hw1

February 3, 2015

1 Homework 1

- Name: Austin
- SID: 23826762
- Repro: Open up hw1.ipynb in IPython Notebook.

```
In [1]: %matplotlib inline
        import matplotlib.pyplot as plt
        import scipy.io
        import numpy
        from sklearn import svm

        # from http://stackoverflow.com/q/4601373/1222351
        def permute_both(a, b):
            p = numpy.random.permutation(len(a))
            return a[p], b[p]
```

```
In [2]: # Load the data
        mat = scipy.io.loadmat('data/digit-dataset/train.mat')
        images = numpy.reshape(mat["train_images"], (1, -1, 60000))[0].T
        labels = mat["train_labels"]
```

```
In [3]: # Shuffle the data in parallel
        images, labels = permute_both(images, labels)
        labels = numpy.ravel(labels)
```

1.1 Problems 1 & 2

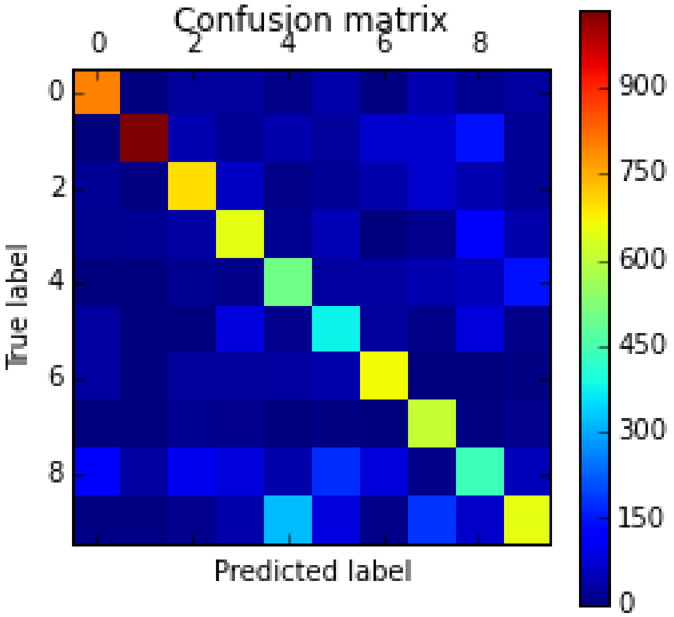
The accuracies approach 80% as the number of training examples increase.

From the confusion matrix, it appears that 5s and 8s are the most commonly misidentified digits, often labeled as the reverse. Meanwhile, 0s and 1s are very accurately identified.

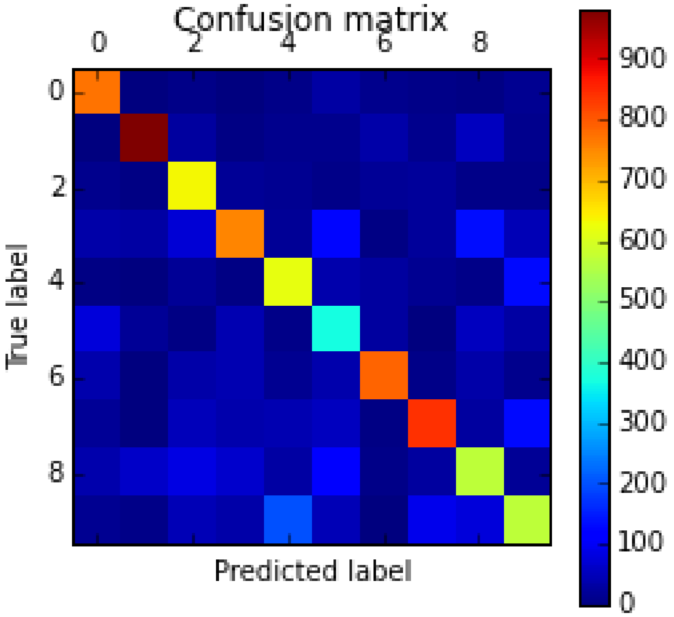
```
In [4]: i = 0
        x, y = [100, 200, 500, 1000, 2000, 5000, 10000], []
        for size in x:
            s = svm.LinearSVC()
            s.fit(images[i:i + size], labels[i: i + size])
            score = s.score(images[-10000:], labels[-10000:])
            y.append(score)
            print(size, " - ", score)
            i += size + 1

            # Make a confusion matrix
            from sklearn.metrics import confusion_matrix
```

```
cm = confusion_matrix(s.predict(images[-10000:]), labels[-10000:])
plt.matshow(cm)
plt.title('Confusion matrix')
plt.colorbar()
plt.ylabel('True label')
plt.xlabel('Predicted label')
plt.show()
```

