Kernelized Pegasos - USPS Dataset

Sara Manfredi

Università degli Studi di Milano, Milan, Italy

Abstract. This project consists in developing from scratch an algorithm capable of predicting the digit contained in an image from USPS Dataset[1] using Pegasos with Gaussian Kernel. The project aims to emphasize how hyperparameters influence the results. The results show that the T parameter has a bigger influence on the result than λ.

Keywords: Kernelized Pegasos · USPS Dataset · Hyperparameters tuning.

I declare that this material, which I now submit for assessment, is entirely my own work and has not been taken from the work of others, save and to the extent that such work has been cited and acknowledged within the text of my work. I understand that plagiarism, collusion, and copying are grave and serious offences in the university and accept the penalties that would be imposed should I engage in plagiarism, collusion or copying. This assignment, or any part of it, has not been previously submitted by me or any other person for assessment on this or any other course of study.

1 Introduction

The prediction of the digit in an image is a multi-class classification problem where the classes are the 10 possible digits. Since the Pegasos Algorithm implements a binary classifier, we are going to implement 10 binary classifiers, one for each digit, and use them to predict the label of the multiclass classification problem.

The paper is structured as follows: in Section 2 we can find the theory and the pseudo-code used to implement the binary classifiers; Section 3 contains the method used to predict the multiclass classification label; Section 4 describes the dataset used; Section 5 illustrates the method used to the hyperparameters tuning; Section 6 shows the obtained results and Section 7 presents the Discussion about the results.

2 Binary Classification

In binary classification we want to find a classifier $h : \mathcal{X} \to \mathcal{Y}$, where $\mathcal{X} = \mathbb{R}^d$ is the feature space from which the instances are drawn and $\mathcal{Y} = \{-1, 1\}$ is the output label space, that minimizes some loss function.

2.1 The Pegasos Algorithm

Pegasos is the name of the algorithm whose goal is to resolve SVM (Support Vector Machine) minimization problem using SOGD (Stochastic Online Gradient Descent). SVM is an algorithm for learning linear classifiers,that, in the case of a non-linearly separable dataset, minimizes the following loss function

$$F(\mathbf{w}) = \frac{1}{m}\sum_{t=1}^{m} h_t(\mathbf{w}) + \frac{\lambda}{2}||\mathbf{w}||^2 \tag{1}$$

where:

- $h_t(\mathbf{w})$ is the hinge loss defined as $h_t(\mathbf{w}) = \left[1 - y_t \mathbf{w}^T \mathbf{x}_t\right]_+$ and has a positive value if the hyperplane $\mathbf{w}$ makes a mistake when the input is the example $(\mathbf{x}_t, y_t)$
- $\frac{1}{2}||\mathbf{w}||^2$ represents the maximum margin separating hyperplane
- λ is the regularization parameter

We can represent $F(\mathbf{w})$ as

$$F(\mathbf{w}) = \frac{1}{m}\sum_{t=1}^{m} \ell_t(\mathbf{w}) \tag{2}$$

where $\ell_t(\mathbf{w}) = h_t(\mathbf{w}) + \frac{\lambda}{2}||\mathbf{w}||^2$ is a λ-strongly convex function. This implies that $F(\mathbf{w})$ is a λ-strongly convex function and we can apply OGD algorithm for strongly convex function to minimize it.

The pseudo-code of the Pegasos Algorithm can be found in **Algorithm 1**.

Algorithm 1: Pegasos

Parameters : T, λ
Input : Training set $(x_1, y_1), ..., (x_m, y_m) \in \mathbb{R}^d \times \{-1, 1\}$
$w_1 = 0$;
for $t \leftarrow 1$ **to** T **do**
  Draw uniformly at random an element $(\mathbf{x}_{Z_t}, y_{Z_t})$ from the training set;
  Set $\mathbf{w}_{t+1} = \mathbf{w}_t - \eta_t \nabla \ell_{Z_t}(\mathbf{w}_t)$; `// update rule`
  ;
end
Output : $\bar{\mathbf{w}} = \frac{1}{T}(\mathbf{w}_1 + ... + \mathbf{w}_T)$

Note that: the analysis of OGD for strongly convex functions gives us a prefix step size $\eta_t = \frac{1}{\lambda t}$.

