HW01_knn_kulikov

April 13, 2016

1 KNN – Digit Recognizer

1.1 Kulikov Alex, gr. 397

In [4]:
```
import matplotlib
import numpy
import pandas
from knn import MatrixBasedKNearestNeighbor, KDBasedKNearestNeighbor

%pylab inline
%load_ext autoreload
%autoreload 2

pandas.options.display.max_colwidth = 0

from IPython.core.display import HTML
HTML("<style>.container { width:90% !important; }</style>")
```

```
Populating the interactive namespace from numpy and matplotlib
```

Out[4]:
```
<IPython.core.display.HTML object>
```

1.2 Let's get some data!

In [5]:
```
df = pandas.read_csv("kaggle_data/train.csv")
```

In [6]:
```
df.head()
```

Out[6]:
```
   label  pixel0  pixel1  pixel2  pixel3  pixel4  pixel5  pixel6  pixel7  \
0  5      0       0       0       0       0       0       0       0
1  0      0       0       0       0       0       0       0       0
2  4      0       0       0       0       0       0       0       0
3  1      0       0       0       0       0       0       0       0
4  9      0       0       0       0       0       0       0       0

   pixel8    ...     pixel774  pixel775  pixel776  pixel777  pixel778  \
0  0         ...     0         0         0         0         0
1  0         ...     0         0         0         0         0
2  0         ...     0         0         0         0         0
3  0         ...     0         0         0         0         0
4  0         ...     0         0         0         0         0

   pixel779  pixel780  pixel781  pixel782  pixel783
0  0         0         0         0         0
```

```
   1  0          0          0          0          0
   2  0          0          0          0          0
   3  0          0          0          0          0
   4  0          0          0          0          0

   [5 rows x 785 columns]
```

```
In [7]: print "Head:"
        print df.head()
        print "Shape:"
        print shape(df)
```

```
Head:
   label  pixel0  pixel1  pixel2  pixel3  pixel4  pixel5  pixel6  pixel7  \
0  5      0       0       0       0       0       0       0       0
1  0      0       0       0       0       0       0       0       0
2  4      0       0       0       0       0       0       0       0
3  1      0       0       0       0       0       0       0       0
4  9      0       0       0       0       0       0       0       0

   pixel8    ...     pixel774  pixel775  pixel776  pixel777  pixel778  \
0  0         ...     0         0         0         0         0
1  0         ...     0         0         0         0         0
2  0         ...     0         0         0         0         0
3  0         ...     0         0         0         0         0
4  0         ...     0         0         0         0         0

   pixel779  pixel780  pixel781  pixel782  pixel783
0  0         0         0         0         0
1  0         0         0         0         0
2  0         0         0         0         0
3  0         0         0         0         0
4  0         0         0         0         0

[5 rows x 785 columns]
Shape:
(20000, 785)
```

```
In [8]: X_train, y_train = df[df.columns[1:]].values, df["label"].values
```

1.2.1 OK, we have the initial data and we understand its structure

1.3 Visualizing it “as is”, binarized and with image centering

```
In [12]: def plot_image(img, im_size=28):
             pylab.imshow(img.reshape(im_size, im_size), cmap = "gray")

         def plot_grid(imgs, nrows, ncols, dataset = X_train, im_size = 28):
             fig = pyplot.gcf()
             fig.set_size_inches(17.5,15.5)
             for pylab_index, img in enumerate(imgs):
                 pylab.subplot(nrows, ncols, pylab_index + 1)
                 plot_image(img)
                 pylab.axis('off')
```

```
In [24]: plot_grid(X_train[0:10], 1, 10)
```

```
In [25]: def binarize(img, bborder = 100):
             binarizator = MatrixBasedKNearestNeighbor(k = 3)
             img = img.reshape(binarizator.size, binarizator.size)
             img = binarizator.center_image(img)
             img = binarizator.binarize(img, black_border = bborder)

             img = img.reshape(binarizator.size * binarizator.size)

             return img
```

```
In [26]: def binarize_batch(img_set, bborder = 100):
             for index, img in enumerate(img_set):
                 img_set[index] = binarize(img.copy(), bborder)
             return img_set
```

```
In [27]: def plot_grid_bin(imgs, nrows, ncols, bborder = 100, dataset = X_train, im_size = 28):
             fig = pyplot.gcf()
             fig.set_size_inches(17.5,15.5)
             binarizator = MatrixBasedKNearestNeighbor(k = 3)
             for pylab_index, img in enumerate(imgs):
                 img = img.reshape(binarizator.size, binarizator.size)
                 img = binarizator.binarize(img, bborder)
                 pylab.subplot(nrows, ncols, pylab_index + 1)
                 plot_image(img)
                 pylab.axis('off')
```

```
In [28]: plot_grid_bin(X_train[0:20].copy(), 1, 20, bborder = 0)
```

```
In [29]: plot_grid_bin(X_train[0:20].copy(), 1, 20, bborder = 100)
```

```
In [385]: plot_grid_bin(X_train[0:20].copy(), 1, 20, bborder = 200)
```

```
In [386]: def plot_grid_bin_centered(imgs, nrows, ncols, bborder = 100, dataset = X_train, im_size = 28
              fig = pyplot.gcf()
              fig.set_size_inches(17.5,15.5)
              for pylab_index, img in enumerate(imgs):
                  img = binarize(img, bborder)
                  pylab.subplot(nrows, ncols, pylab_index + 1)
                  plot_image(img)
                  pylab.axis('off')
```

```
In [387]: plot_grid_bin_centered(X_train[0:20].copy(), 1, 20, bborder = 0)
```

1.3.1 OK, now we can transform images in a simple way (binarization, centering) and draw them. Hopefully, this helps.

1.4 Plot means

```
In [388]: average_class_imgs = []

          figures = [[] for i in xrange (0, 10)]

          for cur_fig in xrange(0, 10):
              for image_id in xrange (1, len (X_train)):
                  if (y_train[image_id] == cur_fig):
                      figures[cur_fig].append (X_train[image_id])

              np_figures = np.array (figures[cur_fig])
              np_figures = np.mean (np_figures, axis = 0)
              average_class_imgs.append (np_figures)
              pass

          average_class_imgs = np.array (average_class_imgs)
```

```
In [389]: # Plot your means, note that is should be similar on to real smooth numbers -- done
          plot_grid(average_class_imgs, nrows = 1, ncols = 10)
```

```
In [390]: plot_grid_bin_centered(average_class_imgs.copy(), 1, 10, bborder = 50)
```

1.5 Matrix-based KNN

In [391]:
```
# Code matrix-based KNN with L2 norm (see MatrixBasedKNearestNeighbor class)
# Use predict on to X_train[:100] only for debug
```

In [392]:
```
plot_grid(X_train[110:120], 1, 10)
```

In [393]:
```
# KNN two loops

knn_clf_loop2 = MatrixBasedKNearestNeighbor(num_loops = 2, k = 3)
knn_clf_loop2 = knn_clf_loop2.fit(X_train[:110], y_train[:110])
%time knn_clf_loop2.calc_dist(X_train[110:120], metric="pixel_L1")
%time y_pred2 = knn_clf_loop2.predict_labels(X_train[110:120])
print y_pred2
```

```
CPU times: user 853 ms, sys: 43.3 ms, total: 897 ms
Wall time: 794 ms
[(9, 2)]
[(3, 2)]
[(1, 3)]
[(1, 3)]
[(0, 3)]
[(4, 2)]
[(9, 2)]
[(2, 2)]
[(0, 1)]
[(0, 3)]
CPU times: user 0 ns, sys: 0 ns, total: 0 ns
Wall time: 703 µs
[ 9.  3.  1.  1.  0.  4.  9.  2.  0.  0.]
```

In [394]:
```
# KNN one loop

knn_clf_loop1 = MatrixBasedKNearestNeighbor(num_loops = 1, k = 3)
knn_clf_loop1 = knn_clf_loop1.fit(X_train[:110], y_train[:110])
%time knn_clf_loop1.calc_dist(X_train[110:120], metric="pixel_L1")
%time y_pred1 = knn_clf_loop1.predict_labels(X_train[110:120])
print y_pred1
```

```
CPU times: user 787 ms, sys: 30 ms, total: 817 ms
Wall time: 763 ms
[(9, 2)]
[(3, 2)]
[(1, 3)]
[(1, 3)]
[(0, 3)]
[(4, 2)]
[(9, 2)]
[(2, 2)]
[(0, 1)]
[(0, 3)]
CPU times: user 3.33 ms, sys: 0 ns, total: 3.33 ms
Wall time: 648 µs
[ 9.  3.  1.  1.  0.  4.  9.  2.  0.  0.]
```

```
In [395]: print 'good' if np.linalg.norm(y_pred2 - y_pred1) < 1e-4 else 'fail'
```

```
good
```

```
In [396]: # KNN no loops

          knn_clf_loop0 = MatrixBasedKNearestNeighbor(num_loops = 0, k = 3)
          knn_clf_loop0 = knn_clf_loop0.fit(X_train[:110], y_train[:110])
          %time knn_clf_loop0.calc_dist(X_train[110:120], metric="pixel_L1")
          %time y_pred0 = knn_clf_loop0.predict_labels(X_train[110:120])
          print y_pred0
```

```
CPU times: user 707 ms, sys: 30 ms, total: 737 ms
Wall time: 710 ms
[(9, 2)]
[(3, 2)]
[(1, 3)]
[(1, 3)]
[(0, 3)]
[(4, 2)]
[(9, 2)]
[(2, 2)]
[(0, 1)]
[(0, 3)]
CPU times: user 0 ns, sys: 0 ns, total: 0 ns
Wall time: 652 µs
[ 9.  3.  1.  1.  0.  4.  9.  2.  0.  0.]
```

```
In [397]: print 'good' if np.linalg.norm(y_pred1 - y_pred0) < 1e-4 else 'fail'
```

```
good
```

