Kaggle Handwriting Digit Database Competition Report

1. Averaging the data and plotting

To achieve a basic understanding of the kaggle handwriting digit database (MINIST), I picked up the average value of each pixel of the digits from 0 to 9 in the database, and plotted as the following:

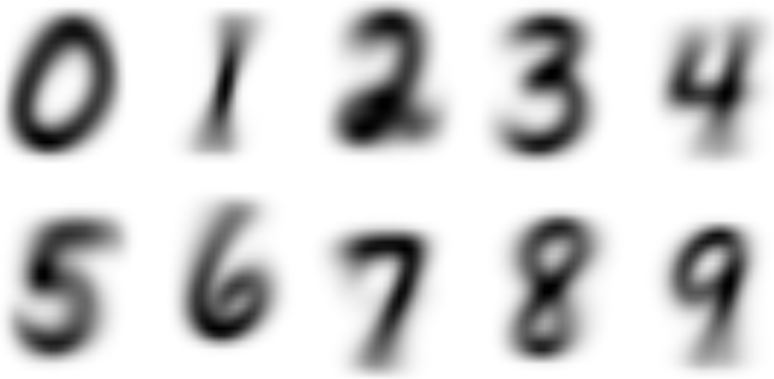

Figure 1, Averaged imagines of digits from 0 to 9

From the averaged images of the 42000 records, it seems that certain digits like 0 and 7 would be easier to be recognized than the digits like 4 and 8, since the 4 and 8 are more vague which might means less stable patterns with these digits.

2. Blending Random Forest, KNN and SVM

Since Random Forest and KNN are the most popular methods with this competition, which can be confirmed from the forum discussions and leaderboard, they are chosen as the basic models. The models are all implemented using available functions from Scikit-learn package of Python. These functions could not only give the predictions, but also the probabilities with these predictions, which greatly facilitated the blending work. The main blending idea is choosing the predictions with the maximum probabilities that developed by these models. Different blending methods are examined, which include RF, KNN, RF+KNN, RFs+KNN, RFs+KNN+SVMs. Through experiments with small database, it's turned out that the RFs+KNN model is the most efficient one, but too many RF produced would bring the problem of overfitting which vividly shown by the

decreasing prediction accuracy with the increasing number of RF when the number of RF rises above a certain threshold.

3. Blend SIFT

So far as I know, Scale-Invariant Feature Transform (SIFT) is actually a neighbor searching method used to find matching points of two imagines. To implement this method, I chose the predictions produced by the RFs+KNN model with probability values less than 45%, and then compare the imagines of these records with the averaged imagines from 0 to 9, and identify the most similar one. Since the KNN model already implemented neighbor-searching technique, this method failed to improve the predication accuracy of the RFs+KNN model.

4. Submission and discussion

Submission timeline:

Table 1, Improved submissions, would be updated until Monday

Kaggle Name	Time	Score	Method
lianzg	Wed, 05 Feb 2014 00:37:10	0.96857	3RF+KNN
lianzg	Fri, 07 Feb 2014 02:45:17	0.97686	Nudge+Rotation+RF
lianzg	Fri, 07 Feb 2014 08:29:20	0.98043	Nudge+Rotation+RF+KNN

Nudging and rotating the original training data and thus produce more training data significantly improved the predication accuracy. Due to the hardware limitations of laptop, several ideas are not carried out: the blending method used now only blends the predication models at the same level, no sub-blending methods explored; prediction method might be improved by dividing the dataset according to different digits, and specify different models used for different digits.

