COMP 540 HW 4

Peiguang Wang, Xinran Zhou

Due: 3/5/2018

1: Intuitions about support vector machines

1. We typically frame an SVM problem as trying to maximize the margin. Explain intuitively why a bigger margin will result in a model that will generalize better, or perform better in practice.

   Solution. Because with larger margin, we have more confidence to our result. That is, we have more confidence to classify a given sample to its class. The points that are most difficult to classify, which lies close to the decision boundary, can be classify using the boundary or hyperplane found by maximum margin. Also, we can reduce overfitting with bigger margin.

2. Will moving points which are not support vectors further away from the decision boundary effect the SVMs hinge loss?

   Solution. hinge loss function

   $$hingeloss = max(0, 1 - y^{(i)}(\theta^T x^{(i)} + \theta_0))$$

   No. For those points that lie on both sides, the right hand of the max function is already negative. And that will stay negative as moving them further away from the decision boundary. So the hinge loss will always be zero. The loss function and decision boundary are only determined by the support vectors.

2: Fitting an SVM classifier by hand

1. Write down a vector that is parallel to the optimal vector θ.

   Solution. After mapping the points to the 3-D using feature vector $\phi(x)$, new data $D^{'}$ is

   $$D^{'} = \{([1, 0, 0], -1), ([1, 2, 2], +1)\}$$

   since there are only two points, the vector θ is parallel to the direction of two connected points, so

   $$\theta = (0, 2, 2)$$

2. What is the value of the margin that is achieved by this θ?

   Solution. Because we only two sample points, so the margin achieved by this θ is equal to the distance of two points. That is $2\sqrt{2}$

3. Solve for the .

   Solution. From the previous part, we know that the margin is equal to $2\sqrt{2}$. Therefore,

   $$\frac{2}{||\theta||} = 2\sqrt{2}$$

   subject to $\theta = k(0, 2, 2)$ so we can solve the equation and get $\theta = (0, \frac{1}{2}, \frac{1}{2})$

4. Solve for the intercept θ_0 using your value for θ and the inequalities above.

Solution. We already know that for dataset $D^{'}$.

$$y^{(1)}(\theta^T x(1) + \theta_0) \geq 1$$

$$y^{(2)}(\theta^T x(2) + \theta_0) \geq 1$$

using the θ that we got from the previous part, we can finally get

$$-1 \leq \theta_0 \leq -1$$

so $\theta_0 = -1$

5. Write down the equation for the decision boundary in terms of θ, θ_0 and x.

   Solution. The decision boundary should be $\theta^T X + \theta_0 = 0$ where $X = \Phi(x) = (1, \sqrt{2}x, x^2)$. So the decision boundary is

$$\frac{\sqrt{2}}{2}x + \frac{1}{2}x^2 - 1 = 0$$

Problem 3: Support vector machines for binary classification - Part 1

1. The hinge loss function and gradient.

   Solution. The implementation of the hinge loss cost function and its gradient for support vector machines.

   The cost J and the gradient of the loss function with respect to an all-zeros vector are computed.

```
J =  1.0  grad =  [-0.12956186 -0.00167647]
```

Figure 1: the result of the hinge loss and gradient

2. Example dataset 1: impact of varying C.

   Solution. The C parameter is a positive value that controls the penalty for misclassified training examples. Large C will tell the SVM model try to classify all the samples correctly. Here I change the value of C on the decision boundary to see the different decision boundary.

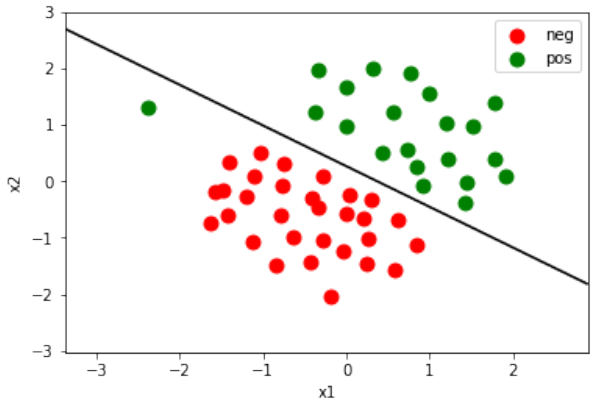

Figure 2: SVM decision boundary with C = 1 for Example dataset 1

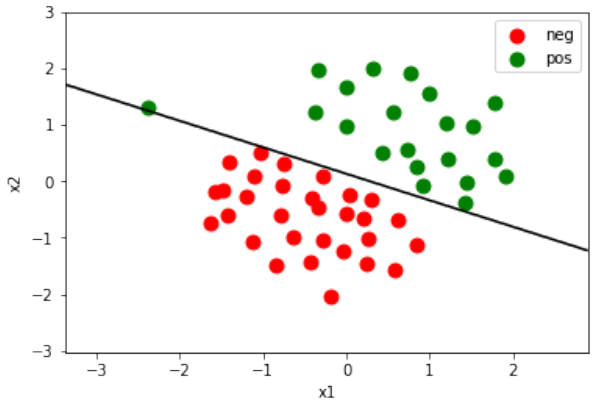

Figure 3: SVM decision boundary with C = 100 for Example dataset 1

When C = 1, we can see that the boundary misclass an example. It's because of the weak regulation. As C increase, the penalty for misclassification will be larger, resulting less misclassified points.

3. Gaussian kernel

Solution. Implement the Gaussian kernel function and test our implementation.

```
##########################################################################
#  Part  3: Training SVM with a kernel                                   #
#  We train an SVM with an RBF kernel on the data set and the plot the   #
#  learned decision boundary                                             #
##########################################################################

# test your Gaussian kernel implementation

x1 = np.array([1,2,1])
x2 = np.array([0,4,-1])
sigma = 2

print "Guassian kernel value (should be around 0.324652) = ", utils.gaussian_kernel(x1,x2,sigma)
```

```
Guassian kernel value (should be around 0.324652) =  0.32465246735834974
```

Figure 4: the Guassian kernel value

4. Example dataset 2: learning non-linear boundaries.

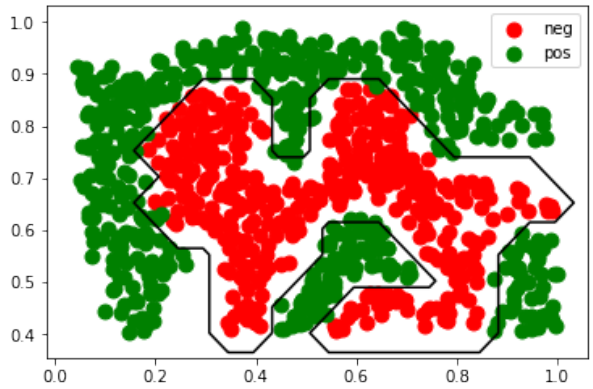

Figure 5: SVM gaussian kernel decision boundary for Example dataset 2, C = 1 and σ = 0:02

5. Example dataset 3: selecting hyper parameters for SVMs.

Solution. Selecting the best C and σ parameter using cross-validation. The best C and σ is shown in figure 6

```
print best_C, best_sigma
0.3 0.1
```

Figure 6: selection the best C and σ

The corresponding decision boundary given the best C and σ is shown in figure 7

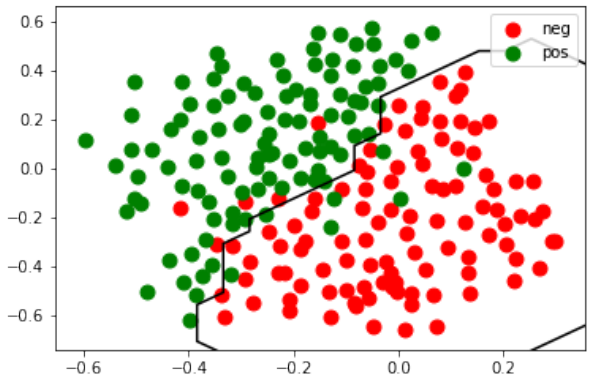

Figure 7: SVM Gaussian kernel decision boundary for Example dataset 3 with the best hyper parameters.

Problem 3: Support vector machines for binary classification - Part 2 Spam Classification

Because the data X we got is one-hot encoded, which consist only 0 and 1. So we think there is no need to use Gaussian kernel. And X is near 0, so there is no need to scale either. Using cross-validation to select the hyperparameters, the best accuracy is reached by setting C = 1, learning rate = 0.3, and iteration number is 10000. Here are some results, more detailed results can be found in the .HTML file.

```
C: 0.1  learning rate: 0.01  val acc: 0.96375
C: 0.1  learning rate: 0.03  val acc: 0.96625
C: 0.1  learning rate: 0.1  val acc: 0.965
C: 0.1  learning rate: 0.3  val acc: 0.96625
C: 0.3  learning rate: 0.01  val acc: 0.97125
C: 0.3  learning rate: 0.03  val acc: 0.975
C: 0.3  learning rate: 0.1  val acc: 0.97375
C: 0.3  learning rate: 0.3  val acc: 0.9725
C: 1  learning rate: 0.01  val acc: 0.9825
C: 1  learning rate: 0.03  val acc: 0.9825
C: 1  learning rate: 0.1  val acc: 0.9825
C: 1  learning rate: 0.3  val acc: 0.97375
C: 3  learning rate: 0.01  val acc: 0.97375
C: 3  learning rate: 0.03  val acc: 0.97625
C: 3  learning rate: 0.1  val acc: 0.9775
C: 3  learning rate: 0.3  val acc: 0.9775
C: 10  learning rate: 0.01  val acc: 0.975
C: 10  learning rate: 0.03  val acc: 0.975
C: 10  learning rate: 0.1  val acc: 0.9775
C: 10  learning rate: 0.3  val acc: 0.98
C: 30  learning rate: 0.01  val acc: 0.9725
C: 30  learning rate: 0.03  val acc: 0.9725
C: 30  learning rate: 0.1  val acc: 0.98
C: 30  learning rate: 0.3  val acc: 0.9775
Best C, learning rate and accuracy 1 0.01 0.9825
```

Figure 8: selecting the best hyperparameters

```
################################################################################
# YOUR CODE HERE for testing your best model's performance                     #
# what is the accuracy of your best model on the test set? On the training set?  #
################################################################################
yy_test = np.ones(y_test.shape)
yy_test[y_test == 0]=-1
svm.train(X,yy,learning_rate = 0.3, reg = 1, batch_size = 500, num_iters = 10000)
y_pred = svm.predict(X_test)
print "Accuracy", np.sum(y_pred == yy_test)/float(len(y_pred))
```

```
Accuracy 0.989
```

Figure 9: the test accuracy using the best hyperparameter

In order to speed up the process of selecting best parameters, we set the batch size to a smaller number. Using the best parameters we found, we achieve a test accuracy over 98.5

```
################################################################################
# YOUR CODE HERE for testing your best model's performance                     #
# what is the accuracy of your best model on the test set? On the training set?  #
################################################################################
yy_test = np.ones(y_test.shape)
yy_test[y_test == 0]=-1
svm.train(X,yy,learning_rate = 0.3, reg = 1, batch_size = 500, num_iters = 10000)
y_pred = svm.predict(X_test)
print "Accuracy", np.sum(y_pred == yy_test)/float(len(y_pred))
```

```
Accuracy 0.989
```

Figure 10: the test accuracy using the best hyperparameter

```
 1191 our 0.5216618252354417
298 click 0.5103177587835251
1398 remov 0.44735554582585224
739 guarante 0.4300091878079063
156 basenumb 0.40868365513504595
1796 visit 0.3958199383610757
477 dollar 0.3366393236589053
1852 will 0.302937418862586
966 lo 0.2984218930045666
1299 price 0.29197185890924676
1089 nbsp 0.284525720672859
1264 pleas 0.27521472322388 83
792 hour 0.27497394100869976
391 da 0.2569661420136832
1067 most 0.2545557806067744
```

Figure 11: the 15 words most indicative of spam

```
 1561 spamassassin -0.62142180892488
1881 wrote -0.4387318695676192
1437 rpm -0.41682263967213384
401 date -0.4081938444932535
1666 the -0.40789523927155724
1765 url -0.4057233039621416
961 list -0.40391242167243324
1773 user -0.3380736451539691
1362 razor -0.33234089360985186
1759 until -0.3299785573230686
1153 numbertnumb -0.3245584519799971
135 author -0.3234526627098289
1886 yahoo -0.29002709219161477
1444 said -0.2759951954690061
314 comment -0.2725033526749356
```

Figure 12: the 15 words most indicative of ham

Part 4: Support vector machines for multi-class classification

1. It is possible that once in a while a dimension in the gradient check will not match exactly. What could such a discrepancy be caused by? Is it a reason for concern?

   Solution. The discrepancy is caused by the non-differentiable part of the loss function. The loss function is non-differentiable in when $\theta^{(j)^T} x(i) - \theta^{y_i^T} x(i) + \Delta = 0$.
   It is not a reason for concern. The loss will not increase if gradient is computed this way. So it is not a reason for concern.

2. Loss history figure.

Solution. The loss changes with iteration times. The loss history plot is shown in Figure 13.

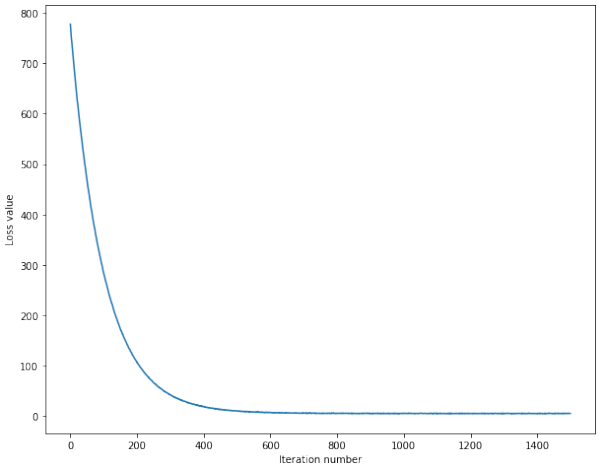

Figure 13: Loss History

3. Search for the best svm model.

Solution. Experiment with 4 learning rates and 4 regularization strengths. The result are:
lr 1.000000e-08 reg 1.000000e+04 train accuracy: 0.230592 val accuracy: 0.249000
lr 1.000000e-08 reg 5.000000e+04 train accuracy: 0.264000 val accuracy: 0.258000
lr 1.000000e-08 reg 1.000000e+05 train accuracy: 0.301898 val accuracy: 0.321000
lr 1.000000e-08 reg 5.000000e+05 train accuracy: 0.323796 val accuracy: 0.340000
lr 5.000000e-08 reg 1.000000e+04 train accuracy: 0.315531 val accuracy: 0.323000
lr 5.000000e-08 reg 5.000000e+04 train accuracy: 0.373837 val accuracy: 0.392000
lr 5.000000e-08 reg 1.000000e+05 train accuracy: 0.360265 val accuracy: 0.374000
lr 5.000000e-08 reg 5.000000e+05 train accuracy: 0.313184 val accuracy: 0.331000
lr 1.000000e-07 reg 1.000000e+04 train accuracy: 0.372735 val accuracy: 0.376000
lr 1.000000e-07 reg 5.000000e+04 train accuracy: 0.368714 val accuracy: 0.376000
lr 1.000000e-07 reg 1.000000e+05 train accuracy: 0.354429 val accuracy: 0.357000
lr 1.000000e-07 reg 5.000000e+05 train accuracy: 0.317000 val accuracy: 0.332000
lr 5.000000e-07 reg 1.000000e+04 train accuracy: 0.369204 val accuracy: 0.369000
lr 5.000000e-07 reg 5.000000e+04 train accuracy: 0.339816 val accuracy: 0.336000
lr 5.000000e-07 reg 1.000000e+05 train accuracy: 0.291224 val accuracy: 0.292000
lr 5.000000e-07 reg 5.000000e+05 train accuracy: 0.280633 val accuracy: 0.289000
lr 1.000000e-06 reg 1.000000e+04 train accuracy: 0.362796 val accuracy: 0.375000
lr 1.000000e-06 reg 5.000000e+04 train accuracy: 0.254204 val accuracy: 0.266000
lr 1.000000e-06 reg 1.000000e+05 train accuracy: 0.265082 val accuracy: 0.277000
lr 1.000000e-06 reg 5.000000e+05 train accuracy: 0.227735 val accuracy: 0.232000
best validation accuracy achieved during cross-validation: 0.392000

Visualized the results and the results are shown in Figure 14.

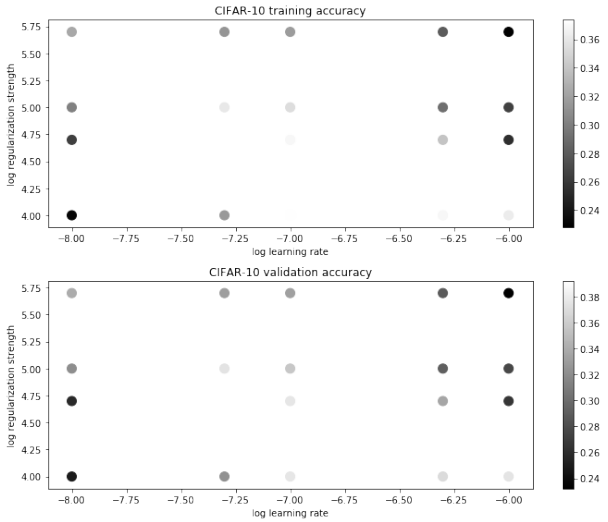

Figure 14: Visualized Results for Parameter Selection

4. Evaluate the test set accuracy on the best svm learned.

   Solution. After searching for the best values of learning rate and regularization. We obtain the best svm model. The results on the test set is: *linear SVM on raw pixels final test set accuracy: 0.368300.* The accuracy is similar to the accuracy of validation set. Visualization of the results is shown in Figure 15.

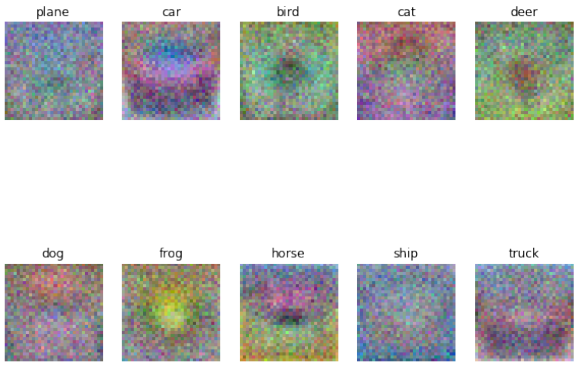

Figure 15: Visualized Results for Best SVM Multiclass Model

5. Comparing the performance of multi-class SVM and softmax regression. Which approach takes longer to train, which approach achieves higher performance? Compare the visualizations of the θ parameters learned by both methods - do you see any differences? Comment on hyper parameter selection for both methods.

   Solution. First compare the accuracy.

   - Softmax on raw pixels final test set accuracy: 0.405100
   - Linear SVM on raw pixels final test set accuracy: 0.368300.

   The accuracy on Softmax and SVM are almost the same level. Softmax is slightly higher than linear SVM method.

Then compare the visualization of these two methods.
Figure 16 shows the results on softmax model.

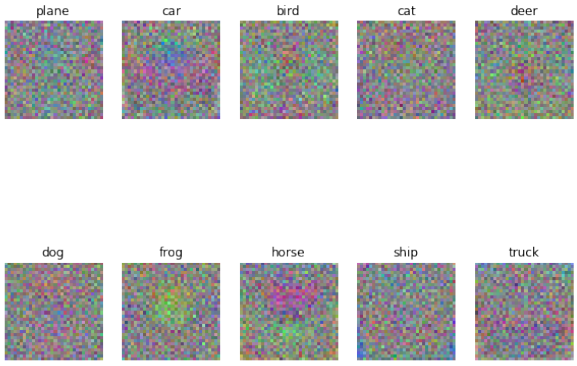

Figure 16: Visualized Results for Best Softmax Model

Figure 17 shows the results on SVM multiclass model.

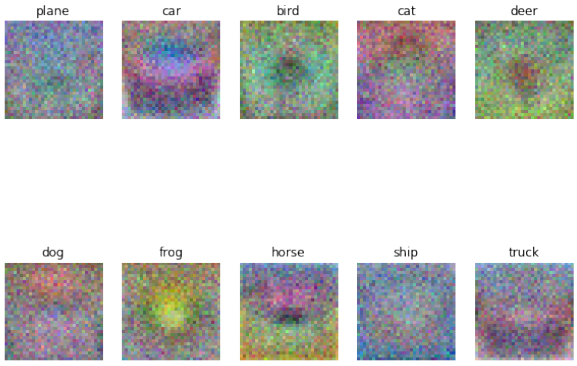

Figure 17: Visualized Results for Best SVM Multiclass Model

From the figure it seems the SVM model extracts a more "meaningful" features. Here "meaningful" means understandable for humans. The contour and edges are more clear in SVM model results.

Compare the time of these two methods.

- Softmax vectorized loss: 2.352202e+00 computed in 0.483000s
- Linear SVM Vectorized loss: 9.293820e+00 computed in 0.016000s

In terms of time cost. Softmax takes a much longer time to train. The reason is that SVM learns a sparse kernel.