MSc in Computer Science
at University of Milan

Statistical Methods for Machine Learning
Kernelized Linear Predictor

course held by **Nicoló Cesa-Bianchi**

Email:
davide.cologni@studenti.unimi.it

Created by:
Davide Cologni
mat. 09732A

Academic year of 2023/2024

Contents

1 introduction

2 Dataset analysis and preprocessing

2.1 Dataset description

The provided dataset contains 10000 points with 10 features named from $x1$ to $x10$ and a label column named y.
All the feature are floating point values and the dataset is well formed (in the sense that there are no missing values).
The label column contains values that are either -1 or $+1$.
I collect the major statistics from each feature in the dataset: mean, standard deviation, min and max
It's clear that the different features are not normalized and follow different probability distributions.

2.2 Feature Scaling

Because the dataset is already well formed and there are no missing values The first thing I do is scale the features.
This step should ensure that the values of the features are in a comparable range.
I tried two approaches: normalisation and standardisation.
Scaling sets each feature to have a mean of 0 and a standard deviation of 1. This is achieved by subtracting the feature mean from each value and dividing by its standard deviation:

$$x' = \frac{x - \mu}{\sigma}$$

(where μ is the mean and σ is the standard deviation).
In the case of **normalisation**, the features are rescaled in a fixed range between 0 and 1 in the following way:

$$x' = \frac{x - x_min}{x_max - x_min}$$

.
It's important to note that we should perform both approaches using a sound procedure, more specifically, we should avoid data leakage: When calculating the minimum and maximum of the feature, we should only consider the training set and scale the test set only according to these values, without deriving any information from it.
In the same way, we should calculate the mean and standard deviation for the standardisation process.
Both of these approaches lead to important improvements and can be demonstrated both from an empirical point of view (see table TODO) and from a theoretical perspective.

2.3 Outliers removal

another approach I tried is removing the outliers from the dataset using the Z-score methods. I found 265 outliers (recall that the dataset has size 10000).
I calculated the score $Z = (x - \mu)/\sigma$ for each value where μ is the mean and σ is the variance of the feature. I then removed all the points with a Z-score greater or equals than 3 in absolute value.
I tried training non-kernelized Perceptron and Pegasos over the modified dataset but the results shows that the dataset is already sufficently cleaned: in fact it affects the performance of the models in a minimum way with no significative changes, and even in some cases it is (even if only slightly) worsening

comparison data table

2.4 Feature correlation

I sorted all the point according to each axis and plot on the other axis all the other features to spot eventual correlation and I observed that the feature 2 and 5 have a linear correlation (with a negative coefficent, see 1) as the feature 5 and 9 (with positive coefficent, see 2).

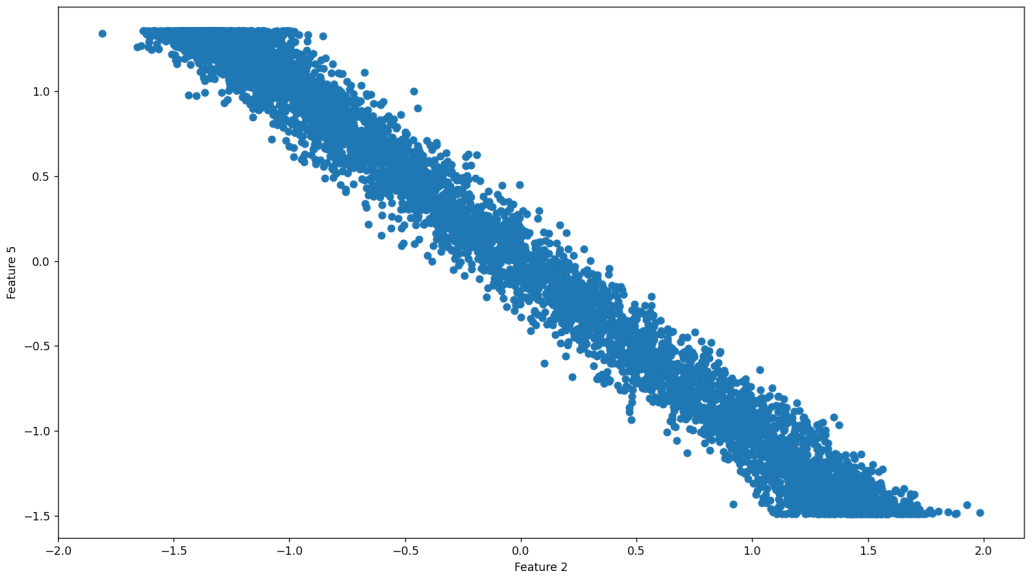

Figure 1: Here there is a clear linear correlation between feature 2 and 5, but very noisy

One possibility in this case during the preprocessing of the data is to remove the correlated features and leave only one of them to avoid redundancy of the data.
I don't follow this approach because there is a sensibile noise in the correlation and removing some features can lead to also removing this noise that can encode important information on the model.