1.5.1 It works sometimes!

1.5.2 Lets' try the clever IMED metric

http://www.cis.pku.edu.cn/faculty/vision/wangliwei/pdf/IMED.pdf

```
In [13]: def precalc_G(sigma, size = 28):

             G = numpy.zeros(size * size * size * size).reshape(size, size, size, size)
```

```
            sigma_squared = math.pow(sigma, 2)

            # fill G
            for row_i in range(size):
                for col_i in range(size):
                    for row_j in range(size):
                        for col_j in range(size):
                            G[row_i][col_i][row_j][col_j] = math.exp(((row_i - row_j) * (row_i - row_j

            return G

        G_0p4 = precalc_G(0.4) # 0.04
        G_0p5 = precalc_G(0.5) # 0.14
        G_0p6 = precalc_G(0.6) # 0.25
        G_0p9 = precalc_G(0.9) # 0.53
        G_1p01 = precalc_G(1.01) #0.61
```

In [399]:
```
from collections import Counter
print(Counter(y_train[:110]).keys())
print(Counter(y_train[:110]).values())
```

```
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
[14, 17, 7, 12, 11, 6, 12, 12, 8, 11]
```

In [403]:
```
knn_IMED = MatrixBasedKNearestNeighbor(num_loops = 2, k = 1)
knn_IMED = knn_IMED.init_G(G_1p5)
# print(knn_IMED.G) # prints the pixel distance matrix
knn_IMED = knn_IMED.fit(knn_IMED.ST_batch(X_train[:510].copy()), y_train[:510])
%time knn_IMED = knn_IMED.calc_dist(X_train[1110:1120], metric="IMED")
%time y_predIMED = knn_IMED.predict_labels(X_train[1110:1120])
print y_predIMED
```

```
CPU times: user 56.7 ms, sys: 0 ns, total: 56.7 ms
Wall time: 59 ms
[(4, 1)]
[(8, 1)]
[(9, 1)]
[(8, 1)]
[(7, 1)]
[(1, 1)]
[(9, 1)]
[(6, 1)]
[(1, 1)]
[(3, 1)]
CPU times: user 3.33 ms, sys: 0 ns, total: 3.33 ms
Wall time: 1.31 ms
[ 4.  8.  9.  8.  7.  1.  9.  6.  1.  3.]
```

In [356]:
```
plot_grid(X_train[1110:1120], 1, 10)
```

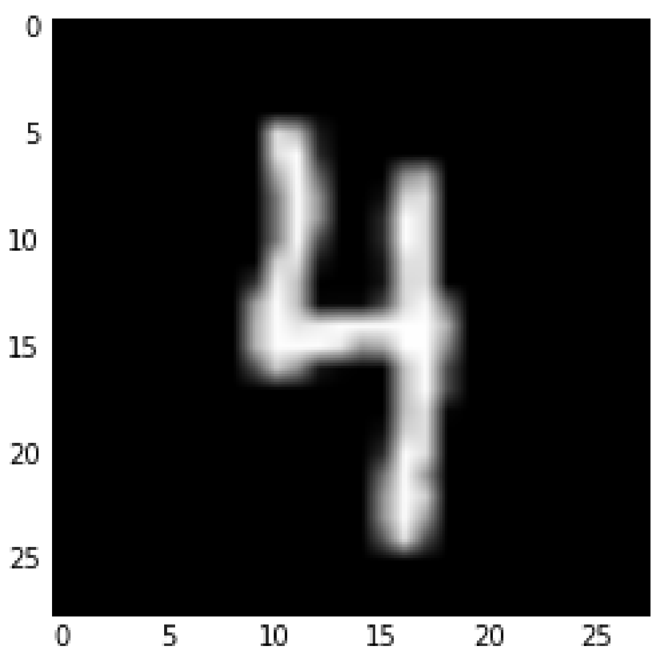
1.5.3 Seems that this metric is also working

1.6 KDTree-based KNN

```
In [9]: # KNN kd_tree

        knn_clf_Tree = KDBasedKNearestNeighbor(k = 3)
        knn_clf_Tree = knn_clf_Tree.fit(X_train[:110], y_train[:110])
        knn_clf_Tree = knn_clf_Tree.calc_dist(X_train[1110:1120], "minkowski")
        neighbors = knn_clf_Tree.get_neighbors(X_train[1110:1120])
        %time y_pred = knn_clf_Tree.predict_labels(X_train[1110:1120])
        print y_pred
        print neighbors
```

```
CPU times: user 6.67 ms, sys: 0 ns, total: 6.67 ms
Wall time: 7.76 ms
[ 4.  5.  9.  6.  7.  1.  4.  1.  1.  0.]
[[ 26  92  48]
 [ 11  99  35]
 [ 26  84  54]
 [ 73  66  22]
 [ 52  84 103]
 [ 59  23   3]
 [ 58  92   4]
 [105  77  59]
 [ 67  40  72]
 [  1  21   0]]
```

```
In [13]: plot_image(X_train[1110]) # one of the target images
```




```
In [14]: plot_grid(X_train[[26, 92, 48]], 1, 3) # and the 3 nearest neighbors (default 'minkowski' L2 m
```

1.7 It's time to test it 4real!

1.8 Code Accuracy score and Cross Validation Prosses

In [412]:
```
# Accuracy
from sklearn import metrics

def accuracy(y_true, y_predict):
    return metrics.accuracy_score(y_true, y_predict)
```

In [413]:
```
print 'good' if accuracy([1, 1, 1, 0], [1, 1, 1, 5]) == 0.75 else 'fail'
```

```
good
```

In [442]:
```
# Cross validation
from sklearn import cross_validation
from datetime import datetime

def my_cross_validation(X, y, predictor, metric, q_fold = 5, r_fold = 5):
    scores = []

    total_size = X.shape[0]

    seed = datetime.now().microsecond + datetime.now().second * 1000000

    shuffled_split = cross_validation.ShuffleSplit(total_size, n_iter=r_fold, test_size=1.0/q

    for train_index, test_index in shuffled_split:
        X_train = X[train_index]
        y_train = y[train_index]
        X_test = X[test_index]
        y_test = y[test_index]

        predictor = predictor.fit(X_train, y_train)
        predictor = predictor.calc_dist(X_test, metric)
        y_predicted = predictor.predict_labels(X_test)

        scores.append(accuracy(y_test, y_predicted))

    return np.mean(scores)
```

```
In [473]: # ShuffleSplit test using datetime.now() random seed

          seed = datetime.now().microsecond + datetime.now().second * 1000000
          shuffled_split = cross_validation.ShuffleSplit(10, n_iter=2, test_size=0.1, random_state=seed)
          for train_index, test_index in shuffled_split:
              print(train_index)
              print(test_index)
```

```
[0 6 3 5 9 7 1 2 4]
[8]
[5 1 3 8 4 0 7 2 9]
[6]
```