2.2 Kernel

Linear predictors, like SVM, may potentially suffer from a large approximation error because they are always described by a number of coefficients that can not be larger than the number of features: d. A common way to reduce this bias is feature expansion, which adds new parameters by constructing new features through a nonlinear combination of the base features. Formally, this can be viewed in terms of a function $\phi : \mathbb{R}^d \to \mathcal{H}$ often called map function. The problem with this expansion is that the number of features in the new data domain can become quickly exponential in the number of the base features[5].

To overcome this computational barrier we can use the kernel trick: that is, we have a function that is able to compute the inner product between two points in the expanded data domain without actually needing to calculate the representation of the points in the expanded feature space.

In order to be able to apply this trick, we need to have in our algorithm a point where we calculate the inner product between two points.

2.3 Kernelized Pegasos

As we said the kernel trick consists of computing the inner product between data points in the expanded space without actually computing the expansion of the data points. Since

$$\nabla \ell_t(\mathbf{w}) = \nabla(h_t(\mathbf{w}) + \frac{\lambda}{2}||\mathbf{w}||^2) = -y_t \mathbf{x}_t \mathbb{1}\left[y_t \mathbf{w}^T \mathbf{x}_t <= 1\right] + \lambda \mathbf{w} \tag{3}$$

The update rule in **Algorithm 1** becomes:

---

if $y_{Z_t} \boldsymbol{w}_t^T \boldsymbol{x}_{Z_t} <= 1$ **then**
  $\mathbf{w}_{t+1} = \left(1 - \frac{1}{t}\right) \mathbf{w}_t + \eta_t y_{Z_t} \mathbf{x}_{Z_t}$
else
  $\mathbf{w}_{t+1} = \left(1 - \frac{1}{t}\right) \mathbf{w}_t$
end

And since there exists **Theorem 3**[4] that said that the minimizer of $F(\mathbf{w})$, $\mathbf{w}^*$, can be written as

$$\mathbf{w}^* = \sum_{t=1}^{m} \alpha_t y_t \mathbf{x}_t \tag{4}$$

we can see that inside the if condition of the update rule now compares an inner product between data points because we can represent $\mathbf{w}$ as a linear combination of training points.

We decided to implement the Kernelized Pegasos as done here [2] to directly minimize the primal problem using kernels. Here, $\forall t$, $\alpha_{t+1} \in \mathbb{R}^{m\,1}$ is a vector

[1] m is the cardinality of the training set

such that $\alpha_{t+1}[j]$ counts how many times example j has been randomly selected so far and causes non-zero loss, a mistake:

$$\alpha_{t+1}[j] = |\{t' \leq t : i_{t'} = j \wedge y_j \mathbf{w}_{t'}^T \mathbf{x}_j < 1\}| \tag{5}$$

Instead of keeping in memory the weight vector $\mathbf{w}_{t+1}$, we'll represent $\mathbf{w}_{t+1}$ using α_{t+1} according to

$$\mathbf{w}_{t+1} = \frac{1}{\lambda t} \sum_{j=1}^{m} \alpha_{t+1}[j]\, y_i \phi(\mathbf{x}_j) \tag{6}$$

It is now easy to implement the Pegasos algorithm by maintaining the vector α. The calculi that bring to this formulation can be found here [3].

The pseudo-code of the Kernelized Pegaso is in **Algorithm 2**.

Note that even though the solution is represented in terms of the variables α, we are still calculating the sub-gradient with respect to the weight vector $\mathbf{w}$.