2.5 Feature expansion

To being able to express non-homogeneous linear separators (hyperplane that don't pass through the origin) we add a constant feature of value 1 to each point in the dataset.
Let $(\mathbf{x})$ be any point in the dataset and $(\mathbf{w})$ be the linear separator, if we define $x' = (\mathbf{x}, 1)$ we can define $w' = (w, c)$, in that way:

$$w'^T x' = (\mathbf{w}^T \mathbf{x} + c)$$

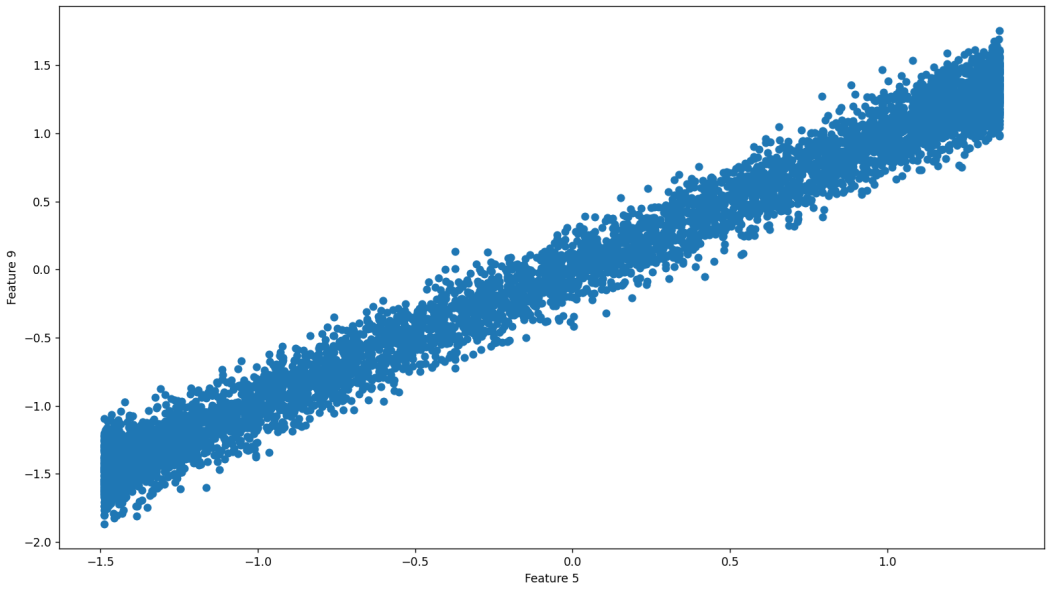

Figure 2: Another correlation is also present between feature 5 and 9 (with a negative coefficent)

3 Perceptron

4 Support Vector Machine

4.1 Naive

4.1.1 implementation details

I adapted this version of the algorithm from https://home.ttic.edu/ nati/Publications/PegasosMPB.pdf
The name of the variables are adapted to be consistent with the pseudo code reported
Between the code presented in the lecture and this one presented in the paper there are some differences with the pseudocode presented during the lectures:

- The gradient descent update is written in a slightly different way using a conditional statement instead of the classical indicator function, I choose to remain consistent also with this stilistic choice.

- Instead of return the average of all the weight vector calculated at each step, the paper returns only the last one.
  The authors indicates that they notate an improvment in performance returning the last vector instead of the average.
  I embrace also this variation.

- The author also provide a mini-batch version of the Pegasos algorithm, with another hyperparameter (the mini-batch size k).

Another approach proposed by the paper is *sampling without replacement*: so a random permutation of the training set is choosen and the updates are performed in order on the new sequence of data. In this way, in one epoch, a training point is sampled only once.
After each epoch we can choose if we restart to sample data sequentially according to the same permutation or create a new one and sampling according that new order.
Although the authors report that this approaches gives better results than uniform sampling as I did, I haven't experiment this variant of the algorithm.

4.1.2 Hyperparameter tuning

To choose the best hyperparameter for this algorithm (and the other implement) I choose to use the grid search method.
I divide the dataset into 3 different subset: training, test and validation set.
The validation set is used as a surrogate test set to obtain an estimate of the risk.
After splitting the data I choose a finite subset of the possible hyperparameter values $\Theta_0 \subseteq \Theta$ and for each of it create a predictor h_θ trained with the choosen hyperparameter on the training set.
The risk is then estimate using the validation error on each predictor, and the one with the lower risk is then choosen.

5 Kernel