1.9 Knn Matrix stats plotting

1.9.1 Dumb L1, L2, L3 metrics

Not a perfect metric for an image, frankly

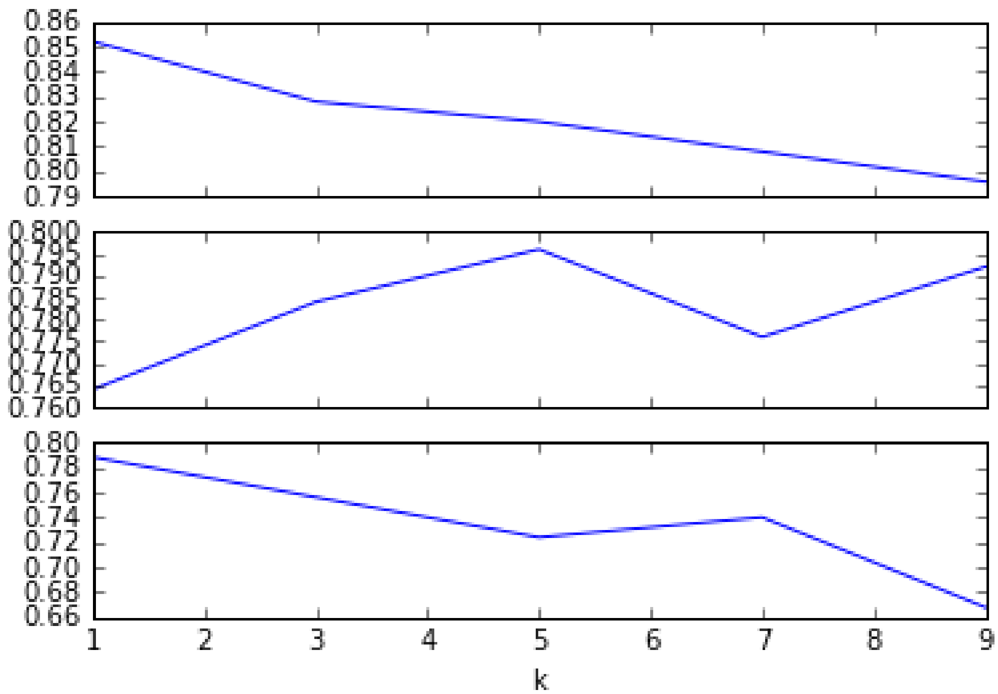
```
In [475]: predictor = MatrixBasedKNearestNeighbor(k = 3)
          print("L1: " + str(my_cross_validation(X_train[:100], y_train[:100], predictor, "pixel_L1", q
          print("L2: " + str(my_cross_validation(X_train[:100], y_train[:100], predictor, "pixel_L2", q
```

```
L1: 0.74
L2: 0.48
```

```
In [447]: # without binarization

          stats_k = [1, 3, 5, 7, 9]
          stats_result_L1 = []
          stats_result_L2 = []
          stats_result_L3 = []

          for k in stats_k:
              predictor = MatrixBasedKNearestNeighbor(k)
              stats_result_L1.append(my_cross_validation(X_train[:500], y_train[:500], predictor, "pixel
              stats_result_L2.append(my_cross_validation(X_train[:500], y_train[:500], predictor, "pixel
              stats_result_L3.append(my_cross_validation(X_train[:500], y_train[:500], predictor, "pixel

          print "L1 " + str(stats_result_L1)
          print "L2 " + str(stats_result_L2)
          print "L3 " + str(stats_result_L3)

          from matplotlib import pyplot

          f, axarr = plt.subplots(3, sharex=True)
          axarr[2].set_xlabel("k")
          axarr[0].plot(stats_k, stats_result_L1)
          axarr[1].plot(stats_k, stats_result_L2)
          axarr[2].plot(stats_k, stats_result_L3)
```

```
L1 [0.85199999999999998, 0.82800000000000007, 0.82000000000000006, 0.80800000000000005, 0.7960000000000
L2 [0.76400000000000001, 0.78400000000000003, 0.79599999999999993, 0.77600000000000002, 0.7920000000000
L3 [0.78800000000000003, 0.75600000000000001, 0.72399999999999998, 0.73999999999999999, 0.6680000000000
```

```
Out[447]: [<matplotlib.lines.Line2D at 0x7fcee8d29710>]
```

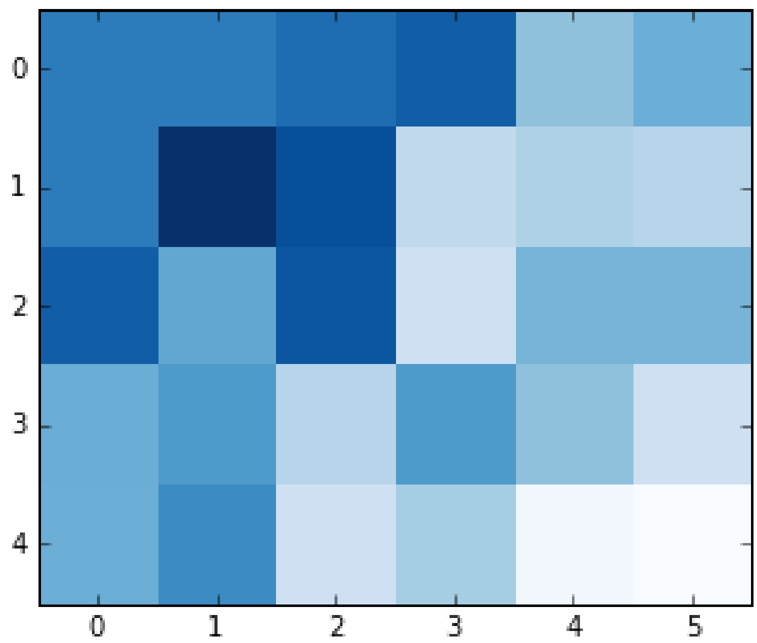


In [448]:
```
plot_grid(binarize_batch(X_train[:500].copy(), 10)[0:10], 1, 10)
```

In [449]:
```
# with binarization

stats_k = [1, 3, 5, 7, 9]
borders = [10, 25, 50, 75, 100, 150]
stats_result_bin = []

for k in stats_k:
    this_result = []
    predictor = MatrixBasedKNearestNeighbor(k)

    for b in borders:
        this_result.append(my_cross_validation(binarize_batch(X_train[:500].copy(), b), y_trai
    stats_result_bin.append(this_result)

print "bin " + str(stats_result_bin)

from matplotlib import pyplot
```

```
plt.imshow(stats_result_bin,
           interpolation="nearest", cmap = "Blues")
plt.show()
```

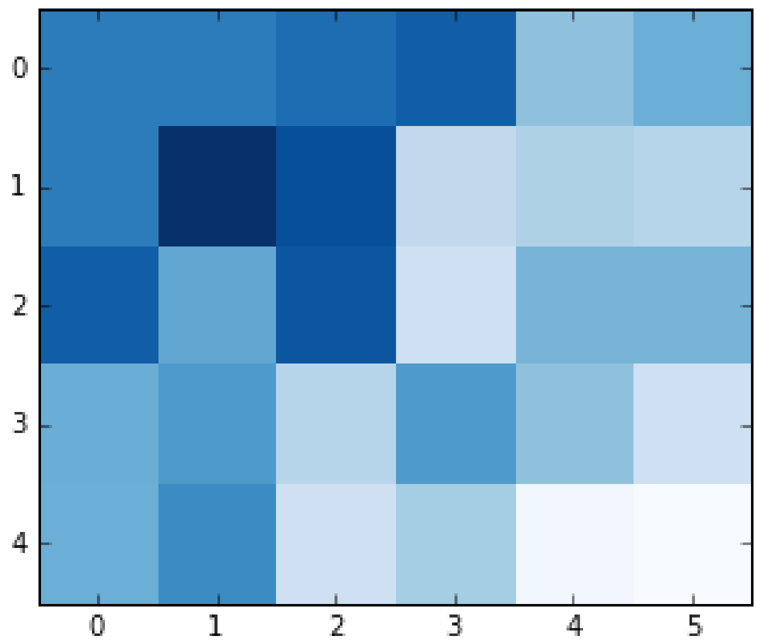
```
bin [[0.8640000000000001, 0.8640000000000001, 0.8720000000000011, 0.8800000000000012, 0.824000000000000
```




```
In [450]: import pprint

          pp = pprint.PrettyPrinter(indent=4)

          pp.pprint(stats_result_bin)

          from matplotlib import pyplot

          plt.imshow(stats_result_bin,
                     interpolation="nearest", cmap = "Blues")
          plt.show()
```

```
[   [   0.8640000000000001,
        0.8640000000000001,
        0.8720000000000011,
        0.8800000000000012,
        0.8240000000000007,
        0.8359999999999997],
    [   0.8640000000000001,
        0.9039999999999991,
        0.8880000000000012,
        0.8040000000000005,
        0.8120000000000006,
        0.8080000000000005],
```

```
[   0.88000000000000012,
    0.84000000000000008,
    0.88400000000000001,
    0.79600000000000004,
    0.83200000000000007,
    0.83200000000000007],
[   0.83600000000000008,
    0.84799999999999986,
    0.80800000000000005,
    0.84800000000000009,
    0.82400000000000007,
    0.79600000000000004],
[   0.83599999999999997,
    0.85600000000000009,
    0.79600000000000004,
    0.81600000000000006,
    0.77200000000000002,
    0.76800000000000013]]
```



