6.867 PSET #2

Anonymous authors

1. Logistic Regression (LR)

1.1. L₂ Regularized Logistic Regression

We first consider logistic regression with a L_2 penalty on the weight vector. The objective function in this case is:

$$E_{LR}(w, w_0) = L(w, w_0) + \lambda \|w\|_2^2 \quad where \ \ L(w, w_0) = \sum_{i=1}^{N} \log\left(1 + exp\left(-y_i\left(w^T x_i + w_0\right)\right)\right)$$

We implement an optimization algorithm utilizing batch gradient descent that minimizes the above objective function with respect to w, w_0 given a dataset and value for λ. To test the algorithm, we use the given dataset: *data1_train.csv* for $\lambda = 0, 1$. The weight vector as a function of iteration number is shown below in Figure 1.

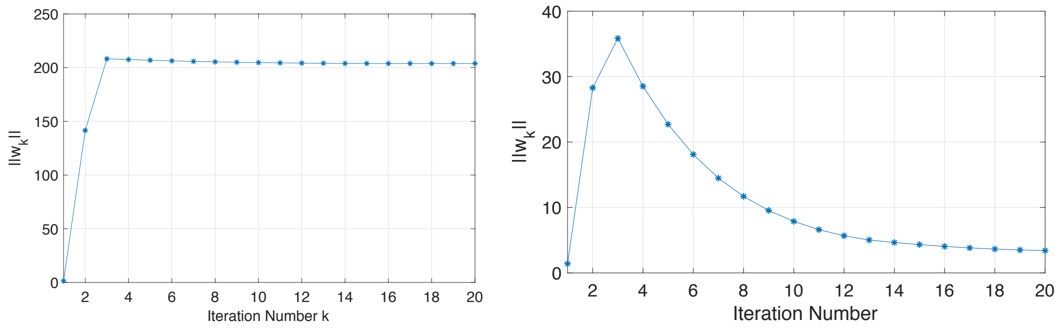

Figure 1. Weight vector magnitude as a function of iteration number for $\lambda = 0$ (left) and $\lambda = 1$ (right)

The weight vector for $\lambda = 0$ rises until it reaches a "steady" state. Since there is no penalty its magnitude, the weight vector is set as to best fit the data (minimize the first term in objective function) without regard for weight values. In contrast, for $\lambda = 1$, $\|w_k\|$ increases initially then decreases until it reaches a "steady" state at a low magnitude. This is because there is a penalty on the weight vector values and the solver has to "balance" the magnitude of the first and second terms of the objective function.

1.2. L₁ vs. L₂ Regularization for Logistic Regression

Next the use of an L_1 regularizer is compared to the L_2 regularizer. The objective function is shown below. Note the normalization term in front of the loss function. This is because the *l1_lorgreg* package used to conduct L_1 regularized LR defines the objective function in this way. To compare to L_2 regularization, the loss function takes the below definition as well.

$$E_{LR}(w, w_0) = L(w, w_0) + \lambda \|w\|_1 \quad where \ \ L(w, w_0) = \frac{1}{N} \sum_{i=1}^{N} \log\left(1 + exp\left(-y_i\left(w^T x_i + w_0\right)\right)\right)$$

First to determine the effect of λ on the weight values, the L_1 and L_2 objective functions were minimized for four data sets for various values of λ. For data set 1, the weight vector components as a function of λ are shown below:

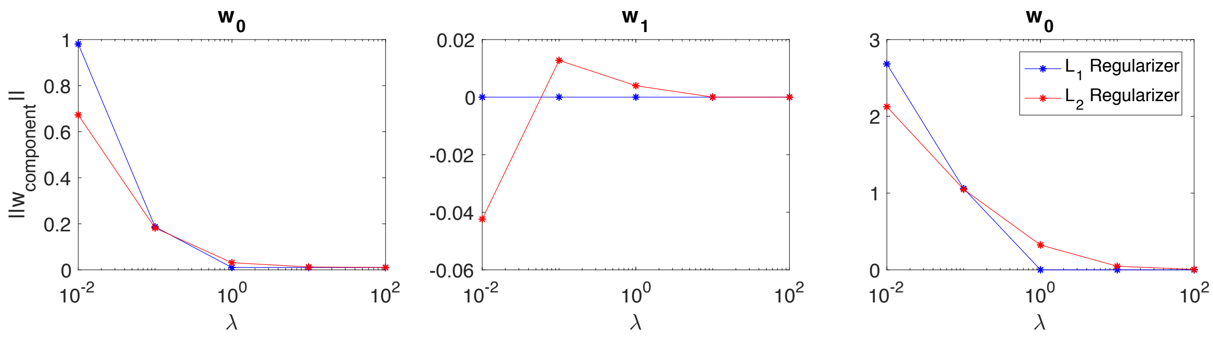

Figure 2. Weight vector component magnitudes for data set 1 for L_1 and L_2 LR

L_1 regularized LR drives the weight components down to zero while L_2 regularized LR drives down the magnitude of all the weights at, not necessarily to zero, as expected from previous analysis of LASSO vs. Ridge regressions. While only data set 1 is shown, the trend was found for the three other data sets. The decision boundaries for the four data sets are shown for $\lambda = 0.1$ for data sets 1-3 and $\lambda = 0.01$ for data set 4 are shown in Figure 3. The reason for differing λ is that a large λ for L_1 LR drives the weight vector w to zero for all components, such that there is no decision boundary – all points are classified the same. This is contrast to L_2 LR which tends to keep the weight vector components of similar magnitude, causing the slope of the boundary to remain relatively constant with λ. We believe that this is due to the fact that the LR penalty for misclassification reaches an asymptote (since $\mathbb{P}(y = 1 \mid x, w, w_0)$ is bounded between 0 and 1), unlike SVM whose penalty for misclassification increases linearly with distance from the decision boundary. The resulting decision boundaries are also used to classify test data sets; the resulting misclassification rates for [L_1, L_2] are [0%, 0%], [19.5%, 19%], [3.5%, 3%] and [49.5%, 50.25%] for test data sets, 1, 2, 3, and 4 respectively. The authors found that λ had little effect on misclassification error rates except when the value drove $\|w\| = 0$ for L_1 LR. The authors suspect that due to the 2-D nature of the data set, the optimization can always attain a constant value of $\frac{w_1}{w_2}$, which maintains the slope of the decision boundary.

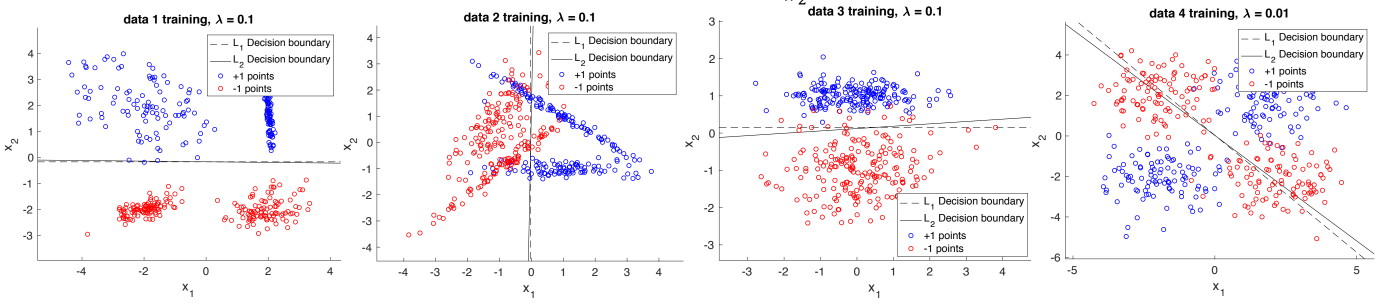

Figure 3. Decision boundary for L_1 and L_2 LR for data sets 1-4

1.3. Model Selection for Logistic Regression

We conduct a mock model selection simulation on the four data sets, splitting them into training, validation and test sets. The two LR models are trained on the training data for a variety of λ values, from which the $\lambda_{opt} = argmin_\lambda \left(N_{misclassify}(X_{validate}, Y_{validate}, \lambda)\right)$ is found, where $N_{misclassify}$ is the function which returns number of misclassified points on the validation set. The performance of the chosen model is then found by finding misclassification rates for the test sets. The resulting data is summarized in Table 1.

