Kernelized Linear Classification

Project of Statistical Methods for Machine Learning

Mattia Lecchi (964860)

September 1, 2024

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 Preprocessing

The given dataset is made of 10000 examples. First, an affine mapping on data points is applied by adding a new feature and setting it to 1:

$$x = (x_1, ..., x_d) \rightarrow x' = (1, x_1, ..., x_d) \tag{1}$$

This allows the bias term to be included in the weight vector of the linear predictors and learn it like every other weight component. After that, the dataset is split in a training set of 7000 examples and a test set of 3000 examples, ensuring that the number of positively and negatively classified entries is balanced in both sets. The next step is to apply z-score to obtain a standardization of the provided dataset through the following formula:

$$X_i' = \frac{X_i - \mu_i}{\sigma_i} \tag{2}$$

for every feature i. μ_i and σ_i are the sample mean and the sample standard deviation on feature i, respectively; they are computed considering only the training set, as the test set should represent unknown data.

2 Perceptron

Figure 1 shows the performances of the perceptron algorithm on training and test sets, based on the number of epochs. The errors on both sets float around

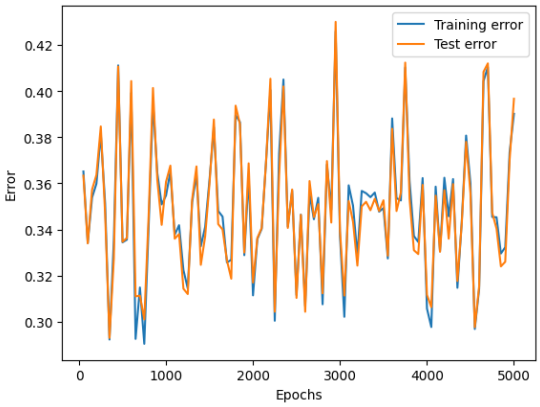


Figure 1: Training and test errors of the perceptron varying the number of epochs

0.36 without lowering; looking at this results, the dataset is probably non linearly separable. With a non linearly separable training set, the perceptron can't terminate.

Quadratic feature extraction An attempt to improve the quality of predictions can be made by applying quadratic feature extraction, at the cost of constructing a predictor considering a more complex dataset: starting from 10 feature, the quadratic feature extraction generate data points of 66 features, including the feature for the bias term learning. Figure 2 shows the achieved errors, which are clearly lower than the ones seen above. Even in this case, the perceptron didn't terminate and the errors stayed around the same values, suggesting that a curve of degree 2 can't perfectly separate the dataset.

2.1 Kernel perceptron

There may be room for improvement by applying a polynomial extraction of higher degree, but the training tends to be computationally intensive even for relatively small degrees and training sets, as the number of features explodes: considering 10 features, with a degree of 3, the number of expanded features would be 286, while with a degree set to 4 the number would be 1001. Kernel methods can be used to overcome this problem.

Polynomial kernel Considering the polynomial kernel:

$$K_d(x, x') = (1 + x^T x')^d \tag{3}$$

figure 3a shows errors using the kernel perceptron with different degrees $d \in \mathbb{N}$; the quality of predictions is far worse than the perceptron with quadratic feature expansion. Furthermore, predictors that used an odd degree performed visibly

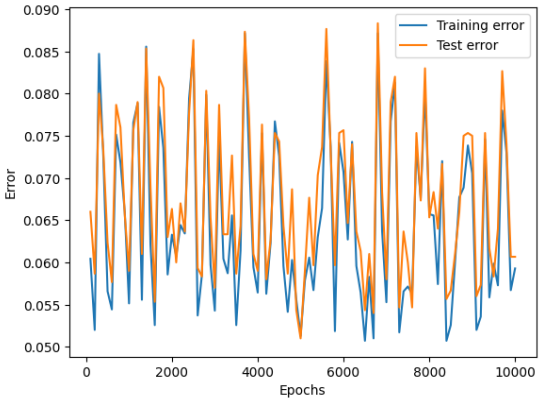


Figure 2: Training and test errors of the perceptron with quadratic feature extraction varying the number of epochs

