# *KERNELIZED PEGASOS - MULTICLASS CLASSIFICATION OF HANDWRITTEN DIGITS*

*Magdalena Beata Skowerska*

## Contents

## List of Figures

## Introduction

This project describes the implementation of Pegasos algorithm with Gaussian kernels, with the aim of training 10 binary classifiers for the multiclass classification of handwritten digits. The dataset is retrieved from the online community Kaggle (Kaggle, n.d.).

The first part describes the dataset, how the labels are transformed to do the binary classification and how the noise, time and space complexity of the algorithm are reduced by using a smaller number of records from the original dataset and the principal component analysis (PCA).

The second part describes the key concepts behind the implemented algorithm. There is an overview of the SVM algorithm, kernels, and one-vs-all encoding. Furthermore, there is also a brief explanation of the advantages and disadvantages using the Support vector machine (SVM) algorithm.

In the last part, there is an analysis of the cross-validated risk estimates for different values of T (the number of epochs) and λ (the regularization parameter) reported on a heatmap. In this analysis the parameter gamma of the Gaussian kernel is kept constant.

Another heatmap representing cross-validated risk estimates is plotted to illustrate the performance of the algorithm for different values of the regularization parameter and the hyperparameter gamma of the Gaussian kernel.

There is also an analysis of the impact on the Kernelized Pegasos performance by using different types of kernels: linear kernels, polynomial kernels, and Gaussian kernels.

At the end of this section it is demonstrated that the use of PCA with 30 components leads to a better complexity-performance trade-off compared to the algorithm used without PCA.

## The dataset

The dataset is composed by handwritten digits (Figure 1). It has 7291 train and 2007 test images. The images are 16*16 grayscale pixels rescaled in a 0-1 range that can be represented as vectors of 256 elements (Figure 2).

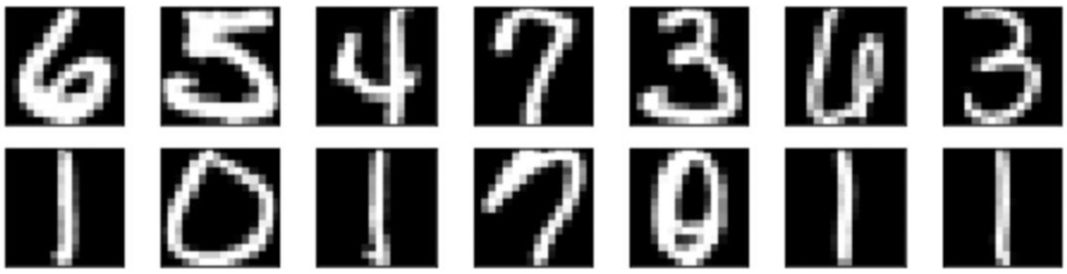
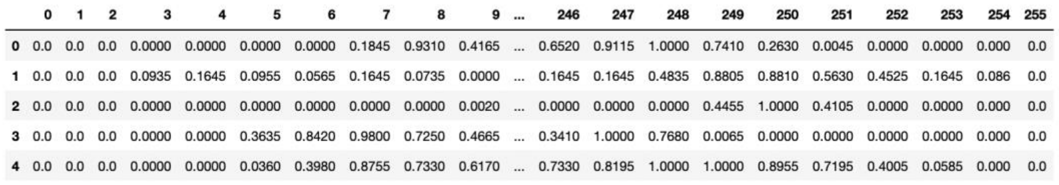