Parameter	Data set 1	Data set 2	Data set 3	Data set 4
L_1 LR λ	0.001	0.001	0.0065	0.013
L_2 LR λ	0.001	0.001	0.001	30.88
L_1 Validation error rate	0.0%	17.5%	2.5%	50.0%
L_2 Validation error rate	0.0%	17.5%	2.5%	45.25%
Chosen model	L_2	L_2	L_2	L_2
Test data error rate	0.0%	19.5%	3.5%	43.0%

Table 1. LR model selection summary

The classifier models chosen seem to perform reasonably well on the test sets (when considering the validation set error) and in general the method to do the model selection seems to be a good approach. It is interesting to note that the L_2 LR was the best model in all cases, suggesting that for these data sets L_1 LR performs poorly in general.

2. Support Vector Machine (SVM)

2.1. Linear Soft SVM test

The dual form of linear soft SVMs was implemented taking as input data observation matrix X and corresponding class data Y, which each row corresponding to one observation. The constrained optimization problem corresponding to this type of SVM is as follows (Bishop 7.32, in minimization form, adapted to matrix form):

$$\min_a \frac{1}{2} a^T \boldsymbol{K} a - fa\,, \qquad where\ f = ones(1, N)\ \ and\ \boldsymbol{K_{ij}} = Y_i Y_j x_i^T x_j$$
$$s.t\quad 0 \le a_i \le C, \qquad Y^T a = 0$$

The decision boundary corresponding to linear soft SVM is:

$$y(x) = 0 = \sum_{i=1}^{N} a_i Y_i x^T x_i + b, \qquad where\ b = \frac{1}{N_P} \sum_{i \in P} \left(Y_i - \sum_{j \in S} a_j Y_j x_i^T x_j \right)$$
$$P\ is\ the\ set\ of\ points\ where\ 0 < a_i < C\ and\ S\ is\ the\ set\ of\ support\ vectors$$

To test the implementation of the created linear soft SVM, a simple test case with positive examples (2, 2), (2, 3) and negative examples (0, -1), (-3, -2) is used. C is set to positive infinity to use hard SVM. Support vectors are found to be (2, 2) and (0, -1). The points, decision boundary and margins are shown in Figure 4.

2.2. Soft SVM on Datasets

Next, we set C = 1 and run the soft SVM algorithm on four sets of training and validation data. The resulting decision boundaries and misclassification rates are shown in Figure 5 and Table 2 respectively. There are several things that are apparent. Since the decision boundary is linear in the input space (linearly separable refers to the input space X for the rest of the discussion), the SVM algorithm works well for data that is (almost) linearly separable in this space and poorly for data that is not. In particular, the SVM performs well for data sets 1 (strictly linearly separable) and 3 (almost linearly separable).

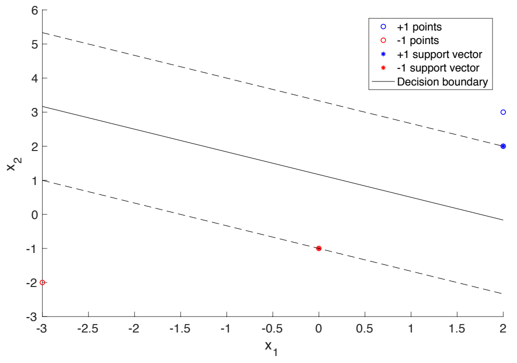

Figure 4. Simple test case for soft SVM

It performs poorly for data set 2 which has overlapping class distributions and very poorly for data set 4 whose classes are split such that a linear decision boundary will always misclassify ~50% of the points (by visual inspection). In order to improve classification for data sets 2 and 4, the input space must be mapped to a different space such that the classes can be separated by a linear decision boundary.

Data set	**1**	**2**	**3**	**4**
Training	0 (0.0%)	76 (19.0%)	10 (2.5%)	117 (29.25%)
Validation	0 (0.0%)	37 (18.5%)	12 (6.0%)	119 (29.75%)

