# Multiclass Classification with Pegasos

Gabriele Cerizza

Università degli Studi di Milano
gabriele.cerizza@studenti.unimi.it
https://github.com/gabrielecerizza/smml

## Introduction

In this report we detail the results of our experiments on the Pegasos algorithm [1] over a multiclass classification task, pursuant to the project specifications set out for the Statistical Methods for Machine Learning course of the Università degli Studi di Milano[1].

In Section 1 we illustrate the dataset used in the experiments. In Section 2 we briefly describe the algorithm and its implementation. In Section 3 we show the results of our experiments and provide comments thereon. Finally, Section 4 contains our concluding remarks.

## 1 Dataset

Our analysis was carried out on the USPS dataset[2], comprising 9298 $16 \times 16$ grayscale images. These images depict handwritten digits ranging from 0 to 9. A sample of these digits can be seen in Figure 1. The classes do not present severe imbalance issues, as shown in Figure 2.

To gauge the complexity of multiclass classification on this dataset, we performed dimensionality reduction by way of t-distributed Stochastic Neighbor Embedding (t-SNE) [2] and projected the examples onto a two-dimensional space. The result is given in Figure 3. Within this figure the examples are clustered into ten clearly distinguishable groups, thus suggesting a low complexity classification task.

The USPS dataset is made available with predetermined training and test sets. In our experiments, however, we merged these sets and run a 5-fold cross-validation on the whole dataset, as illustrated in Section 3.

## 2 Pegasos Algorithm

In the present section we describe the Pegasos algorithm (Section 2.1), the kernel functions employed in our experiments (Section 2.2), and some implementation details to reduce the runtime of the algorithm (Section 2.3).

[1] https://cesa-bianchi.di.unimi.it/MSA/index_21-22.html
[2] https://www.kaggle.com/datasets/bistaumanga/usps-dataset