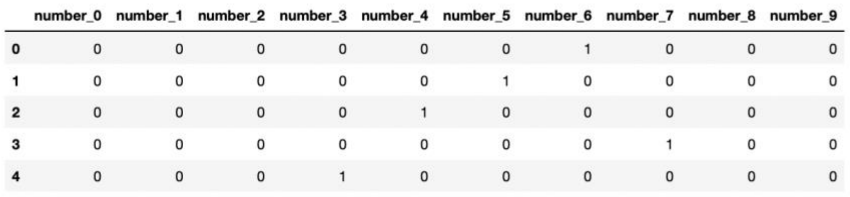
*Figure 1 - images sample from the dataset*



| | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | ... | 246 | 247 | 248 | 249 | 250 | 251 | 252 | 253 | 254 | 255 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **0** | 0.0 | 0.0 | 0.0 | 0.0000 | 0.0000 | 0.0000 | 0.0000 | 0.1845 | 0.9310 | 0.4165 | ... | 0.6520 | 0.9115 | 1.0000 | 0.7410 | 0.2630 | 0.0045 | 0.0000 | 0.0000 | 0.000 | 0.0 |
| **1** | 0.0 | 0.0 | 0.0 | 0.0935 | 0.1645 | 0.0955 | 0.0565 | 0.1645 | 0.0735 | 0.0000 | ... | 0.1645 | 0.1645 | 0.4835 | 0.8805 | 0.8810 | 0.5630 | 0.4525 | 0.1645 | 0.086 | 0.0 |
| **2** | 0.0 | 0.0 | 0.0 | 0.0000 | 0.0000 | 0.0000 | 0.0000 | 0.0000 | 0.0000 | 0.0020 | ... | 0.0000 | 0.0000 | 0.0000 | 0.4455 | 1.0000 | 0.4105 | 0.0000 | 0.0000 | 0.000 | 0.0 |
| **3** | 0.0 | 0.0 | 0.0 | 0.0000 | 0.0000 | 0.3635 | 0.8420 | 0.9800 | 0.7250 | 0.4665 | ... | 0.3410 | 1.0000 | 0.7680 | 0.0065 | 0.0000 | 0.0000 | 0.0000 | 0.0000 | 0.000 | 0.0 |
| **4** | 0.0 | 0.0 | 0.0 | 0.0000 | 0.0000 | 0.0360 | 0.3980 | 0.8755 | 0.7330 | 0.6170 | ... | 0.7330 | 0.8195 | 1.0000 | 1.0000 | 0.8955 | 0.7195 | 0.4005 | 0.0585 | 0.000 | 0.0 |

*Figure 2-sample dataset*

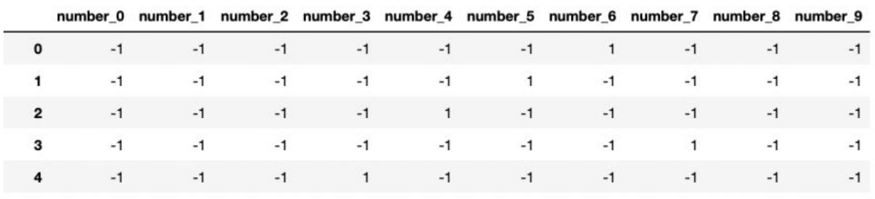
The labels can take the following values: 0,1,2,3,4,5,6,7,8,9. For multiclass classification using Pegasos we need to transform the column of categorical labels into ten different columns of binary values (Figure 3).



| | number_0 | number_1 | number_2 | number_3 | number_4 | number_5 | number_6 | number_7 | number_8 | number_9 |
|---|---|---|---|---|---|---|---|---|---|---|
| **0** | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 |
| **1** | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 |
| **2** | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 |
| **3** | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 |
| **4** | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 |

*Figure 3-One-Hot Encoding labels*

Since the model try to find whether the output is either +1 or -1, we need to convert the values of 0 to -1 (Figure 4).



| | number_0 | number_1 | number_2 | number_3 | number_4 | number_5 | number_6 | number_7 | number_8 | number_9 |
|---|---|---|---|---|---|---|---|---|---|---|
| **0** | -1 | -1 | -1 | -1 | -1 | -1 | 1 | -1 | -1 | -1 |
| **1** | -1 | -1 | -1 | -1 | -1 | 1 | -1 | -1 | -1 | -1 |
| **2** | -1 | -1 | -1 | -1 | 1 | -1 | -1 | -1 | -1 | -1 |
| **3** | -1 | -1 | -1 | -1 | -1 | -1 | -1 | 1 | -1 | -1 |
| **4** | -1 | -1 | -1 | 1 | -1 | -1 | -1 | -1 | -1 | -1 |