Table 2. Misclassified points for soft SVM (C = 1) for the 4 data sets

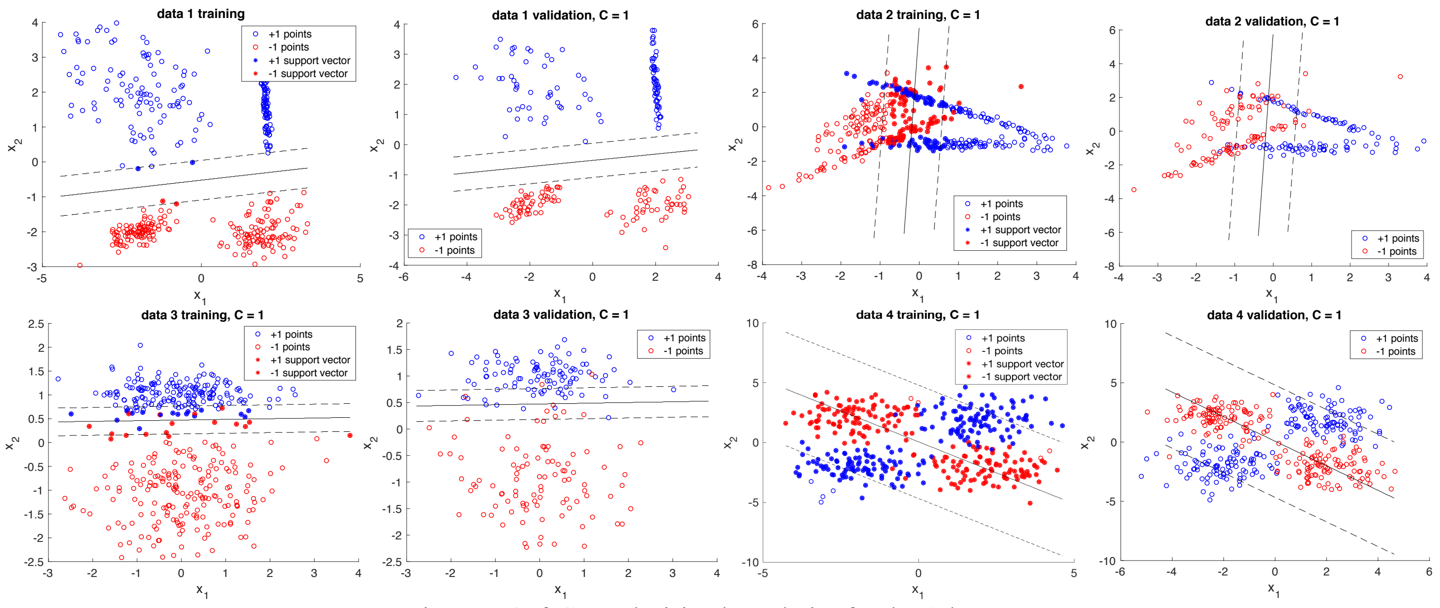

Figure 5. Soft SVM decision boundaries for the 4 data sets

2.3. Kernelized Soft SVM

The soft SVM algorithm (shown in Section 2.1) was updated to allow for the use of Kernels by replacing the following:

$$\boldsymbol{K_{ij}} = Y_i Y_j k(x_i, x_j), \qquad y(x) = 0 = \sum_{i=1}^{N} a_i Y_i k(x, x_j) + b, \qquad where\ b = \frac{1}{N_P} \sum_{i \in P} \left(Y_i - \sum_{j \in S} a_j Y_j k(x_i, x_j) \right)$$

We explore the effect of the kernelized SVM for $C \in \{0.01, 0.1, 1, 10, 100\}$ with a linear kernel $\left(k_{linear}(x_i, x_j) = x_i^T x_j\right)$ and Gaussian RBF kernel $\left(k_{RBF}(x_i, x_j) = \exp\left(-\lambda_{RBF} \|x_i - x_j\|^2\right)\right)$. For the Gaussian RBF kernel, we explore $\lambda_{RBF} \in \{0.1, 1, 10\}$. First, to illustrate the effect of C, data set 1 is used with a linear kernel. The decision boundary and margins are shown below in Figure 6. As C is increased, it can be seen that the decision boundary becomes more dependent on x_1 (ie. slanted) and the margins size decreases until a minimum number of support vectors is found (in this case, 3). If the minimum number of support vectors is achieved, then increasing C will not decrease the margin size anymore, as there must always be at least 2 support vectors (minimum can be greater than 2 if other points are the same distance from the decision boundary). As C is increased (margin size decreases) the number of support vectors decreases as more points are outside the margins.