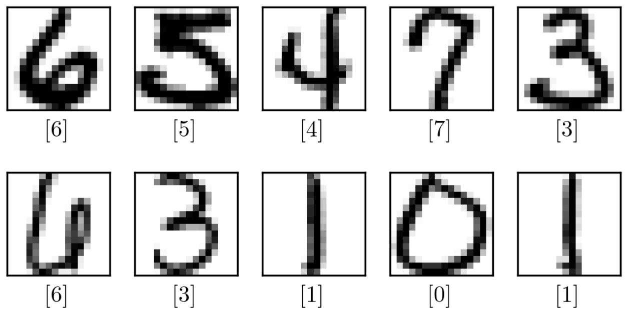

**Fig. 1.** Sample of images from the USPS dataset. The label is indicated below each image.

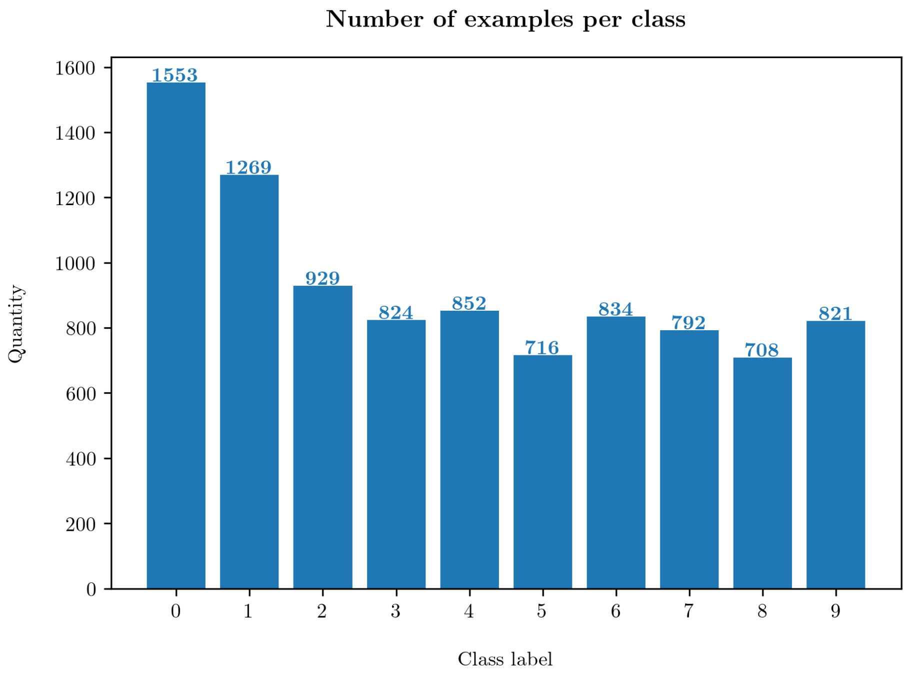

**Fig. 2.** Frequency of the classes in the USPS dataset.

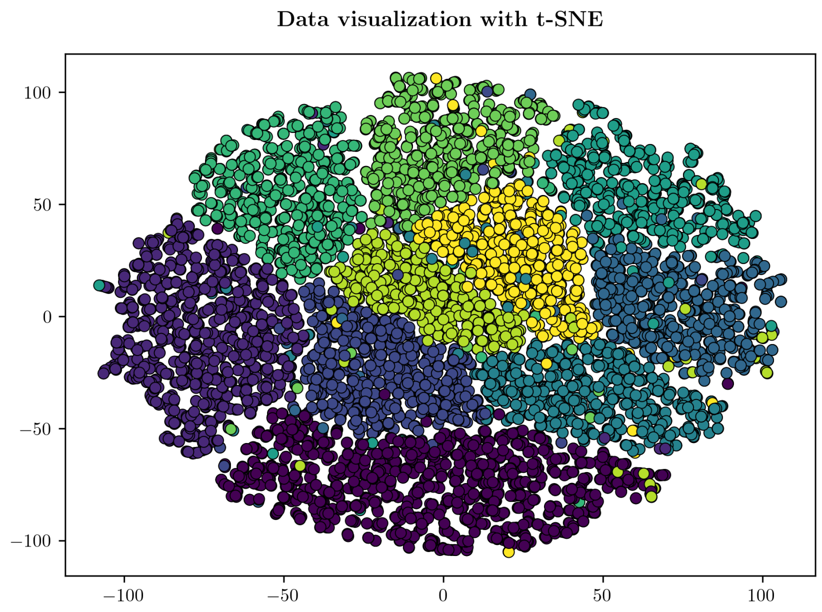

**Fig. 3.** Projection of the USPS dataset onto a two-dimensional space using t-SNE.

## 2.1 Description

The Pegasos algorithm [1] was introduced to solve the Support Vector Machine (SVM) [3] convex optimization problem for binary classification, which can be formalized as follows.

Let $S = \{(\boldsymbol{x}_1, y_1), \dots, (\boldsymbol{x}_m, y_m)\}$ be a training set with $\boldsymbol{x}_t \in \mathbb{R}^d$, $y_t \in \{-1, 1\}$ $\forall t = 1, \dots, m$; and let $\phi_K : \mathbb{R}^d \to \mathcal{H}_K$ be a function such that $\dim(\mathcal{H}_K) \gg d$ and $\langle \cdot, \cdot \rangle_K$ is the inner product in $\mathcal{H}_K$. The optimization problem aims to find the maximum margin separating hyperplane in $\mathcal{H}_K$, which separates the examples according to their labels and maximizes the distance between the hyperplane and the closest example. This yields the following quadratic programming problem:

$$
\begin{array}{lll}
\displaystyle\min_{g \in \mathcal{H}_K, \boldsymbol{\xi} \in \mathbb{R}^m} & \displaystyle\frac{\lambda}{2}\|g\|_K^2 + \frac{1}{m}\sum_{t=1}^{m} \xi_t \, , & \\
\text{subject to} & y_t \langle g, \phi_K(\boldsymbol{x}_t) \rangle_K \geq 1 - \xi_t & t = 1, \dots, m \, , \\
& \xi_t \geq 0 & t = 1, \dots, m \, ,
\end{array}
$$

where $\lambda$ is a regularization coefficient dictating the relative importance of the two terms in the objective function; and $\xi_t$ are slack variables that allow, but penalize, examples lying on the wrong side of the hyperplane.

