6.867 Homework 2

Logistic Regression

Unregularized Logistic Regression

We started by implementing logistic regression with stochastic gradient descent and plotting the norm of the weight vector over time for both the unregularized (λ = 0) and L2 regularized (λ=1) objective.

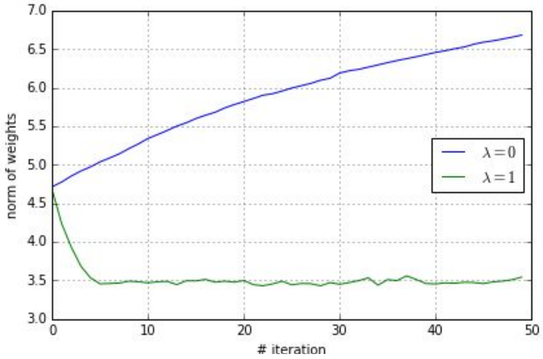


Figure 1. The effect of regularization on the weights.

As shown in Figure 1, the size of the unregularized weight vector increases indefinitely while the regularized weight vector converges. This can be explained by the fact that if we do not penalize the size of the weights, they will keep growing. This will move the output values closer to 1.0 or 0.0 for positive and negative samples respectively, producing a more confident prediction.

Note, however, that a more confident predictor is not necessarily better. In fact, in many cases such as medical applications, a model that is both confident and incorrect is worse than a model that is uncertain on some inputs.

Hyperparameters for Logistic Regression

We examined each of the four datasets with various regularizers and λ values using the sklearn implementation of logistic regression. In our code, we explicitly adjust the intercept scaling so as not to regularize the bias term.

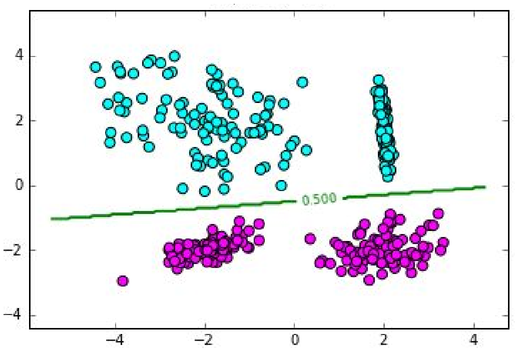


Figure 2. Decision boundary for L2, λ=0.0

The first dataset, shown in figure 2, is linearly separable and every combination of regularizer and λ between 0.0 and 20.0 produced a decision boundary which perfectly separated the two classes on the training, validation, and test set. As λ increases, the norm of the weight decreases.

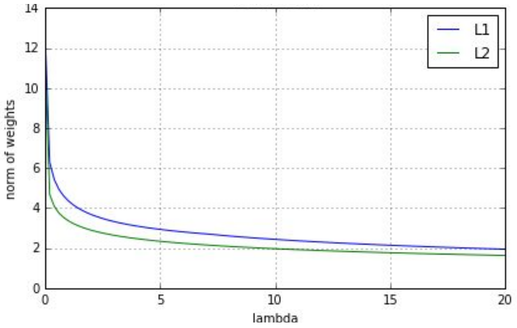


Figure 3. The size of the weight vector vs λ.

On this particular dataset, we can completely ignore the first dimension of the input vector and still separate the two classes. This is reflected by the fact that with a large regularization constant (λ = 20), the L1 regularizer produces the sparse weight vector [0.0, 1.999] with bias 0.596. While the L2 regularizer also tries to drive the weight vector to zero, it doesn't achieve sparsity and produces [-0.007, 1.6517] with bias 0.4257.

The second dataset, however, is not linearly separable and there is no linear decision boundary capable of perfectly discriminating between positive and negative samples.