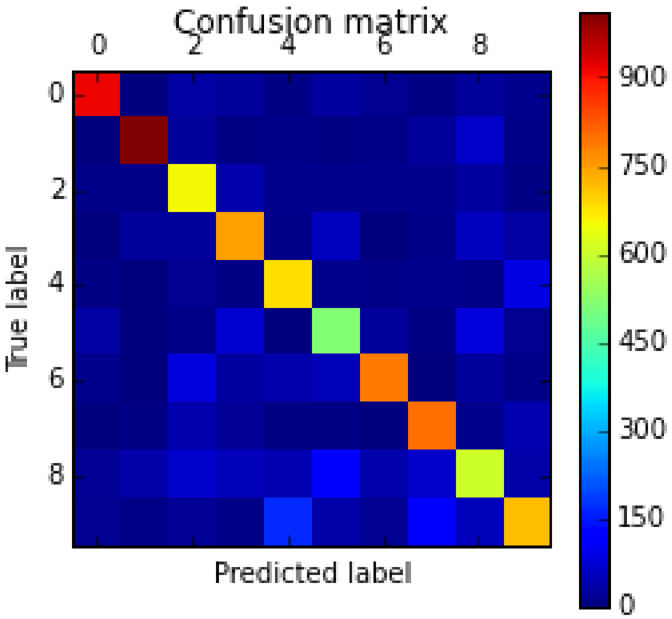
```
100  -  0.6429
```



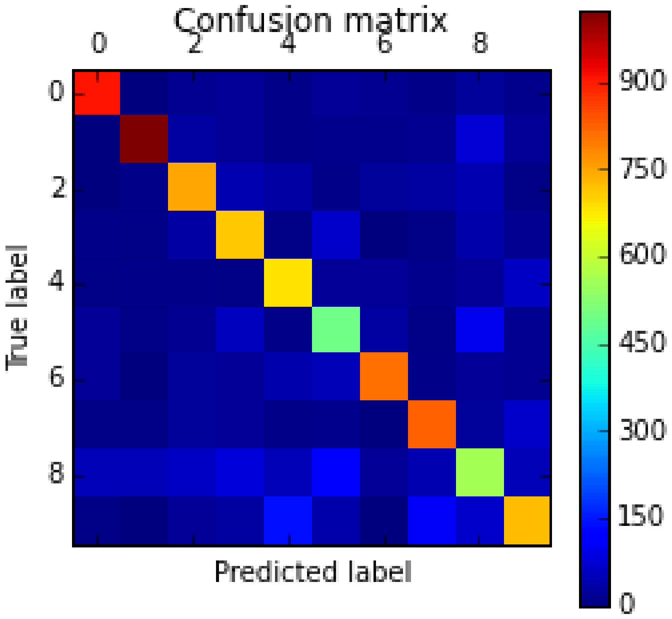
```
200  -  0.6908
```



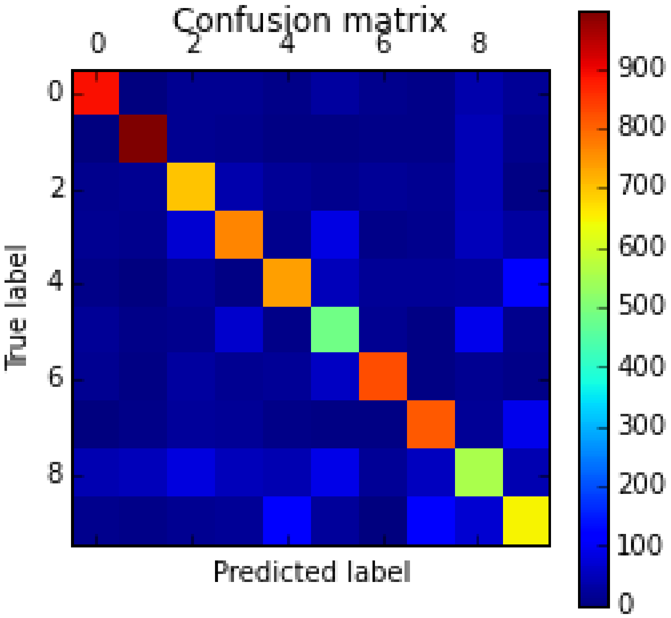
```
500  -  0.7459
```



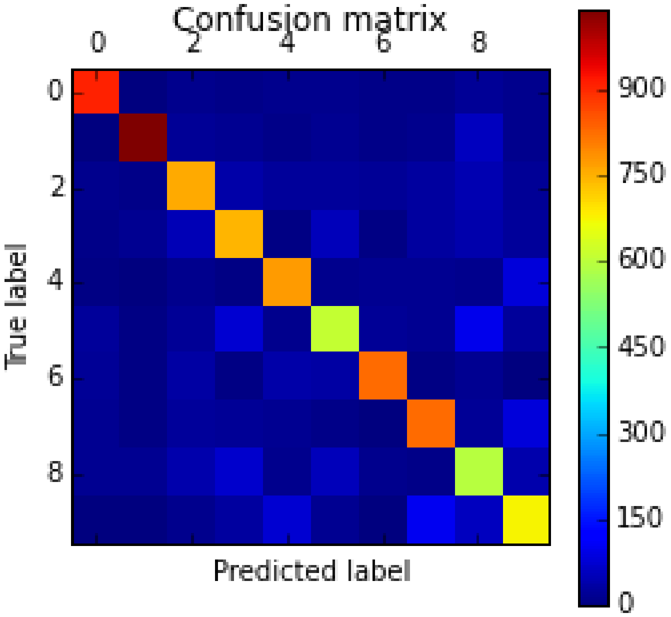
```
1000  -  0.7503
```



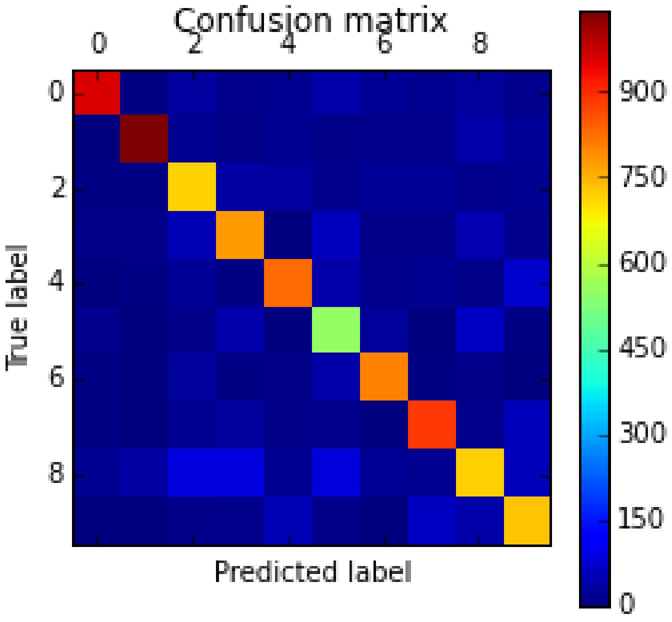
```
2000  -  0.7418
```