This formulation can be turned into the following unconstrained optimization problem:

$$\min_{g \in \mathcal{H}_K} \quad \frac{\lambda}{2}\|g\|_K^2 + \frac{1}{m}\sum_{t=1}^{m} h_t(g)\,,$$

where $h_t(g) = [1 - y_t\langle g, \phi_K(\boldsymbol{x}_t)\rangle_K]_+$ is the hinge loss.

Pegasos solves this optimization problem using a stochastic gradient descent approach. The algorithm performs $T$ iterations, each of which consists in drawing a training example uniformly at random and updating $g$ through a gradient step. In particular, at iteration $t$ Pegasos computes

$$g_{t+1} = \frac{1}{\lambda t}\sum_{r=1}^{t} \mathbb{I}\{h_{s_r}(g_r) > 0\} y_{s_r} K(\boldsymbol{x}_{s_r}, \cdot)\,, \tag{1}$$

where $\mathbb{I}$ is the indicator function, $s_r$ is the index of the example randomly drawn at iteration $r$, and $K : \mathbb{R}^d \times \mathbb{R}^d \to \mathbb{R}$ is a kernel function such that $K(\boldsymbol{x}, \boldsymbol{x}') = \langle \phi_K(\boldsymbol{x}), \phi_K(\boldsymbol{x}')\rangle_K$.

Note that $g_{t+1}$ is a function, not a vector. Indeed, the dot in $K(\boldsymbol{x}_{s_r}, \cdot)$ stands for a missing argument, corresponding to the example on which the function is evaluated. Therefore, unlike vectors, $g_{t+1}$ cannot be stored in memory and used to perform inference on new examples, since we have to evaluate $g_{t+1}$ on them first.

In order to simplify the process, it was observed in [1] that (1) can be rewritten as

$$g_{t+1} = \frac{1}{\lambda t}\sum_{j=1}^{m} \boldsymbol{\alpha}_{t+1}[j] y_j K(\boldsymbol{x}_j, \cdot)\,,$$

where $\boldsymbol{\alpha}_{t+1} \in \mathbb{R}^m$ is a vector representing how many times each example $\boldsymbol{x}_j$ in the training set has been drawn and resulted in a non-zero loss, that is

$$\boldsymbol{\alpha}_{t+1}[j] = |\{r \le t : s_r = j \wedge y_j\langle g_r, \phi_K(\boldsymbol{x}_j)\rangle_K < 1\}|\,.$$

The vector $\boldsymbol{\alpha}_{t+1}$ can be kept in memory and used to quickly evaluate $g_{t+1}$ on new examples during both training and test. Algorithm 1 shows the pseudocode for the kernelized Pegasos algorithm using the abovementioned approach.

Once the vector $\boldsymbol{\alpha}_{T+1}$ has been obtained, the label of a new example $\boldsymbol{x}$ can be predicted by computing

$$\hat{y} = \operatorname{sgn}\left(\sum_{j=1}^{m} \boldsymbol{\alpha}_{T+1}[j] y_j K(\boldsymbol{x}_j, \boldsymbol{x})\right)\,. \tag{2}$$

It is not necessary to multiply the sum by $\frac{1}{\lambda T}$, since scaling does not affect the sign.

So far we illustrated the Pegasos algorithm for binary classification. In order to handle multiclass classification, we can adopt the one-vs-all multiclass strategy. We prepare a different binary classifier for each class. These binary

**Algorithm 1:** Pegasos algorithm in a kernel space

```
**Hyperparameters:** $T$, $\lambda > 0$
**Data:** Training set $S = \{(\boldsymbol{x}_1, y_1), \ldots, (\boldsymbol{x}_m, y_m)\}$
$\boldsymbol{\alpha}_1 \leftarrow (0, \ldots, 0)$
for $t = 1$ to $T$ do
    Choose $s_r \in \{0, \ldots, |S|\}$ uniformly at random
    if $y_{s_r} \frac{1}{\lambda t} \sum_{j=1}^{m} \boldsymbol{\alpha}_{t+1}[j] y_j K(\boldsymbol{x}_j, \boldsymbol{x}_{s_r}) < 1$ then
        $\boldsymbol{\alpha}_{t+1}[s_r] = \boldsymbol{\alpha}_t[s_r] + 1$
    end
end
return $\boldsymbol{\alpha}_{T+1}$
```

classifiers are trained using labels encoded as 1 for the examples belonging to the class of the classifier, and encoded as $-1$ for the examples belonging to all the other classes.

