Homework 9
Accomplished by Zhiping Xiu and Xinzuo Wang

Please note that the two .p files (classifier_2.p and classifier_1.p) we submitted have different training data size. This is because that the .p file of KNN is too large (more than 1GB). To make is available for submitting, we used a smaller data size.

**1):**

**A):** There are several cases that may bring challenges:

1: Different people may have different writing habits. Thus images of one digit may have significant difference. For example, the following image are two '4's which have large differences;

2: Ambiguous digit may also affect classification results, following images show a '9', but it looks very similar to '4'; Also the right image is labeled as '6', but it is very similar to '1'.

3: Different size, shape, and positions of one digit. According to [1], shallow structures such as kernel machines may hard to recognize the same digits at different positions or with different sizes as the same thing. This may occur especially when we do not have enough training samples to cover all sizes and positions.

**B):**

There are 60000 images, The number of each images are: 1: 6742; 2: 5958; 3: 6131; 4: 5842; 5: 5421; 6: 5918; 7: 6265; 8: 5851; 9: 5949; 0: 5923

When adjusting parameters, we chose 6000 images as the data set for 5-fold cross validation. By looking at MNIST data set, we realized that the data set has been shuffled, which means randomness is already satisfied. Thus the 6000 are images are the first 6000 from MNIST data set. Choosing 6000 images make it efficient when adjusting parameters. And a 5-fold cross validation help us evaluating model performance comprehensively.

When comparing algorithm performance, we use all the images, i.e. 60000 images, to do two 10-fold cross validation. Using all the images and cross validation ensures a comprehensive comparison of the two methods.

**2):**

**A):**

**SVM(support vector machine):**

SVM finds hyperplanes by the margin maximization criteria. SVM will mark the margin using data points lying on it, which is called support vector. The training phase is basically the process of finding those support vector. The input in this homework is a set of 784-dimension array. Each array is formed by flattening a 28 * 28 digit image. The output of a SVM is its parameters that define the hyperplane.

After training phase, identified support vectors and only support vectors will be used to classify incoming new data points, and the classification process is similar with KNN (except we only use support vectors at this time).

**KNN(k-nearest neighbors):**

KNN is an instance based lazy learning method used for classification and regression. When using k-NN for image classification, the output is a class membership, i.e. digit numbers from 0 to 9. For KNN algorithm, the input in this homework is a set of 784-dimension array. Each array is formed by flattening a 28*28 digit image. An image array is classified by a majority vote of its neighbors, with the image being assigned to the class most common among its k nearest neighbors (k is a positive integer, typically small). If k = 1, then the image array is simply assigned to the class of that single nearest neighbor.

**B):**

1: We trained our SVM with Radial Basis Function (RBF) kernel, thus it has two parameters to consider C and gamma:

a) **C** is the trades off misclassification of training examples against simplicity of the decision surface. Smaller c makes the decision surface smooth, high c trying to capture more training examples correctly.

b) **Gamma** defines how much defines how much a single example has, the larger the gamma, the closer other examples must be in order to be affected by this example.

2: In KNN, the only parameter we have to adjust is **k**, the number of nearest neighbors. When k is small, the model only considers those few points that are closest to query points. Thus it tends to generate a high curved classification boundary, which may result in overfitting problem. While when k is large, the classification boundary tends to be smoother. But also, a overly large k may bring underfitting problem.

