INF 552 Project Report: Classify Handwritten Digits using MNIST Dataset

Aamir Goriawala, Kalyanee Chendke, Rohit Date

University of Southern California, California, USA

1. Introduction:

The aim of this project is to implement a classification algorithm to recognize handwritten digits (0- 9). This report presents our implementation of the Principal Component Analysis (PCA) combined with k-Nearest Neighbor to recognize the numeral digits. Other methods that we have used are support vector machine, Stochastic Gradient Descent and Neural Networks. We were able to achieve an accuracy of 97.25% using k-Nearest neighbor and 98.16% using support vector machines. The accuracy using stochastic gradient descent was 90.32%, whereas we consistently obtained an accuracy of more than 97% using neural networks.

2. Related Work:

The paper "Comparison of learning algorithms for handwritten digit recognition" gives us a detailed account of what goes on into the classification of handwritten digits. The paper "Handwritten digit classification" provided a lot of information about the classifiers and tools to be used for an efficient classification of digits. Although the major work done by them was in Matlab, we planned to code the classifiers in python.

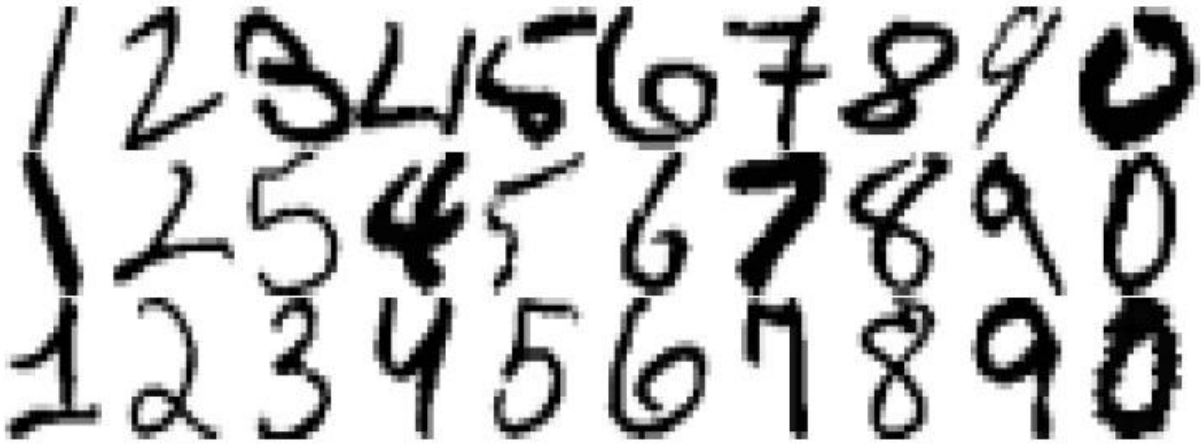
The paper mentioned above gave us a clear idea of what exactly is the role of principal component analysis to provide ease of computation as well as to avoid the curse of dimensionality. The k-nearest neighbor classifier was chosen to experiment with a non-parametric method in the absence of a known distribution for the data.

The tools majorly used by them to work in Matlab were PRtools toolbox. We have majorly used Scikit learn toolbox for implementing the classifiers in python.

3. Data Description:

The MNIST database of handwritten digits has a training set of 60,000 examples, and a test set of 10,000 examples. The data files train.csv and test.csv contain gray-scale images of hand-drawn digits, from zero through nine.

Each image is 28 pixels in height and 28 pixels in width, for a total of 784 pixels in total. Each pixel has a single pixel-value associated with it, indicating the lightness or darkness of that pixel, with higher numbers meaning darker. This pixel-value is an integer between 0 and 255, inclusive. The training data set, (train.csv), has 785 columns, each column consisting of the respective pixel value at that location. The pixel value denotes the brightness at that pixel.



4 Feature Extraction using Principal Component Analysis:

Principal component analysis (PCA) is a fundamental multivariate data analysis method which is encountered into a variety of areas in neural networks, signal processing, and machine learning. It is an unsupervised method for reducing the dimensionality of the existing data set and extracting important information. The digit dataset has 784 dimensions, which will naturally take a lot of time to give a result. Using PCA, we reduced the dimensions of the dataset from 784 to 100. This was obtained through the plot of 'number of components vs variance', with the aim of preserving maximum possible variance along with keeping the accuracy maximum possible.

5. Classifiers Used:

5.1. Support Vector Machine:

We have used libsvm for training and testing the support vector machine (SVM) classifiers.

We have used the Radial Basis Function kernel to tackle the issue of huge amount of features. By default, libsvm selects gamma=1/d, where d is the number of features. However, the training in this case was time-consuming. We therefore changed the value of gamma to 0.03125. The value of gamma was calculated by using grid search.

5.2. K- Nearest Neighbors:

As a non-parametric approach, the k-Nearest Neighbor classifier uses all the training patterns as prototypes. The classification accuracy is influenced by the number of nearest neighbor k. We thus try different k (k=1, 3, 5, 7, 9) and obtain the test error rate for each classifier. The dataset was shuffled to be assured that there will not be a corresponding one-to-one mapping which will give a misdirected result. The highest accuracy is mostly given by k = 3. We hence have used 3-NN classifier.

5.3. Stochastic Gradient Descent:

We have implemented the stochastic gradient descent method for classification of handwritten digits. We have made use of the DBLearn library which provides a variety of parameters and tools which make implementation of stochastic gradient descent convenient and efficient. We have taken parameters loss='hinge', learning rate='optimal', which basically provides us with the optimal learning rate and hinge loss. Accuracy obtained using the stochastic gradient descent algorithm is 90.32%.

5.4. Neural Networks:

We have implemented a 2-layer neural network to do the digit recognition on the MNIST dataset. The neural network has 300 hidden units. The learning rate is taken as 0.3 and the increment rate is taken as 0.9. The learning rate was multiplied by the increment rate after each epoch. We used the nolearn.dbn module for implementing the neural network algorithm.

6. Experimental Setup:

The training data consists of 42000 samples of digits denoted through their pixel brightness. The brightness values lie between 0 and 255 which are normalized before the samples are fed to the algorithm for training. 28000 test samples are then fed to the classifier model.

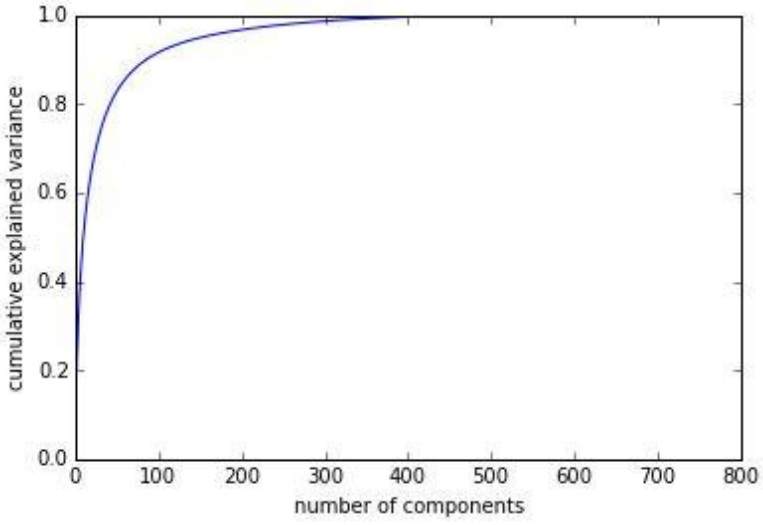
7. Results:

7.1. k-Nearest Neighbor:

We implemented k-Nearest Neighbor classifier for multiple values of k (k = 3,5,10,20)

The best accuracy among our test cases was provided by k=3 and was 97.25%

We have used principal component analysis to reduce the dimensionality of the datatset from 784 to 100. The number of components = 100 was obtained using the plot of 'number of components' vs 'variance'. We aimed to preserve about 92% of the variance, while keeping the accuracy as best as possible. With this motive, we have come up with components = 100.



