Jose Caballero  
Dr. Olac Fuentes  
CS 5356 Computer Vision

**Problem Description**

The problem given to us for this lab was to just experiment with features and learning algorithms for image-based object classification. The task was to find the best classifier for MNIST dataset using SKlearn library.

**Algorithms Implemented**

The algorithms implemented were both K Nearest Neighbors and MLP (Multi-layer Perceptron) that are already come from the SKLearn package. Another library used was the PCA (Principal comeponent analysis) decomposition of data as well.

For both runs of MLP and KNN, I had set a PCA algorithm in order to increase accuracy results. The following line is where I create said class with 70 components. The number of components is tricky, and I really just experimented with numbers to see how my accuracy changed with the number of components. It really did not make such a big difference but I got consistent results with 70 and decided to leave it like that.

**pca = PCA(n\_components=70, svd\_solver='full')**

**Experimental Results**

**MLP**

Using the MLP classifier for the MNIST dataset, the maximum I could get was around an 85-86% accuracy using the PCA decomposition first. Without PCA, I was getting 84% accuracy, which means using PCA only improved my accuracy by 1 percent. With MLP, I was getting time completions of about 0.90-1.0 seconds.

The following line of code is how I initiated my MLP classifier. I’m using different parameters that are passed to the constructor. To find these, I brute forced all of them since they are 3-4 choices per attribute, and these are the ones that I had the greatest accuracy with.

MLP = MLPClassifier(solver='adam', activation='relu', alpha=1, batch\_size='auto')

*MLP accuracy:* ***0.855***

**precision recall f1-score support**

0 0.94 0.95 0.95 175

1 0.94 0.98 0.96 234

2 0.91 0.85 0.88 219

3 0.87 0.76 0.81 207

4 0.88 0.83 0.86 217

5 0.74 0.78 0.76 179

6 0.85 0.90 0.87 178

7 0.83 0.82 0.83 205

8 0.84 0.78 0.81 192

9 0.75 0.88 0.81 194

**avg / total 0.86 0.85 0.85 2000**

Elapsed time: 0.95s

*MLP accuracy :* ***0.858***

**precision recall f1-score support**

0 0.93 0.97 0.95 175

1 0.94 0.99 0.96 234

2 0.90 0.85 0.88 219

3 0.86 0.78 0.82 207

4 0.88 0.84 0.86 217

5 0.77 0.78 0.77 179

6 0.84 0.88 0.86 178

7 0.84 0.84 0.84 205

8 0.83 0.77 0.79 192

9 0.78 0.87 0.82 194

**avg / total 0.86 0.86 0.86 2000**

Elapsed time: 0.94s

These tables were produced using the classification report method that is part of the MLP library in SKLearn.

**KNN**

Using the KNN classifier for the MNIST dataset, I was getting consistent results of 83% accuracy with an elapsed time of ~0.04s.

The following line of code is how I initiated my KNN classifier. I passed 3 parameters which were the number of neighbors to use, weights, and the algorithm to use. The number of neighbors was basically again trial and error, using increments of odd numbers. The algorithm surprisingly did not change the accuracy results for some reason, but I did notice it greatly affected the time it took. For the other algorithms (ball tree, kd tree, and auto), the time increased by atleast a second, and brute greatly reduced the time, which seemed weird to me. I would say because we are only taking a small amount of data, and that is why brute is performing better. If we were to take the whole data, I am sure it would not be the case where brute performs faster than the others.

knn = KNeighborsClassifier(n\_neighbors=7, weights='distance', algorithm='brute')

KNN accuracy : 0.834

Elapsed time: 0.04

**precision recall f1-score support**

0 0.91 0.95 0.93 175

1 0.79 1.00 0.88 234

2 0.92 0.72 0.81 219

3 0.93 0.78 0.85 207

4 0.86 0.74 0.79 217

5 0.82 0.77 0.79 179

6 0.84 0.93 0.88 178

7 0.83 0.86 0.85 205

8 0.86 0.71 0.78 192

9 0.68 0.89 0.77 194

**avg / total 0.84 0.83 0.83 2000**

**Discussion of Results**

The algorithms work right out of the box, the only thing needed for this lab was to optimize the accuracy and running times. I pretty much used a trial and error/brute force approach to experiment on this lab. As I said before, I paired a PCA preprocessing algorithm before actually doing both KNN and MLP. It helped increase the accuracy, but not by much. Though, I will take any accuracy gain for sure.

Something I noticed which is weird was that when running MLP classifier, I would get different results each time. But, for the KNN classifier, I was getting the same exact result each time, which is why I only put one table for that algorithm. I tried resetting Spyder/Python because I thought that it was caching the results, but I continued getting the same results for KNN. Of course, the only time it changed was when I actually changed the parameters for the classifier, such as the number of neighbors to use when running the algorithm.

**Conclusions**

I learned about machine learning algorithms and I got a very basic idea on how it works. I really enjoyed using a library because 1) It is way easier than implementing my own, and 2) because it allowed to quickly start messing with different parameters and observe different results. It was a challenge increasing the accuracy completely because whenever I changed one attribute/parameter for the classifiers, sometimes it would only increase by not much, and if I screwed up, my accuracy would be on the 20%-40% accuracy. Since there are so many things you can change/add to your classifiers, it was basically a trial and error process to see what worked and what did not.

Searching around in the internet for MNIST dataset and KNN/MLP classifiers did yield many results, but sometimes the people were using KERAS, or sometimes they would implement their own versions. It was helpful seeing others because I could tell what worked for them and I was able to use that as well.

Something I could not get to work was the preprocessing **Standard Scaler** algorithm, also available on SKLearn. The resources that included examples showed improvements on their accuracy results, but on mine, it greatly decreased them. I tried many different configurations but to no avail.

Another thing I wanted to try was **feature selection** but the documentation on it was not that great in my opinion, and there was only one example in the SKLearn website.

**Appendix**

import os

from urllib.request import urlretrieve

import numpy as np

import time

from sklearn.neural\_network import MLPClassifier

from sklearn.metrics import classification\_report,confusion\_matrix

from sklearn.neighbors import KNeighborsClassifier

from sklearn.preprocessing import StandardScaler

from sklearn.decomposition import PCA

def download(filename, source='http://yann.lecun.com/exdb/mnist/'):

print("Downloading %s" % filename)

urlretrieve(source + filename, filename)

# We then define functions for loading MNIST images and labels.

# For convenience, they also download the requested files if needed.

import gzip

def load\_mnist\_images(filename):

if not os.path.exists(filename):

download(filename)

# Read the inputs in Yann LeCun's binary format.

with gzip.open(filename, 'rb') as f:

data = np.frombuffer(f.read(), np.uint8, offset=16)

# The inputs are vectors now, we reshape them to monochrome 2D images,

# following the shape convention: (examples, channels, rows, columns)

data = data.reshape(-1, 1, 28, 28)

# The inputs come as bytes, we convert them to float32 in range [0,1].

# (Actually to range [0, 255/256], for compatibility to the version

# provided at http://deeplearning.net/data/mnist/mnist.pkl.gz.)

return data / np.float32(256)

def load\_mnist\_labels(filename):

if not os.path.exists(filename):

download(filename)

# Read the labels in Yann LeCun's binary format.

with gzip.open(filename, 'rb') as f:

data = np.frombuffer(f.read(), np.uint8, offset=8)

# The labels are vectors of integers now, that's exactly what we want.

return data

X\_train = load\_mnist\_images('train-images-idx3-ubyte.gz')

y\_train = load\_mnist\_labels('train-labels-idx1-ubyte.gz')

X\_test = load\_mnist\_images('t10k-images-idx3-ubyte.gz')

y\_test = load\_mnist\_labels('t10k-labels-idx1-ubyte.gz')

train\_ex = 1000

test\_ex = 2000

X\_train = X\_train.reshape((X\_train.shape[0], -1))[:train\_ex]

X\_test = X\_test.reshape((X\_test.shape[0], -1))[:test\_ex]

y\_train = y\_train[:train\_ex]

y\_test = y\_test[:test\_ex]

pca = PCA(n\_components=70, svd\_solver='full')

pca.fit(X\_train)

X\_train = pca.transform(X\_train)

X\_test = pca.transform(X\_test)

start = time.time()

MLP = MLPClassifier(solver='adam', activation='relu', alpha=1, batch\_size='auto')

MLP.fit(X\_train, y\_train)

MLPredictions = MLP.predict(X\_test)

MLPAccuracy = np.sum(MLPredictions == y\_test)/y\_test.shape[0]

print('MLP accuracy : ' , MLPAccuracy)

print(classification\_report(y\_test,MLPredictions))

elapsed\_time = time.time()-start

print('Elapsed time: {0:.2f} '.format(elapsed\_time))

start = time.time()

knn = KNeighborsClassifier(n\_neighbors=5, weights='distance', algorithm='brute')

knn.fit(X\_train, y\_train)

KnnPredictions = knn.predict(X\_test)

KnnAccuracy = np.sum(KnnPredictions == y\_test)/y\_test.shape[0]

print('KNN accuracy : ' , KnnAccuracy)

elapsed\_time = time.time()-start

print('Elapsed time: {0:.2f} '.format(elapsed\_time))

print(classification\_report(y\_test, KnnPredictions))

**Sources**

KNN - <https://scikit-learn.org/stable/modules/generated/sklearn.neighbors.KNeighborsClassifier.html>

PCA - <https://scikit-learn.org/stable/modules/generated/sklearn.decomposition.PCA.html>

MLP - <https://scikit-learn.org/stable/modules/generated/sklearn.neural_network.MLPClassifier.html>

Standard Scaler - <https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.StandardScaler.html>

Lessons sample KNN classification - <https://gurus.pyimagesearch.com/lesson-sample-k-nearest-neighbor-classification/>

MNIST with PCA and KNN - <https://www.kaggle.com/gregnetols/mnist-with-pca-and-knn>

Feature selection using SelectFromModel - <https://scikit-learn.org/stable/auto_examples/feature_selection/plot_select_from_model_boston.html>