Algorithm 2: Kernelized Pegasos

Parameters : T, λ
Input : Training set $(x_1, y_1), ..., (x_m, y_m) \in \mathbb{R}^d \times \{-1, 1\}$
$\alpha_1 = 0$;
for $t \leftarrow 1$ **to** T **do**
  i_t = element drawn uniformly at random from the training set;
  forall $j \neq i_t$ **do**
    $\alpha_{t+1}[j] = \alpha_t[j]$;
  end
  if $y_{i_t} \frac{1}{\lambda t} \sum_j \alpha_t[j] y_j K(\boldsymbol{x}_{i_t}, \boldsymbol{x}_j) < 1$ **then**
    $\alpha_{t+1}[i_t] = \alpha_t[i_t] + 1$;
  else
    $\alpha_{t+1}[i_t] = \alpha_t[i_t]$;
  end
end
Output : $\alpha = \frac{1}{\lambda T} \alpha_{T+1}$

As stated in [2], even though the analysis of convergence of the Pegasos algorithm is based on the fact that the final hyperplane corresponds to the mean of all the other hyperplane computed in the T steps, that is on $\bar{\mathbf{w}} = \frac{1}{T} \sum_{t=1}^{T} \mathbf{w}_t$, in practice the final hyperplane $\mathbf{w}_{T+1}$ often provides better results and so we are going to predict using $\mathbf{w}_{T+1}$.

2.4 The Gaussian Kernel

The Gaussian Kernel is defined as

$$K_\gamma(\mathbf{x}, \mathbf{x}') = exp\left(-\frac{1}{2\gamma}||\mathbf{x} - \mathbf{x}'||^2\right) \tag{7}$$

It represents a linear combination of infinitely many polynomial kernels of increasing degree.

The Gaussian Kernel is universal in the sense that for each $\gamma > 0$ and for each continuous function $f : \mathbb{R}^d \to \mathbb{R}$ and for each $\epsilon > 0$, there exists a function g of the form

$$g = \sum_{i=1}^{N} \alpha_i K_\gamma(\mathbf{x}_i, \cdot) \tag{8}$$

for some $\mathbf{x}_1, ..., \mathbf{x}_N \in \chi$, $\alpha_1, ..., \alpha_N \in \mathbb{R}$ and $N \in \mathbb{N}$ such that g approximate f with error bounded by ϵ[5].

The value of γ determines the width of the gaussian centered in each point. This width represents the sphere of influence of each point in the binary classification prediction. The more γ is narrow, the few points will be part of the decision and so we'll tend to have overfitting; on the other hand, the more γ is large, the more points will be part of the decision and we'll tend to have a constant classifier.

3 The Multi-class classifier

As we said Pegasos is an algorithm for solving binary classification problems. In order to use it to solve a multiclass classification problem we are going to train 10 binary classifiers using Kernelized Pegasos and predict using the following formula:

$$\hat{y}(x) = \arg\max_i h_i(x) \tag{9}$$

where h_i is the prediction made by the classifier of the class i given x as input.

Note that for binary classification problem, the final prediction consists in $sgn(h_i(x))$ where

$$sgn(x) = \begin{cases} -1 & \text{if } x \leq 0 \\ 1 & \text{otherwise} \end{cases} \tag{10}$$

while there we don't take the $sgn()$ function over the prediction but we interpret the prediction as a sort of probability to be that class: greater is the value of $h_i(x)$ and greater is the probability that the multiclass label corresponds to that value. The loss used in this problem is the zero-one loss described as $\ell(\hat{y}, y) = \mathbb{1}\{\hat{y} \neq y\}$. Since the labels of the dataset are the labels of the multiclass classification problem, is needed to convert them to binary classification labels. In order to do that, for each class, a label will be set to 1 if it is equal to the class of the binary classification problem and -1 otherwise. In other words, $\mathcal{Y}$ passes form being $\{0, 1, 2, ..., 9\}$ to be $\{-1, 1\}$. For example, $h_0(x) = \sum_{i=1}^{m} \alpha_t y_i K(\mathbf{x}_i, \mathbf{x})$ represents the binary classifier of the class 0 where the label set $\mathcal{Y}$ has been trasformed to have a 1 if the label of the multiclass classification problem had a 0 and -1 otherwise.

