Recognising Hand-Written Digits

Machine Learning

Name : Lo Bue Oddo Giulia

Link to the repository

https://github.com/zulai98/ML-project

Executive Summary

This project wishes to assess the suitability of the Pegasos learning algorithm on the USPS dataset, a dataset composed of 10 types of hand-written digits: from number 0 to number 9. We wish to observe the optimal hyperparameters (maximum number of iterations T (referred to as ite in this project) and regularizing parameter lambda) change when implementing a different kernel (e.g., Gaussian vs polynomial vs linear) and understand which kernel is the most suitable when applied to the complete dataset. We first provide a brief overview of the Pegasos algorithm and the use of kernels before delving into the practical part which comprises of the following sections:

- Perform some pre-processing and data visualization.

This section will deal with normalizing/standardizing the dataset if necessary, removing null values and assessing whether the dataset would benefit from PCA. As we are dealing with multi-class classification, we shall use a one-to-all encoding, building 10 different subclasses composed of positive 1's or negative 1's (positive if the number is equal to the subclasses' label and negative otherwise).

- Perform some exploratory data analysis

In this section we wish to visualize the resulting predictions of the Pegasos algorithm, to see whether we can come up with any other interesting observations. For computational reasons, we choose to apply the Pegasos algorithm to the first digit of the dataset, 0. We apply the (Gaussian kernel) Pegasos algorithm with standard hyperparameters on the training dataset, evaluate metrics on the test dataset. We then visualize the predictions to detect which numbers are mistaken for 0 and note some of our observations down with regards to the choice of the metrics and the algorithm. Finally we use 5-fold cross-validation to select the best hyperparameters (this will later be applied to the whole dataset) and assess the test error by training the "best model" on the whole training set and evaluating it on the test dataset.

- Assess Pegasos on the whole dataset

In this section we wish to find the best hyperparameters for the Pegasos algorithm and assess which kernel is the best in finding the separating hyperplane on the dataset. We use three different kernels: the Gaussian one with gamma=0.25, a fourth order polynomial and a linear kernel. Similarly to the previous section, after selecting the kernel to use, we take the following steps:

- Use k-fold cross validation to select the model thus to find the best hyperparameters: number of iterations and regularizing parameter for the Pegasos algorithm.
- Fit the "best model" to the complete dataset and use 5-fold cross-validation to evaluate the model's performance.

After comparing the performance of the model (with different kernels), we try to explain the reasons behind the choice of the best kernel.

In the last section we describe further works to be performed including refinements to our work, to ensure the robustness of our findings, and further ideas for data-pre-processing to improve the algorithm's performance.

Theoretical Interlude

SVMs: Pegasos

Pegasos is a version of the Support Vector Machine classifier which stands for Primal Estimated sub-GrAdient SOlver. In its primal form, the SVM is an unconstrained minimization problem of empirical risk plus a regularization term. To find the optimal weights which minimize the objective function, Pegasos performs stochastic gradient descent on this objective function, with a learning rate (equal to $\frac{1}{\lambda(t+1)}$) progressively decreasing to guarantee convergence.

Taking a step back the goal of a binary SVM classifier is to find the best separating hyperplane between the two classes (the one labelled as positive and the one labelled as negative). Theoretically the best hyperplane is the maximum-margin one, which maximizes the distance between the two classes (such that the distance between the hyperplane and the nearest point from either group is maximised). For datasets which are non-linearly separable, we adapt the objective function by allowing for some incorrect classifications (the margin is referred to "soft margin").

Looking into Pegasos, we notice that it iteratively employs a Stochastic sub-Gradient Solver to find the weights which minimize the objective function.

Indeed, by taking the hinge loss as the empirical risk function (thus a convex and non- function) implies that we must use (sub)-gradients instead of gradients. The stochastic part instead refers to the fact that in performing the gradient descent method, the algorithm considers one sample at a time (instead of the whole dataset). Doing so greatly reduces the computational time, an aspect which is critical for large datasets such as the USPS one. Moreover, by considering one sample at a time Pegasos is also useful as an "online learning" algorithm, as the weights are re-adjusted "on the fly" as new data is added.

Kernels:

Moving onto kernels, we note that kernels allow mapping the original dataset to higher dimensional space. A suitable kernel (for the dataset's function class) will therefore map the original dataset to a higher dimensional space where the dataset becomes linearly separable, mapping the non-linear decision surface in the original space to a linear one in the transformed space. We shall explore the use of three types of kernels: polynomial, gaussian and linear. By accounting for the interdependencies between features, polynomial kernels are quite popular in image processing and Gaussian ones are applied to a wider range of datasets (ie. networks, sparse-text document matrices and images). The linear kernel is the simplest one and generally the least time-consuming one (having no hyperparameter to tune), although may perform poorly when dealing with a dataset which is not linearly separable.