Seems that black_border = 20 is close to the best one

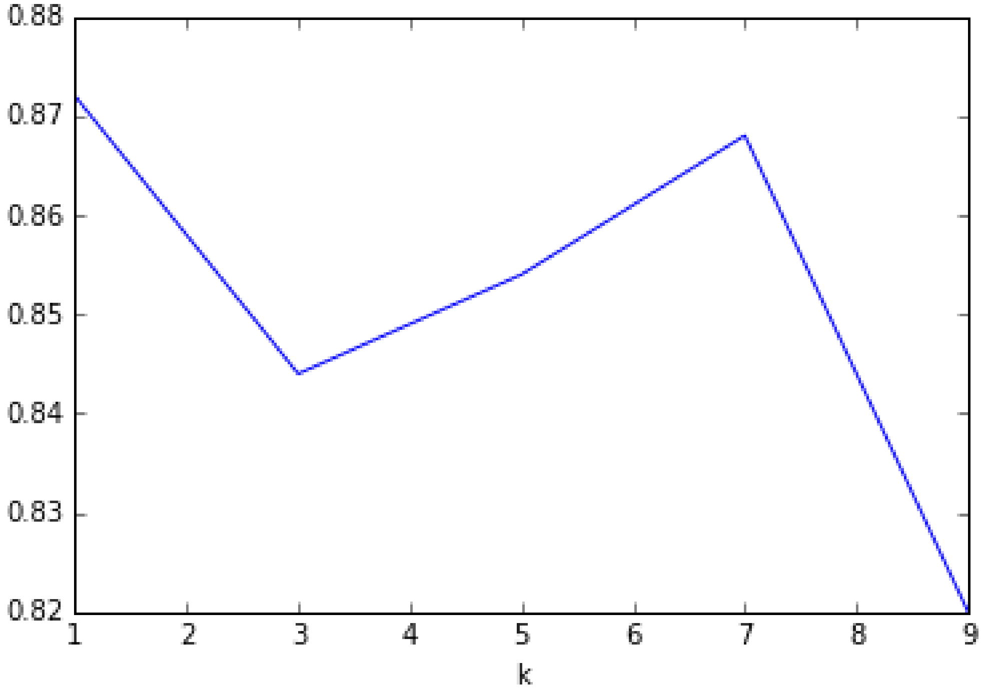
In [451]:
```
plot_grid(binarize_batch(X_train[:500].copy(), 15)[0:10], 1, 10)
```

```
In [452]: plot_grid(binarize_batch(X_train[:500].copy(), 20)[0:10], 1, 10)
```

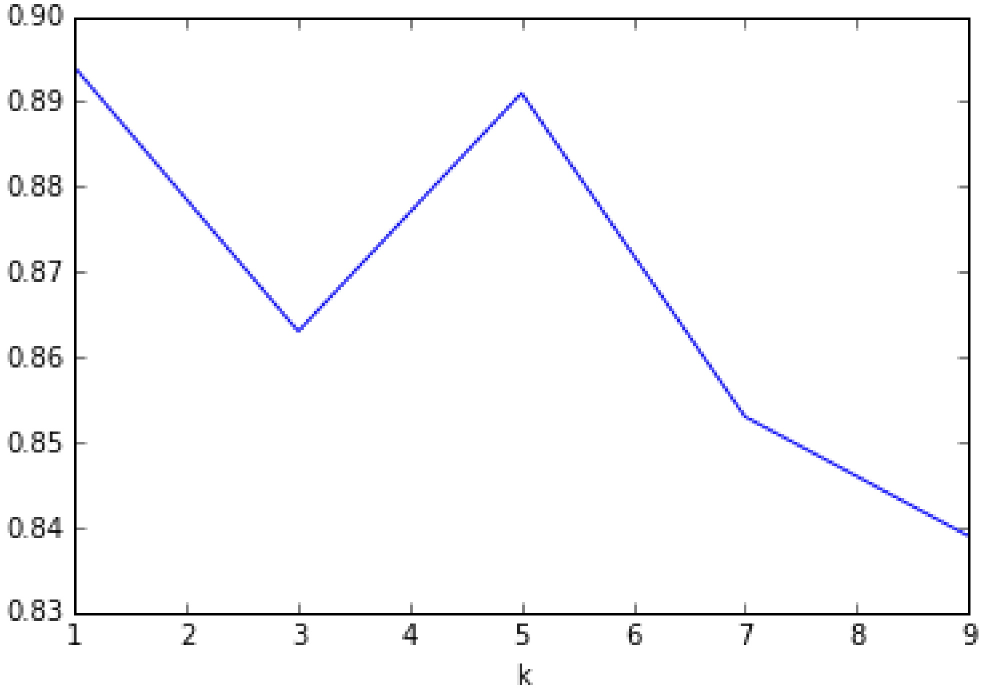
```
In [453]: plot_grid(binarize_batch(X_train[:500].copy(), 25)[0:10], 1, 10)
```

1.9.2 IMED metric – time to set up some parameters

```
In [456]: stats_k = [1, 3, 5, 7, 9]
          stats_result_IMED = []

          for k in stats_k:
              predictor = MatrixBasedKNearestNeighbor(k)
              predictor = predictor.init_G(G_0p5)
              stats_result_IMED.append(my_cross_validation(predictor.ST_batch(X_train[:500].copy()), y_

          print "IMED " + str(stats_result_IMED)

          from matplotlib import pyplot

          f, ax = plt.subplots()
          ax.set_xlabel("k")
          ax.plot(stats_k, stats_result_IMED)

IMED [0.87200000000000011, 0.84400000000000008, 0.85400000000000009, 0.86799999999999999, 0.8200000000000

Out[456]: [<matplotlib.lines.Line2D at 0x7fcee8f58f10>]
```




```
In [458]: stats_k = [1, 3, 5, 7, 9]
          stats_result_IMED = []

          for k in stats_k:
              predictor = MatrixBasedKNearestNeighbor(k)
              predictor = predictor.init_G(G_0p6)
              stats_result_IMED.append(my_cross_validation(predictor.ST_batch(X_train[:1000].copy()), y_

          print "IMED " + str(stats_result_IMED)

          from matplotlib import pyplot

          f, ax = plt.subplots()
          ax.set_xlabel("k")
          ax.plot(stats_k, stats_result_IMED)

IMED [0.89400000000000013, 0.86299999999999988, 0.89100000000000001, 0.85299999999999998, 0.8389999999999

Out[458]: [<matplotlib.lines.Line2D at 0x7fcee9d28f90>]
```




```
In [460]: G_0p4 = precalc_G(0.4) # 0.04
          G_0p6 = precalc_G(0.6) # 0.25
          G_0p9 = precalc_G(0.9) # 0.53
          G_1p01 = precalc_G(1.01) #0.61
          G_1p3 = precalc_G(1.3) #0.74
          G_1p5 = precalc_G(1.5) #0.8
          G_2 = precalc_G(2) #0.88
          G_10 = precalc_G(10) #0.99 -- bad

          stats_G = [G_0p4, G_0p5, G_0p6, G_0p9, G_1p01, G_1p3, G_1p5, G_2, G_10]
          stats_result_IMED_G = []

          for G in stats_G:
              predictor = MatrixBasedKNearestNeighbor(k=3)
              predictor = predictor.init_G(G)
              %time stats_result_IMED_G.append(my_cross_validation(predictor.ST_batch(X_train[:500].cop
```

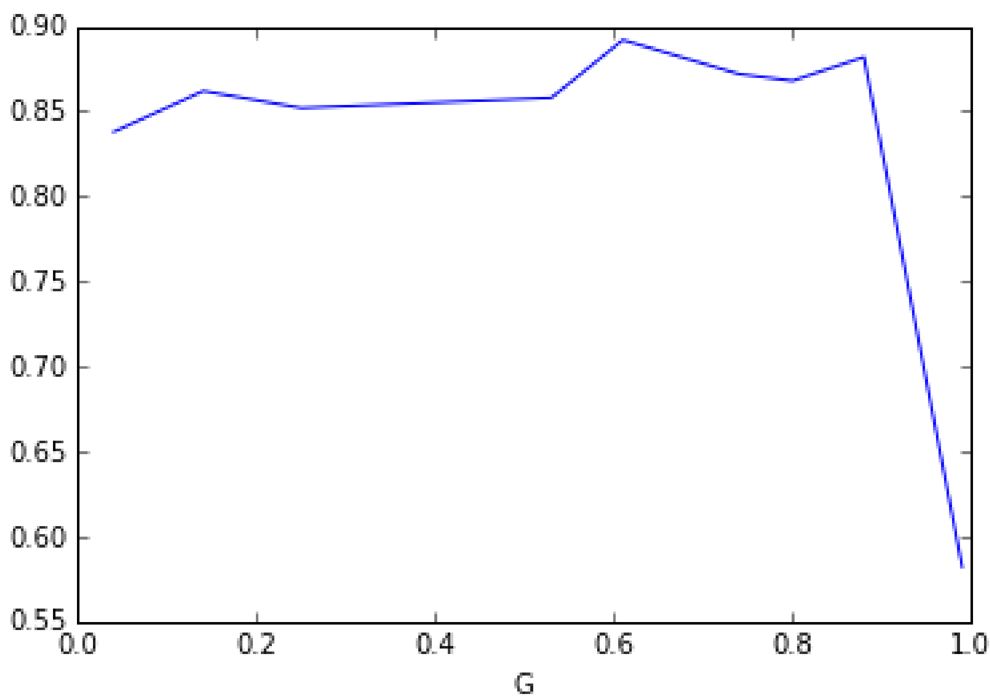
```
CPU times: user 2.61 s, sys: 0 ns, total: 2.61 s
Wall time: 2.62 s
CPU times: user 2.62 s, sys: 0 ns, total: 2.62 s
Wall time: 2.62 s
CPU times: user 2.61 s, sys: 0 ns, total: 2.61 s
Wall time: 2.61 s
CPU times: user 3.81 s, sys: 0 ns, total: 3.81 s
Wall time: 3.81 s
CPU times: user 3.33 s, sys: 0 ns, total: 3.33 s
Wall time: 3.33 s
CPU times: user 3.2 s, sys: 0 ns, total: 3.2 s
```

```
Wall time: 3.2 s
CPU times: user 3.12 s, sys: 0 ns, total: 3.12 s
Wall time: 3.12 s
CPU times: user 3.6 s, sys: 0 ns, total: 3.6 s
Wall time: 3.6 s
CPU times: user 3.54 s, sys: 0 ns, total: 3.54 s
Wall time: 3.54 s
```