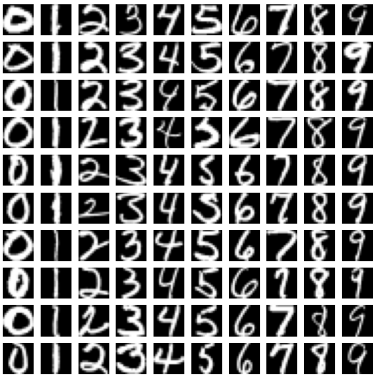

Fig. 1. Examples of digits in the dataset.

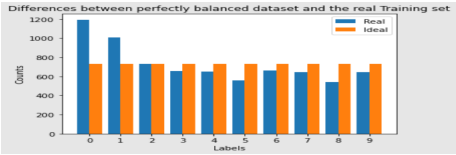

Fig. 2. Bar plot of the number of images for every label in the training set.

4 The USPS Dataset

The dataset consists of 9298 black-and-white images that represent handwritten digits. It is divided into training and test datasets with, respectively, 7291 and 2007 images. In Fig. 1) we can see some of the digits present in the dataset. The training set is overall balanced as we can see in Fig. 2). The orange bars represent how the training set should be divided in order to have a perfectly balanced dataset; that is the case in which every digit is contained in the same number of images. The real distribution of the images, represented by the blue bars, is not so distant from the ideal one. We have decided to consider the training set balanced. Every image is represented by 256 pixels and their value ranges between 0 and 1 where 0 represents black and 1 white. Since the dataset is already balanced and the pixels are already represented in the compact form (usually their value ranges between 0 and 255) there is no need to preprocess the data.

The USPS dataset can be freely downloaded here[1]. Here, $\mathcal{X} = [0,1]^{256}$ and $\mathcal{Y} = \{0, 1, 2, ..., 9\}$

5 Hyperparameter tuning

The hyperparameters of the Kernelized Pegasos are:

- γ that we decided to set to a commonly good value; $\gamma = 0.25$

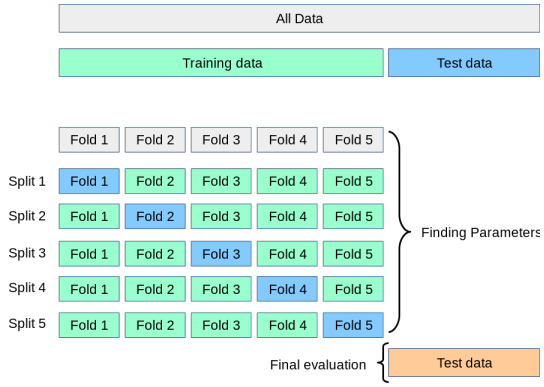

Fig. 3. Explanation of the 5-fold cross-validation.

- λ that is the regularization parameter between the search of the maximum margin hyperplane and the minimization of the mistakes
- T that is the number of steps computed

The γ parameter represents the width of the Gaussian centered in each data point; we can think of it as the sphere of influence that we decide to give to the training points in order to label the new data point.

In order to find the best values of λ and T we have followed the subsequent steps:

1. Put together the training and test data from the USPS Dataset.
2. Shuffle the entire dataset.
3. Divide the shuffled dataset in training and test using a rate of (0.8, 0.2) respectively.
4. Use the training set to compute 5-fold cross-validation in order to select the best hyperparameters.
5. Ritrained the model with the entire training set using the best hyperparameters found.
6. Test the model with the test dataset.

In Fig. 3) there is an explanation of the step computed in the 5-fold cross-validation.

6 Results

In Table 1 we can see the training accuracy and the test accuracy found by the various pairs of hyperparameters. The best pair found in the 5-fold CV is in bold and corresponds to $\lambda = 31.62$ and $T = 1000$.