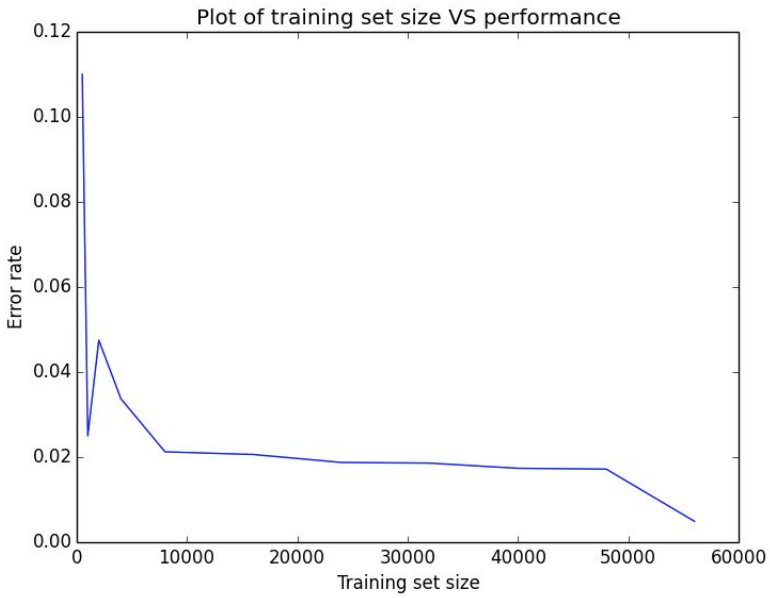
**3):**

**A):** Please see the code in classifier_1.py and classifier_1.p

**B):**

We first identified the best hyper parameters combo for gamma and C (please see the definitions of C and gamma in question 2. Which according to the following graph, based on cross validation result of 6000 sample images, we reached the best accuracy when C = 100.0 and gamma = 0.01.
We then do the rest of the experiment based on this hyper parameters (shown on 3-B-2).

1: Training set size vs performance on a testing set

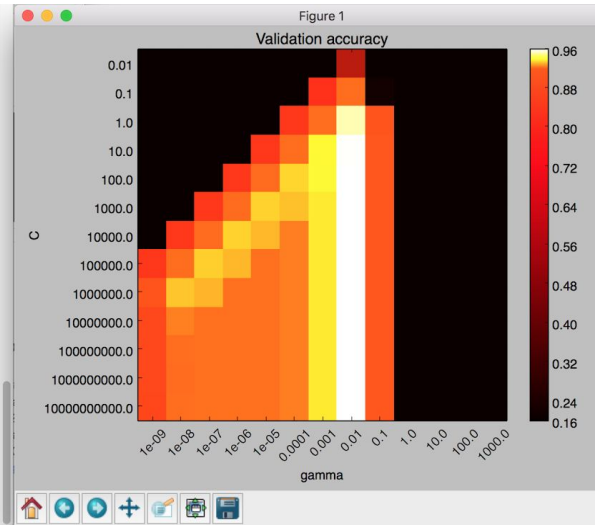



When the training set is small, the result is fluctuate and not very good, with the increase of the training set, the error rate showing a decreasing tendency.

2: Classifier parameters training: reference: [RBF SVM parameters]

```
    gc.collect()
KeyboardInterrupt
Error in sys.exitfunc:
^CTraceback (most recent call last):
  File "/System/Library/Frameworks/Python.framework/Versions/2.7/lib/python2.7/atexit.py", line 24, in _run_exitfuncs
    func(*targs, **kargs)

➜ Zhiping_Xiu_hw9 git:(master) ✗ open .
➜ Zhiping_Xiu_hw9 git:(master) ✗ cd ..
➜ HW atom .
➜ HW cd Zhiping_Xiu_hw9
➜ Zhiping_Xiu_hw9 git:(master) ✗ ls
README.md               cvHelper.py             mnist.py
boosting.py             eecs349-fall16-hw9.html mnist.pyc
classifier_1.p          eecs349-fall16-hw9.ipynb train-images-idx3-ubyte
classifier_1.py         helper.py               train-labels-idx1-ubyte
classifier_1.pyc        helper.pyc              training_labels.p
classifier_2.py         lecun-98.pdf            training_set.p
➜ Zhiping_Xiu_hw9 git:(master) ✗ python classifier_1.py
The best parameters are {'C': 10.0, 'gamma': 0.01} with a score of 0.82
/System/Library/Frameworks/Python.framework/Versions/2.7/Extras/lib/python/matplotlib/collections.py:548: FutureWarning: elementwise comparison failed; returning scalar instead, but in the future will perform elementwise comparison
  if self._edgecolors == 'face':
➜ Zhiping_Xiu_hw9 git:(master) ✗ python classifier_1.py
The best parameters are {'C': 100.0, 'gamma': 0.01} with a score of 0.96
/System/Library/Frameworks/Python.framework/Versions/2.7/Extras/lib/python/matplotlib/collections.py:548: FutureWarning: elementwise comparison failed; returning scalar instead, but in the future will perform elementwise comparison
  if self._edgecolors == 'face':
```




The best error rate  is obtained when C = 100.0 and gamma = 0.01. It seems the result is mostly governed by gamma, as we can see, when gamma == 0.01 and c > 1.0, all the result seems not bad. Combined with the definition of c and gamma, we may assume that the overfitting is not very much a big problem.