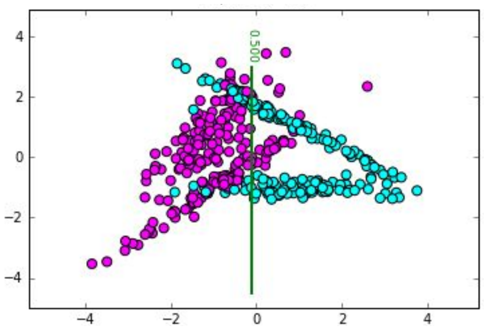


Figure 4. Decision boundary for L1, λ=1.

As we increased λ, we observed that the norm of the weights behaved exactly like it did for the first dataset. On the other hand, the training and validation accuracy changed significantly as we adjusted λ and the regularizer.

A subset of the regularizer and λ values we examined is shown below in figure 5, where the bolded entry indicates a pair of values which minimizes the validation error.

L	λ	Train Error	Validation Error	Test Error
-	0	0.1700	0.1750	0.1950
L1	**1**	**0.1675**	**0.1750**	**0.1950**
L2	1	0.1675	0.1750	0.1950
L1	2	0.1700	0.1800	0.1900

L2	2	0.1650	0.1750	0.1950

Figure 5. L1 with λ=1 is one possible choice.

The third dataset was nearly linearly separable. We determined that L2 regularization with a λ of 0.5 produced the smallest validation error.

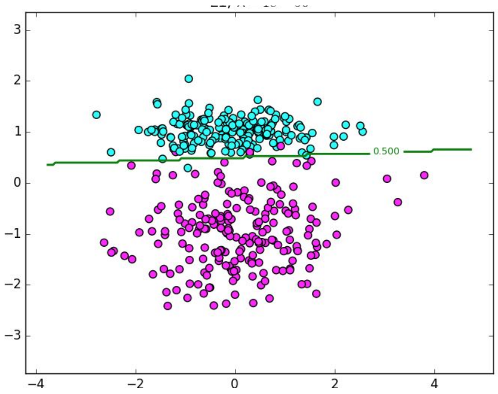


Figure 6. Decision boundary for L2, λ=0.5.

L	λ	Train Error	Validation Error	Test Error
-	0	0.0125	0.035	0.050
L1	0.5	0.0200	0.030	0.045
L2	**0.5**	**0.0125**	**0.025**	**0.400**
L1	1	0.0175	0.035	0.045
L2	1	0.0225	0.030	0.030

Figure 7. L2 with λ=0.5 is optimal.

The fourth dataset resembles the XOR problem and is not linearly separable. Any linear decision boundary will return the wrong answer for roughly half of the data points.

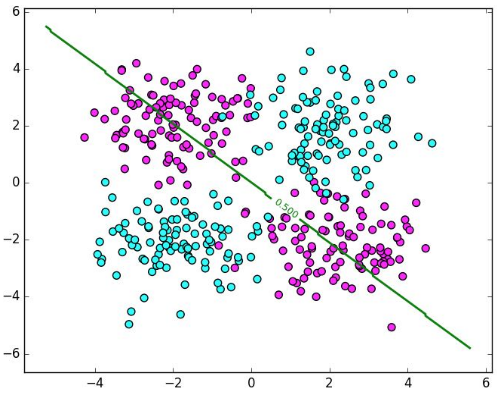


Figure 8. Decision boundary for L1, λ=1.

L	λ	Train Error	Validation Error	Test Error
-	0	0.4850	0.5075	0.5025
L1	**1**	**0.4825**	**0.5025**	**0.5000**
L2	1	0.4850	0.5075	0.5025
L1	3	0.4825	0.5050	0.5000
L2	3	0.4850	0.5075	0.5025

Figure 9. L1 with λ=1 is optimal.