Finally, to determine the class of a new example $\boldsymbol{x}$, we first compute

$$h_i = \sum_{j=1}^{m} \boldsymbol{\alpha}_{T+1}[j] y_j K(\boldsymbol{x}_j, \boldsymbol{x}) \tag{3}$$

for each classifier $c_i$, and then predict with

$$\hat{y} = \arg\max_i h_i \, .$$

Also in this case scaling $h_i$ is unnecessary, given that $\frac{1}{\lambda T}$ is always positive and therefore does not change the order of the $h_i$ values.

### 2.2 Kernel functions

In our experiments with the Pegasos algorithm, we employed two kernel functions: the Gaussian kernel and the polynomial kernel. The Gaussian kernel is defined as

$$K(\boldsymbol{x}, \boldsymbol{x}') = \exp\left(-\frac{1}{2\gamma} \|\boldsymbol{x} - \boldsymbol{x}'\|^2\right) , \tag{4}$$

with $\gamma > 0$. The polynomial kernel is defined as

$$K(\boldsymbol{x}, \boldsymbol{x}') = (1 + \boldsymbol{x}^\top \boldsymbol{x}')^n , \tag{5}$$

where $n \in \mathbb{N}$ is the degree of the polynomial.

### 2.3 Implementation

In this section we expound on two implementation details that improve the runtime of the algorithm.

**Kernel matrix.** In order to perform multiclass classification, we need to train multiple classifiers, each slated to run the Pegasos algorithm over the same training set. Within this framework, the kernel function between two given examples may be computed multiple times, resulting in redundant computations.

Moreover, at each iteration the Pegasos algorithm computes the kernel function between a randomly drawn training example and all the other training examples. Drawing the same example multiple times results in further redundant computations.

Redundant computations adversely affect the runtime of the algorithm and should therefore be avoided. To this effect, we opted to compute ahead of time the kernel matrix $K$, such that

$$K_{i,j} = K(\boldsymbol{x}_i, \boldsymbol{x}_j) \quad \forall i, j = 1, \ldots, m\,.$$

This matrix was then dispatched to each classifier and used in place of the kernel function evaluation.

The drawback of the kernel matrix lies in its memory footprint. However, given the number of examples in the USPS dataset, such a matrix would require at most $\frac{9298 \times 9298 \times 32}{2^3 \times 10^9} \approx 0.34$ gigabytes of memory, which in modern machines constitutes a negligible amount.

**Vectorization.** The summations within the Pegasos algorithm can be vectorized, thus allowing to exploit the highly optimized array operations made available by math libraries.

In particular, the condition of the `if` statement within Algorithm 1 can be rewritten as

$$y_{s_r} \frac{1}{\lambda t} (\boldsymbol{\alpha}_t \odot y)^\top K_{s_r} < 1\,,$$

where $\odot$ is the Hadamard product and $K$ is the kernel matrix between the training examples. Likewise, the formula for $h_i$ in (3) can be rewritten as

$$\boldsymbol{h}_i = (\boldsymbol{\alpha}_{T+1} \odot y)^\top K'\,,$$

where $K'$ is the kernel matrix between the training examples and the test examples and $\boldsymbol{h}_i$ is the vector containing the predictions for all test examples returned by classifier $c_i$.

Also the computation of the kernel matrices can be vectorized. In the case of the polynomial kernel, the vectorization is trivial, since the dot product between vectors in (5) directly translates into a matrix product. In the case of the Gaussian kernel, however, an analogous approach would fail, because subtracting the training set matrix from itself in (4) would result in a null matrix. We therefore follow another approach.