*Figure 4-Converting labels values from 0 to -1*

## Reducing time and space complexity

To reduce time and space complexity there will be used the Principal Component Analysis (PCA) algorithm and considered only 700 records. PCA reduces the dimensions of feature space and the noise in the data. At first, there are selected 200 orthogonal components, and the variance ratio is calculated. By summing up the variance ratio we can see that about 30 components are contributing for approximately 80% of the total variance. So, the reduction to 30 features will be chosen. Additionally, considering just the first 700 records of the dataset, already randomly distributed, will reduce the execution time.

## Key Concepts

In this chapter there will be explained the key concepts for the implementation of multiclass classification algorithm used in this project.

### Support Vector Machines

A support vector machine (SVM) is a supervised machine learning algorithm for learning linear models. It classifies data into two categories by finding the hyperplane with the largest margin between them.

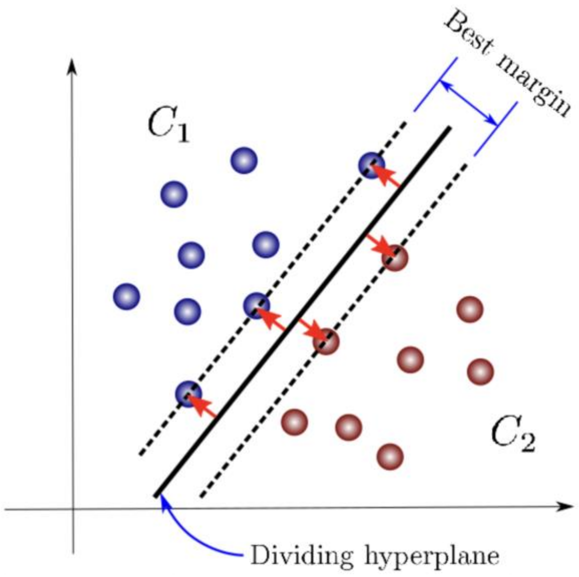


*Figure 5- graphical representation of SVM*

It can be used also in case of data not linearly separable by introducing soft margin which consists of allowing SVM to make a certain number of mistakes. In this case, the problem consists of minimizing the following objective:

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

*Figure 6- SVM objective*

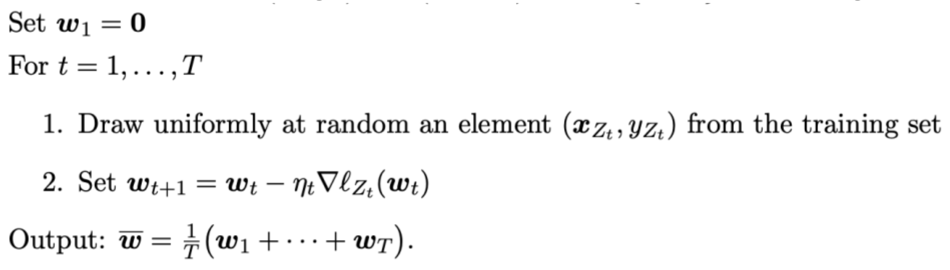
Where $h_t(\omega)$ is the hinge loss of $\omega$ and represents the values of slack variables $\xi_{t=}[1-y_t\omega^T x_t]_+$ for each t=1,….,m , with m number of training examples.

F represents a strongly convex function, and it can be minimized by using Online Gradient Descent (OGD) on which is based the Pegasos algorithm.

### Pegasos

Pegasos is an OGD algorithm applied in case of SVM. It consists in updating the predictor, at each new data point, moving in the negative direction of the gradient of the loss of the predictor calculated including the new data point.