Appendix: Codes

```
# Averaging Images
def average(data):
    newdata = data.groupby('label').mean()
    fig, axes = plt.subplots(1, 1, sharex= True, sharey= True)
    a = 0
```

```
    for i in range(0,2):
        for j in range(0,5):
            mm = np.array(newdata.ix[a+j])
            xx = mm.reshape((28, 28))
            plt.imshow(xx, cmap= plt.cm.gray_r, extent = [0,28,0,28], aspect = 'auto')
            plt.gca().get_xaxis().set_visible(False)
            plt.gca().get_yaxis().set_visible(False)
            for spine in ['top','right','left','bottom']:
                plt.gca().spines[spine].set_visible(False)
            plt.show()
            plt.savefig('average.png')
        a = 5
    plt.subplots_adjust(wspace=0, hspace=0)
    plt.show()
    plt.clf()

# Random Forest
def rf(target, train, test):
    ranfor = RandomForestClassifier(n_estimators=500, n_jobs=2)
    ranfor.fit(train, target)
    print 'Predicting'
    predictpro = ranfor.predict_proba(test)
    predict = ranfor.predict(test)
    bestpro = predictpro.max(axis =1)
    return predict, bestpro

# KNN
def kn(target, train, test):
    knn = neighbors.KNeighborsClassifier()
    knn.fit(train, target)
    predictpro = knn.predict_proba(test)
    print 'Predicting'
    predict = knn.predict(test)
    bestpro = predictpro.max(axis=1)
    return predict, bestpro

# Rotate data, copied from WiseRF_98.py
def rotate_dataset(X, Y):
    """
    This produces a dataset 2 times bigger than the original one,
    by rptating the 28x28 images in 10 degrees clockwise and counter clockwise
    """
    angles = [-10,10]


    rotate = lambda x, w: imrotate(x.reshape((28, 28)), w).ravel()
```

```
    X = np.concatenate([X] +
                       [np.apply_along_axis(rotate, 1, X, angle)
                        for angle in angles])
    Y = np.concatenate([Y for _ in range(3)], axis=0)
    return X, Y

# Nudge data, copied form WiseRF_98.py
def nudge_dataset(X, Y):
    """
    This produces a dataset 5 times bigger than the original one,
    by moving the 28x28 images in X around by 1px to left, right, down, up
    """
    direction_vectors = [
        [[0, 1, 0],
         [0, 0, 0],
         [0, 0, 0]],

        [[0, 0, 0],
         [1, 0, 0],
         [0, 0, 0]],

        [[0, 0, 0],
         [0, 0, 1],
         [0, 0, 0]],

        [[0, 0, 0],
         [0, 0, 0],
         [0, 1, 0]],
]

    shift = lambda x, w: convolve(x.reshape((28, 28)), mode='constant',
                                  weights=w).ravel()
    X = np.concatenate([X] +
                       [np.apply_along_axis(shift, 1, X, vector)
                        for vector in direction_vectors])
    Y = np.concatenate([Y for _ in range(5)], axis=0)
    return X, Y

# Main
if __name__ == '__main__':

    traindata = pd.read_csv('train.csv')
    testdata = pd.read_csv('test.csv')
    target = np.array(traindata.ix[:,0])
    train = np.array(traindata.ix[:,1:])
    train = pd.DataFrame(train)
```

```
    test = np.array(testdata)
    average(traindata)
    ntrain,ntarget = rotate_dataset(train,target)
    ntrain,ntarget = nudge_dataset(train,target)
    np.save('ntrain', ntrain)
    np.save('ntarget',ntarget)
    ntrain = np.load('ntrain.npy')
    ntarget = np.load('ntarget.npy')
    knnpredict, knnpro = kn(ntarget, ntrain, test)
    mm = list(knnpro)
    yy = list(knnpredict)
    pro = []
    predict = []
    pro.append(mm)
    predict.append(yy)
    for i in range (0, 10):
      rfpredict, rfpro = rf(ntarget, ntrain, test)
      print 'done ' + str(i)
      pro.append(rfpro)
      predict.append(rfpredict)
      promatrix = np.array(pd.DataFrame(np.array(pro)).T)
      predictmatrix = np.array(pd.DataFrame(np.array(predict)).T)
      firstpredict = predictmatrix[np.arange(len(predictmatrix)), promatrix.argmax(axis=1)]
      output = pd.DataFrame({'ImageId': np.arange(1,len(test)+1),'Label':
list(firstpredict)} )
      output.to_csv(str(i) +'_benchmark.csv', index = False)
```