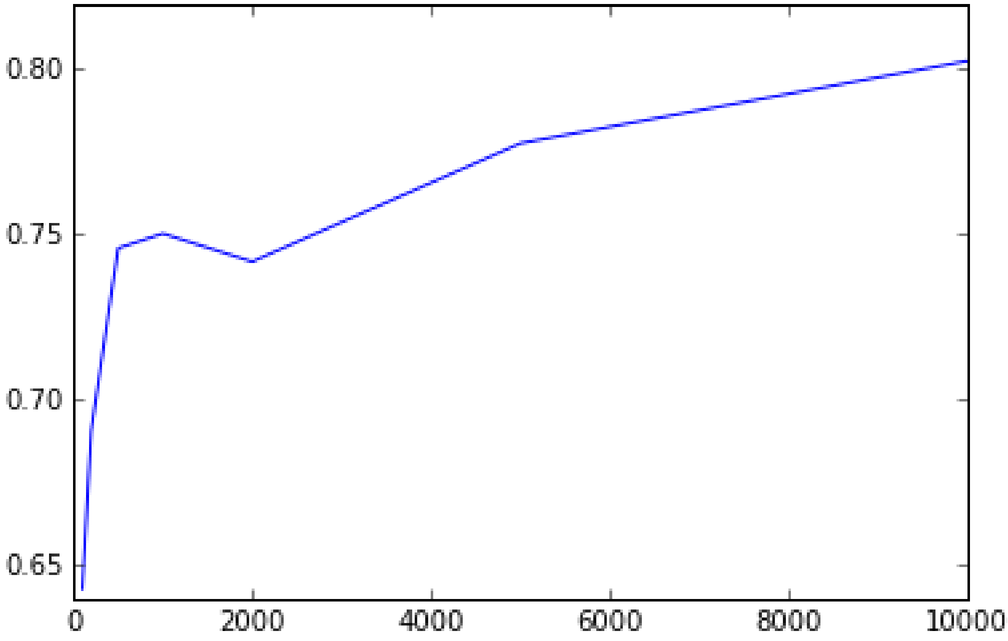
```
5000  -  0.7776
```



```
10000  -  0.8025
```



```
In [5]: plt.plot(x, y)

Out[5]: [<matplotlib.lines.Line2D at 0xadf4d30>]
```



1.2 Problem 3

Cross-validation allows you to train and test against the same set of data, with an overall effect of improving accuracy by allowing more training data to be used.

The optimal ‘C’ I’ve found is $5 * 10^{-7}$, providing a cross-validation accuracy of 0.86. To converge on this value, I first started C on 10^{-10}, then repeatedly multiply by a factor of 100 up to 10^{10}, to get an order of magnitude approximation. Then I tried increasing C with a linear step size.

```
In [ ]: # Reshuffle the data
        images, labels = permute_both(images, labels)
        labels = numpy.ravel(labels)

        c = 1e-7

        for _ in range(5):

            # 10-fold cross validation
            total = 0
            for k in range(10):
                kimages, klabels = [], []
                s = svm.LinearSVC(C=c)
                for i in range(10):
                    if i != k:
                        start, end = i * 1000, i * 1000 + 1000
                        kimages.append(images[start:end])
                        klabels.append(labels[start:end])
                s.fit(numpy.concatenate(kimages), numpy.concatenate(klabels))

                start, end = k * 1000, k * 1000 + 1000
                total += s.score(images[start:end], labels[start:end])
                print(c, "-", total / (k + 1))


            c += 2e-7
```

1.3 Kaggle - Digits

With $C = 5 * 10^{-7}$ on a LinearSVC, I got a Kaggle score of 0.87820.

```
In [4]: # Load the data
        mat = scipy.io.loadmat(’data/digit-dataset/train.mat’)
        images = numpy.reshape(mat["train_images"], (1, -1, 60000))[0].T
        labels = mat["train_labels"]
        labels = numpy.ravel(labels)

        # Train the SVM on all the training data
        s = svm.LinearSVC(C=1e-7).fit(images, labels)

        # Predict the labels for the test data
        mat = scipy.io.loadmat(’data/digit-dataset/test.mat’)
        test = numpy.reshape(mat["test_images"], (1, -1, 10000))[0].T
        result = s.predict(test)
```

```
In [16]: # Write the results to a csv
         f = open(’digits.csv’, ’w’)
```

```
f.write('Id,Category\n')
for i in range(len(result)):
    f.write("{0},{1}\n".format(i + 1, result[i]))
f.close()
```

1.4 Problem 4

The optimal 'C' I've found is 90, providing a cross-validation accuracy of 0.813. This was derived with the same method – starting C on 10^{-10}, then repeatedly multiply by a factor of 100 up to 10^{10}, to get an order of magnitude approximation, and finally increasing C with a linear step size.

```
In [2]: # Load the data
        mat = scipy.io.loadmat('data/spam-dataset/spam_data.mat')
        emails = mat['training_data']
        labels = numpy.ravel(mat['training_labels'])
        emails, labels = permute_both(emails, labels)
```

```
In [ ]: # k-fold cross validation to optimize C
        emails, labels = permute_both(emails, labels)

        folds = 12
        interval = 431

        c = 90

        for _ in range(10):
        # cross validation
            total = 0
            for k in range(folds):
                kemails, klabels = [], []
                s = svm.SVC(C=c)
                for i in range(folds):
                    if i != k:
                        start, end = i * interval, (i + 1) * interval
                        kemails.append(emails[start:end])
                        klabels.append(labels[start:end])
                s.fit(numpy.concatenate(kemails), numpy.concatenate(klabels))

                start, end = k * interval, (k + 1) * interval
                total += s.score(emails[start:end], labels[start:end])
            print(c, "-", total / (folds))
            c *= 3
```

1.5 Kaggle - Spam

With $C = 90$ on a LinearSVC, I got a Kaggle score of 0.82548.

```
In [6]: # Load the data
        mat = scipy.io.loadmat('data/spam-dataset/spam_data.mat')
        emails = mat['training_data']
        labels = numpy.ravel(mat['training_labels'])

        # Train the SVM on all the training data
        s = svm.SVC(C=90).fit(emails, labels)
        # Predict the labels for the test data
        result = s.predict(mat[’test_data’])
```

```
In [7]: # Write the results to a csv
        f = open(’spam.csv’, ’w’)
        f.write(’Id,Category\n’)
        for i in range(len(result)):
            f.write("{0},{1}\n".format(i + 1, result[i]))
        f.close()
```