Given the training set ($x_1$,$y_1$),...,($x_m$,$y_m$) ∈ $R^d$ x {-1,-1} , the number of T rounds and the regularization coefficient λ>0, the algorithm can be summarized as follows (Figure 7):



```
Set $\boldsymbol{w}_1 = \boldsymbol{0}$
For $t = 1, \ldots, T$
    1. Draw uniformly at random an element $(\boldsymbol{x}_{Z_t}, y_{Z_t})$ from the training set
    2. Set $\boldsymbol{w}_{t+1} = \boldsymbol{w}_t - \eta_t \nabla \ell_{Z_t}(\boldsymbol{w}_t)$
Output: $\overline{\boldsymbol{w}} = \frac{1}{T}(\boldsymbol{w}_1 + \cdots + \boldsymbol{w}_T)$.
```

*Figure 7-Pegasos algorithm*

Where the loss function is represented by the SVM objective.

## Kernelized Pegasos

In general, Kernel functions represents the similarity between two points, and they can be represented as a dot product. Most of the times in order to use linear classifier minimizing the approximation error it is necessary to map the data to a higher dimensional space. This mapping is given by the function ϕ: $R^d$→H and the linear classifier of the space H is a non-linear classifier in the space $R^d$. Therefore, the kernel function can be defined as follows:

$$k(\mathbf{x},\mathbf{z}) = \langle \phi(\mathbf{x}), \phi(\mathbf{z}) \rangle$$

Where x,z ∈ $R^d$ .
The kernel trick helps us finding an efficiently computable kernel function.

It is possible to solve the SVM problem in a reproducing kernel Hilbert space (RKHS) as an element of $H_K$ such that:

$$\mathcal{H}_K \equiv \left\{ \sum_{i=1}^{N} \alpha_i K(\boldsymbol{x}_i, \cdot) \,:\, \boldsymbol{x}_1, \ldots, \boldsymbol{x}_N \in \mathcal{X},\, \alpha_1, \ldots, \alpha_N \in \mathbb{R},\, N \in \mathbb{N} \right\}$$

The kernelized Pegasos generates predictors of the form:

$$g_{t+1} = \frac{1}{\lambda t} \sum_{r=1}^{t} f_r$$

Where $f_r = y_{s_r} K(x_{s_r}, \cdot) I\{h_{s_r}(g_r) > 0\}$ and $s_1, \ldots, s_t$ are the training set examples.

The above equation can be rewritten using the parameters $\alpha \in \Re^m$. Given that:

$$\alpha_{t+1}[j] = |\{t' \leq t : st' = j \wedge y_j g_{t'}(x_j) < 1\}|$$

We obtain:

$$g_{t+1} = \frac{1}{\lambda t} \sum_{j=1}^{m} \alpha_{t+1}[j] y_j K(x_j, \cdot)$$

Therefore, the training objective can be written as:

$$\min_{\alpha} \frac{\lambda}{2} \sum_{i,j=1}^{m} \alpha[i]\alpha[j]K(\mathbf{x}_i, \mathbf{x}_j) + \frac{1}{m} \sum_{i=1}^{m} \max\{0, 1 - y_i \sum_{j=1}^{m} \alpha[j]K(\mathbf{x}_i, \mathbf{x}_j)\}$$

Given the training set $(x_1,y_1),\ldots,(x_m,y_m) \in R^d \times \{-1,-1\}$ , the number of T rounds and the regularization coefficient $\lambda>0$, the Kernelized Pegasos Algorithm can be summarized in the following steps:

```
Set $\alpha_1 = 0$
For $t = 1,2,\ldots,T$
    Draw uniformly at random an element k from the set {0,…,m}
    Set $\alpha_{t+1}[j] = \alpha_t[j]$, for all $k \neq j$
    If $y_k \frac{1}{\lambda t} \sum_j \alpha_t[j] y_j K(x_k, x_j) < 1$, then:
        set $\alpha_{t+1}[k] = \alpha_t[k] + 1$
    else
        set $\alpha_{t+1}[k] = \alpha_t[k]$

Obtaining $\alpha_{T+1}$
```

For the multiclass classification there will be used the Gaussian kernel:

$$K_\gamma(\boldsymbol{x}, \boldsymbol{x}') = \exp\left(-\frac{1}{2\gamma} \left\|\boldsymbol{x} - \boldsymbol{x}'\right\|^2\right) \qquad \gamma > 0$$

The gamma parameter characterizes the reach of a given training data point. High values of gamma will determine a far reach, on the other hand low values will determine a close reach. With small values of gamma, the decision boundary is more flexed, and it causes the dependence on just the closest points of the one considered. Too small value of this parameter can lead to overfitting. This parameter can be used to control the bias variance trade-off.

