6.867: Homework 2

Anonymous authors

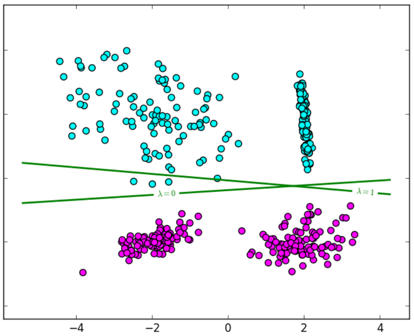


Fig. 1: LR decision boundaries for L_2 $\lambda = 0$ and $\lambda = 0$.

***Abstract*— This is Machine Learning homework 2. Topics include Logistic Regression, SVM, Gaussian RBF Kernel, Pegasos, and classifying the MNIST dataset. All work has been done in Python.**

I. Logistic Regression (LR)

In this section, we consider Logistic Regression (LR) for binary classification of several 2D datasets, and compare the effect of different model parameters and regularization.

A. Part 1

We used SGD to optimize the LR objective with L_2 regularization, according to the equation:

$$E_{LR}(w, w_0) = \sum_i log(1+exp(-y_i(w^T x_i + w_0))) + \lambda ||w||_2^2 \tag{1}$$

The decision boundary is shown in Fig. 1 for the cases of no regularization and some regularization ($\lambda = [0, 1]$). When $\lambda = 0$, $w = [1.73, -0.34, 4.55]$, and training accuracy is 100%. When $\lambda = 1$, $w = [0.03, 0.03, 0.32]$, and there are some misclassified training samples, but the size of the weight vector is much smaller.

As iteration number increases, the weight vector converges to the learned weight, and the change between iterations gets smaller, shown in Table I.

Iteration	w_1	w_2
1	-0.0069498	0.00896533
2	0.00249062	0.02189003
...		
39,999	-0.34175886	4.55454265
40,000	-0.34173718	4.55456526

TABLE I: Accuracy of LR on four datasets.

B. Part 2

An alternative regularization, L_1, was then compared against L_2 from the previous section. In Fig. 2, the elements of the weight vector are compared. The top row shows L_1 regularization for each of the four datasets, and the bottom row shows L_2. In general, increased regularization parameter, λ, causes smaller magnitude weight vector, since the blue and green points (small λ) are further from zero than the magenta and yellow points (large λ). This is true for both L_1 and L_2 regularization. Especially clear in the 2nd and 4th columns, L_1 regularization creates a more sparse weight vector, as all elements are zero (yellow) for the highly regularized case, whereas with the same regularization constant with a L_2 penalty has some non-zero elements.

In addition to the weight vector, the decision boundary is affected by λ and regularization method. In Fig. 3, the decision boundaries for various λ values are shown for same dataset with L_1 on the left and L_2 on the right. As λ increases, the boundary is placed in a position that will cause training errors ($\lambda > 1$). The boundaries from the two regularization strategies are slightly different, but not by very much. In Fig. 4, the same concept is shown across all four datasets, with the top row using L_1 and bottom row using L_2. Green, blue, and yellow lines are decision boundaries with low λ, and red magenta, cyan are high λ. Again, as λ increases, training error increases. The first dataset (left) is linearly separable, the middle two have some class overlap, and the right dataset is not even close. Accordingly, the decision boundary does not appear in the range of the data on the rightmost plot.

Finally, the test-set classification error is also compared across λ values and regularization methods in Fig. 5. When λ is very high for L_1 regularization (blue), test accuracy suffers, since model is too sparse. L_2 reg. has almost constant accuracy throughout, but there is some small variation. This is somewhat unexpected, because high regularization usually lowers model accuracy. It could be that the effect is not seen in the range of λ's shown.