The algorithm:

The Pegasos algorithm's steps can be summarized as follows:
For each iteration ite:
- Choose a random training example
- Replace the objective function with an approximation based on the training example (This computes if the current inclination of the hyperplane in the dimension of the sample feature correctly predicts the label.)
- Update the weight, alpha, by subtracting the product of the step size and the sub gradient of the objective function from the previous weight.

Pre-processing and Data Visualisation

We notice that there are 9298 samples (of digits) and 256 columns, and that the dataset does not contain any null or N/A values. Taking each column as the nth pixel, each picture is composed of a 16 by 16 grid of pixels (see Fig.1) with the pixels' value, representing the intensity, ranging from 0-1. This suggests that the data has already been normalized, seeing as, generally, grayscale images are stored in a matrix of numbers where the pixels' intensity ranges from 0 (black) to 255 (white), with values in between making up the different shades of grey.

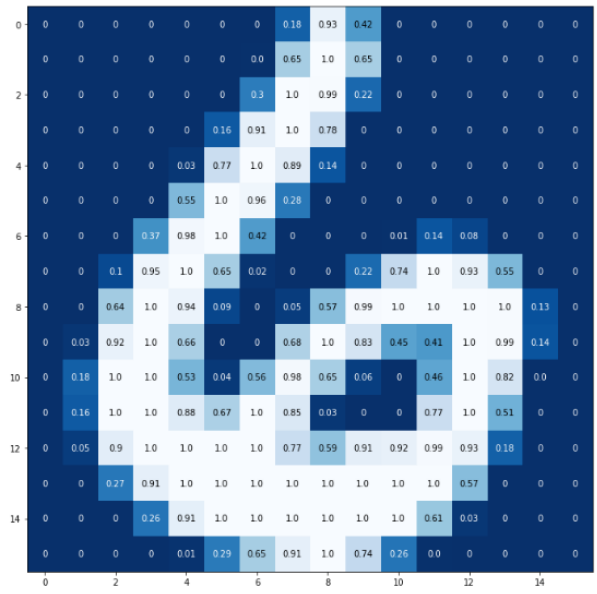

Figure 1 – Visualizing a sample

Referring to Fig.2 and Fig.3, we notice that the hand-written digits range from 0 to 9., and we have a greater representation of the digits 0 and 1, with 5 and 8 being the least represented. As we choose a one-vs-all encoding, after hot-encoding the labels, we notice that the +1 and -1s are not evenly distributed: indeed the -1s are overrepresented in all classes. This leads us to suggest that the F1 metric should be used when assessing the performance of the algorithm, as we could end up with a very good accuracy metric but a lower precision.

Taking 10 random samples of each picture (see Fig. 4) and combining our previous observations we note that:

- Digits from 2 to 9 have considerable variance (i.e., the digits sometimes occupy different positions and are not all equally "sharp", e.g. digit 2 or digit 5)
- Digit 1 should be fairly easy to estimate as it generally consists of a line through the middle of the grid and should not be easily confused with other numbers. Similarly digit 6 shouldn't be severely misclassified, although we do have a lesser number of samples. Lastly, a similar logic applies to digit 0. Although it could be confused with digit 6 or digit 9, we have a fair number of samples, and number 0 clearly remains relatively distinguishable when compared to other digits.
- Digits 4, 5,9 instead could be easily misclassified (as they are not optimally represented) and a 4 could be interpreted as a 9 or a 5 could be interpreted as a 7.