Let $S \in \mathbb{R}^{n \times m}$ be a training set. In order to vectorize the kernel matrix with respect to the Gaussian kernel, we need to compute the difference between each row in $S$ and all the rows in $S$. To accomplish this, we first turn $S$ into a tensor $T \in \mathbb{R}^{n \times 1 \times m}$, in which each row $T_i$ is a $1 \times m$ matrix containing only the corresponding row $S_i$ of the original matrix. Then, we simply compute

$$R = T - S\,,$$

where $R \in \mathbb{R}^{n \times n \times m}$ is a tensor having $R_i = S_i - S\ \forall i = 1, \ldots, n$, in which $S_i$ is casted into a $\mathbb{R}^{n \times m}$ matrix using broadcasting rules[3]. Once $R$ is obtained, the Gaussian kernel formula in 4 can be applied straightforwardly. The same procedure can be applied when computing the kernel matrix between the training and test examples.

One caveat of this approach is that $R$ has a much larger memory footprint than the $n \times n$ kernel matrix. At most, the required memory would be $0.34 \times 256 \approx 87$ gigabytes. For this reason, it is necessary to split $S$ into chunks, apply the procedure on each chunk and then concatenate the results.

**Comparison.** The training runtime reduction obtained by applying the abovementioned optimizations on a single binary classifier is given in Figure 4. The shown improvement becomes even more promiment when accounting the fact that, for multiclass classification, multiple classifiers need to be trained.

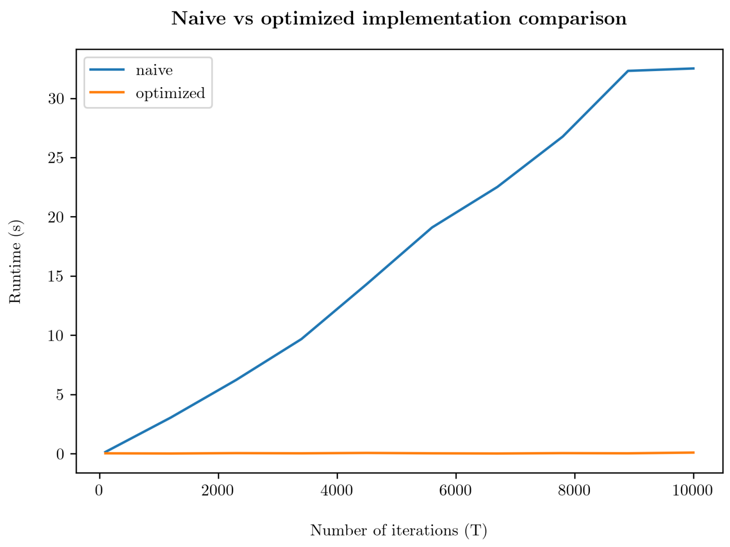

**Fig. 4.** Runtime improvement when training a single binary classifier with the optimized Pegasos algorithm. The comparison was carried out using a custom training set with 1000 examples and a Gaussian kernel.

[3] https://numpy.org/doc/stable/user/basics.broadcasting.html

## 3 Experiments

In this section we show the results of our experiments on the multiclass classification task using the Pegasos algorithm. The experiments were run on a machine with 16 gigabytes of RAM and a CPU Intel(R) Core(TM) i7-9700K 3.60GHz with 8 cores.

**Cross-validation.** The algorithm performance was evaluated with a 5-fold cross-validation on the entire USPS dataset. The folds were stratified so as to retain the proportion of the classes found in the complete dataset.

**Hyperparameters.** We investigated the behavior of the algorithm for different choices of the hyperparameters $T$ and $\lambda$, respectively representing the number of iterations and the regularization coefficient.

Concerning $T$, we gauged the performance of the algorithm over an increasing number of iterations, starting from a number much smaller than the cardinality of the training set and ending with a number several times bigger than the cardinality of the training set. More precisely, we chose the following values: 1000, 5000, 10000, 25000, 50000.

Concerning $\lambda$, in [1] the Pegasos algorithm was evaluated on the USPS dataset using $\lambda$ equal to $1.36 \times 10^{-4}$. It was also observed therein how, given a fixed number of iterations, the performance of the algorithm was inversely proportional to the value of $\lambda$, where $\lambda$ ranged from 1e-7 to 1e-3. We followed this insight and took the analysis one step further, experimenting with even higher values of $\lambda$, ranging from 1e-5 to 10.

We ran the algorithm with the polynomial and Gaussian kernel. The polynomial kernel was experimented upon with $n$ equal to 2, 3, 4 and 7, in order to assess the effectiveness of the algorithm at both low and high degrees of the polynomial. The Gaussian kernel was run with $\gamma$ equal to 0.25, as suggested in the project requirements, although we note that in [1] the authors used $\gamma$ equal to 2 on the USPS dataset.

**Results.** The cross-validated test error (zero-one loss) of the Pegasos algorithm over the USPS dataset with respect to the different training settings is shown in Figures 5, 6.

Considering the settings employing a polynomial kernel, we observe

zero-one loss

Namely, we hose values that emphasized both the capacity of the algorithm to quickly the runtime overhead of the naive implementation This is exacerbated Remove tags for equations unless needed Under scrutiny PCA?, Standard Scaler? Fare un unico plot con i tempi regenerate frequency classes img check again runtime, that 0 is strange

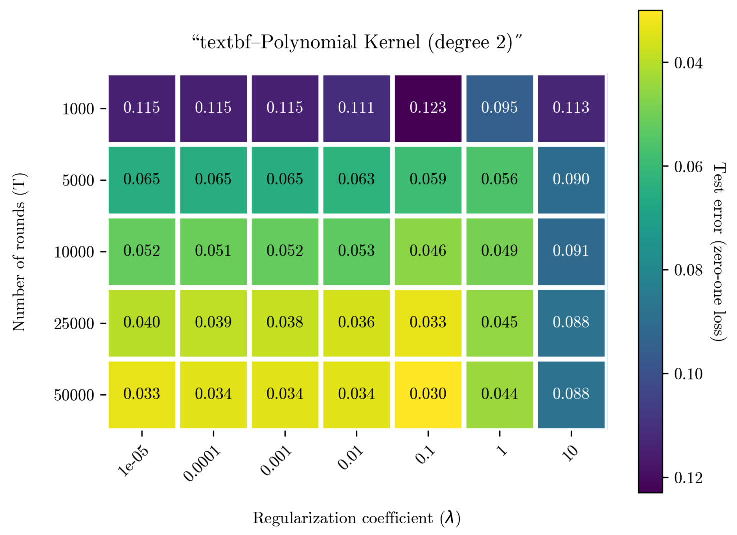

**Fig. 5.** Cross-validated test error (zero-one loss) of the Pegasos algorithm for multiclass classification over the USPS dataset, employing a polynomial kernel of degree 2.

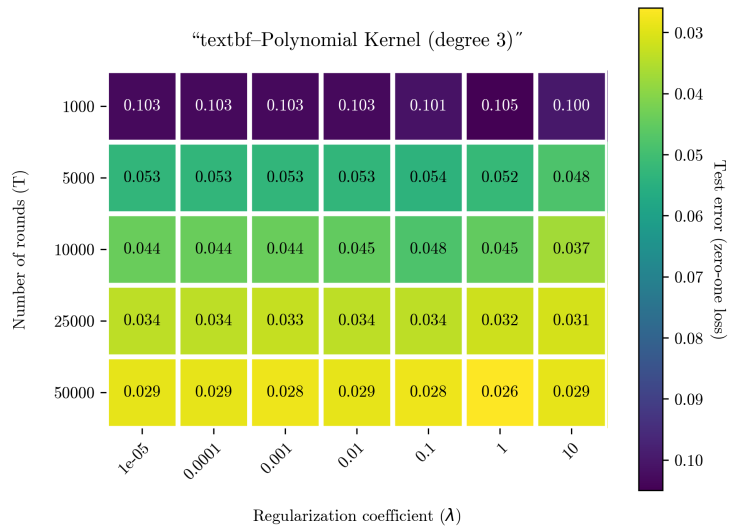

**Fig. 6.** Cross-validated test error (zero-one loss) of the Pegasos algorithm for multiclass classification over the USPS dataset, employing a polynomial kernel of degree 3.

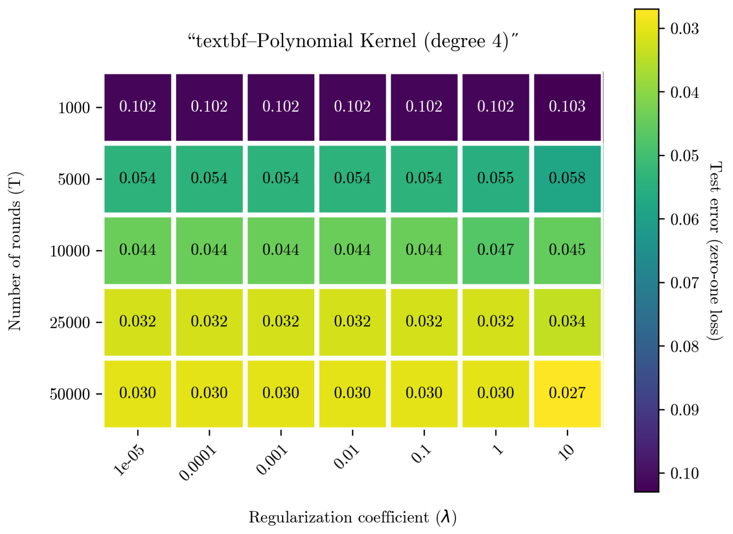

**Fig. 7.** Cross-validated test error (zero-one loss) of the Pegasos algorithm for multiclass classification over the USPS dataset, employing a polynomial kernel of degree 4.

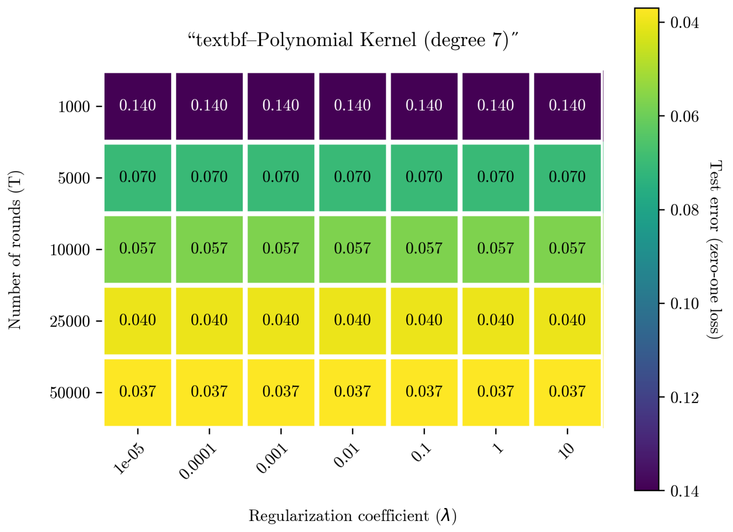

**Fig. 8.** Cross-validated test error (zero-one loss) of the Pegasos algorithm for multiclass classification over the USPS dataset, employing a polynomial kernel of degree 7.

## 4 Conclusions

## Statement

*I/We declare that this material, which I/We now submit for assessment, is entirely my/our 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/our work. I/We 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/us or any other person for assessment on this or any other course of study.*

## References

1. Shalev-Shwartz, S., Srebro, N., and Cotter, A. Pegasos: Primal estimated sub-gradient solver for svm. *Math. Program. 127* (03 2011), 3–30.
2. van der Maaten, L., and Hinton, G. Visualizing data using t-sne. *Journal of Machine Learning Research 9* (11 2008), 2579–2605.
3. Vapnik, V. N. *The Nature of Statistical Learning Theory*, second ed. Springer, November 1999.