## One-vs-all encoding

To implement the Kernelized Pegasos for multiclass classification of the dataset used in this project it is necessary to build 10 binary classifiers, one for each binary label. To predict the final label of a data point the technique used is one-vs-all encoding. It is based on two facts:

- If a data point belongs to a particular category, the classifier generated for the considered category predicts +1, while the other classifiers will predict -1.
- To determine if a data point belongs to a particular category, we compute the maximum value predicted by the ten classifiers.

## The advantages and disadvantages of using SVMs

Support vector machine is usually used for classification and regression problems. As any algorithm has its advantages and disadvantages.

Some of the advantages of SVMs are:

- The convex optimization method leads to a guaranteed optimality with a solution that represents a global minimum.
- It works well in cases where number of features is greater than the number of data points.
- It works well on smaller training datasets
- It works well when there is a clear margin of separation between classes

Some of the disadvantages are:

- It is less effective on datasets with more noise
- It is less effective on larger datasets. In these scenarios, the training time can be high and computationally intensive.

SVMs can be applied in different settings and for different purposes such as: intrusion detection, face detection, bioinformatics, handwriting remembrance, grouping of portrayals.

## The performance with respect to T and λ

To evaluate the performance of the Kernelized Pegasos algorithm in predicting the labels, the dataset was divided into train and test set using the cross-validation technique with 5 folds. This allows us to do a more accurate prediction.
By using annotate_heatmap() and heatmap() functions, taken from Matplotlib website (Matplotlib, s.d.), below we can see the heatmap with the risk estimates for different values of the number of rounds T and the regularization parameter (Figure 8), keeping the gaussian kernel parameter gamma set to 0.25.

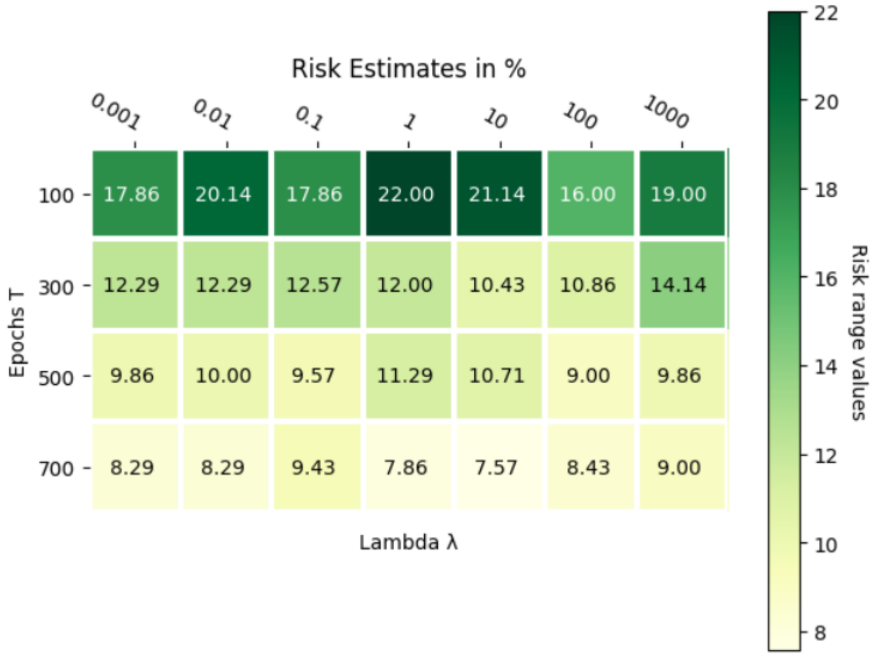


*Figure 8 – Risk Estimates for different values of T and λ*

To measure the risk estimates, there were used the zero-one loss function and the following values of T and lambda:

- T= [100,300,500,700]
- λ= [0.001,0.01,0.1,1,10,100,1000]

The regularization parameter allows misclassification and controls the trade-off between maximizing the margin and minimizing the loss. It helps in controlling the bias-variance trade-off. High values of λ give us a larger margin, avoiding overfitting, at the expense of a greater number of misclassifications. On the other hand, low values of λ give us smaller margin and less misclassified data.
In general, the optimized value of λ depends on how a particular dataset is distributed. However, in this experiment with gamma set to 0.25, there is no evidence of a particular pattern in the risk when changing the value of λ which effect is negligible.

The parameter T indicates the number of examples that are drawn from the uniformly distributed training set. We can see that for small values of T the estimated risk values are higher compared to those calculated when T is high. In fact, small values of T can lead to underfitting.

## The performance with respect to λ and γ

When using the Gaussian Kernel both λ and γ need to be optimized. If gamma is small, the effect of lambda becomes negligible, while if it is large, lambda affects the model just like how it affects a linear model. In the first scenario, since gamma determines the curvature in the decision boundary, low gamma leads to higher curvature (each point has less influence on classifying others) and by adding more complexity it can cause overfitting. In the second scenario, higher the lambda higher the number of the allowed misclassifications, and so higher the risk because of high bias and underfitting. To demonstrate this affirmation, I run Kernelized Pegasos for some values of gamma: 0.00025,0.025,2.5,2. The performance is measured for values of λ= [0.001,0.1,100,1000] and T set to 700 (Figure 9):

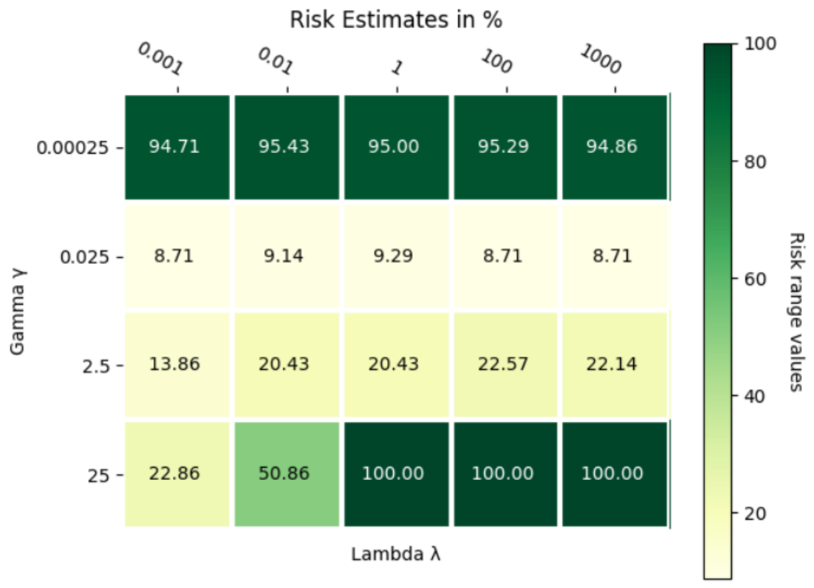


*Figure 9 - Risk Estimates for different values of λ and γ*

## Linear, polynomial, Gaussian kernels

In this project, I want to explore the performance of different kernel types. In particular, there are compared: linear kernel, polynomial kernel of degree 3 and Gaussian kernel. I used the value of T that gives the lowest risk estimate when using the Gaussian kernel in the previous section: T set to 700 and $\gamma$ set to 0.25. Below we can see the heatmaps with the results obtained:

For linear kernel (Figure 10):