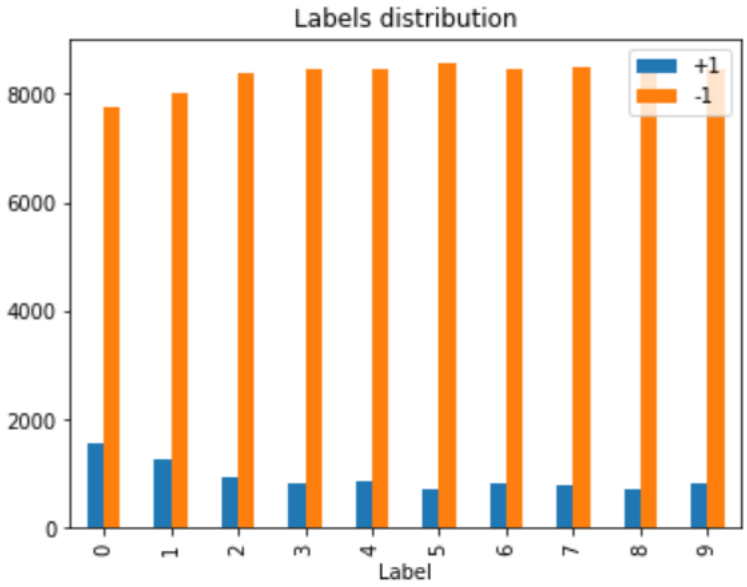

Figure 2 – Digits' distribution in the dataset

	Label	set size	+1	-1	ratio +1 to -1
0	0	9298	1553	7745	0.200516
1	1	9298	1269	8029	0.158052
2	2	9298	929	8369	0.111005
3	3	9298	824	8474	0.097239
4	4	9298	852	8446	0.100876
5	5	9298	716	8582	0.083430
6	6	9298	834	8464	0.098535
7	7	9298	792	8506	0.093111
8	8	9298	708	8590	0.082421
9	9	9298	821	8477	0.096850

Figure 3 – Digits' distribution in the dataset

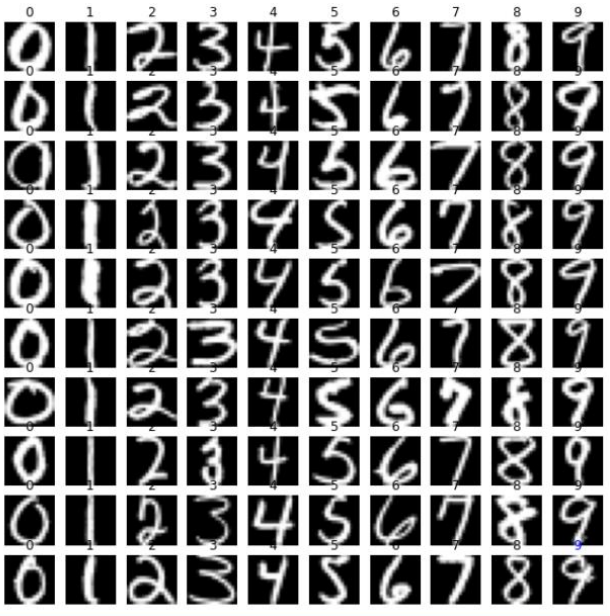

Figure 4 – Visualising samples

Referring to Fig.4 and Fig. 5 we wonder if we could benefit from PCA as there seems to be some correlation between the pixel's intensities (features) and the final label (i.e., pixel 0 and 16, pixel 8 and 24…). The PCA plot in Fig. 6 shows that 90% of the variance is covered by taking the first 70 features. However running Pegasos's algorithm (with a Gaussian kernel) on the reduced dataset with 5-fold cross-validation (thus training it and using it for prediction purposes) takes up more computational time when compared to running the algorithm on the original dataset (total computational time: 290 minutes vs 257min).

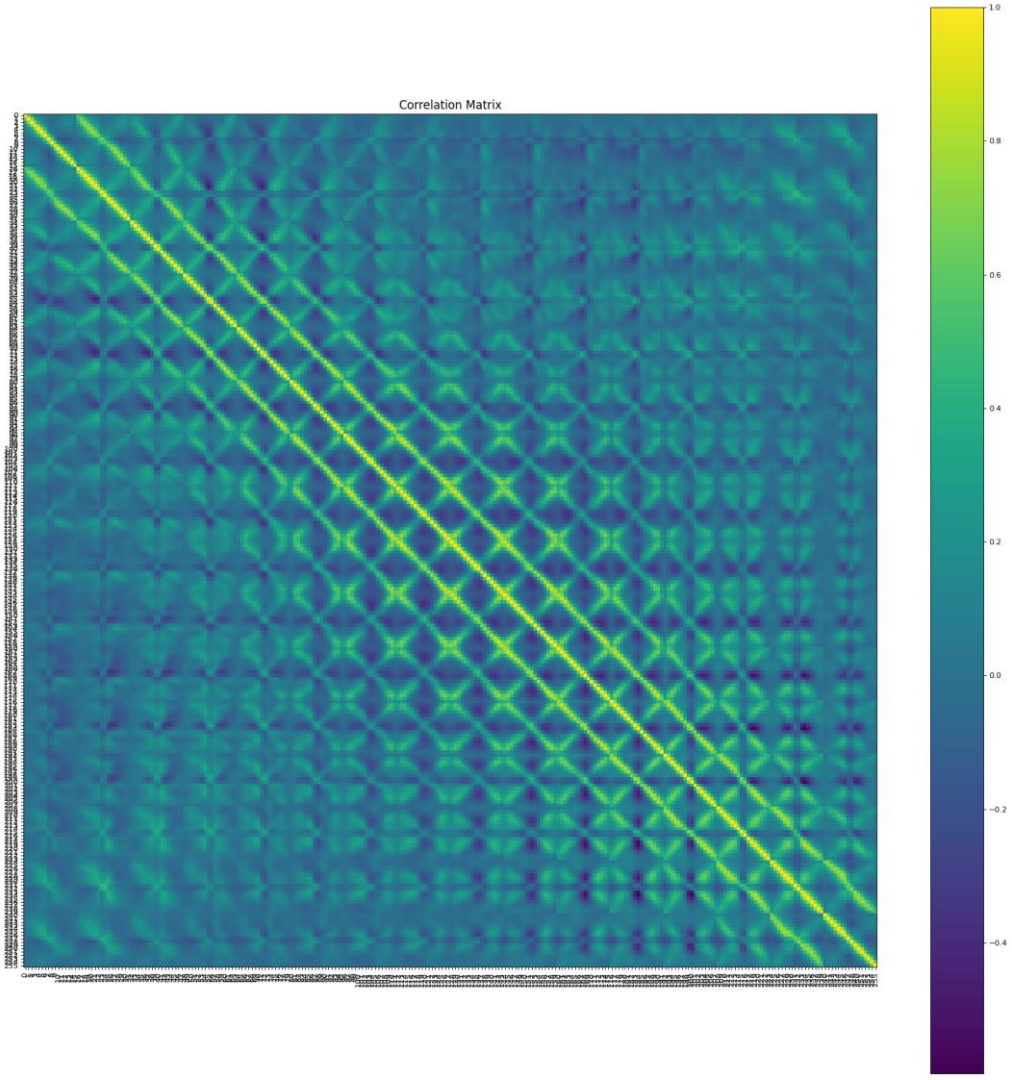

Figure 5 – correlation matrix

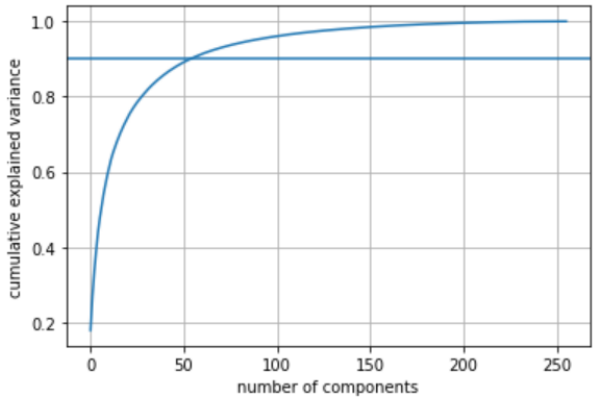

Figure 6: PCA – explained variance

When selecting the best hyper-parameters for each digit (see Fig.7) , we notice that although the mean computational time is 459mins per digit (30% more w.r.t the complete features and kernel with gamma=0.25, see Fig.11) it wildly varies per digit, with it taking up to 45% more of the time compared to the standard kernel run for digits 5 and 7. As the performances are quite similar to the standard kernel run, we opt not to use PCA for the remaining analysis.

	number	ite	lambda	acc_score	prec_score	f1_score	time
0	0.0	1000.0	1.000000e-05	0.991491	0.959964	0.972945	421.061556
1	1.0	1000.0	1.000000e+00	0.995344	0.964093	0.981161	552.030957
2	2.0	1000.0	1.000000e+00	0.988923	0.961722	0.942762	341.021768
3	3.0	1000.0	1.000000e-07	0.985391	0.896324	0.915673	328.921572
4	4.0	1000.0	1.000000e+00	0.982662	0.919158	0.902345	344.027110
5	5.0	1000.0	1.000000e+00	0.986193	0.924998	0.906079	619.971496
6	6.0	1000.0	1.000000e-07	0.991170	0.943547	0.952758	463.126835
7	7.0	1000.0	1.000000e-05	0.988923	0.917981	0.936860	576.462787
8	8.0	1000.0	1.000000e-05	0.985070	0.913036	0.888098	509.840585
9	9.0	1000.0	1.000000e-07	0.982501	0.869229	0.903282	435.574572

Figure 7 – Best hyperparameters for each digit. PCA.

Exploratory Data Analysis

Looking at the first digit 0, we split the data in two sets, the train and the test set, where the test set's size is 33% of the complete set. This results in 6229 observations in the training set and 3069 for the test set. Out of the 1553 samples of 0, the train set has 1020 samples of the digit 0, while the test set has 533.

After fitting the standard Gaussian-kernelized Pegasos algorithm on the training set (gamma=0.25, T=10, lambda=1e-3) we predict the labels on the test set. We repeat the process to check whether Pegasos is consistent in delivering results. Looking at Table 1 and Table 2 we notice that Pegasos is not consistent in delivering the same performance. Indeed the first run has a lower ratio of false negative results (thus the likelihood that it correctly estimates a digit to be different from 0 is higher), and a higher ratio of false positives. This means that there is less of a likelihood that the first run correctly estimates a digit to be 0 (lower precision).

The reason for which the results differ per run are mainly due to the algorithm being, in nature, stochastic and therefore the results (thus the performance) of the algorithm) depending on the order of the training examples. Indeed, by using stochastic (sub)gradient descent, Pegasos arbitrarily (and iteratively) chooses a sample (to adjust the weights if needed). This coupled with the fact that the sets are unbalanced (having many more -1's than +1's) implies that the precision can vary wildly per iteration. Although running the algorithm repeatedly and averaging the results could aid with the robustness of the results, we notice that choosing the f1score is also a good solution to the problem.

Set	**Train**	**Test**
TP, FP, TN, FN	810,572,4637,210	414,282,2254,119
Precision (1-FP ratio)	58.61%	59.48%
FN ratio	4.33%	5.01%
Accuracy	87.45%	86.93%
F1score	67.44%	67.37%

Table 1 – Run 1, standard kernelized Pegasos (gamma=0.25)

Set	**Train**	**Test**
TP, FP, TN, FN	568,89,5120,452	315,68,2648,218
Precision (1-FP ratio)	86.45%	82.25%
FN ratio	8.11%	8.12%
Accuracy	91.31%%	90.68%
F1score	67.74%	68.78%

Table 2 – Run 2, standard kernelized Pegasos (gamma=0.25)

The following figures highlight examples of misclassified digits in red. While the first figure shows two examples of a 0 misclassified as a 2 (false positive), the second figure shows three examples where a 0 is incorrectly predicted and classified as a digit different from 0 (false negatives).

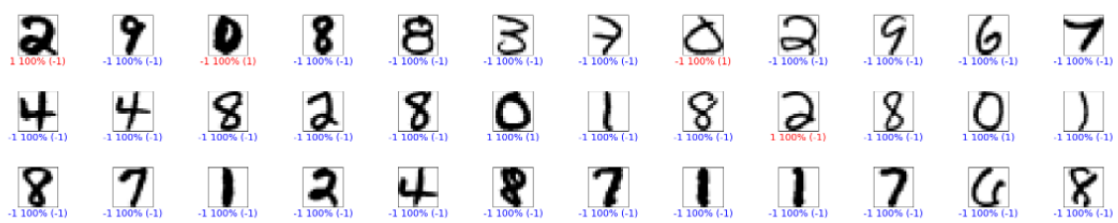

Figure 8 – Visualising predicted digits

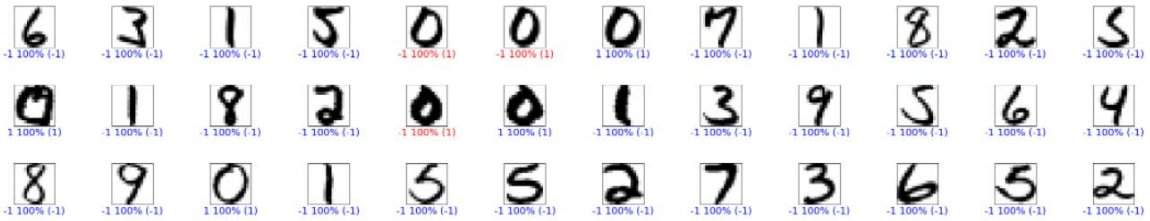

Figure 9 – Visualising predicted digits

Continuing our analysis on digit 0, we now wish to find the optimal hyper-parameters for the Gaussian-kernel (gamma=0.25) Pegasos algorithm, thus the hyper-parameters which maximize the f1score metric. After performing 5-fold cross-validation on the train set to select the "best model" we fit the "best model" on the whole training set and evaluate the performance on the test set. Taking ite=[10,100] and lambda=[1e-5,1e-3], we can see that <ite,lambda>=<100,1e-5> leads to highest f1score (90%) (see Fig. 10).

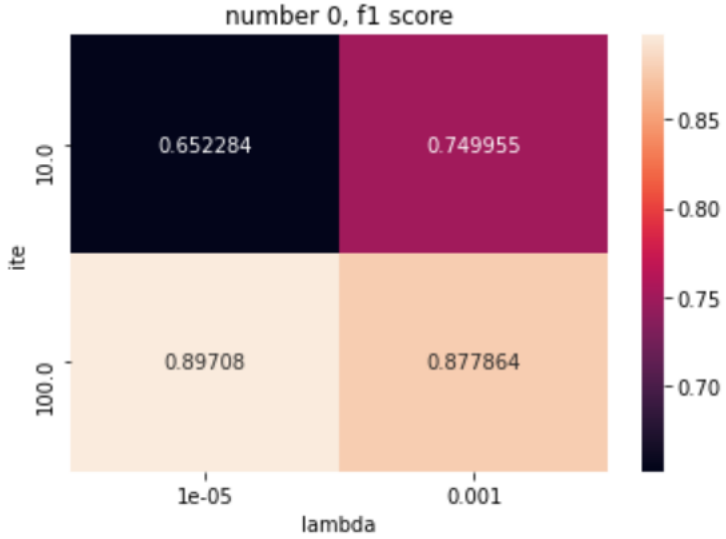

Figure 10 – Heatmap for the standard Kernelized Pegasos - digit 0.

Training the model with these hyperparameters(<ite,lambda>=<100,1e-5>) on the whole dataset and applying it to the test set, we achieve decent performances both on the training set and the test set (see Table 3).

Set	**Train**	**Test**
TP, FP, TN, FN	946,103,5106,74	487,78,2458,46
Precision (1-FP ratio)	90.18%	86.19%
FN ratio	1.43%	1.84%
Accuracy	97.16%	95.96%
F1score	91.45%	88.71%

Table 3 – Train-test results. Digit 0.

Assessing Pegasos on the whole dataset

We now wish to find the optimal hyper-parameters for the Pegasos algorithm by using different kernels: a fourth degree polynomial kernel, a Gaussian kernel with gamma=0.25 and a linear kernel. Taking ite=[10,100,1000] and lambda=[1e-5,1e-3,1] we perform 5-fold cross-validation on the train set and compute the "best hyperparameter" for each digit, where "best" means best CV f1_score (see Fig. 11 14 17 for the Gaussian, Linear and Polynomial kernel respectively).We then select the "best hyperparameters" by selecting the ones which are the "best" for most digits (see Fig. 12 15 18 for the Gaussian, Linear and Polynomial kernel respectively). We then fit the "best model" on the whole set (training and test) and use 5-fold cross-validation to evaluate the performance on the model (see Fig. 13 16 19 for the Gaussian, Linear and Polynomial kernel respectively).

Gaussian Kernel

Looking at the following figures we note that:

- The "best" f1_scores range from 0.87 to 0.98. The CV's model computational time is approx. 340s on the training set and 550s on the whole dataset.
- We select <ite,lambda>= <1000,1e-7> as the best hyperparameter and fit it on the whole dataset.

	number	ite	lambda	acc_score	prec_score	f1_score	time
0	0.0	1000.0	1.000000e-07	0.989244	0.948808	0.965819	337.221102
1	1.0	1000.0	1.000000e-05	0.993738	0.953417	0.976064	353.737011
2	2.0	1000.0	1.000000e-07	0.983142	0.966085	0.908854	349.820663
3	3.0	1000.0	1.000000e-07	0.984749	0.896377	0.911852	350.755101
4	4.0	1000.0	1.000000e-05	0.978647	0.921799	0.877747	350.542014
5	5.0	1000.0	1.000000e-05	0.986033	0.923911	0.903073	357.779297
6	6.0	1000.0	1.000000e+00	0.990850	0.948903	0.951450	356.225983
7	7.0	1000.0	1.000000e+00	0.985872	0.892589	0.921595	357.724705
8	8.0	1000.0	1.000000e-07	0.982823	0.936475	0.870977	349.665379
9	9.0	1000.0	1.000000e-07	0.978487	0.851475	0.880008	358.260377

Figure 11 – Best hyperparameters for each digit. Gaussian kernel.

ite	lambda	
1000.0	1.000000e-07	5
	1.000000e-05	3
	1.000000e+00	2

Figure 12 – Choosing the best hyperparameters. Gaussian kernel.

Looking at the following figure we note that the Kernelized algorithm is quite good at predicting 0 and 1, scoring around 96-97% while finds it trickier to predict numbers 5, 9, 8, 4 (around 86%). Overall, it takes at most 10 minutes to fit (and predict) each digit.

	CV acc score	CV prec score	CV f1score	CV time
0	0.987201	0.942197	0.962474	512.137736
1	0.993009	0.963542	0.974513	599.640972
2	0.982470	0.946470	0.907879	620.110789
3	0.982900	0.887731	0.900020	617.982191
4	0.978598	0.926359	0.880157	613.125949
5	0.981286	0.908456	0.870842	616.046312
6	0.989460	0.931074	0.941450	610.740649
7	0.984190	0.882543	0.908006	616.460874
8	0.982038	0.911351	0.879775	612.958525
9	0.977629	0.838666	0.873993	613.746796

Figure 13 – Final results: Gaussian kernel Pegasos with the best hyperparameters.

Linear Kernel

Looking at the following figures we note that:

- The “best” f1_scores range from 0.70 to 0.98. The CV’s model computational time is approx. 90s on the training set and 140s on the whole dataset. Thus although the computational time is greatly reduced when compared to the Gaussian kernel, the f1_score is consistently lower.
- We select <ite,lambda>= <1000,1e-7> as the best hyperparameter and fit it on the whole dataset.

	number	ite	lambda	acc_score	prec_score	f1_score	time
0	0.0	1000.0	1.000000e-05	0.959067	0.907291	0.856231	106.600813
1	1.0	1000.0	1.000000e+00	0.991492	0.986550	0.971026	99.398973
2	2.0	1000.0	1.000000e-07	0.969176	0.904548	0.831767	96.140272
3	3.0	1000.0	1.000000e-07	0.958418	0.764459	0.774522	97.017404
4	4.0	1000.0	1.000000e-07	0.965805	0.846713	0.817944	95.836789
5	5.0	1000.0	1.000000e-07	0.961310	0.826641	0.707077	103.786010
6	6.0	1000.0	1.000000e-07	0.957300	0.697568	0.805160	77.848448
7	7.0	1000.0	1.000000e-07	0.975599	0.861381	0.867661	78.494490
8	8.0	1000.0	1.000000e-07	0.926638	0.563080	0.592424	78.259350
9	9.0	1000.0	1.000000e-07	0.951841	0.762451	0.768578	77.446670

Figure 14 – Best hyperparameters for each digit. Linear kernel.

```
ite     lambda
1000.0  1.000000e-07    8
        1.000000e-05    1
        1.000000e+00    1
dtype: int64
```

Figure 15 – Choosing the best hyperparameters. Linear kernel.

Looking at the following figure we note that the linear algorithm is quite good at predicting 0 and 1, scoring around 92-96% while finds it extremely tricky to predict numbers 9 8, 5 respectively. Overall the best model’s performance is consistently weaker when compared to the Gaussian kernel.

	CV acc score	CV prec score	CV f1score	CV time
0	0.973003	0.890150	0.922650	142.848120
1	0.989138	0.942537	0.960347	139.756606
2	0.950954	0.861181	0.742624	146.136509
3	0.960529	0.859993	0.725513	141.242198
4	0.958916	0.808535	0.759072	141.056234
5	0.955904	0.804342	0.660004	140.956689
6	0.948811	0.777517	0.759685	140.742676
7	0.962465	0.759734	0.826910	141.058415
8	0.945900	0.745326	0.584407	142.042783
9	0.930309	0.756811	0.440389	140.278532

Figure 16 – Final results: Linear kernel Pegasos with the best hyperparameters.

Polynomial Kernel

Looking at the following figures we note that:

- The "best" f1_scores range from 0.81 to 0.98. The CV's model computational time is approx. 190s on the training set and 300s on the whole dataset. Although the model's performance is slightly worst than the Gaussian kernel it leads to a 30% reduction in computational-time-
- We select <ite,lambda>= <1000,1> as the best hyperparameter and fit it on the whole dataset.

	number	ite	lambda	acc_score	prec_score	f1_score	time
0	0.0	1000.0	1.000000e-07	0.979934	0.923650	0.938009	226.747650
1	1.0	1000.0	1.000000e-07	0.993739	0.965478	0.976964	154.182693
2	2.0	1000.0	1.000000e+00	0.978809	0.891560	0.885553	220.058078
3	3.0	1000.0	1.000000e-07	0.976077	0.919861	0.867833	218.870325
4	4.0	1000.0	1.000000e+00	0.978167	0.910217	0.873342	199.432981
5	5.0	1000.0	1.000000e-05	0.973028	0.847760	0.812079	215.194795
6	6.0	1000.0	1.000000e-05	0.982181	0.914250	0.902553	224.180353
7	7.0	1000.0	1.000000e+00	0.984910	0.935036	0.914256	184.939009
8	8.0	1000.0	1.000000e+00	0.967891	0.722822	0.787899	159.150890
9	9.0	1000.0	1.000000e+00	0.971101	0.835671	0.842451	157.010562

Figure 17 – Best hyperparameters for each digit. Polynomial kernel.

ite	lambda	
1000.0	1.000000e-07	3
	1.000000e-05	2
	1.000000e+00	5

dtype: int64

Figure 18 – Choosing the best hyperparameters. Polynomial kernel.

Looking at the following figure we note that the polynomial algorithm of degree 4 is quite good at predicting 1's and 0's, scoring 97% and 93% respectively while digits 8,4,2,5,9 and (76% to 84%).

Overall the best model's performance is consistently slightly weaker when compared to the Gaussian kernel although it takes up half of the computational time.

	CV acc score	CV prec score	CV f1score	CV time
0	0.977306	0.913017	0.933987	281.697678
1	0.991825	0.964521	0.970186	298.938129
2	0.961710	0.803461	0.814757	311.049094
3	0.975047	0.891722	0.841038	299.782368
4	0.968165	0.818602	0.814398	314.093904
5	0.973649	0.833973	0.821771	319.786124
6	0.975479	0.832351	0.872210	315.836500
7	0.979886	0.921742	0.871119	314.462663
8	0.959885	0.756295	0.764626	318.658673
9	0.971392	0.821411	0.841453	343.577536

Figure 19 – Final results: Polynomial kernel Pegasos with the best hyperparameters.

Ranking

Ref. to the following figure, we notice that the Gaussian kernel (with gamma=0.25) consistently leads to better performance whilst the linear kernel leads to the worst (around 20% loss of performance). This, coupled with the fact that a polynomial kernel also fares quite well, strongly suggests the that the images are non-linearly separable in the original space. We also note that there is a trade-off in choosing between the Gaussian kernel and polynomial kernel as, although the polynomial kernel leads to a 5% loss of performance, it takes half of the computational time.

```
Using a Gaussian kernel the best results we get are
CV acc score       0.983878
CV prec score      0.913839
CV f1score         0.909911
CV time          603.295079
dtype: float64
Using a linear kernel the best results we get are
CV acc score       0.957593
CV prec score      0.820613
CV f1score         0.738160
CV time          141.611876
dtype: float64
Using a polynomial kernel of order 4 the best results we get are
CV acc score       0.973435
CV prec score      0.855709
CV f1score         0.854555
CV time          311.788267
dtype: float64
```

Figure 20 – Final average results: Gaussian, linear, polynomial kernel.

Conclusion

We conclude by noting the main points relevant to our analysis:

- We are dealing with an unbalanced dataset, with digits 5 and 8 represented in ~8% of the dataset (worst-case) while 1 and 2 in ~17% (best-case).
- We are dealing with a considerable amount of variation in the handwritten digits ranging from 2 to 9.
- Digit 1 and 0 should be easy to predict. Digits 4, 5,9 instead could easily be misclassified (as they are not optimally represented and can be mistaken for other digits).

After running the Pegasos algorithm on the original dataset we conclude that

- We shall not apply PCA on the dataset seeing as it leads to an increased training time.
- Digits 9 8 5 4 are difficult to predict.
- Digits 0 and 1 are relatively well-predicted.
- Overall the Gaussian kernel (with gamma=0.25) consistently leads to better results (in all digits) while the polynomial kernel of degree 4 leads to a ~5% performance loss and a reduction in computational time of around 50%. Although the linear kernel leads to further reduction in time, it performs poorly performance-wise, especially on digit 9.
- The “best” model is found to be the kernelized Pegasos algorithm (with gamma=0.25 and hyperparameters equal to <ite,lambda>=<1000,1e-7>), with an average CV f1score of 91.3%.

Further Works

Further works could range from minor improvements and experiments to more creative studies when dealing with the pre-processing of the data. Minor improvements are listed below:

- Splitting the training dataset in folds which ensures an accurate representation of the sets (e.g., +1 -1). Indeed we saw that, the combination of our simple splitting algorithm and the fact that +1's were underrepresented sometimes led to one fold having no +1s (thus this fold was left out when averaging the precision).
- Fitting the Pegasos algorithm n number of times per fold and averaging the results to improve the results' (and consequent observations) robustness.

Experiments could range from:

- Testing a wider range of the hyperparameters on Pegasos (number of iterations and regularizing parameter) and testing the effect of the kernel's hyperparameter (gamma for Gaussian or d for polynomial) on the results to
- Using a one-vs-one approach when encoding the labels or
- Oversampling and under sampling the dataset when creating the training and test folds and verifying whether it improves the model's performance.

Moreover an interesting study would be to verify whether adding some random "virtual examples" (e.g. rotating, shifting and distorting the digits) and additional normalization would improve the model's performance on the dataset.