C. Part 3

To pick the best regularizer and λ, the LR weights are trained with many values of λ for each regularizer, using the training set. Then, the generalization is evaluated on the validation set, by measuring the classification accuracy. A model that generalizes well will have high accuracy on data not seen during training. In cases where the validation accuracy is identical for multiple values of λ, the higher λ is chosen, because this corresponds to lower model complexity (more regularized). It is difficult to pick between L_1 and L_2 regularization because the boundaries aren't that different,

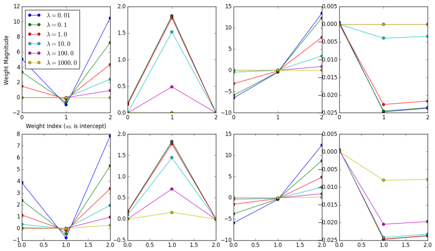


Fig. 2: Weight vectors of L_1 regularization (top row) and L_2 reg. (bottom row) for four datasets. Each column is the model for the same dataset. Within each subplot, several values of λ are shown. For high lambda, weights are smaller in magnitude for all plots. L_1 reg causes sparser weight vector (more zero elements) than L_2.

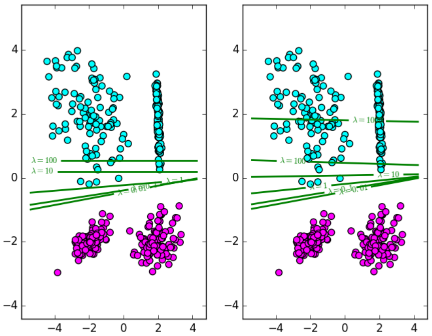


Fig. 3: LR on one dataset with L_1 reg on left, L_2 on right. Green lines show decision boundaries for various λs. As λ increases, training error increases.

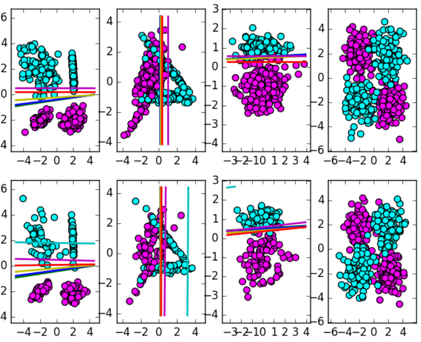


Fig. 4: LR on four datasets, with the top row using L_1 and bottom row using L_2. Green, blue, and yellow lines are decision boundaries with low λ, and red magenta, cyan are high λ. As λ increases, training error increases. The first dataset (left) is linearly separable, the middle two have some class overlap, and the right dataset is not even close. Accordingly, the decision boundary does not appear in the range of the data on the rightmost plot.

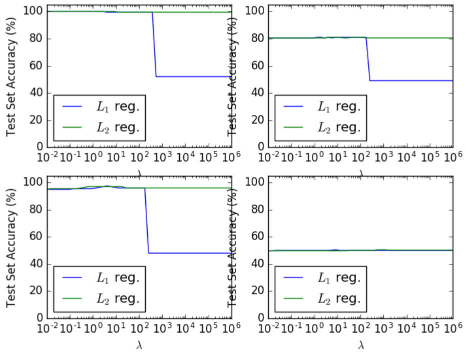


Fig. 5: For each dataset, the test accuracy is plotted across a wide range of λ values. When λ is very high for L_1 regularization (blue), test accuracy suffers, since model is too sparse. L_2 reg has almost constant accuracy throughout.

as discussed earlier. In case both validation accuracies are the same, L_1 may be slightly preferable to L_2 because of its sparsity.

Table II shows the accuracy of LR on the four datasets with optimized hyperparameters. Training accuracy is greater than or equal to test accuracy for all datasets, as expected. The accuracy numbers seem appropriate given the boundaries in Fig. 4: first and third columns are classified well by linear separator, second is decent, and fourth is not linearly separable in this feature space (the datasets are also described in later sections).

Dataset	Regularizer	λ	Train Acc. (%)	Val. Acc. (%)
1	L_2	0.01	100.0	100.0
2	L_2	0.01	83.0	80.5
3	L_1	1.78	97.5	96.5
4	L_2	1000	51.75	50.5

TABLE II: Accuracy of LR on four datasets.