In [462]:
```
# best G =  (G_1p01) -- 0.61

print "IMED " + str(stats_result_IMED_G)

from matplotlib import pyplot

f, ax = plt.subplots()
ax.set_xlabel("G")
ax.plot([0.04, 0.14, 0.25, 0.53, 0.61, 0.74, 0.8, 0.88, 0.99], stats_result_IMED_G)
```

```
IMED [0.83800000000000008, 0.86199999999999988, 0.85199999999999998, 0.85799999999999998, 0.892000000000
```

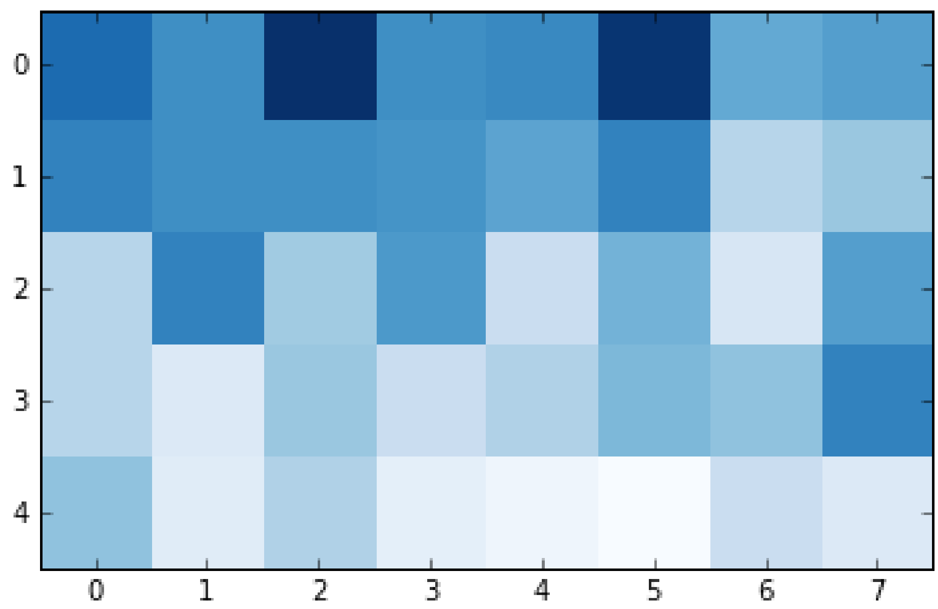
Out[462]:
```
[<matplotlib.lines.Line2D at 0x7fcee9c8ad50>]
```



In [463]:
```
#### Binarize!

stats_k = [1, 3, 5, 7, 9]
borders = [0, 10, 15, 20, 50, 80, 100, 120]
stats_result_bin_IMED = []

for k in stats_k:
```

```
    this_result = []
    predictor = MatrixBasedKNearestNeighbor(k)
    predictor = predictor.init_G(G_1p01)

    for b in borders:
        this_result.append(my_cross_validation(predictor.ST_batch(binarize_batch(X_train[:500
    %time stats_result_bin_IMED.append(this_result)

print "bin " + str(stats_result_bin_IMED)

from matplotlib import pyplot

plt.imshow(stats_result_bin_IMED,
           interpolation="nearest", cmap = "Blues")
plt.show()
```

```
CPU times: user 0 ns, sys: 0 ns, total: 0 ns
Wall time: 5.01 µs
CPU times: user 0 ns, sys: 0 ns, total: 0 ns
Wall time: 26 µs
CPU times: user 0 ns, sys: 0 ns, total: 0 ns
Wall time: 5.96 µs
CPU times: user 0 ns, sys: 0 ns, total: 0 ns
Wall time: 5.96 µs
CPU times: user 0 ns, sys: 0 ns, total: 0 ns
Wall time: 5.96 µs
bin [[0.89200000000000002, 0.88000000000000012, 0.91199999999999992, 0.88000000000000012, 0.882000000000
```

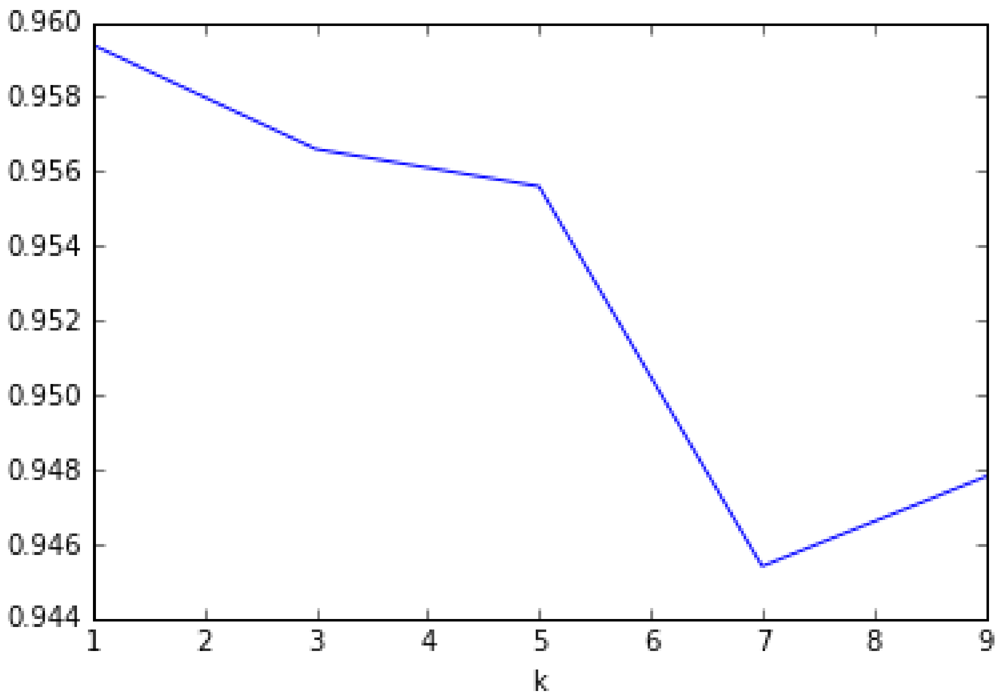



```
In [465]: # best b = 15

          stats_k = [1, 3, 5, 7, 9]
```

```
stats_result_IMED = []

for k in stats_k:
    predictor = MatrixBasedKNearestNeighbor(k)
    predictor = predictor.init_G(G_1p01)
    %time stats_result_IMED.append(my_cross_validation(predictor.ST_batch(binarize_batch(X_tra

print "IMED " + str(stats_result_IMED)

from matplotlib import pyplot

f, ax = plt.subplots()
ax.set_xlabel("k")
ax.plot(stats_k, stats_result_IMED)
```

```
CPU times: user 4min 25s, sys: 223 ms, total: 4min 26s
Wall time: 4min 26s
CPU times: user 4min 17s, sys: 257 ms, total: 4min 17s
Wall time: 4min 17s
CPU times: user 4min 4s, sys: 337 ms, total: 4min 5s
Wall time: 4min 5s
CPU times: user 4min 8s, sys: 163 ms, total: 4min 8s
Wall time: 4min 8s
CPU times: user 4min 9s, sys: 233 ms, total: 4min 9s
Wall time: 4min 9s
IMED [0.95939999999999992, 0.95660000000000012, 0.95559999999999989, 0.94539999999999991, 0.947799999999
```

```
Out[465]: [<matplotlib.lines.Line2D at 0x7fcee8f9b3d0>]
```