3:

```
Confusion matrix:
[[543   0   2   0   0   1   1   1   0   1]
 [  0 651   3   1   1   1   0   0   0   0]
 [  5   1 614   4   2   1   0   4   1   0]
 [  0   0   3 607   0   1   0   1   1   0]
 [  0   2   0   0 567   1   0   2   0   4]
 [  2   0   2   4   0 541   3   0   0   5]
 [  2   0   1   0   0   2 621   0   1   0]
 [  0   1   4   2   2   0   0 596   1   2]
 [  0   2   1   4   2   1   2   0 616   3]
 [  1   0   0   1   4   3   0   7   3 531]]
error rate 0.018833
```

From the previous problems we see that when size of training set is larger than 20000,  C = 100.0 and gamma = 0.01, the model will get good results. So we tested performance of the model by using 54000 instances as training, set C = 100.0 and gamma = 0.01 as parameters. We choose the other 6000 data points as test set. The confusion matrix is shown in the above figure.

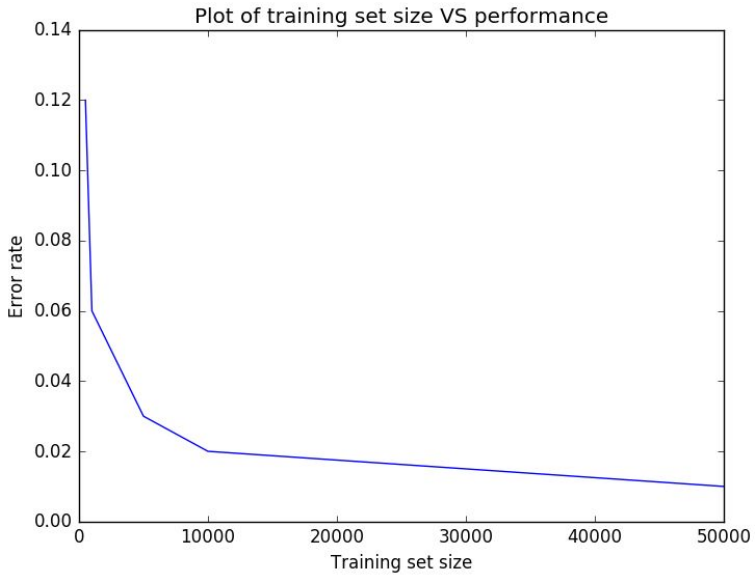
**4):**

**A):** Please see the code in classifier_2.py and classifier_2.p

**B):**

1: Training set size vs performance on a testing set

To test the relationship between performance and training set size, we select 500, 1000, 5000, 10000, 50000 data points as training set. And use a data set formed by 100 the other data points as test set. The relationship between performance and training set size is shown as follows.



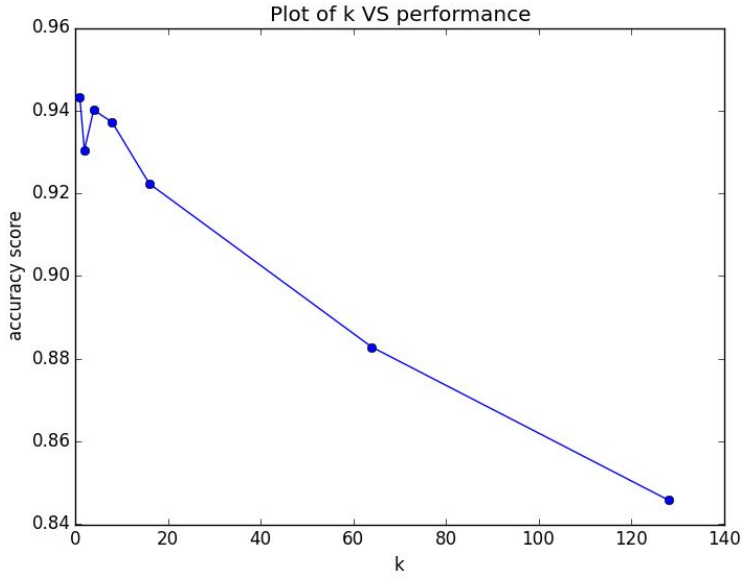

2: Classifier parameters vs performance on a testing set:

To test the relationship between performance and the parameter k, we set k to 1, 2, 4, 8, 16, 64, 128 and test KNN on a 6000-image data set by doing a 5-fold cross validation for each parameter respectively. Their relationship is shown as follows:

The best parameters is  k = 1 with a score of 0.94.

The mean accuracy scores under these parameters are: 0.94333333, 0.9305, 0.94016667, 0.93716667, 0.92233333, 0.88283333, 0.84583333.

The following picture shows the relationship between accuracy and k:




The best parameter is when k = 1. We used 54000 instances as training points and the other 6000 points as test set. The reason why k = 1 is the best parameter may because that the training set is large enough. Because only in this case

3: confusion matrix for the data. (KNN)

```
Confusion matrix:
[[546   0   1   0   0   1   1   0   0   0]
 [  0 653   2   0   1   0   0   1   0   0]
 [  6   5 609   1   2   0   0   8   1   0]
 [  0   3   3 594   0   7   0   1   2   3]
 [  0   5   0   0 560   0   2   1   0   8]
 [  3   2   2   4   0 538   6   0   0   2]
 [  4   2   0   0   0   2 619   0   0   0]
 [  0   3   0   1   1   0   0 597   1   5]
 [  1   9   2   9   2   7   1   1 593   6]
 [  1   1   1   3   9   2   0   6   2 525]]
error rate 0.027667
```

**5):**

1: For SVM, we used 6000 images as training set and tested on 1000 testing sample, we list our misclassified image in our homework folder, ~/classifier_1/misclassified_image. Now we see some examples here:

Misclassified as '0':

Misclassified as '1':

Misclassified as '2':

Misclassified as '3':

None

Misclassified as '4':

Misclassified as '5':

Misclassified as '6':

Misclassified as '7':

Misclassified as '8':

Misclassified as '9':

2: For KNN, we used 6000 images as training set and tested 1000 images based on the trained model. The following are some selected examples that are misclassified.

Misclassified as ‘0’ (true label: ‘2’, ‘8’, ‘9’):

Misclassified as ‘1’ (true label: ‘4’):

Misclassified as ‘3’ (true label: ‘5’, ‘3’, ‘5’):

Misclassified as ‘5’ (true label: ‘8’, ‘9’):

Misclassified as ‘6’ (true label: ‘5’):

Misclassified as ‘7’ (true label: ‘9’):

Misclassified as ‘8’ (true label: ‘1’):

Misclassified as ‘9’ (true label: ‘4’)

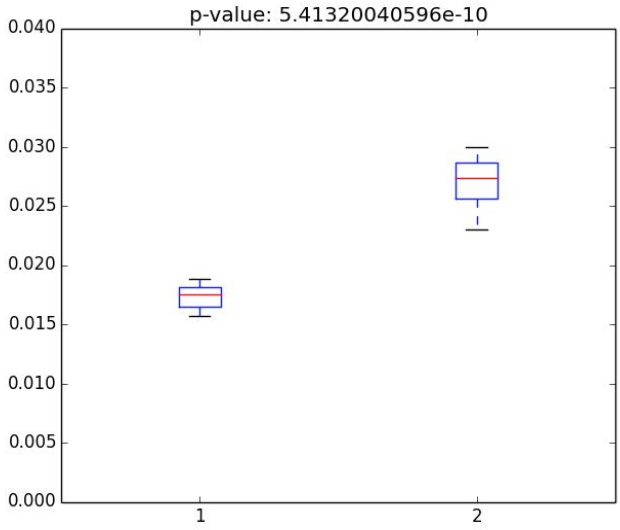
Some new understanding of the classifier and data:

1. Both KNN and SVM have limited power of recognizing ‘shape’ of images. they do classification only by using features generated by the flattened gray images. Thus it cannot identify images well if the images are skewed, at different sizes or positions. For example, it is relatively easy for human to recognize the ‘1’ misclassified as ‘8’. For our learner (KNN or SVM), however, as the ‘1’ is bolder, it contains those pixels that are usually ‘white’ in ‘8’. Thus our learner may be more likely to identify it as a ‘8’, which fails to consider its shape. This shows that our learner has limited power to identify an image’s shape.
2. Better features may help us to get a better classification. At here we only used the flattened gray images as features, thus may result in a poor classification result because we did not abstract important information from images, i.e. shape information. If using more sophisticated features, we may get a better result.

**6):**

We demonstrate this result from a 10-fold cross validation on 60000 images. The result are shown as follow, the p-value is 5.41e-10 < 0.05, thus we are confident about that SVM outperform KNN on given dataset.

The test is processed by cvHelper.py, you can perform the cross validation again with python cvHelper.py 0. (uncomment cv(10, int(sys.argv[1]), '.') in '__main__' area first, the experiment could take time).