II. Support Vector Machine (SVM)

In this section, we consider the Support Vector Machine (SVM) for binary classification of several 2D datasets.

A. Part 1

We implemented the dual form of a linear SVM with slack variables. I first converted the input data into a standard format, then we used the cvxopt Python package to execute the quadratic programming to find the desired α values.

The inputs to the cvxopt solver were:

- $P \in \Re^{nxn}$, where $P[i,j] = Y[i]Y[j]K(X[i], X[j])$
- $q = [-1, -1, ...] \in \Re^n$
- $G = [-I, I]^T \in \Re^{2nxn}$
- $h = [0, 0, ...; C, C, ...]$
- $A = Y^T$
- $b = 0$

With the simple 4 element dataset $\{((2,2),+1),((2,3),+1),((0,-1),-1),((-3,-2),-1)\}$, the algorithm outputs $(2,2)$ and $(0,-1)$ as the support

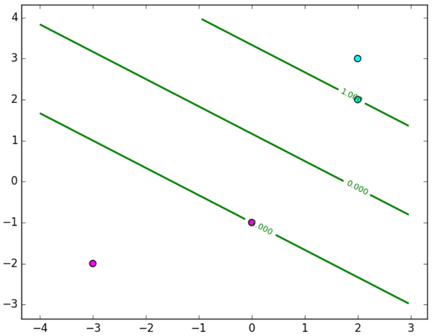


Fig. 6: Simple 4-element dataset (positive elements in cyan, negative in magenta) is separated by SVM. The three lines are the decision boundary ($\hat{y} = 0.0$) and positive and negative margin boundaries ($\hat{y} = \pm 1.0$) and two elements (support vectors) lie directly on boundaries of margin. Training error is zero.

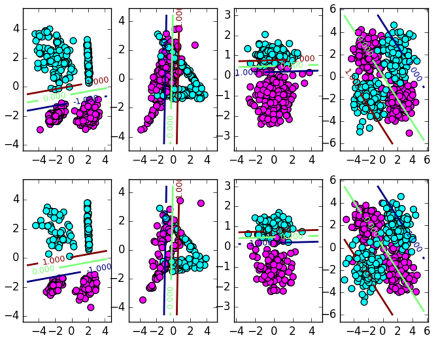


Fig. 7: Four 2D datasets (positive elements in cyan, negative in magenta) are separated by SVM. The top row is the four training datasets, and the bottom row is the four corresponding validation sets. The three lines are the decision boundary ($\hat{y} = 0.0$ in green) and positive and negative margin boundaries ($\hat{y} = \pm 1.0$ in red/blue).

vectors, meaning they lie on the boundaries of the margin, seen in Fig. 6. The weight vector has elements $w = [0.308, 0.462]$ and bias, $b = -0.538$. The training error is zero for this simple dataset. We use regularization term $C = 1$ for all results in this and the next subsection.

B. Part 2

Next, we used SVM to classify data from four much larger 2D datasets. The weights and bias of the decision boundary are generated using the training data, shown on the top row of Fig. 7, and then tested on the validation set shown in the bottom row of the same figure. Each column represents one dataset, presumably from the same distribution but with slightly different data. The leftmost dataset is easiest to linearly separate, whereas the middle two datasets have significant overlap between classes in this feature space. The rightmost dataset is not well-suited for a linear classifier because the data seems to have four distinct clusters at opposite ends of a rectangle.

The classification accuracy on the validation set matches the intuition from the plots. The first and third datasets are classified most accurately, while the fourth dataset is essentially equivalent to a random coinflip. For three of the four datasets, the training accuracy is at least as high as the validation accuracy. This is expected, because the model typically performs worse on data that was not seen during testing. In these cases, though, the accuracy difference is very minor.

Dataset	Training Accuracy (%)	Training Accuracy (%)
1	100.0	100.0
2	82.75	84.0
3	98.75	96.5
4	51.0	48.5

TABLE III: Accuracy of linear SVM on four datasets in Fig. 7.

C. Part 3

Next, we add an ability to handle kernel functions to the dual SVM implementation. Two kernels are considered: linear kernel ($K(x, x') = x^T \cdot x$) and gaussian RBF kernel ($K(x, x') = exp(-\gamma ||w - w'||^2)$. The regularization parameter, C, was set to 1 in previous sections, but here we are interested in how its value affects the results.

The decision boundaries with the two kernels on the four datasets are shown in Fig. 9. The left two datasets can be separated pretty well by a linear kernel (top row), and while the Gaussian RBF (bottom row) can be tuned to classify reasonably well, but not to an extent that is worth the extra computational cost. The rightmost dataset shows a clear example of a linear kernel failing where a gaussian RBF kernel shines. The data is not linearly separable in the original feature space, so the linear kernel's accuracy is about as bad as guessing. The Gaussian RBF kernel encloses the four clusters and classifies accurately. It could be argued that this decision boundary overfits the data, especially clear around the edges of the boundaries. The C parameter could be tuned to adjust this effect, but the computation time involved is significant. The bandwidth parameter, γ was adjusted for each dataset to get this performance (0.01 for the left two plots, 1.0 for the right two), and this parameter affects the spread of the kernel.

On the left of Fig. 8, the geometric margin ($1/||w||$) decreases as C increases in general. These results are shown for each of the four datasets, for the two kernel functions, so there are 8 curves. In one case, the margin increases after an increase in C. It makes sense that margin would generally decrease with an increase in C, because more penalty is being applied to incorrectly classified samples, and therefore $||w||$ will increase to overfit the dataset slightly more and reduce the number of incorrectly classified samples.

On the right of Fig. 8, a similar concept applies to the number of support vectors as C increases. Any sample with non-zero α_i is a support vector, including mis-classified samples, and with small C, the penalty of having many support

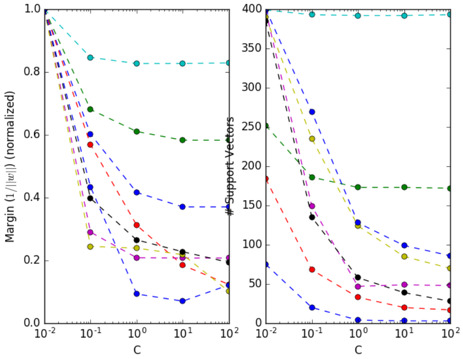


Fig. 8: For each of four datasets, and for each of two kernel functions, on the right, the margin is plotted against regularization paramter C. As C increases, generally margin decreases. The margins are normalized by their value when $C = 10^{-2}$. On the left, the number of support vectors decreases as C increases.

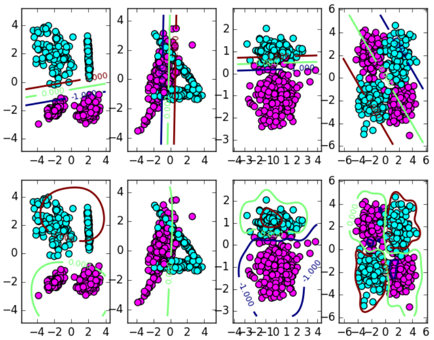


Fig. 9: Each column is one validation set, with the top row showing the SVM decision boundary using a linear kernel, and the bottom row using a Gaussian RBF kernel. The Gaussian RBF can generate curved decision boundaries which is necessary for the rightmost dataset. The left datasets can be accurately/simply classified with a linear kernel, which is also less computationally intensive.

vectors is low. As C increases, the margin shrinks and the number of support vectors decreases.

Maximizing the geometric margin on the training set is not an appropriate criterion for selecting C because that method will simply choose the weights corresponding to C=0. This eliminates any allowance of slack, where misclassified points can be ignored to improve the model's ability to generalize. An alternative method to choose C is to train many different SVM classifiers with different C values, and evaluate the classification accuracy on a validation set. Then, choose the value of C that corresponds to high accuracy.

III. Pegasos

In this section, we use the Pegasos algorithm for soft-margin SVM binary classification of one of the 2D datasets.

A. Part 1

The Pegasos algorithm is a method to solve soft-margin SVM. We implemented the algorithm according to the provided pseudocode, and added a bias term, w_0 that is incremented by $\eta_t y_i$ for misclassified sample x_i each iteration. The bias term is not penalized by the regularization. The formulation solved by Pegasos is:

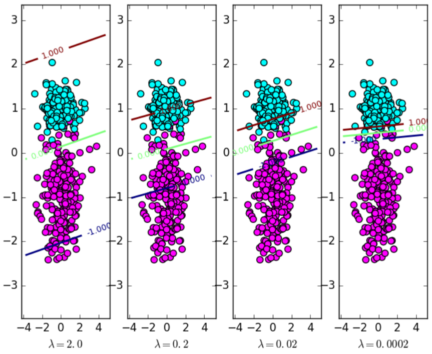


Fig. 10: The Pegasos algorithm is applied to the same dataset with four different regularization parameter values, λ. For large λ (left), magnitude of w is penalized heavily, so the decision boundary is not very accurate. As λ decreases, the regularization penalty becomes less dominant compared with accuracy, so the accuracy improves.

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

B. Part 2

The algorithm is applied to the same 2D dataset for several values of λ ($\lambda = [2, 2^{-1}, 2^{-2}, 2^{-4}$), shown in Fig. 10. For large λ (left), magnitude of w is penalized heavily, so the decision boundary is not very accurate. As λ decreases, the regularization penalty becomes less dominant compared with accuracy, so the accuracy improves. This intuition agrees with the objective function (2).

C. Part 3

Next, we extended our implementation to handle a kernel matrix input, and tested it with a gaussian RBF kernel as in Section II. After learning α, we predict the class of a new sample, x_i by first calculating $c = \alpha \cdot K(X, x_i; \gamma)$, where X is the entire training set and γ is the Gaussian kernel bandwidth. If $c > 0$, it's part of the positive class, otherwise it's the negative class.

D. Part 4

We tested our algorithm on multiple values of γ with a fixed $\lambda = 0.02$. For large γ (left), the kernel is very narrow, and the decision boundary overfits the data, evident in the decision boundary's jagged shape tracing around individual borderline samples. As γ decreases, the accuracy decreases, but the decision boundary is much less complicated. This usually leads to better generalization to unseen data.

The number of support vectors decreases as γ increases (192, 165, 10, 22). This observation aligns with the decreasing complexity of the decision boundary, because each support vector affects the shape of the decision boundary.

These observations associated with increasing γ match the trends seen in Section II with increasing C. Both

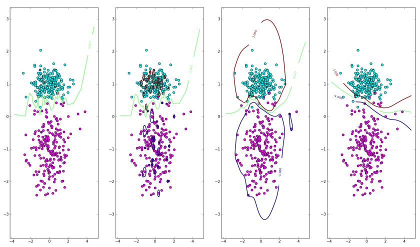


Fig. 11: The Pegasos algorithm is applied to the same dataset with four different kernel bandwidth values, $\gamma = [2^2, 2^1, 2^0, 2^{-1}]$. For large γ (left), the kernel is narrow, and the decision boundary overfits the data. As γ decreases, the accuracy decreases, but the decision boundary is much less complicated.

implementations (Pegasos and SVM) are able to correctly classify the dataset. Our Pegasos implementation runs much faster, making it easier to test with different parameters for increased performance.

IV. MNIST Dataset

In this section, we use each of the algorithms to classify handwritten digits from the MNIST dataset.

A. Part 1

The MNIST dataset contains labeled, handwritten digits. We split the dataset into multiple classification tasks, shown in the first column of Table IV. We also split it into training, validation, and testing sets.

We follow the typical procedure: train with many hyperparameters (C and λ for L_1-, L_2-regularized LR, and C for SVM), choose the hyperparameters that maximize performance on the validation set with lowest model complexity (low C), and report performance on the test set for the best hyperparameters.

The two classifiers here, Logistic Regression and Linear SVM, perform similarly on each classification task. For all tasks, the training accuracy was better than the testing accuracy, as expected.

Normalization of the data did not make a large difference ($< \pm 1\%$) for these classifiers, and results presented are for non-normalized data. The only significant difference after normalization was the size of regularization constant C; in all cases, normalization caused the optimal C value to increase by a few orders of magnitude. This is likely because the size of the data elements decreased, so the size of the learned weight vector increased, meaning a smaller regularization cost had to be applied for the same effect.

A couple misclassified digits are shown in Section IV-A. Some handwriting is very difficult even for humans to classify, so it makes sense that our learned classifiers are not perfect.

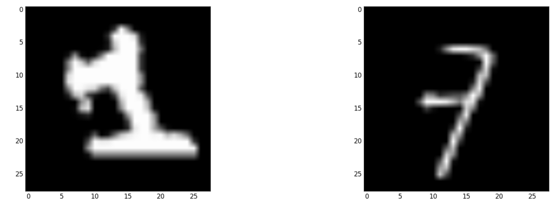
B. Part 2

Next, we applied the Gaussian RBF SVM classifier on the MNIST dataset for the same binary classification tasks. Again, there are two parameters, C (regularization) and γ (bandwidth) that must be tuned with the validation set. It is difficult to tune these two in parallel, especially without a method of visualizing the dataset, as was possible in the simple 2D data case. Our approach was to train of each parameter in the range $[10^{-5}, 10^{-4}, ..., 10^{5}]$, and then compare accuracy on the validation set. Many models had a validation accuracy of 99%, so for these, the least complex model (low γ, low C). Even so, it's very hard to select the best model because the relative importance of C and γ's size is not obvious (i.e. is a low γ and high C preferable to a high γ and low C?). The two are related to some extent; we observed roughly that an increase in order of magnitude of γ decreases the C for maximum validation accuracy, by an order of magnitude as well. Also, $\gamma > 1$ always has poor validation accuracy ($< 60\%$) regardless of C.

The choice of C, γ, and training and test accuracy are shown in **??**. Even with the poor choice of parameters in the stated range, validation accuracy was often around 70-80%, so the classifier would not be completely useless. For each classification task, the chosen hyperparameters are listed in **??**'s rightmost column.

The RBF classifier results are better than the LR and linear SVM from Table IV. This is expected because the RBF takes into account higher dimensional features, and the learning determines which of these features should affect classification. Simply learning weights based on pixel values is insufficient for high accuracy classification because there are relationships between pixels that help distinguish the digit.

According to our observations, normalization is important for the RBF kernel. The optimal hyperparameters shrink by several orders of magnitude when normalization is turned off, just as observed for linear classifiers. However test accuracy drops without normalization for the RBF SVM classifier (e.g. 92% for 1 vs. 7 without normalization, from 99% with normalization). It's possible our hyperparameter search was not broad/precise enough, but the procedure could easily be scaled up given more time/computational resources.



(a) Misclassified 1

(b) Misclassified 7

Fig. 12: The MNIST dataset has some ambiguous entries, in accordance with real human handwriting, that are difficult to classify correctly.

Dataset	LR Tr.	LR Test (C)	SVM Tr	SVM Test (C)
1 vs. 7	100.0	98.3 (1, L_1)	100.0	98.7 (0.2)
3 vs. 5	100.0	93.3 (60, L_1)	100.0	94.7 (0.02)
4 vs. 9	100.0	94.7 (2, L_1)	100.0	94.7 (0.02)
odds vs. evens	92.9	89.0 (0.1, L_2)	93.9	89.2 (0.02)

TABLE IV: Accuracy of LR and Linear SVM on MNIST datasets.