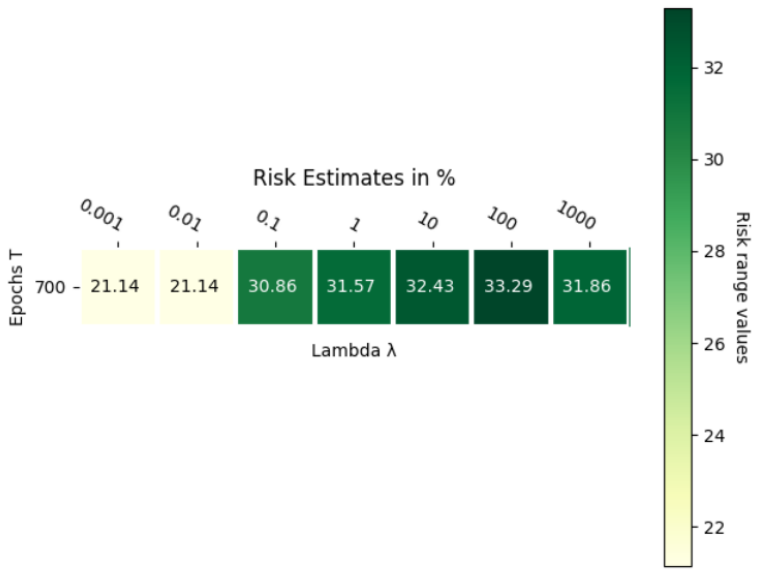


*Figure 10 - Risk Estimates using linear kernel*

For polynomial kernel of degree 3 (Figure 11):

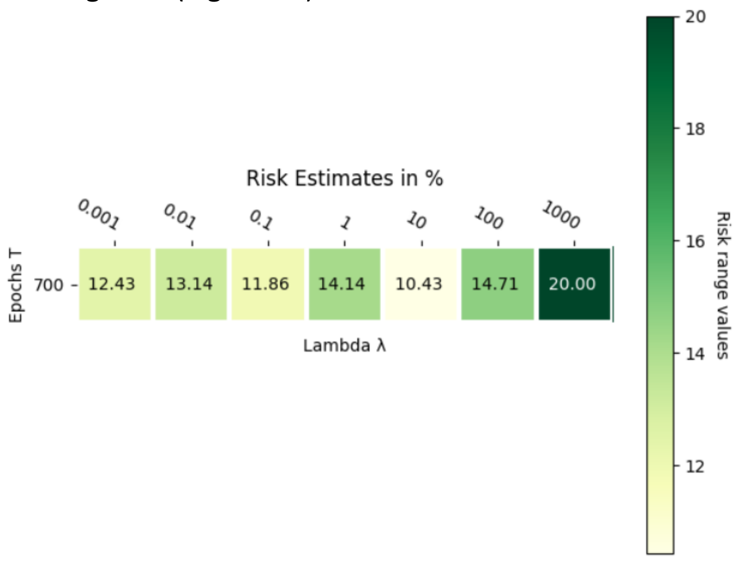


*Figure 11 - Risk Estimates using polynomial kernel*

The estimated risks using linear and polynomial kernels are higher than the risks calculated using Gaussian Kernel. In fact, Gaussian kernel can be seen as the sum of polynomial kernels $(x \times y + r)^{d}$ with r=0 and of degree going from 0 to infinity, and so, it defines a function space that is a lot larger, leading to a better performance.

## PCA performance for different numbers of components

In this section, it is demonstrated that by using PCA with 30 components the performance does not increase compared to those calculated without PCA (Figure 12) and when PCA is used with the number of components equal to 80 (Figure 13), as we can see from the heatmaps represented below:

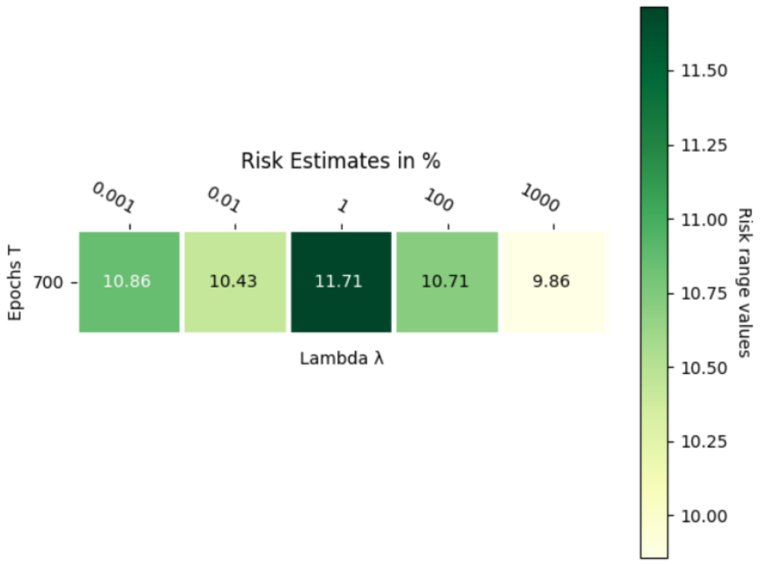


*Figure 12 - Risk Estimates without PCA*

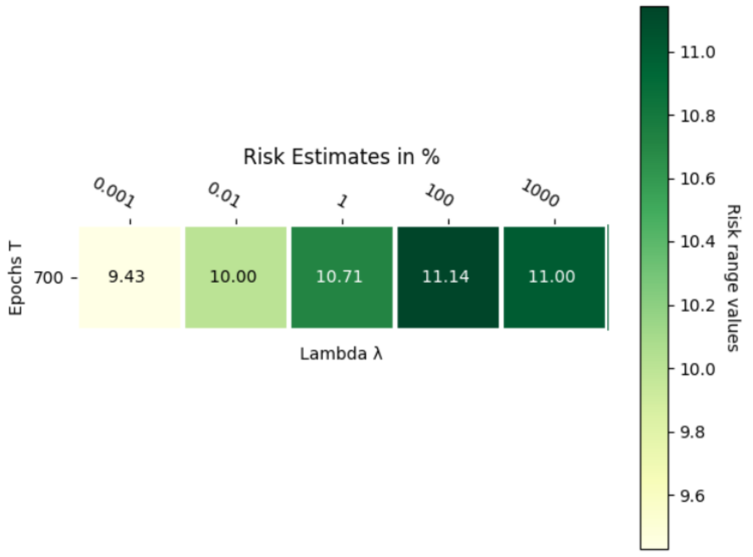


*Figure 13 - Risk Estimates applying PCA with 80 components*

For the same number of epochs and records from the original dataset the performance is even better when using PCA algorithm with 30 components.

## Conclusion

In conclusion, using the Kernelized Pegasos with PCA algorithm and considering a lower number of data points it is possible to reduce the time and space complexity. In the context of training 10 binary classifier for the multiclass classification of handwritten digits it has been shown that:

- Using the Gaussian kernels, keeping its parameter constant set to 0.25, and increasing T the number of epochs there is a decrease in the risk estimates.
- Using the Gaussian kernels and keeping its parameter constant set to 0.25, there is no evidence of a pattern in the performance of the algorithm when changing the value of $\lambda$.
- For small values of the Gaussian kernel parameter the estimated risks are high for any value of $\lambda$ which impact is negligible. On the other hand, for larger values of the hyperparameter gamma the impact of $\lambda$ can be compared to the impact it has when using linear kernels.
- Using the same amount of data and keeping the number of epochs T constant set to 700, the linear and polynomial kernels, lead to higher risk estimates than those calculated with Gaussian kernel.

## Bibliography

Kaggle. (n.d.). *Kaggle*. Retrieved from https://www.kaggle.com/datasets/bistaumanga/usps-dataset

Matplotlib. (s.d.). Tratto da https://matplotlib.org/stable/gallery/images_contours_and_fields/image_annotated_heatmap.html