In [5]:
```
#1 import required modules
import time
import io
import matplotlib.pyplot as plt
import numpy as np

from sklearn import neighbors

from scipy.io.arff import loadarff
from sklearn.datasets import get_data_home
from sklearn.externals.joblib import Memory
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.utils import check_random_state
from urllib.request import urlopen
```

In [6]:
```
#2.  read the NMIST dataset
memory = Memory(get_data_home())
@memory.cache()
def fetch_mnist():
    content = urlopen(
        'https://www.openml.org/data/download/52667/mnist_784.arff').read()
    data, meta = loadarff(io.StringIO(content.decode('utf8')))
    data = data.view([('pixels', '<f8', 784), ('class', '|S1')])
    return data['pixels'], data['class']
X, y = fetch_mnist()
```

In [7]:
```
# rescale the data, use the traditional train/test split

X = X / 255.

###### NEW  Refromat the the labels to be integers rather than byte arrays
y_trans = []
for i in range(len(y)):
    y_trans.append(int(y[i]))
y = np.asarray(y_trans)

X_train, X_test = X[:2000], X[2000:2200]
y_train, y_test = y[:2000], y[2000:2200]
```

In [4]:
```
y_trans = []
for i in range(len(y)):
    y_trans.append(int(y[i]))
y = np.asarray(y_trans)
```

KNN neighbor = 3

In [5]:
```
import time
start_time = time.time()
#Classifier Declaration
KNN = neighbors.KNeighborsClassifier(n_neighbors=3)
#Train the classifier
KNN.fit(X_train,y_train)
train_time = time.time() - start_time
start_time = time.time()
print("Training time %.3f seconds" % train_time)
#Evaluate the result
score = KNN.score(X_test,y_test)
print("Test score with 3NN is: %.4f" % score)
test_time = time.time() - start_time
print("Test time %.3f seconds" % test_time)
print("Test score with 3NN is: %.4f" % score)
```

```
Training time 0.131 seconds
Test score with 3NN is: 0.9150
Test time 0.990 seconds
Test score with 3NN is: 0.9150
```

KNN neighbor=5

In [6]:
```
import time
start_time = time.time()
#Classifier Declaration
KNN = neighbors.KNeighborsClassifier(n_neighbors=5)
#Train the classifier
KNN.fit(X_train,y_train)
train_time = time.time() - start_time
start_time = time.time()
print("Training time %.3f seconds" % train_time)
#Evaluate the result
score = KNN.score(X_test,y_test)
print("Test score with 5NN is: %.4f" % score)
test_time = time.time() - start_time
print("Test time %.3f seconds" % test_time)
print("Test score with 5NN is: %.4f" % score)
```

```
Training time 0.140 seconds
Test score with 5NN is: 0.9200
Test time 0.957 seconds
Test score with 5NN is: 0.9200
```

KNN neighbor=1

In [7]:
```
import time
start_time = time.time()
#Classifier Declaration
KNN = neighbors.KNeighborsClassifier(n_neighbors=1)
#Train the classifier
KNN.fit(X_train,y_train)
```

```
train_time = time.time() - start_time
start_time = time.time()
print("Training time %.3f seconds" % train_time)
#Evaluate the result
score = KNN.score(X_test,y_test)
print("Test score with 1NN is: %.4f" % score)
test_time = time.time() - start_time
print("Test time %.3f seconds" % test_time)
print("Test score with 1NN is: %.4f" % score)
```

```
Training time 0.161 seconds
Test score with 1NN is: 0.9300
Test time 0.952 seconds
Test score with 1NN is: 0.9300
```

single hidden layer perceptron

In [8]:
```
# Standard scientific Python import
import time
import io
import matplotlib.pyplot as plt
import numpy as np
from scipy.io.arff import loadarff

from sklearn.neural_network import MLPClassifier
from sklearn.datasets import get_data_home
from sklearn.externals.joblib import Memory
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.utils import check_random_state
from urllib.request import urlopen
```

In [3]:
```
# rescale the data, use the traditional train/test split

X = X / 255.

###### NEW  Refromat the the labels to be integers rather than byte arrays
y_trans = []
for i in range(len(y)):
    y_trans.append(int(y[i]))
y = np.asarray(y_trans)

X_train, X_test = X[:10000], X[10000:10200]
y_train, y_test = y[:10000], y[10000:10200]
```

In [10]:
```
import time
start_time = time.time()
#Classifier Declaration
# Single hidden layer
mlp = MLPClassifier(hidden_layer_sizes=(50,), max_iter=40, alpha=1e-4,
                    solver='sgd', verbose=10, tol=1e-4, random_state=1,
                    learning_rate_init=.1)
# Note, the iteration limit will be reached!
#Train the classifier
mlp.fit(X_train,y_train)
train_time = time.time() - start_time
start_time = time.time()
print("Training time of Single hidden layer perceptron is: %.3f seconds" % train_time)
#Evaluate the result
score = mlp.score(X_test,y_test)
print("Test score with Single hidden layer perceptron is: %.4f" % score)
test_time = time.time() - start_time
print("Test time of of Single hidden layer perceptron is: %.3f seconds" % test_time)
```

```
Iteration 1, loss = 2.30454215
Iteration 2, loss = 2.30125955
Iteration 3, loss = 2.30120262
Iteration 4, loss = 2.30058722
Iteration 5, loss = 2.30042938
Iteration 6, loss = 2.29943975
Iteration 7, loss = 2.29867319
Iteration 8, loss = 2.29834305
Iteration 9, loss = 2.29773144
Iteration 10, loss = 2.29659269
Iteration 11, loss = 2.29551009
Iteration 12, loss = 2.29415868
Iteration 13, loss = 2.29271219
Iteration 14, loss = 2.29080467
Iteration 15, loss = 2.28815967
Iteration 16, loss = 2.28567883
Iteration 17, loss = 2.28233864
Iteration 18, loss = 2.27804437
Iteration 19, loss = 2.27350144
Iteration 20, loss = 2.26689236
Iteration 21, loss = 2.25943889
Iteration 22, loss = 2.25020769
Iteration 23, loss = 2.23991498
Iteration 24, loss = 2.22809553
Iteration 25, loss = 2.21294113
Iteration 26, loss = 2.19509497
Iteration 27, loss = 2.17492325
Iteration 28, loss = 2.15165413
Iteration 29, loss = 2.12602172
Iteration 30, loss = 2.09749275
Iteration 31, loss = 2.06691795
Iteration 32, loss = 2.03546323
Iteration 33, loss = 1.99887825
Iteration 34, loss = 1.97120796
Iteration 35, loss = 1.93659810
Iteration 36, loss = 1.89858619
Iteration 37, loss = 1.92144571
Iteration 38, loss = 1.83605941
Iteration 39, loss = 1.87642970
Iteration 40, loss = 1.79389169
Training time of Single hidden layer perceptron is: 17.264 seconds
Test score with Single hidden layer perceptron is: 0.5300
Test time of of Single hidden layer perceptron is: 0.019 seconds
```

```
/home/student/anaconda3/lib/python3.6/site-packages/sklearn/neural_network/multilayer_perceptron.py:564: ConvergenceW
```

```
/home/student/anaconda3/lib/python3.6/site-packages/sklearn/neural_network/multilayer_perceptron.py:564: ConvergenceWarning: Stochastic Optimizer: Maximum iterations (40) reached and the optimization hasn't converged yet.
  % self.max_iter, ConvergenceWarning)
```

Double hidden layer perceptron

In [11]:
```
import time
start_time = time.time()
#Classifier Declaration
# Two hidden layers
mlp_2 = MLPClassifier(hidden_layer_sizes=(50,50), max_iter=40, alpha=1e-4,
                    solver='sgd', verbose=10, tol=1e-4, random_state=1,
                    learning_rate_init=.1)
#
#Train the classifier
mlp_2.fit(X_train,y_train)
train_time = time.time() - start_time
start_time = time.time()
print("Training time of Double hidden layer perceptron is: %.3f seconds" % train_time)
#Evaluate the result
score = mlp_2.score(X_test,y_test)
print("Test score with Double hidden layer perceptron is: %.4f" % score)
test_time = time.time() - start_time
print("Test time of of Double hidden layer perceptron is: %.3f seconds" % test_time)
```

```
Iteration 1, loss = 2.30800662
Iteration 2, loss = 2.30288661
Iteration 3, loss = 2.30179307
Iteration 4, loss = 2.30179997
Iteration 5, loss = 2.30213552
Iteration 6, loss = 2.30119653
Iteration 7, loss = 2.30092266
Iteration 8, loss = 2.30070933
Iteration 9, loss = 2.29981106
Iteration 10, loss = 2.30022172
Iteration 11, loss = 2.29862128
Iteration 12, loss = 2.29712283
Iteration 13, loss = 2.29611147
Iteration 14, loss = 2.29387955
Iteration 15, loss = 2.28984783
Iteration 16, loss = 2.28109478
Iteration 17, loss = 2.26712063
Iteration 18, loss = 2.24307923
Iteration 19, loss = 2.25598550
Iteration 20, loss = 2.27592082
Iteration 21, loss = 2.22471399
Iteration 22, loss = 2.25716824
Iteration 23, loss = 2.21686140
Iteration 24, loss = 2.27704808
Iteration 25, loss = 2.25683880
Iteration 26, loss = 2.21989608
Training loss did not improve more than tol=0.000100 for two consecutive epochs. Stopping.
Training time of Double hidden layer perceptron is: 10.984 seconds
Test score with Double hidden layer perceptron is: 0.2000
Test time of of Double hidden layer perceptron is: 0.039 seconds
```

Linear function SVM

In [2]:
```
# Import datasets, classifiers and performance metrics
from sklearn import datasets, svm, metrics
```

In [10]:
```
import time
start_time = time.time()
#Classifier Declaration
## Linear
clf = svm.SVC(kernel = 'linear', C = 1)
##
#Train the classifier
clf.fit(X_train,y_train)
train_time = time.time() - start_time
start_time = time.time()
print("Training time of linear function SVM is: %.3f seconds" % train_time)
#Evaluate the result
score = clf.score(X_test,y_test)
print("Test score with linear function SVM is: %.4f" % score)
test_time = time.time() - start_time
print("Test time of of linear function SVM is: %.3f seconds" % test_time)
```

```
Training time of linear function SVM is: 1.704 seconds
Test score with linear function SVM is: 0.9200
Test time of of linear function SVM is: 0.198 seconds
```

Polynormial function SVM

In [12]:
```
import time
start_time = time.time()
#Classifier Declaration
## Linear
clf = svm.SVC(kernel = 'poly',degree = 3, C = 1)
##
#Train the classifier
clf.fit(X_train,y_train)
train_time = time.time() - start_time
start_time = time.time()
print("Training time of polynormial function SVM is: %.3f seconds" % train_time)
#Evaluate the result
score = clf.score(X_test,y_test)
print("Test score with polynormila function SVM is: %.4f" % score)
test_time = time.time() - start_time
print("Test time of polynormial function SVM is: %.3f seconds" % test_time)
```

```
Training time of polynormial function SVM is: 7.816 seconds
Test score with polynormila function SVM is: 0.1100
Test time of polynormial function SVM is: 0.528 seconds
```

radial basis function SVM

In [9]:
```
import time
start_time = time.time()
#Classifier Declaration
## Radial basis functions
clf = svm.SVC(kernel = 'rbf', C = 1, gamma = 0.5)

#Train the classifier
clf.fit(X_train,y_train)
train_time = time.time() - start_time
start_time = time.time()
print("Training time of radial basis function SVM is: %.3f seconds" % train_time)
#Evaluate the result
score = clf.score(X_test,y_test)
print("Test score with radial basis function SVM is: %.4f" % score)
test_time = time.time() - start_time
print("Test time of ofradial basis function SVM is: %.3f seconds" % test_time)
```

```
Training time of radial basis function SVM is: 8.571 seconds
Test score with radial basis function SVM is: 0.1850
Test time of ofradial basis function SVM is: 0.577 seconds
```

Discuss

KNN:

To speed up the trainning and test time. my trainning set is the first 2000 iamges. my test set is the 2000-2200 images. Previously I tried to use the first 10000 images to train and the last 10000 images to test, but due to the performance of my virtual machine, the trainning time is so long, more than 40 minuets, so I gave up and choosed the 200 images smaller set.

For neighbors = 1: Training time is 0.161 seconds Test score is: 0.9300 Test time 0.952 seconds

For neighbors = 3: Training time is 0.131 seconds Test score is: 0.9150 Test time is 0.990 seconds

For neighbors = 5: Training time is 0.140 seconds Test score is: 0.9200 Test time is 0.957 seconds

As the increase of neighbors, the score is almost the same, they are not in linear relationship. From the result, I guess K should be smaller so that the trainning quality will be better. In k-NN classification, the output is a class membership. An object is classified by a majority vote of its neighbors, with the object being assigned to the class most common among its k nearest neighbors. And k is a small positive integer. If k = 1, then the object is simply assigned to the class of that single nearest neighbor.

Hidden Layer perceptron

Since the tranning and test is a lot more faster for hidden layer perceptron compared with KNN and SVM, so to speed up the trainning and test time. my trainning set is the first 10000 iamges. my test set is the 10000-10200 images.

For single hidden layer perceptron: I set the hidden_layer_sizes=(50,50), max_iter=40, alpha=1e-4, in every iteration, the loss fluctuate a lot, in a large range, the loss decreases from 2.3 to 1.8. Finally the loss is around 1.8. Although after 40 itertions, my model still does not convergent, but the score is already very high: Training time of Single hidden layer perceptron is: 17.264 seconds Test score with Single hidden layer perceptron is: 0.5300 Test time of of Single hidden layer perceptron is: 0.019 seconds

For double hidden layer perceptron: I set the hidden_layer_sizes=(50,50), max_iter=40, alpha=1e-4, in every iteration, the loss fluctuate a lot, in a large range, the loss decreases from 2.3 to 2.21. Finally the loss is around 2.21. 40 iterations is so many, when loss did not improve more than tol=0.000100 for two consecutive epochs, I set it as convergent and let my program stop. So my model convergent at the 26 iterations, the score is pretty high, results are pretty good as below: Training time of Double hidden layer perceptron is: 10.984 seconds Test score with Double hidden layer perceptron is: 0.2000 Test time of of Double hidden layer perceptron is: 0.039 seconds

The classify result of single layer perceptron is better than double layer perceptron.

SVM

To speed up the trainning and test time. My trainning set is the first 2000 iamges. my test set is the 2000-2200 images. Previously I tried to use the first 10000 images to train and the last 10000 images to test, but it only works for linear function SVM,for linear function SVM, the result is :

Training time of linear function SVM is: 17.567 seconds Test score with linear function SVM is: 0.9100 Test time of of linear function SVM is: 0.659 seconds

But due to the performance of my virtual machine, for polynormial and radix basis function SVM, the trainning time is so long, more than 40 minuets, so I gave up and choosed the 200 images smaller set.

Training time of linear function SVM is: 1.704 seconds Test score with linear function SVM is: 0.9200 Test time of of linear function SVM is: 0.198 seconds

Training time of polynormial function SVM is: 7.816 seconds Test score with polynormila function SVM is: 0.1100 Test time of polynormial function SVM is: 0.528 seconds

Training time of radial basis function SVM is: 8.571 seconds Test score with radial basis function SVM is: 0.1850 Test time of ofradial basis function SVM is: 0.577 seconds

the classify result for linear function SVM is the best, and the polynormial function SVM is the worst.

Conclusion

All in all, the best classification result is KNN, and then hidden layer perceptron, the least is SVM.