1.9.3 Perfect result! About 95.9% based on a quarter of the set!

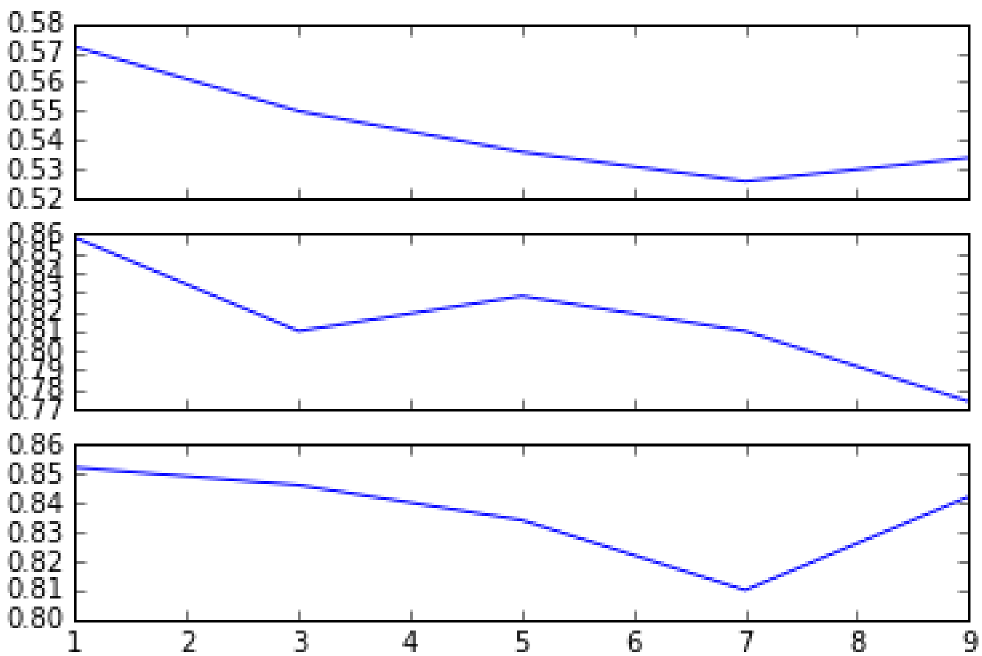
1.10 KDTree stats plotting

1.10.1 Chebyshev, Manhattan, Minkowski p=2 metrics

In [466]:
```
stats_k = [1, 3, 5, 7, 9]
stats_result_Ch = []
stats_result_Ma = []
stats_result_Mi = []

for k in stats_k:
    predictor = KDBasedKNearestNeighbor(k)
    stats_result_Ch.append(my_cross_validation(X_train[:500], y_train[:500], predictor, "cheby
    stats_result_Ma.append(my_cross_validation(X_train[:500], y_train[:500], predictor, "manha
    stats_result_Mi.append(my_cross_validation(X_train[:500], y_train[:500], predictor, "minko

print "Cheb " + str(stats_result_Ch)
print "Manh " + str(stats_result_Ma)
print "Mink " + str(stats_result_Mi)

from matplotlib import pyplot

f, axarr = plt.subplots(3, sharex=True)
axarr[0].plot(stats_k, stats_result_Ch)
axarr[1].plot(stats_k, stats_result_Ma)
axarr[2].plot(stats_k, stats_result_Mi)
```

```
Cheb [0.57200000000000006, 0.55000000000000004, 0.53600000000000003, 0.52600000000000002, 0.534000000000
Manh [0.85799999999999998, 0.80999999999999994, 0.82800000000000007, 0.80999999999999994, 0.774000000000
Mink [0.85199999999999998, 0.84600000000000009, 0.83399999999999996, 0.80999999999999994, 0.841999999999
```

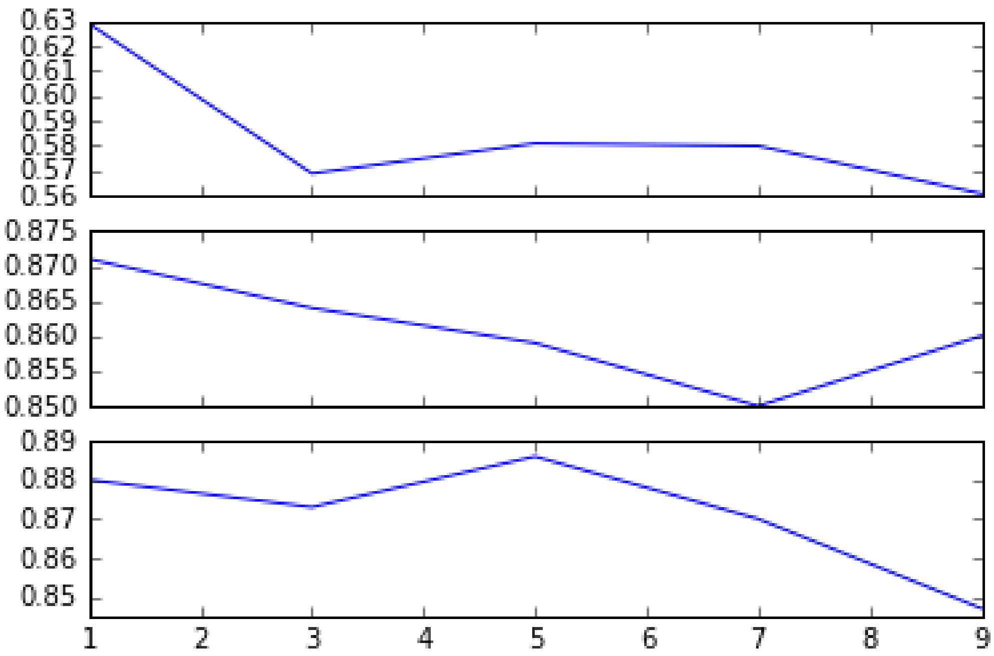
Out[466]:
```
[<matplotlib.lines.Line2D at 0x7fcee8d2ff10>]
```



In [467]:
```
stats_k = [1, 3, 5, 7, 9]
stats_result_Ch = []
stats_result_Ma = []
stats_result_Mi = []

for k in stats_k:
    predictor = KDBasedKNearestNeighbor(k)
    stats_result_Ch.append(my_cross_validation(X_train[:1000], y_train[:1000], predictor, "che
    stats_result_Ma.append(my_cross_validation(X_train[:1000], y_train[:1000], predictor, "man
    stats_result_Mi.append(my_cross_validation(X_train[:1000], y_train[:1000], predictor, "min

print "Cheb " + str(stats_result_Ch)
print "Manh " + str(stats_result_Ma)
print "Mink " + str(stats_result_Mi)

from matplotlib import pyplot

f, axarr = plt.subplots(3, sharex=True)
axarr[0].plot(stats_k, stats_result_Ch)
axarr[1].plot(stats_k, stats_result_Ma)
axarr[2].plot(stats_k, stats_result_Mi)
```

```
Cheb [0.62899999999999989, 0.56900000000000006, 0.58099999999999996, 0.58000000000000007, 0.561000000000
Manh [0.87100000000000011, 0.8640000000000001, 0.85899999999999999, 0.84999999999999998, 0.8599999999999
Mink [0.88000000000000012, 0.873, 0.8859999999999999, 0.87000000000000011, 0.84699999999999986]
```

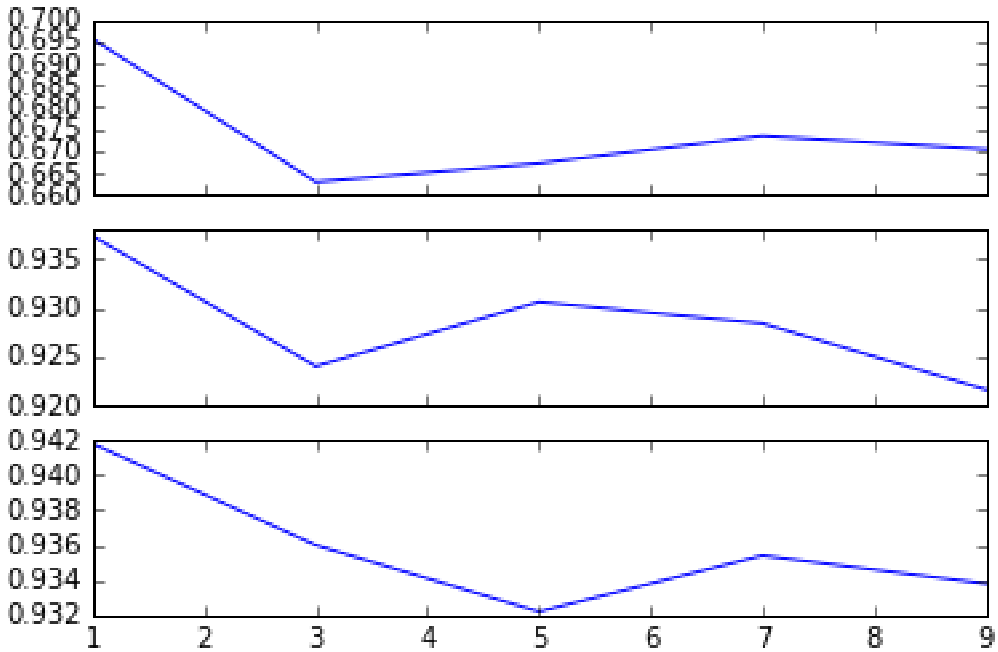
Out[467]: [<matplotlib.lines.Line2D at 0x7fcee91704d0>]