Support Vector Machine

We tested our support vector machine on a toy dataset consisting of (2, 2), (2, 3), (0, -1), and (-3, -2) with two positive and two negative examples. Our implementation produced the below objective function and constraints.

$$\min_\alpha \frac{1}{2} \begin{bmatrix} \alpha_0 \\ \alpha_1 \\ \alpha_2 \\ \alpha_3 \end{bmatrix}^T \begin{bmatrix} 8 & 0.1 & 2.0 & 0.1 \\ 0.1 & 0.13 & 3.0 & 0.12 \\ 2.0 & 3.0 & 1.0 & 2.0 \\ 0.1 & 0.12 & 2.0 & 0.13 \end{bmatrix} \begin{bmatrix} \alpha_0 \\ \alpha_1 \\ \alpha_2 \\ \alpha_3 \end{bmatrix} - \begin{bmatrix} \alpha_0 \\ \alpha_1 \\ \alpha_2 \\ \alpha_3 \end{bmatrix}$$

$$s.t.\ 0 < \alpha_i < C,\ \sum_i alpha_i = 0$$

By solving this quadratic programming problem, we get the following decision boundary which only depends on two support vectors, (2,2) and (0, -1).

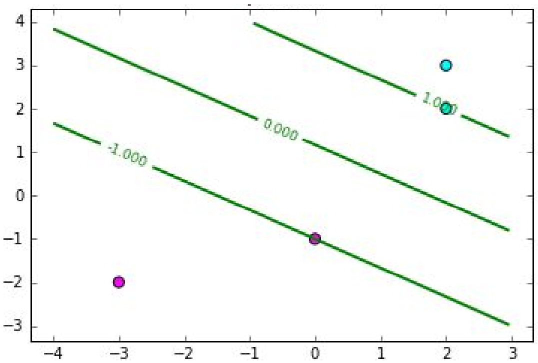


Figure 10. Decision boundary for C=1.0.

Dataset 1

For the first dataset, we noted the various C values in the set {0.01, 0.1, 1, 10, 100} did not have a significant impact on the linear decision boundary; in fact, just like when applying logistic regression, every C value produced a boundary which perfectly separated the two classes.

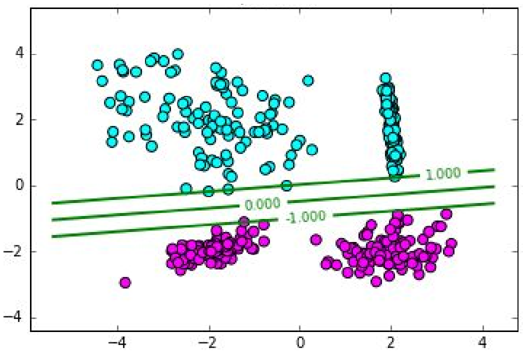


Figure 11. C = 1.0, 0.0 train, 0.0 validation, 0.0 test

When using the radial basis function kernel, we were able to achieve a reasonable decision boundary for all values of C in the above set. By decreasing γ, we were able to produce decision boundaries that ranged from a nearly straight line (on the local scale) to a circular boundary which completely encircled one of the classes.

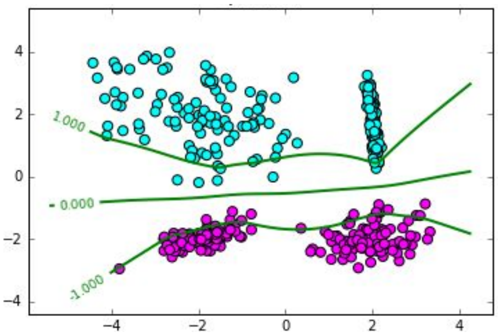


Figure 12. C = 10.0, γ = 0.1

We observed that for this particular dataset, increasing C led to a decrease in both the geometric margin and number of support vectors. This can be explained by the fact when we increase C, the SVM penalizes mistakes more heavily, resulting in larger weights. We further analyze this interaction in the hyperparameters section.

Dataset 2

We started by establishing a baseline for the second dataset using the linear SVM, achieving error rates of 0.165 on the validation set and 0.19 on the test set.

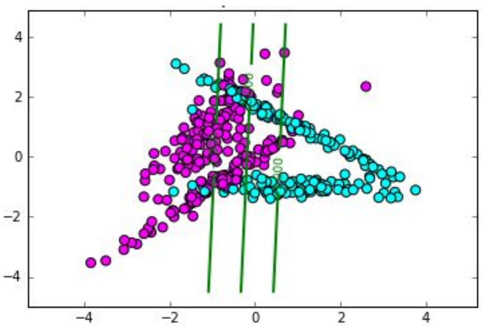


Figure 13. C = 1.0, 0.17 train, 0.165 validation, 0.19 test

For small values of C such as C = 0.01, we observed that the decision boundary was nearly linear. As we increased the C value to C = 10.0 and set γ to 10.0, we observed a decision boundary with validation error 0.099 and test error 0.055, significantly outperforming the linear SVM.

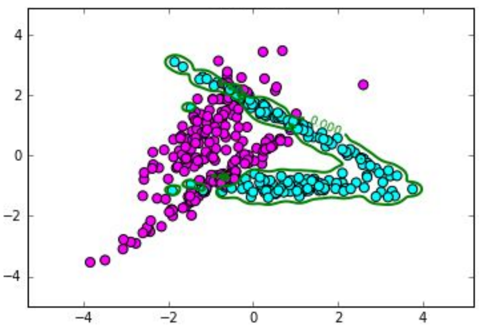


Figure 14. C = 10.0, γ = 10.0, 0.099 validation, 0.055 test

Dataset 3

When applying our SVM to the third dataset, we discovered that for certain combinations of C and bandwidth values, the Gaussian kernel was less accurate than the linear kernel. A visual inspection of the dataset gives some intuition about why this is the case: the classes appear to be naturally separated by a linear boundary.

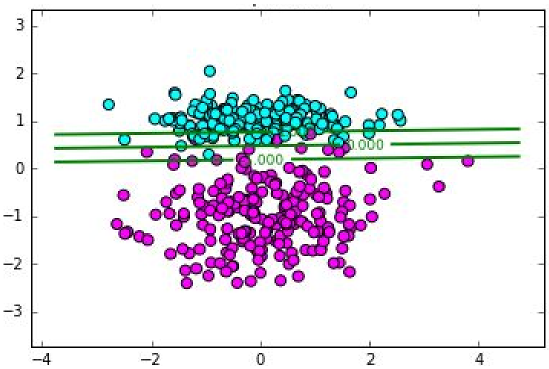


Figure 15. C = 1.0, 0.012 train, 0.035 validation, 0.05 test

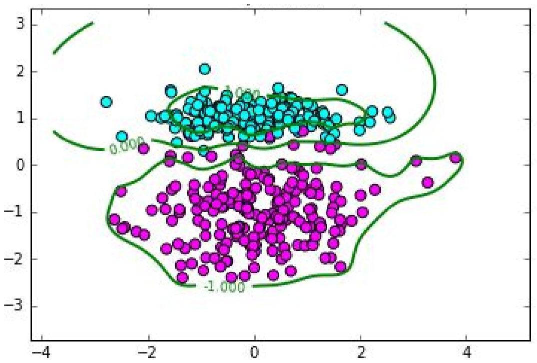


Figure 16. C=0.01, γ=1.0, 0.045 validation, 0.030 test

Dataset 4

Unlike the third dataset, the fourth dataset requires a nonlinear kernel to produce a reasonable classifier. The linear support vector machine produces a model that is no better than to random guessing due to the linearly inseparable nature of the dataset.

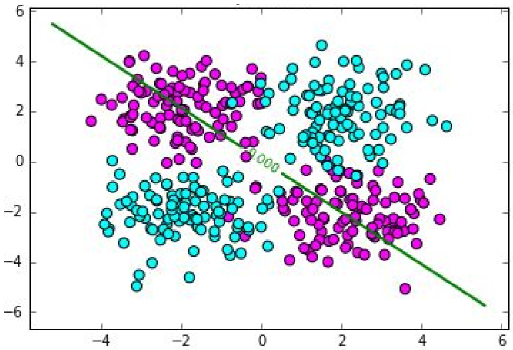


Figure 17. C = 1.0, 0.489 train, 0.515 validation, 0.51 test

By introducing the Gaussian RBF kernel, the model improves dramatically, correctly classifying over 95% of the test data.

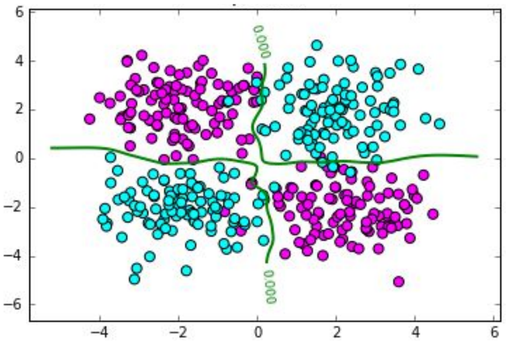


Figure 18. C=1.0, γ=1.0, 0.05 validation, 0.05 test

Hyperparameters

Based on our observations, we believe that in most cases, as C increases, the number of support vectors increases while the geometric margin decreases. This is not true in all cases, but in most situation, heavily penalizing the slack variables will result in a larger weight vector and therefore produce a smaller geometric margin.

In addition, we conclude that maximizing the geometric margin is not an appropriate method for selecting C as it is easy to overfit to the training data, especially when using a RBF kernel. It is always better to use a validation set to tune hyperparameters; if the dataset is too small to be split into training, testing, and validation sets, leave-one-out cross-validation is another good alternative.

Pegasos

To try and get better performance on large datasets, we implemented the Pegasos algorithm for training support vector machines. Unlike our quadratic programming SVM implementation, the Pegasos implementation does not use C to penalize slack variables; instead, this implementation uses the λ regularization constant which is inversely proportional to C.

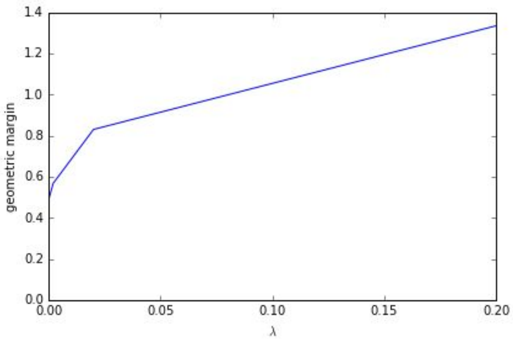


Figure 19. As λ increases, so does the margin.

We observed that as we increased the regularization constant, our model produced smaller weights which resulted in a larger geometric margin. This is consistent with both our understanding of the objective function and our assertion that λ and C are inversely proportional.

Kernels

We extended our Pegasos implementation to accept kernel functions using the below objective function. Unfortunately, this formulation does not have the same sparsity properties as the dual SVM.

$$\min_w \frac{\lambda}{2}\|w\|^2 + \frac{1}{n}\sum_i max\{0, 1 - y_i(w^T\phi(x_i))\}$$

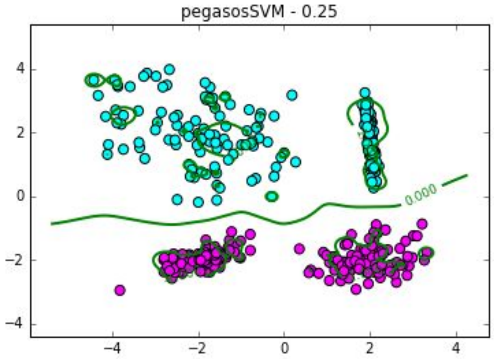
Although this formulation appears to require explicitly computing the feature vector mapping using φ, the training algorithm only requires the kernel operator. Based on this formulation of the objective function, we should use the following prediction rule.

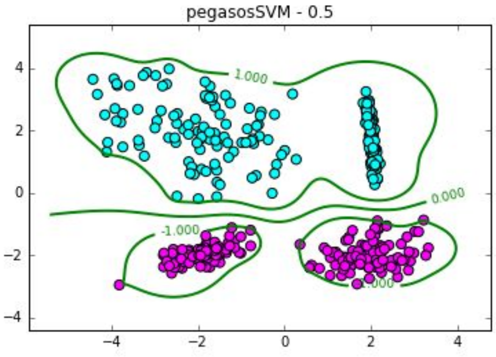
$$sign(\sum_i \alpha_i K(x, x_i))$$

Bandwidth

The relationship between the bandwidth of the Gaussian kernel (γ) and the resulting decision boundary is consistent between the QP and Pegasos algorithms. Smaller bandwidths produced uneven boundaries while larger bandwidths produced smooth boundaries.

As we increased the bandwidth from 2^-2 to 2^2, we observed that the number of support vectors initially decreases before increasing after after γ=1.





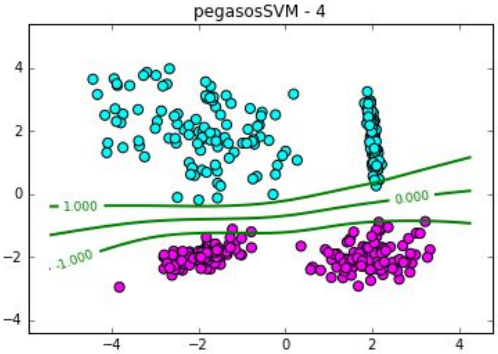


Figure 20. Decision boundaries for various γ values.

MNIST

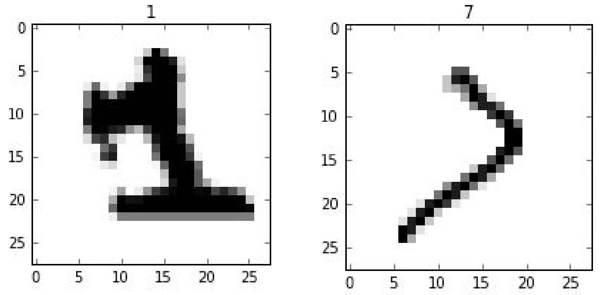
Linear Models

Next, we used logistic regression and linear support vector machine implementations to try and solve four binary classification tasks on the MNIST data. We used C=1.0 for both the logistic regression and support vector machine.

Logistic Regression	Train	Validation	Test
1 vs 7	1.0	0.989	0.989
1 vs 7 (normalized)	1.0	0.989	0.987
3 vs 5	1.0	0.939	0.930
3 vs 5 (normalized)	1.0	0.946	0.946
4 vs 9	1.0	0.959	0.943
4 vs 9 (normalized)	1.0	0.970	0.943
even vs odd	1.0	0.848	0.855
even vs odd (normalized)	0.962	0.883	0.887

Support Vector Machine	Train	Validation	Test
1 vs 7	1.0	0.986	0.989
1 vs 7 (normalized)	1.0	0.989	0.987
3 vs 5	1.0	0.939	0.946
3 vs 5 (normalized)	1.0	0.940	0.947
4 vs 9	1.0	0.943	0.943
4 vs 9 (normalized)	1.0	0.943	0.943
even vs odd	1.0	0.837	0.851
even vs odd (normalized)	0.979	0.869	0.867

In general, it appears that normalization - rescaling the data to [-1.0, 1.0] - improves accuracy on the test data. Some of the misclassified digits are shown below.



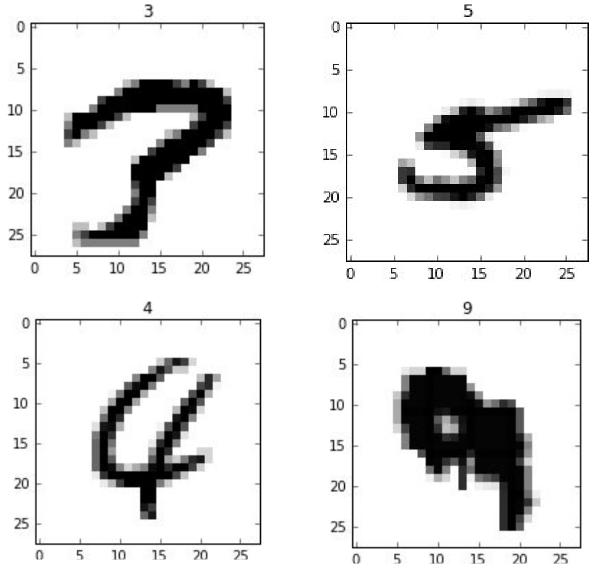


Figure 21. Misclassified digits.

Many of these mistakes seem very reasonable, especially the last row where the 4 and 9 are almost unrecognizable.

Kernels

When we attempted to fit a support vector machine with a Gaussian kernel to the dataset, we discovered that without normalization, it is extremely difficult to find C and γ values which would produce an accurate model. After normalizing the data, it became much easier to find C and γ values which could match and in some cases exceed the accuracy of the linear models.

RBF SVM	C	γ	Train	Validation	Test
1 vs 7	2.7	0.016	1.0	0.989	0.989
3 vs 5	0.2	1.667	1.0	0.947	0.960
4 vs 9	0.4	1.111	1.0	0.963	0.949
even vs odd	.03	2.667	1.0	0.957	0.967

The accuracy contours for each of these pairwise comparisons is shown below.

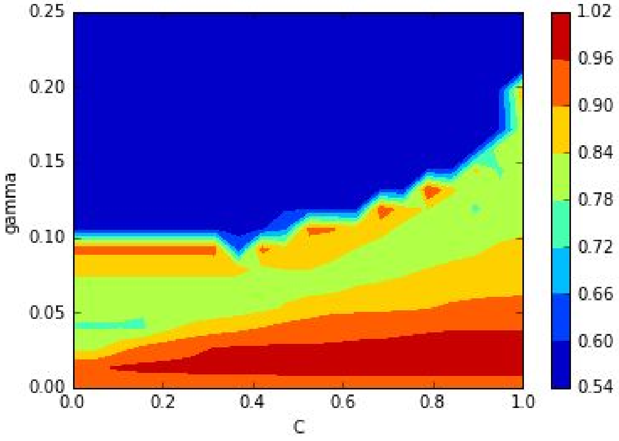


Figure 22. Accuracy contours for 1 vs 7.

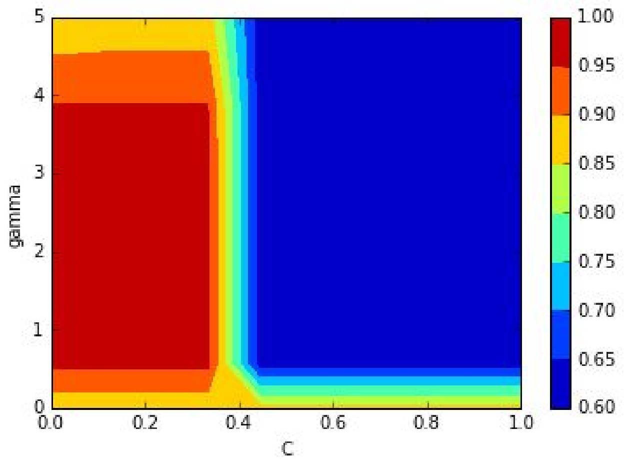


Figure 23. Accuracy contours for 3 vs 5.

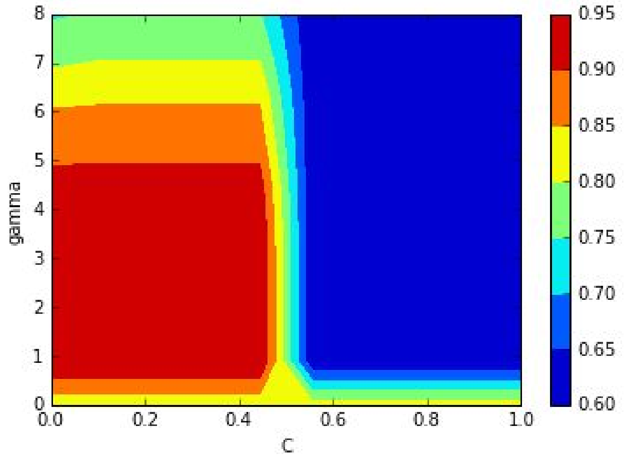


Figure 24. Accuracy contours for 4 vs 9.

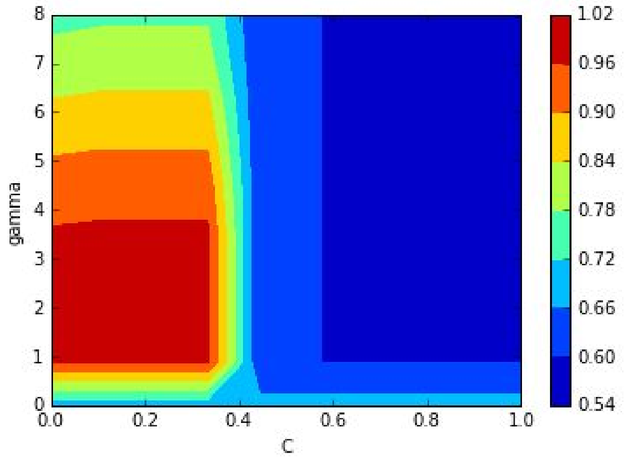


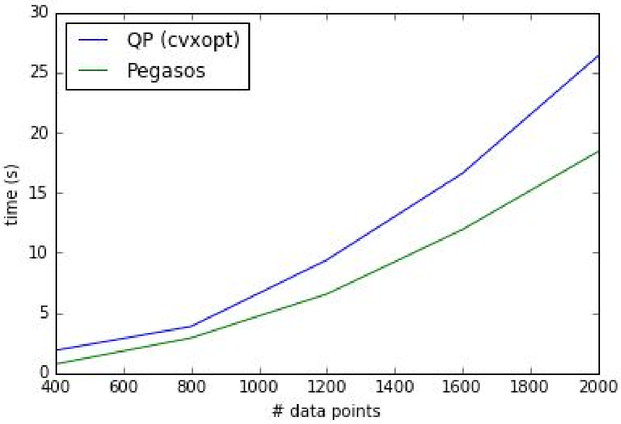
Figure 25. Accuracy contours for even vs odd.

Interestingly, the last three accuracy contours share a similar structure. It appears that as C increases beyond approximately C=0.5, it becomes less beneficial. Meanwhile, as we increase γ from 0 to 10, accuracy increases until approximately γ=3.

Performance

Next, we compared our quadratic programming (QP) and Pegasos implementation of support vector machines by evaluating the speed on different sized datasets for various γ (1/C) values.

We experimentally determined that a max epoch of 20 was sufficient for our Pegasos implementation to converge on the optimal hyperplane for our dataset and γ values. The resulting decision boundary matched or in a few cases exceeded the accuracy of the QP implementation.



As shown in the above figure, the amount of time needed to converge increases with the number of datapoints for both the quadratic programming and the Pegasos algorithm. However, as the number of data points increases, our Pegasos implementation consistently outperforms the quadratic programming algorithm.

Furthermore, we observed that as we increased C (and therefore decreased the γ value), the time required for the quadratic programming implementation to solve the optimization problem tended to decrease. On the other hand, the time required by the Pegasos algorithm was not significantly affected.