We have provided the precision, recall, F1 score, confusion matrix, accuracy and runtime for the cases as given below:

```
FOR K=  3
              precision    recall  f1-score   support

           0       0.98      0.99      0.98       813
           1       0.98      1.00      0.99       961
           2       0.98      0.98      0.98       860
           3       0.97      0.97      0.97       863
           4       0.98      0.95      0.97       827
           5       0.98      0.97      0.97       756
           6       0.98      1.00      0.99       841
           7       0.97      0.97      0.97       899
           8       0.99      0.94      0.96       768
           9       0.94      0.95      0.94       812

avg / total       0.97      0.97      0.97      8400

Confusion matrix:
[[806   0   1   0   1   1   3   0   1   0]
 [  0 957   2   0   0   0   1   1   1   0]
 [  7   3 839   1   0   0   1   7   2   0]
 [  2   0   6 835   0   5   0   5   6   4]
 [  1   6   0   0 789   0   3   1   0  27]
 [  2   0   1   7   1 736   7   0   1   1]
 [  0   1   0   0   1   2 837   0   0   0]
 [  0   6   7   0   2   0   0 876   0   8]
 [  3   6   1  11   4   6   5   0 725   7]
 [  5   1   1   4  10   4   2  16   0 769]]

Accuracy:97.250000
Runtime:
226.94  seconds
```

```
FOR K=  5
              precision    recall  f1-score   support

           0       0.98      0.99      0.98       813
           1       0.98      0.99      0.99       961
           2       0.98      0.97      0.98       860
           3       0.97      0.97      0.97       863
           4       0.98      0.96      0.97       827
           5       0.96      0.97      0.96       756
           6       0.97      1.00      0.98       841
           7       0.96      0.97      0.97       899
           8       0.99      0.94      0.96       768
           9       0.95      0.95      0.95       812

avg / total       0.97      0.97      0.97      8400

Confusion matrix:
[[806   0   2   0   0   2   3   0   0   0]
 [  0 956   2   1   0   0   0   1   1   0]
 [  8   4 834   3   0   0   1   9   1   0]
 [  1   1   4 837   0   6   0   6   4   4]
 [  1   5   0   0 797   0   3   2   0  19]
 [  3   0   1   8   1 731  10   0   0   2]
 [  0   1   0   0   0   3 837   0   0   0]
 [  0   8   5   1   2   0   0 875   0   8]
 [  2   4   0   9   3  15   8   1 719   7]
 [  5   1   2   3  10   4   2  16   0 769]]

Accuracy:97.154762
Runtime:
244.25  seconds
```

```
FOR K=  10
              precision    recall  f1-score   support

           0       0.97      0.99      0.98       813
           1       0.96      0.99      0.98       961
           2       0.98      0.97      0.98       860
           3       0.96      0.97      0.97       863
           4       0.98      0.96      0.97       827
           5       0.96      0.97      0.97       756
           6       0.96      0.99      0.98       841
           7       0.96      0.97      0.96       899
           8       0.99      0.91      0.95       768
           9       0.95      0.94      0.94       812

avg / total       0.97      0.97      0.97      8400

Confusion matrix:
[[805   0   2   0   0   2   4   0   0   0]
 [  0 953   4   0   1   0   1   1   1   0]
 [ 11   5 831   2   0   0   2   7   2   0]
 [  1   2   2 838   0   4   0   7   5   4]
 [  1   7   0   0 795   0   4   2   0  18]
 [  0   2   0   9   0 731  10   0   0   4]
 [  1   1   0   0   1   4 834   0   0   0]
 [  0  10   3   0   2   0   0 874   0  10]
 [  4   6   0  18   4  16   9   2 701   8]
 [  5   2   2   6  10   1   2  21   0 763]]

Accuracy:96.726190
Runtime:
249.23  seconds
```

```
FOR K=  20
              precision    recall  f1-score   support

           0       0.97      0.99      0.98       813
           1       0.95      0.99      0.97       961
           2       0.98      0.95      0.97       860
           3       0.96      0.96      0.96       863
           4       0.98      0.95      0.96       827
           5       0.96      0.96      0.96       756
           6       0.96      0.99      0.98       841
           7       0.95      0.97      0.96       899
           8       0.98      0.90      0.94       768
           9       0.93      0.94      0.93       812

avg / total       0.96      0.96      0.96      8400

Confusion matrix:
[[805   0   2   0   0   2   4   0   0   0]
 [  0 955   3   0   0   0   2   0   1   0]
 [ 13  11 820   2   1   0   1   8   4   0]
 [  1   3   5 830   0   5   0   8   6   5]
 [  1   8   0   0 782   0   6   2   0  28]
 [  0   2   0  10   0 729  10   0   0   5]
 [  1   1   0   0   0   3 836   0   0   0]
 [  0  14   3   0   1   0   0 870   0  11]
 [  5  10   0  18   5  21   7   3 689  10]
 [  5   3   2   6  10   3   1  22   0 760]]

Accuracy:96.142857
Runtime:
245.64  seconds
```

7.2. Support Vector Machine:

We implemented support vector machine with cost value 100000 and gamma = 0.03125. We have used the RBF kernel to tackle the issue of multiple features.We have obtained an accuracy of 98.16%. Given below is the output screenshot of the grid search method we have used to compute the cost value and gamma:

```
scaling
grid search
Fitting 5 folds for each of 20 candidates, totalling 100 fits
[Parallel(n_jobs=-1)]: Done   1 jobs       | elapsed:    8.6s
[Parallel(n_jobs=-1)]: Done  32 jobs       | elapsed: 12.7min
[Parallel(n_jobs=-1)]: Done  94 out of 100 | elapsed: 50.3min remaining:  3.2min
[CV] C=100000.0, gamma=0.03125 ....................................
[CV] ............. C=100000.0, gamma=0.03125, score=0.999500 -   7.4s
[CV] C=100000.0, gamma=0.0625 .....................................
[CV] ............. C=100000.0, gamma=0.0625, score=0.997001 -  27.5s
[CV] C=100000.0, gamma=0.0625 .....................................
[CV] ............. C=100000.0, gamma=0.0625, score=0.996498 -  29.0s
[CV] C=100000.0, gamma=0.125 ......................................
[CV] ............. C=100000.0, gamma=0.125, score=0.988494 - 1.5min
[CV] C=100000.0, gamma=0.25 .......................................
[CV] ............. C=100000.0, gamma=0.25, score=0.961000 - 2.0min
[CV] C=1000000.0, gamma=0.03125 ...................................
[CV] ............. C=1000000.0, gamma=0.03125, score=0.997999 -  29.1s
[CV] C=1000000.0, gamma=0.0625 ....................................
[CV] ............. C=1000000.0, gamma=0.0625, score=0.997001 - 1.9min
[CV] C=1000000.0, gamma=0.0625 ....................................
[CV] ............. C=1000000.0, gamma=0.0625, score=0.998500 - 1.5min
[CV] C=1000000.0, gamma=0.0625 ....................................
[CV] ............. C=1000000.0, gamma=0.0625, score=0.996498 - 1.4min
[CV] C=1000000.0, gamma=0.125 .....................................
[CV] ............. C=1000000.0, gamma=0.125, score=0.988494 - 3.8min
[CV] C=1000000.0, gamma=0.25 ......................................
[CV] ............. C=1000000.0, gamma=0.25, score=0.961000 - 4.8min
[CV] C=10000000.0, gamma=0.03125 ..................................
[CV] ............ C=10000000.0, gamma=0.03125, score=0.999500 -  15.9s
[CV] C=10000000.0, gamma=0.03125 ..................................
[CV] ............ C=10000000.0, gamma=0.03125, score=0.997999 -  16.3s
[CV] C=10000000.0, gamma=0.0625 ...................................
```

```
[CV] ..... C=10000000.0, gamma=0.0625, score=0.997001 - 1.1min
[CV] C=10000000.0, gamma=0.0625 ......................................
[CV] ..... C=10000000.0, gamma=0.0625, score=0.998500 - 1.1min
[CV] C=10000000.0, gamma=0.125 .......................................
[CV] ..... C=10000000.0, gamma=0.125, score=0.989505 - 3.4min
[CV] C=10000000.0, gamma=0.125 .......................................
[CV] ..... C=10000000.0, gamma=0.125, score=0.983992 - 3.4min
[CV] C=10000000.0, gamma=0.25 ........................................
[CV] ..... C=10000000.0, gamma=0.25, score=0.969485 - 4.6min
[CV] C=100000000.0, gamma=0.0625 .....................................
[CV] ..... C=100000000.0, gamma=0.0625, score=0.996498 - 1.3min
[CV] C=100000000.0, gamma=0.125 ......................................
[CV] ..... C=100000000.0, gamma=0.125, score=0.989000 - 4.0min
[CV] C=100000000.0, gamma=0.25 .......................................
[CV] ..... C=100000000.0, gamma=0.25, score=0.949525 - 4.3min
[CV] C=1000000000.0, gamma=0.03125 ...................................
[CV] ..... C=1000000000.0, gamma=0.03125, score=0.999000 -  15.4s
[CV] C=1000000000.0, gamma=0.03125 ...................................
[CV] ..... C=1000000000.0, gamma=0.03125, score=0.999500 -  16.3s
[CV] C=1000000000.0, gamma=0.03125 ...................................
[CV] ..... C=1000000000.0, gamma=0.03125, score=0.998999 -  16.3s
[CV] C=1000000000.0, gamma=0.0625 ....................................
[CV] ..... C=1000000000.0, gamma=0.0625, score=0.998501 - 1.1min
[CV] C=1000000000.0, gamma=0.0625 ....................................
[CV] ..... C=1000000000.0, gamma=0.0625, score=0.995998 - 1.1min
[CV] C=1000000000.0, gamma=0.125 .....................................
[CV] ..... C=1000000000.0, gamma=0.125, score=0.987506 - 4.1min
[CV] C=1000000000.0, gamma=0.25 ......................................
[CV] ..... C=1000000000.0, gamma=0.25, score=0.964518 - 5.1min
[CV] C=1000000000.0, gamma=0.25 ......................................
[CV] ..... C=1000000000.0, gamma=0.25, sc[Parallel(n_jobs=-1)]: Done 100 out of 100 | elapsed: 58.9min finished
predicting
score:  0.2115
```

```
SVC(C=100000.0, cache_size=1000, class_weight=None, coef0=0.0, degree=3,
  gamma=0.03125, kernel='rbf', max_iter=-1, probability=False,
  random_state=None, shrinking=True, tol=0.001, verbose=False)
```

The grid search computed the best value of C (cost value) and gamma out of the 20 possible combinations of C and gamma provided to the grid search algorithm.

The best value of C obtained was 100000. The best value for gamma was 0.03125

Given below is the output of the support vector machine classifier we have used to classify the handwritten digits:

```
Classification report for classifier SVC(C=100000.0, cache_size=1000, class_weight=None, coef0=0.0, degree=3,
  gamma=0.03125, kernel='rbf', max_iter=-1, probability=False,
  random_state=None, shrinking=True, tol=0.001, verbose=False):
             precision    recall  f1-score   support

        0.0       0.98      0.99      0.99       813
        1.0       0.99      0.99      0.99       961
        2.0       0.98      0.99      0.98       860
        3.0       0.98      0.96      0.97       863
        4.0       0.98      0.98      0.98       827
        5.0       0.98      0.98      0.98       756
        6.0       0.98      0.99      0.99       841
        7.0       0.98      0.98      0.98       899
        8.0       0.97      0.99      0.98       768
        9.0       0.98      0.97      0.97       812

avg / total       0.98      0.98      0.98      8400


Confusion matrix:
[[803   0   2   0   1   1   3   0   3   0]
 [  0 953   3   2   0   0   0   1   2   0]
 [  2   0 850   1   0   0   0   4   3   0]
 [  1   1   5 831   1   8   0   4   9   3]
 [  1   3   1   0 809   1   3   2   0   7]
 [  2   0   0   4   0 743   5   0   2   0]
 [  1   0   0   0   2   2 835   0   1   0]
 [  0   3   7   0   3   2   0 880   0   4]
 [  1   2   0   2   1   3   2   0 757   0]
 [  5   1   1   4   6   1   0   6   3 785]]
('Accuracy', 0.98166666666666669)
```

7.3. Stochastic Gradient Descent:

We have implemented stochastic gradient descent. The following are the parameters we have chosen for the algorithm:

The tool Sgdclassifier was used to implement this algorithm with the following parameters:

Number of Iterations= 20, Learning Rate='optimal' , Loss='hinge', Shuffle = 'true'

```
stochastic gradient descent accuracy:  0.903214285714
             precision    recall  f1-score   support

          0       0.97      0.94      0.95       813
          1       0.96      0.97      0.97       961
          2       0.88      0.92      0.90       860
          3       0.93      0.83      0.88       863
          4       0.92      0.92      0.92       827
          5       0.79      0.92      0.85       756
          6       0.93      0.95      0.94       841
          7       0.88      0.95      0.91       899
          8       0.85      0.83      0.84       768
          9       0.95      0.78      0.86       812

avg / total       0.91      0.90      0.90      8400

Confusion matrix:
[[762   1  10   2   2  12  17   2   5   0]
 [  0 936   9   3   0   4   2   1   6   0]
 [  3   5 787   7   8   8   7  15  20   0]
 [  3   3  33 716   0  58   5  16  22   7]
 [  1   8  10   1 761   3   9   5  12  17]
 [  4   2   7  11   3 694   8   5  21   1]
 [  1   0  10   1   4  18 802   0   5   0]
 [  1   4  13   2   9   4   3 853   3   7]
 [  5  13   9  18   4  57  11   6 640   5]
 [  9   3   5   9  40  21   1  70  18 636]]
```

7.4. Neural Networks:

We have implemented neural network with learning rate as 0.3 and increment rate as 0.9. We obtained an accuracy of more than 97% on each run.

```
neural network accuracy:  0.977619047619
Classification report:
             precision    recall  f1-score   support

        0.0       0.99      0.99      0.99       813
        1.0       0.99      0.99      0.99       961
        2.0       0.97      0.98      0.98       860
        3.0       0.98      0.96      0.97       863
        4.0       0.99      0.97      0.98       827
        5.0       0.98      0.98      0.98       756
        6.0       0.97      0.99      0.98       841
        7.0       0.98      0.98      0.98       899
        8.0       0.96      0.97      0.97       768
        9.0       0.97      0.97      0.97       812

avg / total       0.98      0.98      0.98      8400

Confusion matrix:
[[802   0   0   0   1   0   4   1   4   1]
 [  0 950   3   3   0   0   1   2   2   0]
 [  2   0 845   2   0   0   1   3   6   1]
 [  1   1   5 832   0   9   0   2   9   4]
 [  1   4   1   1 800   1   5   3   0  11]
 [  0   0   0   4   0 739   6   3   4   0]
 [  2   0   0   0   1   1 836   0   1   0]
 [  0   4   9   1   2   2   0 878   1   2]
 [  1   5   2   2   0   3   7   0 746   2]
 [  3   0   2   4   5   2   1   8   3 784]]
```

8. Source Code:

8.1. k-Nearest Neighbor:

```
# -*- coding: utf-8 -*-

import pandas as pd

import numpy as np

import time

from sklearn import metrics

from sklearn import decomposition

from sklearn.cross_validation import train_test_split

from sklearn.neighbors import KNeighborsClassifier
```

```
from sklearn.metrics import classification_report
from sklearn.metrics import confusion_matrix
from sklearn.decomposition import PCA
from sklearn.metrics import accuracy_score
import csv
import matplotlib.pyplot as plt
train = pd.read_csv("train.csv")
features = train.columns[1:]
X = train[features]
y = train['label']
X_train, X_test, y_train, y_test= train_test_split(X, y, test_size= 0.2, random_state=0)
pca=PCA().fit(X_train)
plt.plot(np.cumsum(pca.explained_variance_ratio_))
plt.xlabel('number of components')
plt.ylabel('cumulative explained variance')
plt.show()
PCA_COMPONENTS = 100
pca=decomposition.PCA(n_components=PCA_COMPONENTS).fit(X_train)
X_train_reduced = pca.transform(X_train)
accuracy_dict={}
values_dict={}
for k in [3,5,10,20]:
    start_time=time.time()
    print "FOR K= ",k
    clf=KNeighborsClassifier(k)
    clf.fit(X_train_reduced, y_train)
    X_test_reduced = pca.transform(X_test)
    y_pred=clf.predict(X_test_reduced)
    values_dict[k]=y_pred
```

```
    print classification_report(y_test, y_pred)
    print("Confusion matrix:\n%s" % metrics.confusion_matrix(y_test,y_pred))
    acc = accuracy_score(y_test, y_pred)
    accuracy_dict[k]=acc
    print("\n")
    print("Accuracy:%f" %(acc*100))
    print("Runtime:" )
    print round((time.time()-start_time),2)," seconds"
max_key=max(accuracy_dict, key=accuracy_dict.get)
with open('KNN_results.csv','w')as f:
    writer=csv.writer(f,lineterminator='\n')
    writer.writerow(["ImageID","Label"])
    i=1
    for val in values_dict[max_key]:
        writer.writerow([i,val])
        i+=1
```

8.2. Support Vector Machine:

Grid Search:

```
# -*- coding: utf-8 -*-
from sklearn.grid_search import GridSearchCV
from sklearn.cross_validation import StratifiedKFold
from sklearn.svm import SVC
import numpy as np
import csv
from sklearn import *
def main():
    train=[]
    f1=open("train.csv","r")
```

```
    datareader=csv.reader(f1)
    for row in datareader:
        intlist=[]
        #scaling
        intlist = [ round((int(x)/255.),2) for x in row]
        train.append(intlist)
    train_labels=[]
    f2=open("train_labels.csv","r")
    datareader2=csv.reader(f2)
    for row in datareader2:
        labellist=[]
        labellist=[float(x) for x in row]
        train_labels.extend(labellist)
    X_train,X_test,y_train,y_test=
cross_validation.train_test_split(train,train_labels,test_size=0.2,random_state=0)
    svm = SVC(cache_size=1000, kernel='rbf')
    C_range = 10. ** np.arange(5,10)
    gamma_range = 2. ** np.arange(-5, -1)
    parameters = dict(gamma=gamma_range, C=C_range)
    print("grid search")
    grid = GridSearchCV(svm, parameters, cv=StratifiedKFold(y_train, 5), verbose=3, n_jobs=-1)
    grid.fit(X_train, y_train)
    print("predicting")
    print "score: ", grid.score(X_test, y_test)
    print("The best classifier is: ", grid.best_estimator_)
if __name__ == "__main__":
    main()
```

Support Vector Machine:

```
# -*- coding: utf-8 -*-
from sklearn import svm, metrics
from numpy import *
from sklearn import *
from sklearn.metrics import accuracy_score
import csv
import sys
train=[]
f1=open("train.csv","r")
datareader=csv.reader(f1)
for row in datareader:
    intlist=[]
    #scaling
    intlist = [ round((int(x)/255.),2) for x in row]
    train.append(intlist)
train_labels=[]
f2=open("train_labels.csv","r")
datareader2=csv.reader(f2)
for row in datareader2:
    labellist=[]
    labellist=[float(x) for x in row]
    train_labels.extend(labellist)
test=[]
f3=open("test.csv","r")
datareader3=csv.reader(f3)
for row in datareader3:
    intlist=[]
```

```
    intlist = [round((int(x)/255.),2) for x in row]
    test.append(intlist)
print "Applying a learning algorithm..."
clf=svm.SVC(C=64, cache_size=1000, class_weight=None, coef0=0.0, degree=3,
  gamma=0.03125, kernel='rbf', max_iter=-1, probability=False,
  random_state=None, shrinking=True, tol=0.001, verbose=False)
clf.fit(train,train_labels)
predicted = clf.predict(test)
#print predicted[0:100]
with open('SVM_results.csv','w')as f:
    writer=csv.writer(f,lineterminator='\n')
    writer.writerow(["ImageID","Label"])
    i=1
    for val in predicted:
        writer.writerow([i,val])
        i+=1
```

8.3. Stochastic Gradient Descent:

```
# -*- coding: utf-8 -*-
import pandas as pd
from sklearn import cross_validation
from sklearn.linear_model import SGDClassifier
from sklearn.metrics import accuracy_score
import matplotlib.pyplot as plt
from nolearn.dbn import DBN
from sklearn.metrics import classification_report
from sklearn.metrics import confusion_matrix
from sklearn import metrics
```

```
train = pd.read_csv("train1.csv")
features = train.columns[1:]
X = train[features]
y = train['label']
X_train,X_test, y_train, y_test = cross_validation.train_test_split(X/255.,y,test_size=0.2,random_state=0)
clf_sgd = SGDClassifier(learning_rate='optimal', loss='hinge', n_iter=20, shuffle=True)
clf_sgd.fit(X_train, y_train)
y_pred_sgd = clf_sgd.predict(X_test)
acc_sgd = accuracy_score(y_test, y_pred_sgd)
print "stochastic gradient descent accuracy: ",acc_sgd
print classification_report(y_test, y_pred_sgd)
print("Confusion matrix:\n%s" % metrics.confusion_matrix(y_test,y_pred_sgd))
```

8.4. Neural Network:

```
# -*- coding: utf-8 -*-
import numpy as np
from sklearn.cross_validation import train_test_split
from sklearn.datasets import fetch_mldata
from sklearn.metrics import classification_report
from sklearn.metrics import confusion_matrix
from sklearn import *
from sklearn.metrics import classification_report
import csv
train=[]
f1=open("train3.csv","r")
datareader=csv.reader(f1)
for row in datareader:
    intlist=[]
    intlist = [ round((int(x)/255.),2) for x in row]
```

```
    train.append(intlist)
train_labels=[]
f2=open("train_labels.csv","r")
datareader2=csv.reader(f2)
for row in datareader2:
    labellist=[]
    labellist=[float(x) for x in row]
    train_labels.extend(labellist)
X_train,X_test,y_train,y_test=
cross_validation.train_test_split(train,train_labels,test_size=0.2,random_state=0)
print "Applying a learning algorithm..."
from nolearn.dbn import DBN
X_train=np.array(X_train)
X_test=np.array(X_test)
y_train=np.array(y_train)
y_test=np.array(y_test)
clf = DBN(
    [X_train.shape[1], 300, 10],
    learn_rates=0.3,
    learn_rate_decays=0.9,
    epochs=15,
    verbose=1,)
clf.fit(X_train, y_train)
acc_nn = clf.score(X_test,y_test)
print "neural network accuracy: ",acc_nn
y_pred = clf.predict(X_test)
print "Classification report:"
print classification_report(y_test, y_pred)
print("Confusion matrix:\n%s" % metrics.confusion_matrix(y_test,y_pred))
```

```
with open('Neural_Network_results.csv','w')as f:
    writer=csv.writer(f,lineterminator='\n')
    writer.writerow(["ImageID","Label"])
    i=1
    for val in y_pred:
        writer.writerow([i,val])
        i+=1
```

9. Conclusion:

We have implemented Support Vector Machine, k-Nearest Neighbor and Stochastic gradient descent on the MNIST dataset. We used Principal Component Analysis for feature extraction, which was then fed to k-Nearest Neighbor. Dimensionality reduction drastically reduced the runtime of k-NN classifier, which was the major advantage of its implementation. The accuracy obtained from k-Nearest neighbor, precisely for k=3, is 97.25%, while the accuracy obtained using SVM is 98.16%. The accuracy of stochastic gradient descent is 90.32%. Although the runtime for SVM classifier is found to be the highest among the classifiers we have implemented, it gives the best accuracy among the three. The accuracy obtained using neural networks was consistently above 97%.

10. References:

[1] Y. Lecun et al., "Comparison of learning algorithms for handwritten digit recognition," in International Conference on Artificial Neural Networks, France.

[2] Abrita Chakravarty,William Day,"Handwritten Digit Classification", Michigan State University.

[3] http://scikit-learn.org/stable/modules/svm.html

[4] http://yann.lecun.com/exdb/publis/pdf/schaul-icml-13.pdf

[5] B. El Kessab, C. Daoui, B. Bouikhalene and R. Salouan, "A Comparative Study between the Support Vectors Machines and the K-Nearest Neighbors in the Handwritten Latin Numerals Recognition"

[6] Seiji HOTTA, Senya KIYASU, and Sueharu MIYAHARA, "A Classifier Based on Distance Between Test Samples and Average Patterns of Categorical Nearest Neighbors"