worse than the others. Also, higher degrees increase the gap between training and test errors, suggesting that large values lead the model to overfit. The kernel perceptron can be forced to make more than one scan on the dataset: instead of adding to a set S the index of examples misclassified during the training, a counter of mistakes α_s is kept for each example s and used as a multiplier. Thus, instead of computing the following function:

$$h(x) = sgn\left(\sum_{s \in S} y_s K(x_s, x)\right) \tag{4}$$

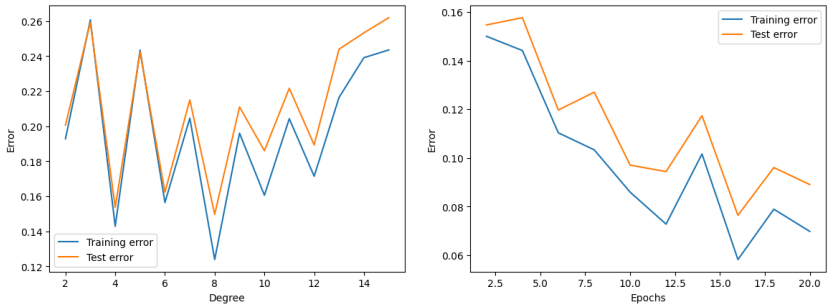
the prediction is evaluated through the following one:

$$h(x) = sgn\left(\sum_{s=0}^{|S|} \alpha_s y_s K(x_s, x)\right) \tag{5}$$

To find a good degree, values can be compared using cross validation: in particular, values between 2 and 10 are used, considering for now only one iteration of kernel perceptron to avoid long execution times. The best performing degree found for a single iteration of the polynomial kernel perceptron is 4, with a loss of 0.1427. To enhance prediction quality, the number of epochs is increased to 20, obtaining the errors shown in figure 3b; the values now approach the ones found in figure 2 for the perceptron with quadratic feature extraction.

Gaussian kernel To initially tune the $\gamma \in \mathbb{R}_{>0}$ hyperparameter of the Gaussian kernel:

$$K_\gamma(x, x') = \exp\left(-\frac{1}{2\gamma}||x - x'||^2\right) \tag{6}$$



(a) (b)

Figure 3: Tuning phases of the perceptron with polynomial kernel. Figure (a) shows an initial search considering degrees from 2 to 10, figure (b) the plot with the error running more epochs on the best selected value using grid search with cross validation, that is $d = 4$

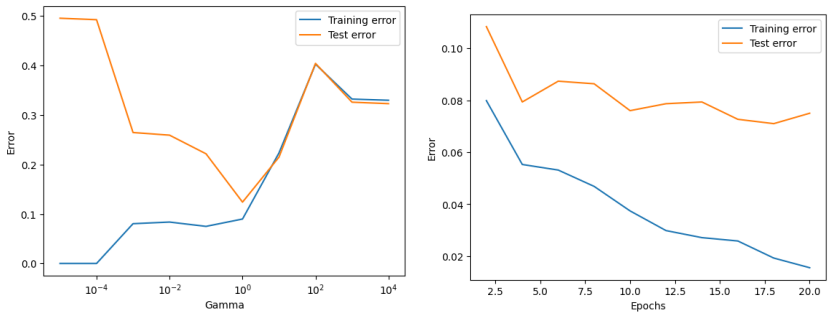
the values picked are in the interval $[10^{-3}, 10^{2}]$ in logarithmic scale, obtaining results shown in figure 4a. The plot clearly shows how values lower than 1 cause overfitting, while values higher than 10 underfitting.

The most promising range is explored performing cross validation considering 10 equidistant values of γ in the interval $[0.5, 5]$. The best value observed during the search is $\gamma = 1.5$ with a loss of 0.1245. A further improvement can be achieved by running the multiple epochs version of the kernel perceptron. The error obtained running up to 10 epochs of perceptron with Gaussian kernel and the selected value of γ is shown in figure 4b.