We then trained the model with these parameters on the whole training data and test it with the test set. The test accuracy obtained was: 0.92

λ	T	**Training accuracy**	**Test accuracy**
0.001	10	0.43969	0.44775
0.001	100	0.78934	0.78104
0.001	500	0.90338	0.89079
0.001	1000	0.93346	0.92186
0.031	10	0.44080	0.44775
0.031	100	0.78457	0.78252
0.031	500	0.90253	0.90007
0.031	1000	0.93319	0.91836
1.0	10	0.45545	0.44264
1.0	100	0.77973	0.77579
1.0	500	0.90213	0.89482
1.0	1000	0.93459	0.91984
31.62	10	0.42570	0.42031
31.62	100	0.79905	0.80000
31.62	500	0.9009	0.89307
31.62	**1000**	**0.93252**	**0.92186**
1000.0	10	0.42342	0.43376
1000.0	100	0.78252	0.77552
1000.0	500	0.90509	0.90168
1000.0	1000	0.93554	0.92011

Table 1. Training and test accuracies for different values of hyperparameters λ and T.

7 Discussion

In Fig. 4) we can see the plots where we fixed a value of λ and we visualize the changes in the training and the test accuracy. In Fig. 5) we did the same but fixing T and seeing how the value of λ influences the accuracies. It seems that the T hyperparameter has a much bigger influence on the results than λ since the plots in Fig. 4) show almost the same results. In Fig. 4) we can also see that for small values of T we have underfitting, and that is something that we can expect since it means that we are letting a small number of training points to be part of the binary decision, and for big values of it (at least the biggest that our resources consent us to test) we are starting to see a difference between the training and the test accuracy even if we probably need a T much bigger in order to get overfitting. We think that the test accuracy could continue to upgrade before starts to decrease.

It could be interesting to see if and how the performances of the multiclass classification problem change if we tune the hyperparameters in each of the binary classifiers and then used them to predict the multiclass label. We think that could be possible that the binary classifiers reach the best prediction for different values of λ and T and that could also influence the prediction of the multiclass classification problem.

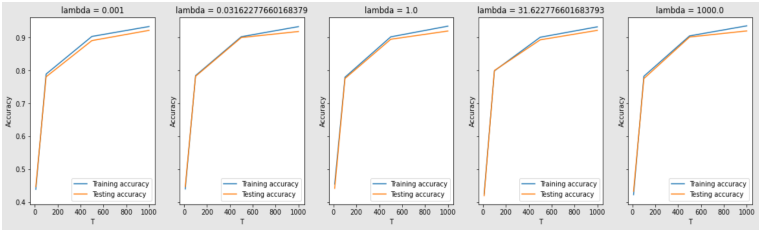

Fig. 4. Plot of training and test accuracy for fixed values of λ.

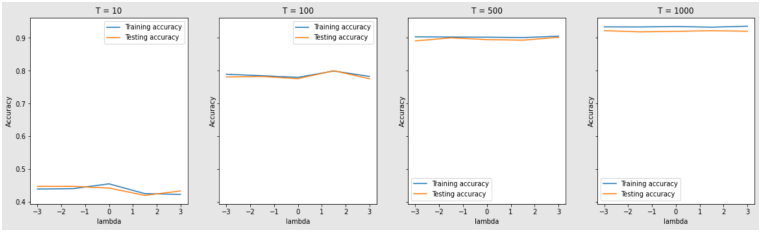

Fig. 5. Plot of training and test accuracy for fixed values of T.

References

1. USPS Dataset — Kaggle, https://www.kaggle.com/datasets/bistaumanga/usps-dataset. Last accessed 17 May 2023.
2. Shalev-Shwartz, s. and Singer, Y. and Srebro, N. and Cotter, A.: Pegasos: Primal Estimated sub-GrAdient SOlver for SVM.
3. Machine Learning Lecture 6 Note, https://people.csail.mit.edu/dsontag/courses/ml16/slides/lecture6_notes.pdf. Last accessed 12 May 2023.
4. Machine Learning-Statistical Methods for Machine Learning, Support Vector Machines, https://cesa-bianchi.di.unimi.it/MSA/Notes/svm.pdf. Last accessed 17 May 2023.
5. Machine Learning-Statistical Methods for Machine Learning, Kernel functions, https://cesa-bianchi.di.unimi.it/MSA/Notes/kernels.pdf. Last accessed 17 May 2023.