernels

6 Overfitting

The parameters of the kernel are fixed so to get the best out of the kernels one should fit the Perceptron on multiple given parameters and choose the one that gives proper accuracy results. Let's plot now something similar, but instead of testing on our test set we'll test on the train set.

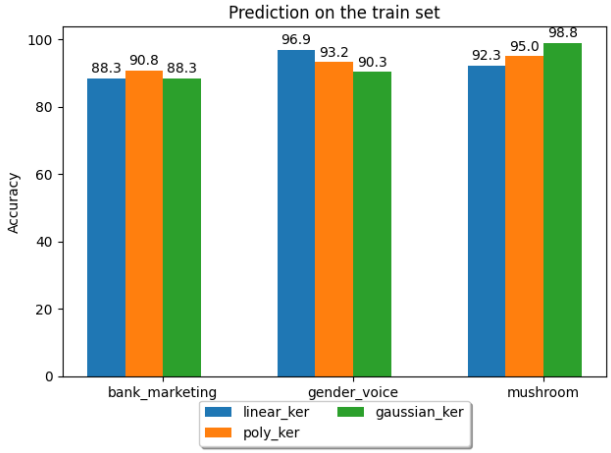

Figure 4: Prediction on the train set

As we can see, performing better on the train set than on the test set means we are overfitting, and this happens with our polynomial kernel in two of the datasets because we did not tune the dimension parameter correctly, so 5 is too high.

7 Final Considerations

We can use cross validation to tune the parameters. The method used in the study to do that takes several pre-determined values for the parameters of the kernels and chooses the one that performs the best. The approach used in NaiveCrossValidation.py is thus prone to overfitting. We can look at the pre-determined values and the one choose by cross validation below for each dataset:

	Values					
dim	1	2	3	4	5	6
sigma	0.001	0.01	0.1	1	2	5

Table 1: Possible Values

	Datasets		
	Bank Marketing	Gender Voice	Mushroom
dim	3	1	5
sigma	0.001	0.1	0.1

Table 2: Chosen values

With the tuned values it is possible to give a more accurate description in terms of performance of the various kernels as shown below:

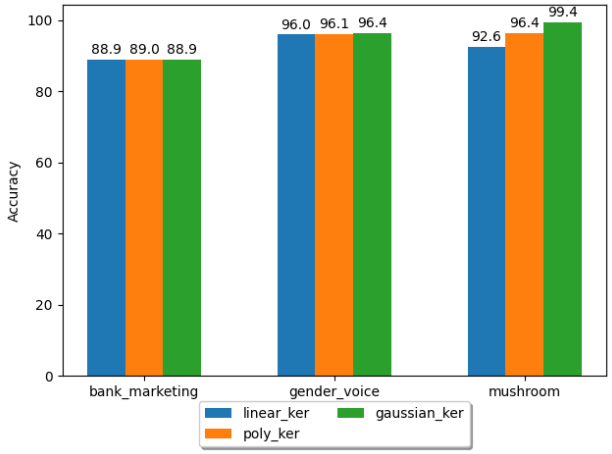

Figure 5: Comparison between kernels

This should answer the question: "What kernel should I use in the context of these three datasets with the dual perceptron?". We know the data is not linearly separable. So why does the perceptron behave similarly on the first two datasets regardless of the kernel?
I think the reason why on the first one and the second one the perceptron performs similarly is because the data itself is "almost linearly separable" so we don't gain much by using the kernels. That's not the case for the Mushroom dataset where we can see that with the gaussian kernel there's a clear discrepancy in terms of accuracy in relation to the other two kernels. (even though the perceptron overfits slightly with the gaussian kernel because of how NaiveCrossValidation.py choses the parameters).

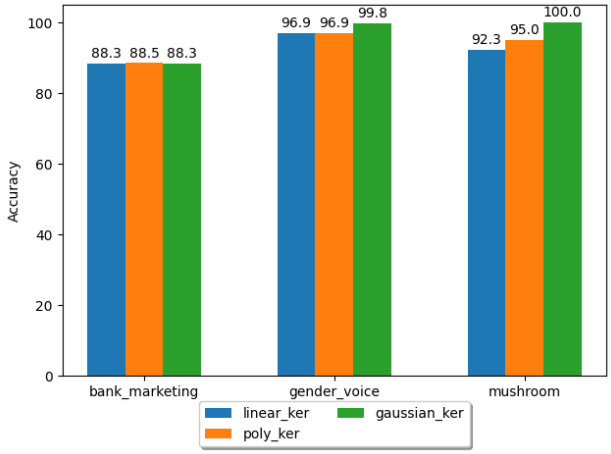

Figure 6: Prediction on the train set

The Dual Perceptron model can be efficient in some cases of binary classification as seen in this study. It also works with non-linearly separable data thanks to the usage of kernels and it's practical in particular cases like the ones we are seeing here, i.e. if the dataset is relatively small, and each feature contains overall information about the relative element.

References

[1] Nello Cristianini, John Shawe-Taylor.
An Introduction to Support Vector Machines and Other Kernel-based Learning Methods.

[2] Bank Marketing Dataset
Predicting client's subscription depending on background.
www.mldata.io/dataset-details/bank_marketing/

[3] Gender Voice Dataset
Determine male or female based on voice cahrac.
www.mldata.io/dataset-details/gender_voice/

[4] Mushroom Dataset
Predict whether a mushroom species is edible or poisonous.
www.mldata.io/dataset-details/mushroom/

[5] Seaborn Library
Seaborn is a Python data visualization library based on matplotlib. It provides a high-level interface for drawing attractive and informative statistical graphics.
https://seaborn.pydata.org/

[6] Paolo Frasconi - Kernels and Deep Networks for Structured Data, part 1
https://youtu.be/bqlJtefxVto

[7] Numba JIT compiler
Numba is an open source JIT compiler that translates a subset of Python and NumPy code into fast machine code.
http://numba.pydata.org/

[8] K-Fold Cross Validation Procedure
https://machinelearningmastery.com/k-fold-cross-validation/

[9] The linear model Perceptron from scikit-learn
https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.Perceptron.html

[10] Credits attributed to Jason Brownlee for the conversion of string features to integers
https://machinelearningmastery.com/implement-perceptron-algorithm-scratch-python/

[11] Credits attributed to Raffael Vogler for the inspiration.
https://www.joyofdata.de/blog/testing-linear-separability-linear-programming-r-glpk

[12] SciPy function to solve linear programming problems
https://docs.scipy.org/doc/scipy/reference/generated/scipy.optimize.linprog.html

[13] Approaches to test linear separability.
https://stats.stackexchange.com/questions/182329/how-to-know-whether-the-data-is-linearly-separable

[14] Numpy, heavily used in the entire study
https://numpy.org/