3 Support vector machines

Training SVM can be more challenging due to the presence of multiple hyperparameters to tune. For this reason, grid search is performed on subsets of hyperparameters using cross validation for model evaluation. The number of folds in which the data is divided is 5. The dataset input for the validation is the one before preprocessing; a preprocessing step as described in section 1 is applied at every iteration of the cross validation, taking into account the division between training and test sets for the considered fold and always assuming that the content of the test set is unknown. In this section, grid search is applied twice: the first time it considers a large interval of values of the hyperparameters. In particular, the epochs considered for tuning are 1000, 5000, 10000, 15000 and the learning rate η and regularization term λ for the SVM objective function:

$$f(w) = \frac{1}{m} \sum_{t=1}^{m} \ell_t(w) + \frac{\lambda}{2} ||w||^2 \tag{7}$$



(a) (b)

Figure 4: Tuning phases of the perceptron with Gaussian kernel. Figure (a) shows an initial search considering a geometric sequence, figure (b) the plot with the error running more epochs on the best selected value using grid search with cross validation, that is $\gamma = 1.5$

are from a geometric sequence of powers of 10 in range $[10^{-3}, 10^{2}]$. The second run of grid search is used to explore the neighborhood of the promising values found in the first iteration.

Starting with SVM, the best loss value obtained through grid search in the first iteration is 0.2777, with 10000 epochs, $\eta = 1$ and $\lambda = 0.001$. For the second iteration, the considered numbers of epochs are 15000, 18000 and 20000, η is selected from 10 equidistant values in the range [0.5, 5], while the considered interval for λ is [0.0005, 0.005]. This time, the selected number of epochs is 20000, $\eta = 0.5$ and $\lambda = 0.003$, with an error of 0.2752. The errors during the training of an instance of SVM is shown in figure 5.

Quadratic extraction As with the perceptron, the results can be improved by using polynomial feature expansion of degree 2. For parameter tuning, The grid search found in the first iteration 15000 as the number of iterations, $\eta = 1$ and $\lambda = 0.001$; the obtained loss is 0.0958. Continuing the exploration, the second run of grid search is performed with numbers of iterations equal to 15000, 18000 and 20000, for η, 10 equidistant values in range [0.3, 3] are considered and for λ, 10 equidistant values in the interval [0.0008, 0.008]. With this second search, a small improvement is found, with a loss of 0.0811; the selected number of iterations is 20000, $\eta = 2.7$ and $\lambda = 0.0008$. Looking at figure 6, it can be seen that the curve of the errors tend to (almost) nullify its steepness after a much greater number of iterations with respect to the predictor on the non-expanded dataset.

3.1 Regularized logistic classification

The hinge loss ℓ_t can be replaced by the logistic loss in the objective function for SVM to obtain a regularized logistic classifier. The evaluation in the first

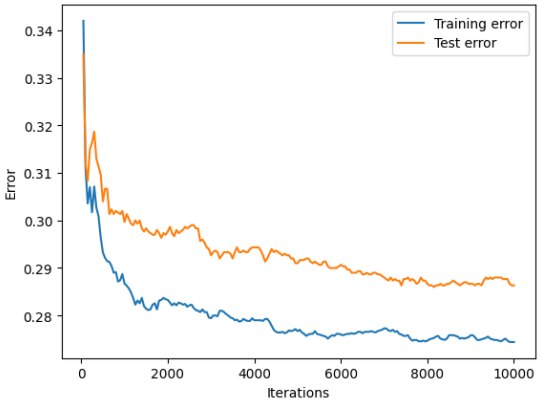


Figure 5: Training and test errors on an instance of SVM with $\eta = 0.5$ and $\lambda = 0.003$

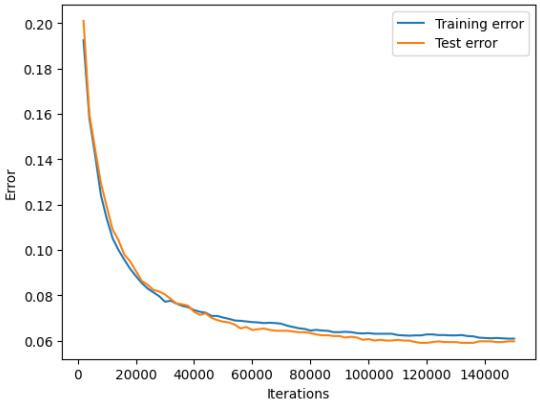


Figure 6: Training and test errors on an instance of SVM with quadratic feature extraction with $\eta = 2.7$ and $\lambda = 0.0008$

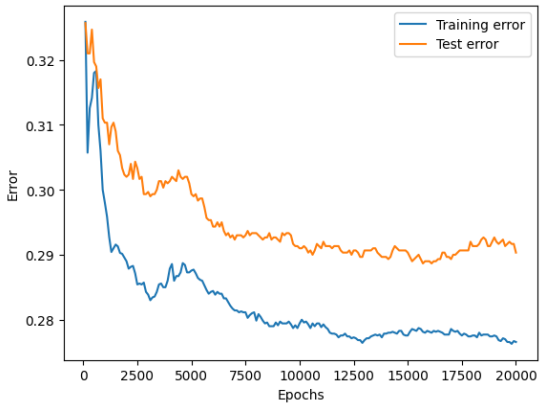


Figure 7: Training and test errors on an instance of regularized logistic classifier with quadratic feature extraction with $\eta = 100$ and $\lambda = 0.0005$

run gave a loss of 0.2799 with 15000 iterations, 100 as η and 0.001 as λ. After that, the neighborhood of the obtained hyperparameters is explored: the new grid search is performed using 15000, 18000 and 20000 as iteration counts, 10 equidistant values in the interval [20, 200] are used for η and 10 equidistant values in the interval [0.0005, 0.005] for λ. the best value found this time is 0.2786 with 18000 iterations, $\eta = 100$ and $\lambda = 0.0005$. The errors on an instance with the parameters just found, varying the number of iterations, is shown in figure 7. The predictor obtained with this version of Pegasos has a slightly higher error than the predictor with hinge loss. The surprising result is the use of a very high learning rate to reach a good quality of prediction.

Quadratic extraction Applying grid search and cross validation with the usual parameters in the first iteration, the regularized logistic classifier with polynomial extraction of degree 2 gives 0.0976 as loss with 15000 iterations, $\eta = 1$ and $\lambda = 0.001$. Next, a grid search is performed with 15000, 18000, 20000 iterations, 10 equidistant values in the interval [0.3, 3] are used as η and 10 equidistant values in the interval [0.0005, 0.005] as λ, obtaining a loss of 0.076 with $\eta = 2.7$, $\lambda = 0.0005$ and 20000 iterations. As in the case of SVM with hinge loss, quadratic extraction seems to require an higher number of iteration to stabilize around a certain value of error. The behavior of errors on a trained instance is shown in figure 8.

3.2 Kernel SVM

Polynomial kernel For the polynomial kernel version of Pegasos, the procedure of hyperparameters selection is quite similar to the one adopted until now. At first, a grid search is run considering degrees d from 2 to 10, λ values are picked from a geometric sequence of powers of 10 in range $[10^{-3}, 10^{2}]$. The

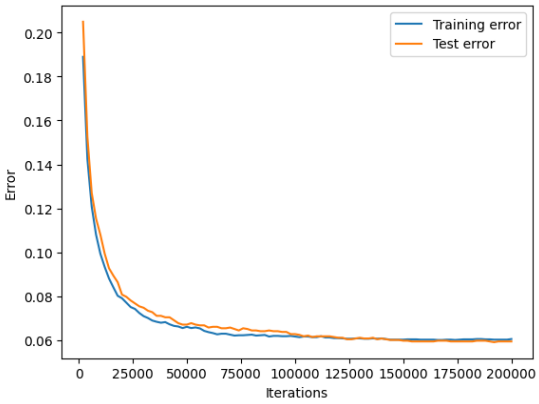


Figure 8: Training and test errors on an instance of regularized logistic classifier with quadratic feature extraction with $\eta = 2.7$ and $\lambda = 0.0005$

obtained loss considering 2000 iterations is 0.2013, $d = 4$ and $\lambda = 0.001$. For the second run, the grid search considers degrees between 3 and 5 and 5 equidistant λ values between 0.0005 and 0.003. The resulting loss didn't change, the selected degree is still 4 and $\lambda = 0.0005$. Figure 9 reports errors on an instance trained in 60000 iterations with the parameters obtained before. The errors through the iterations are overall descending, but there are visible spikes, suggesting that a high number of iterations are needed for reaching a more stable region. However, even training a model with 60000 iterations turned out to be costly in terms of time.

Gaussian kernel For hyperparameters tuning of the Gaussian kernel version of Pegasos, a first run of grid search is performed by considering γ and λ values in a geometric sequence of powers of 10 in the interval $[10^{-3}, 10^2]$. With 2000 iterations, the best parameters found are $\gamma = 1$ and $\lambda = 0.001$ and an error of 0.179. For the second run, the considered γ values are 5 equidistant values in the interval $[0.4, 2]$, while for λ they are 5 equidistant values in the interval $[0.0003, 0.0015]$. Performing 2000 iterations of Pegasos, the best loss obtained is 0.1723 with parameters $\gamma = 0.8$ and $\lambda = 0.0003$. An instance of SVM with Gaussian kernel using the parameters found with the second run of grid search, trained in 30000 iterations, is shown in figure 10.

4 Conclusions

The dataset considered is almost certainly non-linearly separable, as the perceptron algorithm never converged during the experiments, while keeping an error quite stable around the same values through the epochs. Furthermore, the Pegasos algorithm failed to find the separating hyperplane too, even if it

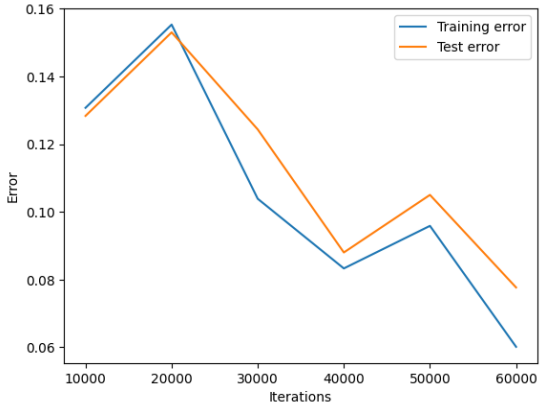


Figure 9: Training and test errors on an instance of SVM with polynomial kernel of degree 4 and $\lambda = 0.0005$

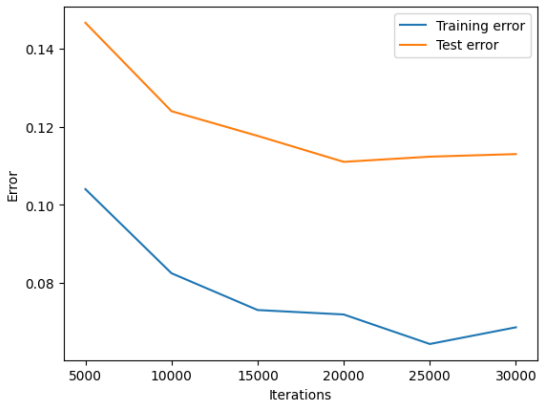


Figure 10: Training and test errors on an instance of SVM with Gaussian kernel with $\gamma = 0.8$ and $\lambda = 0.0003$

was capable of creating more reliable predictors than the perceptron, both optimizing the hinge loss and the logistic loss. A rationale for this result can be found in the way the hyperplanes are chosen by the two algorithms: an SVM explicitly optimize the objective function of the maximum margin separating hyperplane problem, giving a more robust predictor, instead of searching one of the available separating hyperplanes as in the case of perceptron.

The use of quadratic extraction and kernel functions showed a huge improvement on the quality of predictions, with errors near to 0; in particular, the polynomial kernel with $d = 4$ seemed to better describe the curve separating the two classes for the considered problem, both with the perceptron and SVM. The gaussian kernel seemed more prone to overfitting, with a larger gap between test and training errors, even with not too small values of γ.

Computing times The kernelized versions of the considered algorithms turned out to be far more complex in terms of training and prediction times with respect to their basic versions. The comparisons[1] between the perceptron, with and without quadratic extraction, and kernel perceptron is shown in figure 11, considering the best hyperparameters found in the previous sections. The kernelized version shows the steepest curve for the training times measured on a training set of 7000 examples, but, most importantly, the basic perceptron prediction times (measured on 3000 examples) don't depend on the number of epochs run or on the training set size, as it's sufficient to compute the output from the vector weight whose size depends only on the number of features. The kernel perceptron, on the other hand, needs to keep a set of points that can grow unbounded and that needs to be scanned for each prediction. A similar observation can be made for SVM, as it can be seen from figure 12; in this case a larger number of iterations can be performed with the same amount of time with respect to perceptron, as an iteration of Pegasos involve a random extraction and not an entire scan on the training set. This shows that along with better predicting capabilities on non-linearly separable datasets, the kernelized algorithms come also with less scalability, which, in addition, prevented a better exploration of the hyperparameters spaces, the convergence of the stochastic gradient descent towards the minimum of the SVM objective function or to find a separating curve (in case there is) if considering the perceptron, all in reasonable amounts of time. Looking at the results, polynomial feature expansion can be a better choice when the number of features is relatively low and so is the degree d, as in the case with $d = 2$.

[1] The times shown here and in the following are measured on a Ryzen 5 4500u with 8GB of RAM running Arch Linux with kernel 6.10.2-zen1-1-zen. The version of the Python interpreter is 3.12.4

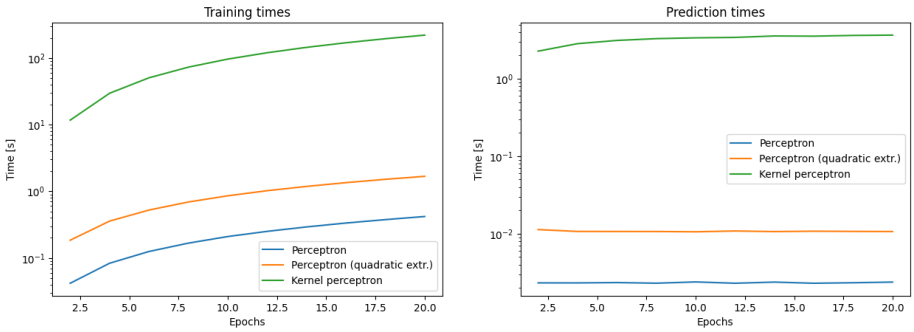


Figure 11: Training and prediction times comparisons between perceptron and kernel perceptron

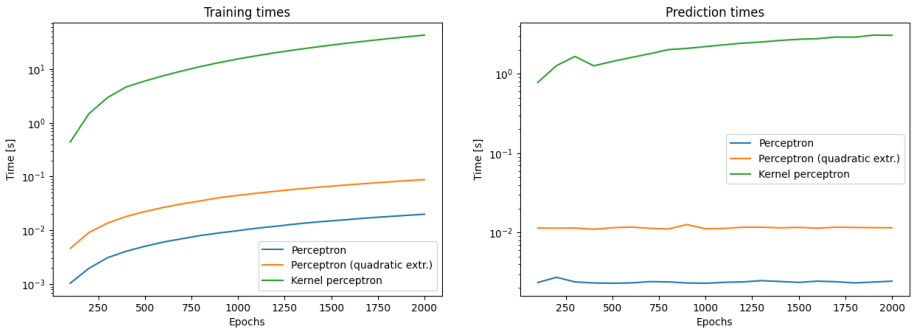


Figure 12: Training and prediction times comparisons between Pegasos and kernel Pegasos