In [471]:
```
stats_k = [1, 3, 5, 7, 9]
stats_result_Ch = []
stats_result_Ma = []
stats_result_Mi = []

for k in stats_k:
    predictor = KDBasedKNearestNeighbor(k)
    stats_result_Ch.append(my_cross_validation(X_train[:5000], y_train[:5000], predictor, "che
    stats_result_Ma.append(my_cross_validation(X_train[:5000], y_train[:5000], predictor, "man
    stats_result_Mi.append(my_cross_validation(X_train[:5000], y_train[:5000], predictor, "min

print "Cheb " + str(stats_result_Ch)
print "Manh " + str(stats_result_Ma)
print "Mink " + str(stats_result_Mi)

from matplotlib import pyplot

f, axarr = plt.subplots(3, sharex=True)
axarr[0].plot(stats_k, stats_result_Ch)
axarr[1].plot(stats_k, stats_result_Ma)
axarr[2].plot(stats_k, stats_result_Mi)
```

```
Cheb [0.69560000000000011, 0.66300000000000003, 0.66720000000000002, 0.6734, 0.6704]
Manh [0.93740000000000001, 0.92400000000000015, 0.93060000000000009, 0.92840000000000011, 0.92160000000
Mink [0.94179999999999997, 0.93600000000000017, 0.93219999999999992, 0.9353999999999999, 0.93379999999
```

Out[471]: [<matplotlib.lines.Line2D at 0x7fcee90fad10>]

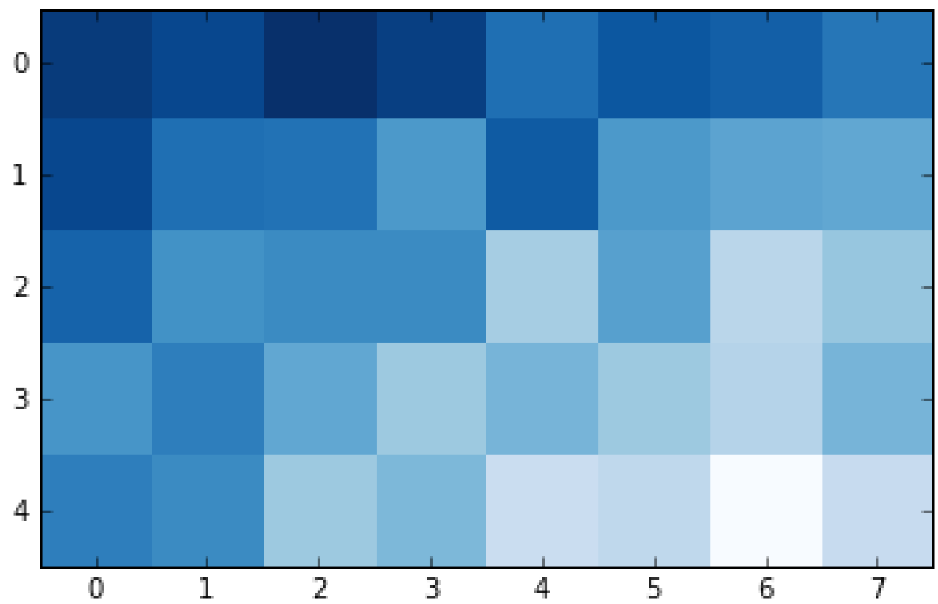


In [ ]:
```
stats_k = [1, 3, 5, 7, 9]
stats_result_Ch = []
```

```
stats_result_Ma = []
stats_result_Mi = []

for k in stats_k:
    predictor = KDBasedKNearestNeighbor(k)
    stats_result_Ch.append(my_cross_validation(X_train, y_train, predictor, "chebyshev", q_fold
    stats_result_Ma.append(my_cross_validation(X_train, y_train, predictor, "manhattan", q_fold
    stats_result_Mi.append(my_cross_validation(X_train, y_train, predictor, "minkowski", q_fold

print "Cheb " + str(stats_result_Ch)
print "Manh " + str(stats_result_Ma)
print "Mink " + str(stats_result_Mi)

from matplotlib import pyplot

f, axarr = plt.subplots(3, sharex=True)
axarr[0].plot(stats_k, stats_result_Ch)
axarr[1].plot(stats_k, stats_result_Ma)
axarr[2].plot(stats_k, stats_result_Mi)
```

In [469]:
```
# let's binarize! -- 15 topping

stats_k = [1, 3, 5, 7, 9]
borders = [0, 10, 15, 20, 50, 80, 100, 120]
stats_result_bin = []

for k in stats_k:
    this_result = []
    predictor = KDBasedKNearestNeighbor(k)

    for b in borders:
        this_result.append(my_cross_validation(binarize_batch(X_train[:500].copy(), b), y_tra
    %time stats_result_bin.append(this_result)

print "bin " + str(stats_result_bin)

from matplotlib import pyplot

plt.imshow(stats_result_bin,
           interpolation="nearest", cmap = "Blues")
plt.show()
```

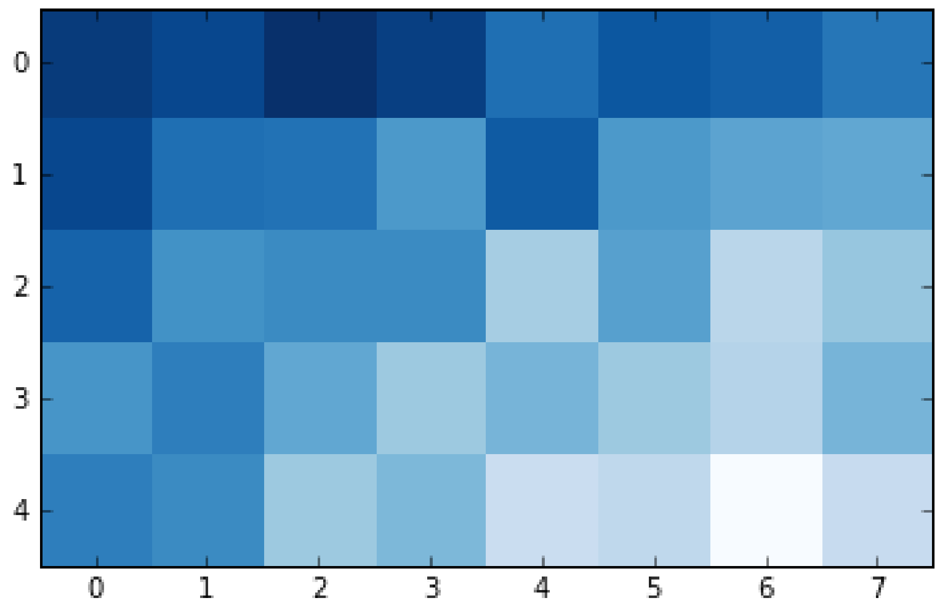
```
CPU times: user 0 ns, sys: 0 ns, total: 0 ns
Wall time: 6.91 µs
CPU times: user 0 ns, sys: 0 ns, total: 0 ns
Wall time: 9.06 µs
CPU times: user 0 ns, sys: 0 ns, total: 0 ns
Wall time: 6.91 µs
CPU times: user 0 ns, sys: 0 ns, total: 0 ns
Wall time: 5.96 µs
CPU times: user 0 ns, sys: 0 ns, total: 0 ns
Wall time: 5.96 µs
bin [[0.8899999999999999, 0.88400000000000001, 0.89600000000000013, 0.8879999999999999, 0.86400000000000
```




```
In [470]: import pprint

          pp = pprint.PrettyPrinter(indent=4)

          pp.pprint(stats_result_bin)

          from matplotlib import pyplot

          plt.imshow(stats_result_bin,
                     interpolation="nearest", cmap = "Blues")
          plt.show()
```

```
[   [   0.8899999999999999,
        0.8840000000000001,
        0.8960000000000013,
        0.8879999999999999,
        0.8640000000000001,
        0.876,
        0.8720000000000011,
        0.8599999999999999],
    [   0.8840000000000001,
        0.8640000000000001,
        0.8619999999999988,
        0.8419999999999997,
        0.874,
        0.8419999999999997,
        0.8359999999999997,
        0.8339999999999996],
    [   0.8699999999999988,
        0.8460000000000009,
        0.8499999999999998,
```

```
     0.84999999999999998,
     0.80999999999999994,
     0.83800000000000008,
     0.80200000000000016,
     0.81600000000000006],
 [   0.84399999999999997,
     0.85600000000000009,
     0.83399999999999996,
     0.81400000000000006,
     0.82599999999999996,
     0.81400000000000006,
     0.80399999999999994,
     0.82599999999999996],
 [   0.85599999999999987,
     0.84999999999999998,
     0.81400000000000006,
     0.82400000000000007,
     0.79399999999999993,
     0.80000000000000004,
     0.76400000000000001,
     0.79600000000000004]]
```



1.11 As long as IMED does realy good, here are the final tests of IMED on the whole dataset

In [493]:
```
predictor = MatrixBasedKNearestNeighbor(k=1)
predictor = predictor.init_G(G_1p01)
%time final_result = my_cross_validation(predictor.ST_batch(binarize_batch(X_train.copy(), 15
```

```
CPU times: user 57min 16s, sys: 4.92 s, total: 57min 21s
Wall time: 57min 21s
```

```
In [264]: print(final_result)
```

```
0.967166666667
```