SVM vs KNN, box-1 is the results (error rate) of SVM from 10-fold cross validation on 60000 images, box-2 is from KNN.

**7):**

**A):**

Testing adaboost algorithm is very time consuming, so we tested it on smaller data set. The adaboost algorithm perform less than KNN and SVM. The algorithm is code up in boosting.py.

```
Confusion matrix:
[[ 6  0  1  0  1  2  0  0  0  2]
 [ 0 15  1  0  0  0  0  0  0  0]
 [ 0  0  8  0  0  0  0  0  2  0]
 [ 1  0  0  4  0  1  0  1  0  0]
 [ 0  0  1  0  6  1  1  0  0  2]
 [ 0  0  0  0  0  7  0  0  0  0]
 [ 0  0  0  0  0  1  4  0  0  0]
 [ 0  0  0  0  2  1  0  6  0  4]
 [ 0  0  0  1  0  1  0  0  7  2]
 [ 0  0  1  0  0  0  0  0  0  7]]
error rate 0.300000
```

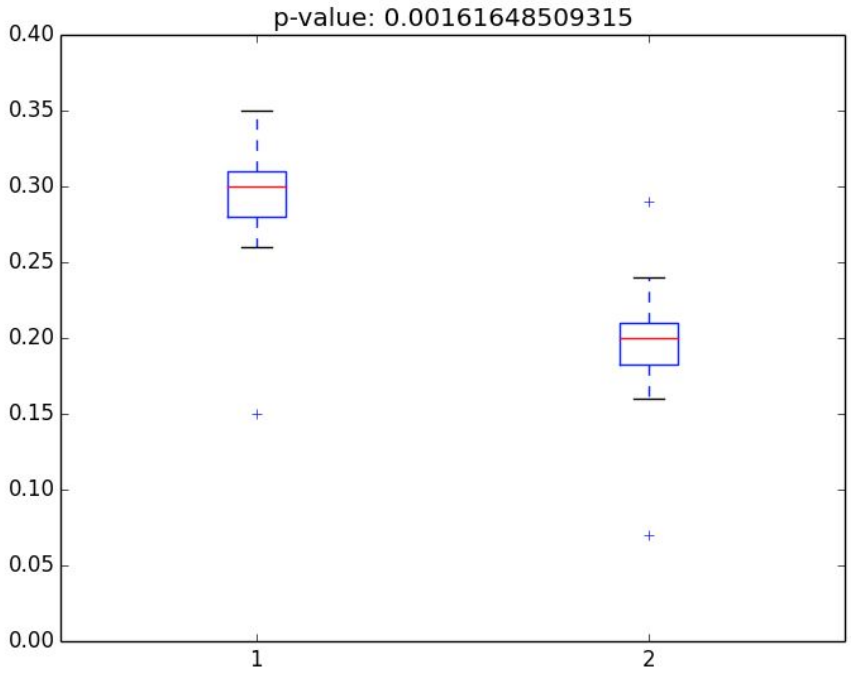
**B):**

The result of adaboost using SVM as base learner is slightly better than previous one, but still not as good as KNN or single SVM. The idea is that boosting is better for improve a weak learner, not very well on improving an already sophisticated algorithm like SVM.

```
Confusion matrix:
[[ 9  0  0  0  0  0  0  0  0  1]
 [ 0  9  2  0  0  0  0  0  0  0]
 [ 0  0 11  0  0  0  0  0  2  0]
 [ 0  0  1 10  0  0  0  0  0  0]
 [ 0  0  0  0  2  0  0  0  0  2]
 [ 0  0  0  0  0 10  0  0  0  0]
 [ 0  0  1  0  0  0  5  0  2  0]
 [ 0  1  0  0  0  0  0  5  0  5]
 [ 0  0  0  1  1  1  0  0  7  0]
 [ 0  0  1  0  0  0  0  0  0 11]]
error rate 0.210000
```

**C):**

We did 10 experiments, trained on 1000 images and tested on 100 images, The result are shown as follow image: we can see that B is better, the adaboost with SVM as base learner outperform the adaboost with decision tree as base leaner.




box 1 is the results using decision tree as base learner, box 2 is using SVM as base learner, we can see the adaboost using SVM as base learner showing better result, with p-value = 0.0016 < 0.05.

Reference:
[1] Bengio, Yoshua, and Yann LeCun. "Scaling learning algorithms towards AI." Large-scale kernel machines 34.5 (2007).