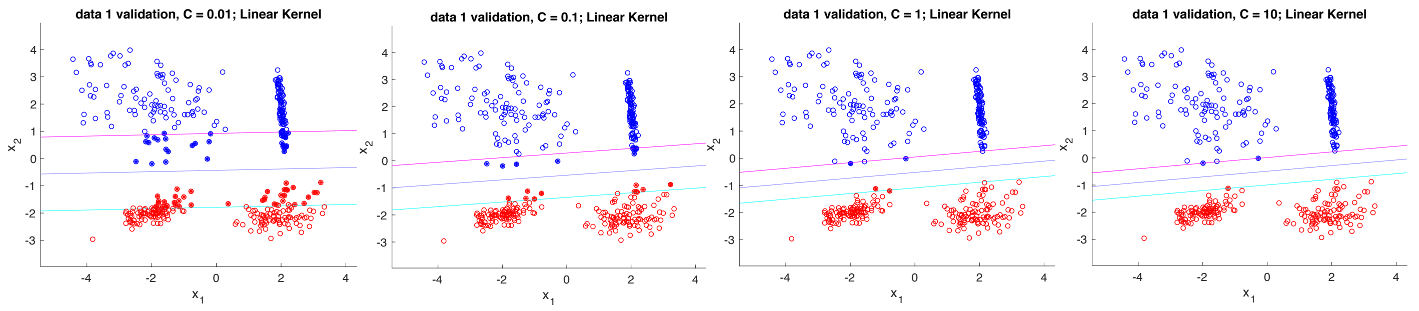

Figure 6. Effect of C on SVM decision boundary, illustrated for data set 1. (Note: $C = 100$ not shown but similar to $C = 10$)

This effect is because C determines how much the objective function increases as a result of margin violations. If C is large, violations are weighted heavily and SVM tolerates fewer violations and visa versa. Note that while only the linear kernel case for data set 1 is shown, the analysis was conducted for the Gaussian RBF kernel and other data sets and a similar trend was found. To choose C, maximizing the geometric margin is a non-ideal strategy because it will lead to a very large margin (C = 0) and has a high probability of misclassifying both training and unseen test data (exceptions exist, such as for linearly separable data). An alternative criterion would be to train an SVM with various hyper-parameter values (C and λ_{RBF}) on training data and calculating the misclassification rate of the resulting SVM on a set of validation data; choose the hyper-parameter values that result in a minimum misclassification rate. This method should reduce the probability of misclassification on unseen test data assuming it is distributed similarly to the training and validation data.

To illustrate the effect of λ_{RBF} on classification with the RBF kernel, data set 2 and C = 1 is used; the resulting decision boundaries are shown in Figure 7. The validation misclassification rate for $\lambda_{RBF} = [0.1, 1, 10]$ are [16.5%, 22.0%, 7.5%] respectively, suggesting an optimum can be found. In general, increasing λ_{RBF} causes the decision boundary to "shrink" around the training data, increasing the number of support vectors as well.

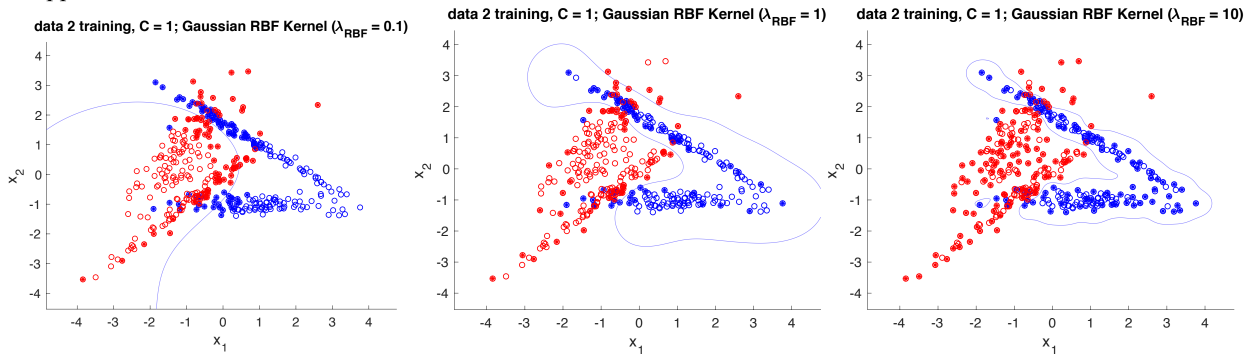

Figure 7. Effect of λ_{RBF} on SVM decision boundary with RBF kernel. Filled points are support vectors.

3. Soft-Margin SVM with Pegasos

3.1. Pegasos linear soft SVM

To overcome traditional SVM implementation's issues with large data sets, the Pegasos algorithm was implemented, which combines the use of a hinge loss, L_2 regularization, stochastic sub-gradient descent, and a decaying step size, allowing it to handle large data sets. The optimization problem for the Pegasos algorithm is:

$$\min_{w,w_0} \frac{\lambda}{2}\|w\|^2 + \frac{1}{N}\sum_{i=1}^{N} \max\{0, 1 - y_i(w^T x_i + w_0)\}$$

The Pegasos algorithm is used on the training data set 3 with various values of λ to determine the effect of λ on the SVM margin, $\frac{1}{\|w\|}$. The resulting values are shown in Table 3. The margin width is shown to increase with increasing λ. This is because as λ increases, the importance of the weight vector magnitude increases and the classifier is willing to tolerate more margin violations (which occurs as the margin increases in width).

λ	2^{-10}	2^{-9}	2^{-8}	2^{-7}	2^{-6}	2^{-5}	2^{-4}	2^{-3}	2^{-2}	2^{-1}	2^{0}	2^{1}
Margin	0.23	0.27	0.34	0.42	0.47	0.56	0.67	0.78	0.93	1.14	1.43	2.17

Table 3. Margin width as a function of λ

3.2. Kernelized Pegasos

We now extend the Pegasos implementation to work with kernels. In this case, the optimization problem becomes:

$$\min_{w} \frac{\lambda}{2}\|w\|_2^2 + \frac{1}{N}\sum_{i=1}^{N} \max\{0, 1 - y_i(w^T\phi(x_i))\}$$

The formulation given retains the sparsity properties of the traditional dual SVM solution as only some subset of the total training data set contributes to the determination of the decision boundary (where $\alpha_i \neq 0$). Predictions are made for a new input x by evaluating $y(x) = sign(\sum_{\alpha_i \neq 0} \alpha_i k(x, x_i))$. To test the kernelized Pegasos algorithm, we use the Gaussian RBF kernel (noted in Section 2.3, here we change notation with $\gamma = \lambda_{RBF}$) for various values of RBF bandwidth, $\gamma \in \{2^2, 2^1, \dots, 2^{-2}\}$, and fixed $\lambda = 0.02$. We focus on determining how γ affects the decision boundary and number of support vectors. The margin width and support vector number found are tabulated in Table 4. The decision boundary, support vectors, and points are shown in Figure 8.

γ	2^{-2}	2^{-1}	2^{0}	2^{1}	2^{2}
Margin width	0.0394	0.0395	0.0329	0.0319	0.0298
# of support vectors	30	30	29	38	58

Table 4. Margin width and support vector number as a function of γ

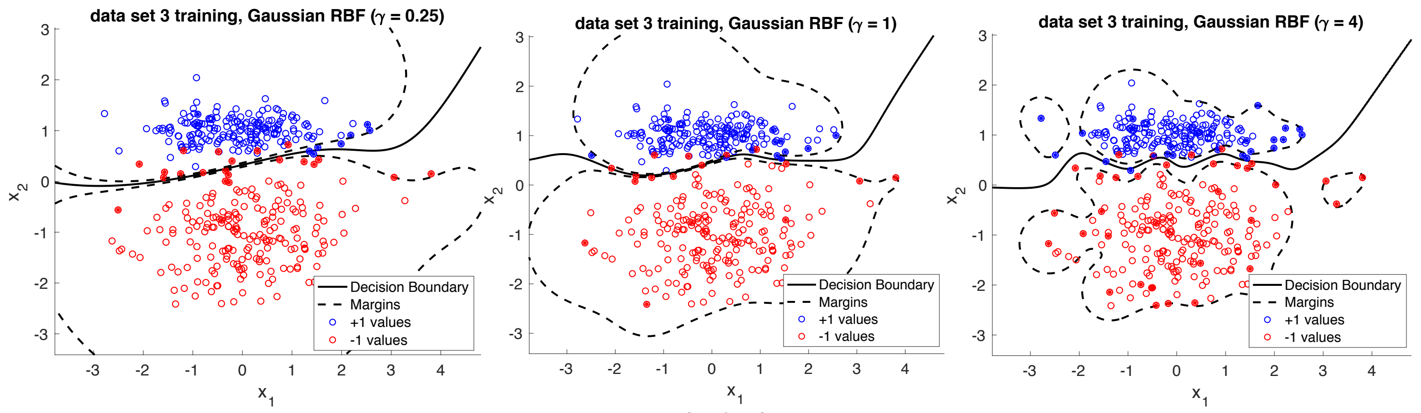

Figure 8. Pegasos with Gaussian RBF for $\gamma = [2^{-2}, 2^0, 2^2]$, left to right. Filled points are support vectors.

As γ is increased, the decision boundary "shrinks" (both visually and quantitatively the margin width decreases) and fits around the data points more tightly and the number of support vectors increase, which matches the observations in the previous section (2.3) as expected. This verifies that the Pegasos algorithm matches the traditional SVM implementation.

4. Handwritten Digit Recognition with MNIST

So far we have developed three different classification methods and tested them on simple two-dimensional data. We are interested in testing our algorithms on real data and for this next section consider the famous MNIST dataset. The MNIST dataset contains images of digits 0 through 9. The methods developed are tested in different scenarios for binary classification tasks. For the first two sections (4.1 and 4.2) the input data used are the raw data and normalized data, where normalized data refers to mapping the input values to the interval [-1 1]. For each digit considered, we take 200 training examples, 150 validation examples, and 150 test examples for all sections.

4.1. Logistic Regression vs. Linear Soft SVM (traditional)

First, the logistic regression algorithm from Section 1 and soft SVM algorithm from Section 2 are compared. The cases considered are: (1 vs. 7), (3 vs. 5), (4 vs. 9), (0, 2, 4, 6, 8) vs. (1, 3, 5, 7, 9) (even vs. odd). For each case, the LR and SVM are trained on the training data set and the classification error is found for the trained classifiers on the training and test data sets. For all cases, we use $\lambda = 0.01$ and L_2 regularizer for LR and $C = 0.01$ for the SVM. The results are shown in Table 5. Overall, LR and SVM perform similarly on the unseen test data, with very similar misclassification rates. Normalization has mixed effects, sometimes showing lower misclassification rates (3 vs. 5) or higher misclassification rates (4 vs. 9); however, it seems that it has a slight benefit on unseen data at the cost of training accuracy. This could be due to the use of constant hyper-parameter values for the two classifiers. Since normalization reduces the magnitude of the input values, this would affect the optimized weights (w for LR, a for SVM); these weights are constrained by C and λ and thus it is likely that normalization requires different hyper-parameter values. Several of the misclassified digits from the test data set for all the cases (1 for each case) are shown in Figure 9. The misclassified digits seem to make sense, in that they are not traditional representations for the digits (in an average sense).

Case	1 vs 7 raw	1 vs 7 normalized	3 vs. 5 raw	3 vs. 5 normalized	4 vs. 9 raw	4 vs. 9 normalized	Even vs. Odd raw	Even vs. Odd normalized
LR Training Error	0.0%	0.0%	0.0%	0.5%	0.0%	0.25%	0.0%	8.5%
SVM Training Error	0.0%	0.0%	0.0%	2.5%	0.0%	1.75%	0.0%	8.05%
LR Test Error	0.66%	1.33%	6.66%	4.0%	5.66%	5.66%	14.13%	11.6%
SVM Test Error	1.33%	1%	5.33%	4.66%	5.33%	6.33%	16.33%	11.2%

Table 5. Misclassification rates for the cases for LR and SVM

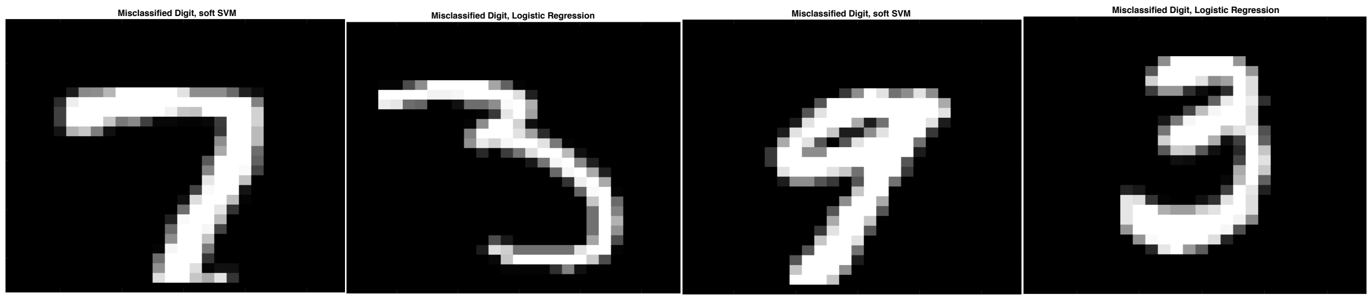
Misclassified Digit, soft SVM
Misclassified Digit, Logistic Regression
Misclassified Digit, soft SVM
Misclassified Digit, Logistic Regression

Figure 9. Misclassified digit examples for the four cases

4.2. Gaussian RBF SVM Classifier on MNIST data

We next investigate the use of SVM with a Gaussian RBF kernel (Section 2.3). The same cases are used, with C and γ chosen by performance on the validation data sets (corresponding to lowest misclassification rate on the validation set). We consider $C \in [10^{-3}\ 10^{3}]$ and $\gamma \in [10^{-4}\ 10^{1}]$. Again, normalization and raw data is considered. We found that the raw data cases resulted in very large misclassification errors (~40%) for all comparisons using the hyper-parameter value ranges considered. It seems that normalization is required if a Gaussian RBF kernel is used for this classification task. This is can be attributed to the difference in formulation between the RBF and linear kernels. The results for the normalized cases are shown in Table 6. It is interesting to note that $\gamma = 0.001$ and $C = 10$ works well for all classification cases. This suggests that Gaussian RBF hyper-parameters depend on what the data sets are (ie. All the classification is for digits, so hyper-parameter values are similar).

Case	1 vs 7 normalized	3 vs. 5 normalized	4 vs. 9 normalized	Even vs. Odd normalized
$\boldsymbol{\gamma}$	0.001	0.001	0.001	0.001
C	10	10	10	10
Test Error	1.66%	3.33%	5%	4.2%

Table 6. Misclassification rates for SVM with a Gaussian RBF kernel.

Compared to the linear kernel, the Gaussian RBF seems to similarly or better than the linear kernel on the test data, suggesting that it is better suited to handling digit recognition classification tasks. This is especially true for the even vs. odd case, with the RBF kernel showing less than half the error of the linear kernel.

4.3. Pegasos vs. SVM with Quadratic Programming

Finally, we investigate the use of Pegasos versus SVM solved with quadratic programming tools. The same cases are used with the Gaussian RBF kernel with normalized data, and the test error and running time are investigated for the two implementations. We vary the number of training points and Pegasos parameters (λ and number of epochs) as well. The training points considered are {200, 300, 400, 500} and the Pegasos parameters are $\lambda = \{10^{-3}, 10^{-2}, 10^{-1}\}$ and $N_{epoch} = \{10, 100, 500\}$. The resulting run times can be seen below (500 epoch case is not included for clarity). We find that both the Pegasos and QP run-times increase with the number of training points which is expected as both algorithms will need to consider more points for each epoch in the Pegasos case and for the kernel Gram matrix calculation for the QP case. Additionally, as the Pegasos algorithm epoch number is increased, the run-time increases and the classification error decreases. This follows logically as epoch number increases, Pegasos cycles through the data more, allowing for better tuning of the α values as more data is used and increasing run-time as more calculations are required. Compared to QP, Pegasos was found to have a much greater classification error for the 10 epoch case (a low epoch number does not allow for proper tuning of the α values) while achieves a similar error for the 100 and 500 epoch cases. Finally, for Pegasos, as λ increases, the initial learning rate η increases, which leads to large initial values of α_i; this results in more iterations for the irrelevant α_i to decay to zero which leads to longer run-times. This is because it takes longer for Pegasos to achieve sparsity in the α vector.

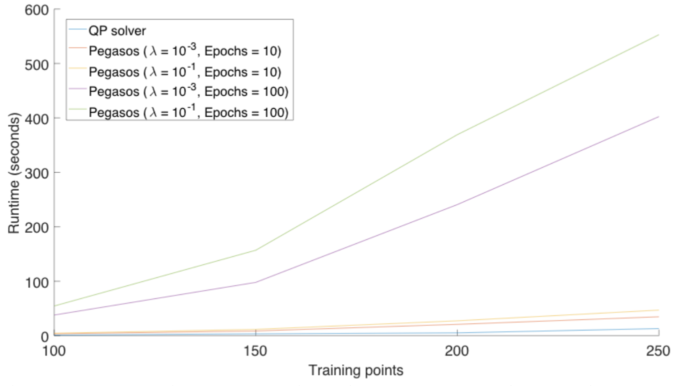

Figure 10. Run time comparison between QP solver and Pegasos