2 WOW that is WORTH it.

2.1 Getting the final results

```
In [477]: df = pandas.read_csv("kaggle_data/train.csv")
          df_test = pandas.read_csv("kaggle_data/test.csv")
          df_random = pandas.read_csv("kaggle_data/test_random_label.csv")
```

```
In [478]: df_test.head()
```

```
Out[478]:    pixel0  pixel1  pixel2  pixel3  pixel4  pixel5  pixel6  pixel7  pixel8  \
          0  0       0       0       0       0       0       0       0       0
          1  0       0       0       0       0       0       0       0       0
          2  0       0       0       0       0       0       0       0       0
          3  0       0       0       0       0       0       0       0       0
          4  0       0       0       0       0       0       0       0       0

             pixel9    ...     pixel774  pixel775  pixel776  pixel777  pixel778  \
          0  0         ...     0         0         0         0         0
          1  0         ...     0         0         0         0         0
          2  0         ...     0         0         0         0         0
          3  0         ...     0         0         0         0         0
          4  0         ...     0         0         0         0         0

             pixel779  pixel780  pixel781  pixel782  pixel783
          0  0         0         0         0         0
          1  0         0         0         0         0
          2  0         0         0         0         0
          3  0         0         0         0         0
          4  0         0         0         0         0

          [5 rows x 784 columns]
```

```
In [479]: df.head()
```

```
Out[479]:    label  pixel0  pixel1  pixel2  pixel3  pixel4  pixel5  pixel6  pixel7  \
          0  5      0       0       0       0       0       0       0       0
          1  0      0       0       0       0       0       0       0       0
          2  4      0       0       0       0       0       0       0       0
          3  1      0       0       0       0       0       0       0       0
          4  9      0       0       0       0       0       0       0       0

             pixel8    ...     pixel774  pixel775  pixel776  pixel777  pixel778  \
          0  0         ...     0         0         0         0         0
          1  0         ...     0         0         0         0         0
          2  0         ...     0         0         0         0         0
          3  0         ...     0         0         0         0         0
          4  0         ...     0         0         0         0         0

             pixel779  pixel780  pixel781  pixel782  pixel783
          0  0         0         0         0         0
```

```
1  0          0          0          0          0
2  0          0          0          0          0
3  0          0          0          0          0
4  0          0          0          0          0

[5 rows x 785 columns]
```

In [515]:
```
df_random["label"].head()
```

Out[515]:
```
0    4
1    1
2    5
3    6
4    6
Name: label, dtype: int64
```

In [544]:
```
X_train, y_train = df[df.columns[1:]].values, df["label"].values
X_test = df_test[df_test.columns].values
y_random = df_random[df_random.columns].values
```

In [486]:
```
predictor = MatrixBasedKNearestNeighbor(k=1)
predictor = predictor.init_G(G_1p01)
%time predictor = predictor.fit(predictor.ST_batch(binarize_batch(X_train.copy(), 15)), y_trai
```

```
CPU times: user 2min 57s, sys: 610 ms, total: 2min 57s
Wall time: 2min 57s
```

In [494]:
```
predictor = predictor.calc_dist(predictor.ST_batch(binarize_batch(X_test.copy(), 15)), "IMED")
%time y_final = predictor.predict_labels(X_test)
```

```
CPU times: user 3.33 s, sys: 0 ns, total: 3.33 s
Wall time: 3.41 s
```

In [497]:
```
print(y_final[0:20]) #looks good
```

```
[ 7.  2.  1.  0.  4.  1.  4.  9.  5.  9.  0.  6.  9.  0.  1.  5.  9.  7.
  3.  4.]
```

In [496]:
```
plot_grid(X_test[0:20], 1, 20)
```

2.1.1 Let's do some checking

Statistically

In [530]:
```
# random_label, strange..

y_test = y_random[:, 1]

numpy.unique(y_test, return_counts = True)
```

Out[530]:
```
(array([0, 1, 2, 3, 4, 5, 6, 7, 8]),
 array([297, 270, 277, 292, 276, 277, 283, 259, 269]))
```

In [499]:
```
# predicted

from collections import Counter
print(Counter(y_final).keys())
print(Counter(y_final).values())
```

```
[0.0, 1.0, 2.0, 3.0, 4.0, 5.0, 6.0, 7.0, 8.0, 9.0]
[222, 300, 265, 266, 259, 221, 229, 250, 227, 261]
```

Manually

In [535]:
```
#quite good

start = 2000

print(y_final[start:start + 20])

plot_grid(X_test[start:start + 20], 1, 20)
```

```
[ 6.  5.  6.  5.  8.  4.  6.  4.  3.  9.  1.  3.  4.  1.  9.  1.  7.  1.
  1.  9.]
```

2.1.2 Looks good, time to submit. :)

2.1.3 Result output

In [541]:
```
for val in y_final:
    open("prediction.txt", "a").write(str(int(y_final[val])) + '\n')
```

```
/usr/lib/python2.7/site-packages/ipykernel/__main__.py:2: DeprecationWarning: using a non-integer number
  from ipykernel import kernelapp as app
```

2.2 Extra: Official kaggle results – not yet submitted

In [15]:
```
kaggle_df = pandas.read_csv("kaggle_data/4real/train.csv")
kaggle_df_test = pandas.read_csv("kaggle_data/4real/test.csv")
```

In [16]:
```
X_traink, y_traink = kaggle_df[kaggle_df.columns[1:]].values, kaggle_df["label"].values
X_testk = kaggle_df_test[kaggle_df_test.columns].values
```

In [30]:
```
predictork = MatrixBasedKNearestNeighbor(k=1)
predictork = predictork.init_G(G_1p01)
%time predictork = predictork.fit(predictork.ST_batch(binarize_batch(X_traink.copy(), 15)), y_
```

```
CPU times: user 5min 29s, sys: 540 ms, total: 5min 30s
Wall time: 5min 29s
```

In [31]:
```
print(X_testk.shape)
```

```
(28000, 784)
```

```
In [32]: predictork = predictork.calc_dist(predictork.ST_batch(binarize_batch(X_testk[:7000].copy(), 15
         %time y_finalk1 = predictork.predict_labels(X_testk[:7000])
```

```
---------------------------------------------------------------------------

KeyboardInterrupt                         Traceback (most recent call last)

<ipython-input-32-c316438ab545> in <module>()
----> 1 predictork = predictork.calc_dist(predictork.ST_batch(binarize_batch(X_testk[:7000].copy(), 1
      2 get_ipython().magic(u'time y_finalk1 = predictork.predict_labels(X_testk[:7000])')


/home/aq/workspace/diht/ML/symbols/knn.pyc in calc_dist(self, X_test, metric, k)
    226             # Fill matrix self.dist_mt by using 0 loops                             #
    227             #########################################################################
--> 228             self.dist_mt = numpy.apply_along_axis(set_row, 1, X_test)
    229
    230         # print(self.dist_mt)


/usr/lib/python2.7/site-packages/numpy/lib/shape_base.pyc in apply_along_axis(func1d, axis, arr,
    126                 n -= 1
    127             i.put(indlist, ind)
--> 128             res = func1d(arr[tuple(i.tolist())], *args, **kwargs)
    129             outarr[tuple(i.tolist())] = res
    130             k += 1


/home/aq/workspace/diht/ML/symbols/knn.pyc in set_row(X_test_i)
    205
    206         def set_row(X_test_i):
--> 207             return numpy.apply_along_axis(self.dist, 1, self.X_train, X_test_i, metric=metric


/usr/lib/python2.7/site-packages/numpy/lib/shape_base.pyc in apply_along_axis(func1d, axis, arr,
    105                 n -= 1
    106             i.put(indlist, ind)
--> 107             res = func1d(arr[tuple(i.tolist())], *args, **kwargs)
    108             outarr[tuple(ind)] = res
    109             k += 1


/home/aq/workspace/diht/ML/symbols/knn.pyc in dist(self, img1, img2, metric)
    176             return self.dist_pixel_L3(img1, img2)
    177         elif metric == "IMED":
--> 178             return self.dist_IMED(img1, img2)
    179         else:
    180             print("Unknown metric")


/home/aq/workspace/diht/ML/symbols/knn.pyc in dist_IMED(self, img_x, img_y, G)
```

```
    156         http://www.cis.pku.edu.cn/faculty/vision/wangliwei/pdf/IMED.pdf
    157         """
--> 158         dist = (numpy.transpose(img_x - img_y)).dot(img_x - img_y)
    159         # for row_i in range(self.size):
    160         #     for col_i in range(self.size):


KeyboardInterrupt:
```

In [3]:
```python
print(y_finalk[0:20])
```

```
9
2
3
2
0
3
2
0
2
0
```

In [24]:
```python
plot_grid(X_testk[0:20], 1, 20)
```

In [6]:
```python
for index in range(len(y_finalk)):
    open("predictionkaggle.csv", "a").write(str(index) + ", " + str(y_finalk[index]) + '\n')
```

In [9]:
```python
f = open("predictionkaggle.txt", "r")
numpy.loadtxt(f)
```

Out[9]:
```
array([ 9.,  2.,  3., ...,  0.,  3.,  9.])
```

In [ ]: