CS498

Applied Machine Learning

Assignment #4

Students:
nidiaib2, Nidia Bucarelli
sunnyk2, Sunny Katiyar
wangx2, Wang Xiang

March 30, 2020
Spring 2020

PROBLEM 8.6.

Question a.

Code is attached

Question b.

Code is attached.

Question c.

The final model was trained using a Random Forest in Python, Sklearn Library. An accuracy of **95.47%** was achieved with the following settings:

- Number of estimators (estimators): 1000
- Criteria: Gini impurity
- Depth: None (until no more splits are possible)
- Using bootstrap
- Euclidean distance (L2)
- Minimum samples split: 2
- Minimum samples leaf: 1

When training, it was noticed that with just 10 trees (default setting in Sklearn), the accuracy of the classifier on the testing set was already around ~91%. This improved to 95.47% when we increased the number of trees to 1000.

Question d.

In part C, a good accuracy was already achieved. We made some experiments to explore if better accuracy was possible to achieve. Some findings include:

- Reducing the number of cluster centers for both cluster levels (Level 1 and Level 2) does not improve the accuracy achieved in part C. Still, good accuracies were achieved (e.g., with K= 20 and K= 40, accuracies were about 85% and 89% respectively).
- Increasing the number of cluster centers does not improve the performance by much. But similar accuracies were achieved with less complex models. For example, with 10,000 patches and 60 clusters centers, 200 trees with a depth of 80 already have similar accuracy as part C (e.g., ~95%) and with the increment of further number of trees no significant improvements were observed (e.g., 0.20-0.30% improvement). In the same sense, with 15,000 patches and 70 clusters centers and a less complex model (200 trees; depth=20) similar accuracy as part C was already achieved. Under this setting, a RF built with 1000 trees until no more splits were possible improved the accuracy of part C by ~0.90-1.00%.
- Using 50 cluster centers in the first level, and changing the number of clusters centers in the second level provide similar accuracy as part c. No improvement was observed.
- Using a set of larger patches of size 12X12 on an overlapping 4X4 grid further improved the test accuracy of the final model from 95.47% to 95.87%.

Note: we haven't uploaded the code for part-d as that's only minor changes in the hyperparameters. However, if the TA needs to cross check, we can share the codes for part-d as well.

Question e.

According to available records:

- The test error achieved with this approach is better than the error rates achieved with linear classifiers.
- The test error achieved with this approach is as good as the test error achieved with K-NN and a simple NN with 2 layers.
- However, the test error achieved with this approach is higher than that of Convolutional nets.

Import necessary libraries

In [604]:

```
import pandas as pd
import numpy as np
from sklearn.model_selection import GridSearchCV
import scipy.misc as smp
import matplotlib.pyplot as plt
import random
from scipy.spatial import distance
import time
import datetime
```

Load the dataset

In [2]:

```
train_data = pd.read_csv("mnist_train.csv", header=None)
test_data = pd.read_csv("mnist_test.csv", header=None)
```

In [7]:

```
train_data.head(2)
```

Out[7]:

	0	1	2	3	4	5	6	7	8	9	...	775	776	777	778	779	780	781	782	783	784
0	5	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	0

2 rows × 785 columns

In [8]:

```
test_data.head(2)
```

Out[8]:

	0	1	2	3	4	5	6	7	8	9	...	775	776	777	778	779	780	781	782	783	784
0	7	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	0
1	2	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	0

2 rows × 785 columns

In [9]:

```
train_data.shape
```

Out[9]:

```
(60000, 785)
```

In [10]:

```
test_data.shape
```

Out[10]:

```
(10000, 785)
```

Split between features and labels

In [11]:

```
X_train, y_train = train_data.iloc[:,1:], train_data.iloc[:,0]
X_test, y_test = test_data.iloc[:,1:], test_data.iloc[:,0]
```

Part-a

Generating 16 patches (10X10) for each training image

Defining a function for generating the patches

In [39]:

```
# This function will return a dictionary that will have 16 10X10 patches
#for each image passed
def generate_patches(training_images):

    patches = {}
    total_images = len(training_images)

    for i in range(total_images):
        image = np.array(training_images.iloc[i,:]).reshape(28,28)
        j = 0 #Row
        patch_count = 1

        while (j+10 <= 28):
            k = 0 #Column
            while(k+10 <= 28):
                patch = image[j:j+10,k:k+10]
                patch_name = "Image_" + str(i) + "_" + str(patch_count)
                patch_count += 1
                patches[patch_name] = patch.flatten() #Updating the dictionary
                k = k + 6
                next
            j = j + 6

    return(patches)
```

Generating the patches for all training images

In [49]:

```
train_data_patches = generate_patches(training_images=X_train)
```

Checking the count of patches generated

In [50]:

```
len(train_data_patches)
```

Out[50]:

```
960000
```

Choosing one patch randomly for each image

In [68]:

```
def select_random_patches(train_data_patches):

    random.seed(10)
    random_patches = {}

    for image_num in range(len(X_train)):
        random_patch = int(np.random.randint(1,17,1))

        patch_name = "Image_" + str(image_num) + "_" + str(random_patch)

        random_patches[patch_name] = train_data_patches[patch_name]

    return(random_patches)
```

In [75]:

```
random.seed(10)
train_random_patches = select_random_patches(train_data_patches=train_data_patch
es)
```

In [77]:

```
len(train_random_patches)
```

Out[77]:

```
60000
```

Sub-Sampling 6000 patches

In [88]:

```
train_random_6000_patches = {}
random_items = random.sample(list(train_random_patches.keys()), 6000)

for i in range(len(random_items)):
    train_random_6000_patches[random_items[i]] =\
    train_random_patches[random_items[i]]
```

Creating a list and then a numpy array of these 6000 patches

In [99]:

```
train_6k_patches_list = []
for i in train_random_6000_patches:
    train_6k_patches_list.append(train_random_6000_patches[i])
train_6k_patches_array = np.array(train_6k_patches_list)
```

Creating the first 50 clusters

In [101]:

```
from sklearn.cluster import KMeans
kmeans = KMeans(n_clusters=50, random_state=10).fit(train_6k_patches_array)
```

K-Means labels

In [107]:

```
kmeans_labels = kmeans.labels_
```

Classifying the 60,000 patches into 50 clusters

In [109]:

```
#Creating a list and then a numpy array of these 6000 patches

train_60k_patches_list = []
for i in train_random_patches:
    train_60k_patches_list.append(train_random_patches[i])
train_60k_patches_array = np.array(train_60k_patches_list)


kmeans_labels_60K_patches = kmeans.predict(train_60k_patches_array)
```

Creating a dataframe with the patches and their cluster class

In [119]:

```
patches_60K_cluster_df = pd.DataFrame(train_random_patches).T
```

In [120]:

```
patches_60K_cluster_df['cluster'] = kmeans_labels_60K_patches
```

In [121]:

```
patches_60K_cluster_df.head()
```

Out[121]:

	0	1	2	3	4	5	6	7	8	9	...	91	92	93	94	95
Image_0_7	170	253	253	253	253	253	225	172	253	242	...	0	45	186	253	253
Image_1_3	0	0	0	0	0	0	0	0	0	0	...	253	190	114	253	228
Image_2_2	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0
Image_3_8	244	251	253	62	0	0	0	0	0	0	...	0	0	0	0	0
Image_4_7	0	0	0	0	0	0	0	0	0	0	...	226	227	252	231	0

5 rows × 101 columns

Clustering each dataset into 50 sub clusters - creating a total of 2500 clusters mean

In [169]:

```
final_cluster_count = 0
final_cluster_mean = {}

for cluster in range(50):

    dataset =\
    patches_60K_cluster_df[patches_60K_cluster_df['cluster'] == cluster].iloc[:,
0:100].values

    kmeans_cluster = KMeans(n_clusters=50, random_state=10).fit(dataset)
    kmeans_cluster_labels = kmeans_cluster.labels_

    for i in range(50):
        cluster_mean = np.mean(dataset[kmeans_cluster_labels == i], axis=0)
        cluster_name = "cluster_" + str(final_cluster_count + i)
        final_cluster_mean[cluster_name] = cluster_mean

    final_cluster_count = final_cluster_count + 50
```

In [170]:

```
pd.DataFrame(final_cluster_mean).T
```

Out[170]:

	0	1	2	3	4	5	
cluster_0	18.520000	0.000000	0.000000	0.000000	0.000000	0.000000	0.00(
cluster_1	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.00(
cluster_2	29.818182	29.000000	13.909091	3.000000	0.000000	0.181818	1.27;
cluster_3	20.105263	0.789474	0.000000	0.000000	0.000000	0.000000	0.00(
cluster_4	16.800000	13.600000	1.133333	0.000000	0.000000	0.000000	0.00(
...	...	...	...	...	...	...	
cluster_2495	15.500000	11.384615	8.769231	11.423077	18.538462	10.038462	0.92;
cluster_2496	1.153846	6.615385	17.461538	14.615385	62.538462	159.923077	215.84(
cluster_2497	236.000000	250.750000	200.750000	146.250000	86.250000	85.250000	95.00(
cluster_2498	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	5.25(
cluster_2499	4.800000	41.200000	95.200000	195.200000	253.000000	240.000000	144.40(

2500 rows × 100 columns

Part-b

Creating a function to create histogram for the query image

In [592]:

```
def create_cluster_histogram(total_images, final_cluster_mean_dictionary):

    cluster_mean_array = np.array(pd.DataFrame(final_cluster_mean_dictionary).T)
    cluster_mean_df = pd.DataFrame(cluster_mean_array)

    image_count = len(total_images)

    #Creating an empty dataframe for storing the final features
    final_features =\
    pd.DataFrame(np.repeat(np.zeros(2500),image_count).reshape(image_count,2500
))

    #patches = []

    for i in range(image_count):
        query_image = np.array(total_images.iloc[i,:]).reshape(28,28)
        query_image_patches = []

        j = 0 #row
        while (j+10 <= 28):
            x = j
            k = 0 #Column
            while(k+10 <= 28):
                y = k

                #1. x,y
                patch = query_image[x:x+10,y:y+10]
                query_image_patches.append(patch.flatten())

                #2. x, y+1
                if (y+11 > 28):
                    q_img_df = pd.DataFrame(query_image)
                    q_img_df[28] = np.zeros(q_img_df.shape[0]) #Padding with zer
oes

                    patch = np.array(q_img_df)[x:x+10,y+1:y+11]
                    query_image_patches.append(patch.flatten())
                else:
                    patch = query_image[x:x+10,y+1:y+11]
                    query_image_patches.append(patch.flatten())

                #3. x, y-1
                if (y-1<0):
                    q_img_df = pd.DataFrame(query_image)
                    q_img_df.insert(0, "a", np.zeros(q_img_df.shape[0])) #Paddin
g with zeroes

                    q_img_df.columns = np.arange(q_img_df.shape[1])
                    patch = np.array(q_img_df)[x:x+10,y:y+10]
                    query_image_patches.append(patch.flatten())
                else:
                    patch = query_image[x:x+10,y-1:y+9]
                    query_image_patches.append(patch.flatten())


                #4. x+1, y
                if (x+11>28):
                    q_img_df = pd.DataFrame(query_image)
                    row_df = pd.DataFrame(np.zeros(q_img_df.shape[1])).T
                    q_img_df = q_img_df.append(row_df, ignore_index=True) #Paddi
ng with zeroes
```

```
                    patch = np.array(q_img_df)[x+1:x+11,y:y+10]
                    query_image_patches.append(patch.flatten())
                else:
                    patch = query_image[x+1:x+11,y:y+10]
                    query_image_patches.append(patch.flatten())

                #5. x-1, y
                if (x-1<0):
                    q_img_old = pd.DataFrame(query_image)
                    row_df = pd.DataFrame(np.zeros(q_img_old.shape[1])).T
                    q_img_df = row_df.append(q_img_old, ignore_index=True) #Padd
ing with zeroes
                    patch = np.array(q_img_df)[x:x+10,y:y+10]
                    query_image_patches.append(patch.flatten())
                else:
                    patch = query_image[x-1:x+9,y:y+10]
                    query_image_patches.append(patch.flatten())

                #6. x+1, y+1
                if (x+11 > 28):
                    q_img_df = pd.DataFrame(query_image)
                    row_df = pd.DataFrame(np.zeros(q_img_df.shape[1])).T
                    q_img_df = q_img_df.append(row_df, ignore_index=True) #Paddi
ng with zeroes

                    if (y+11 > 28):
                        q_img_df[28] = np.zeros(q_img_df.shape[0]) #Padding with
zeroes
                    patch = np.array(q_img_df)[x+1:x+11,y+1:y+11]
                    query_image_patches.append(patch.flatten())

                else:
                    q_img_df = pd.DataFrame(query_image)
                    if (y+11 > 28):
                        q_img_df[28] = np.zeros(q_img_df.shape[0]) #Padding with
zeroes
                    patch = np.array(q_img_df)[x+1:x+11,y+1:y+11]
                    query_image_patches.append(patch.flatten())


                #7. x-1, y-1
                if ((x-1 > 0) and (y-1 > 0)):
                    patch = query_image[x-1:x+9,y-1:y+9]
                    query_image_patches.append(patch.flatten())

                else:
                    if (x-1<0):
                        q_img_old = pd.DataFrame(query_image)
                        row_df = pd.DataFrame(np.zeros(q_img_old.shape[1])).T
                        q_img_df = row_df.append(q_img_old, ignore_index=True) #
Padding with zeroes

                        if (y-1<0):
                            q_img_df.insert(0, "a", np.zeros(q_img_df.shape[0]))
#Padding with zeroes
                            q_img_df.columns = np.arange(q_img_df.shape[1])
                            patch = np.array(q_img_df)[x:x+10,y:y+10]
                            query_image_patches.append(patch.flatten())
                        else:
                            patch = np.array(q_img_df)[x:x+10,y-1:y+9]
                            query_image_patches.append(patch.flatten())
```

```
                else:
                    if (y-1<0):
                        q_img_df = pd.DataFrame(query_image)
                        q_img_df.insert(0, "a", np.zeros(q_img_df.shape[0]))
#Padding with zeroes
                        q_img_df.columns = np.arange(q_img_df.shape[1])
                        patch = np.array(q_img_df)[x-1:x+9,y:y+10]
                        query_image_patches.append(patch.flatten())
                    else:
                        patch = np.array(q_img_df)[x-1:x+9,y-1:y+9]
                        query_image_patches.append(patch.flatten())

            #8. x-1, y+1
            if (x-1<0):
                q_img_old = pd.DataFrame(query_image)
                row_df = pd.DataFrame(np.zeros(q_img_old.shape[1])).T
                q_img_df = row_df.append(q_img_old, ignore_index=True) #Padd
ing with zeroes

                if (y+11 > 28):
                    q_img_df[28] = np.zeros(q_img_df.shape[0]) #Padding with
zeroes

                patch = np.array(q_img_df)[x:x+10,y+1:y+11]
                query_image_patches.append(patch.flatten())

            else:
                q_img_df = pd.DataFrame(query_image)

                if (y+11 > 28):
                    q_img_df[28] = np.zeros(q_img_df.shape[0]) #Padding with
zeroes

                patch = np.array(q_img_df)[x-1:x+9,y+1:y+11]
                query_image_patches.append(patch.flatten())

            #9. x+1, y-1
            if (x+11 > 28):
                q_img_df = pd.DataFrame(query_image)
                row_df = pd.DataFrame(np.zeros(q_img_df.shape[1])).T
                q_img_df = q_img_df.append(row_df, ignore_index=True) #Paddi
ng with zeroes

                if (y-1<0):
                    #q_img_df = pd.DataFrame(query_image)
                    q_img_df.insert(0, "a", np.zeros(q_img_df.shape[0])) #Pa
dding with zeroes
                    q_img_df.columns = np.arange(q_img_df.shape[1])
                    patch = np.array(q_img_df)[x+1:x+11,y:y+10]
                    query_image_patches.append(patch.flatten())

                else:
                    patch = np.array(q_img_df)[x+1:x+11,y-1:y+9]
                    query_image_patches.append(patch.flatten())

            else:
                q_img_df = pd.DataFrame(query_image)

                if (y-1<0):
                    #q_img_df = pd.DataFrame(query_image)
```

```
                        q_img_df.insert(0, "a", np.zeros(q_img_df.shape[0])) #Pa
dding with zeroes
                        q_img_df.columns = np.arange(q_img_df.shape[1])
                        patch = np.array(q_img_df)[x+1:x+11,y:y+10]
                        query_image_patches.append(patch.flatten())
                    else:
                        patch = np.array(q_img_df)[x+1:x+11,y-1:y+9]
                        query_image_patches.append(patch.flatten())

                k = k + 6
                next

            j = j + 6
            next

        #Checking the distance of each patch from the 2500 clusters and assignin
g it to the nearest one.

        for p in range(len(query_image_patches)):

            image_patch = query_image_patches[p]

            dist = (cluster_mean_array - image_patch)**2
            dist = np.sum(dist, axis=1)
            distance_vector = np.sqrt(dist)

            min_dist = np.argmin(distance_vector)

            final_features.iloc[i,:][min_dist] = final_features.iloc[i,:][min_di
st] + 1

    return(final_features)
```

Using the above function and creating the feature space (histogram of patches) for all TRAIN images

In [606]:

```
t0 = time.time()
currentDT = datetime.datetime.now()
print (str(currentDT))

train_features = create_cluster_histogram(total_images = X_train,\
                                          final_cluster_mean_dictionary = final_
cluster_mean)

t1 = time.time()
currentDT = datetime.datetime.now()
print (str(currentDT))
total = t1-t0
```

```
2020-03-21 22:08:35.796702
2020-03-22 05:04:04.598638
```

Using the above function and creating the feature space (histogram of patches) for all TEST images

In [598]:

```
t0 = time.time()
test_features = create_cluster_histogram(total_images = X_test,\
                                         final_cluster_mean_dictionary = final_c
luster_mean)
t1 = time.time()

total = t1-t0
```

Part-c: Training a classifier

Classification using Random Forest

In [615]:

```
from sklearn.ensemble import RandomForestClassifier
```

In [617]:

```
rf_clf = RandomForestClassifier(random_state=0)
rf_clf.fit(X=train_features, y=y_train)
```

```
/anaconda3/lib/python3.7/site-packages/sklearn/ensemble/forest.py:24
6: FutureWarning: The default value of n_estimators will change from
10 in version 0.20 to 100 in 0.22.
  "10 in version 0.20 to 100 in 0.22.", FutureWarning)
```

Out[617]:

```
RandomForestClassifier(bootstrap=True, class_weight=None, criterion
='gini',
            max_depth=None, max_features='auto', max_leaf_nodes=Non
e,
            min_impurity_decrease=0.0, min_impurity_split=None,
            min_samples_leaf=1, min_samples_split=2,
            min_weight_fraction_leaf=0.0, n_estimators=10, n_jobs=No
ne,
            oob_score=False, random_state=0, verbose=0, warm_start=F
alse)
```

In [625]:

```
pred = rf_clf.predict(test_features)
```

In [626]:

```
sum(pred == y_test)/len(y_test)*100
```

Out[626]:

```
90.83
```

Parameter Tuning using Grid Search

In [630]:

```
from sklearn.model_selection import GridSearchCV
```

In [681]:

```
# Create the parameter grid based on the results of random search
param_grid = {
 'max_depth': [4, 15, None],
 'min_samples_leaf': [1, 2],
 'min_samples_split': [2, 5, 10],
 'n_estimators': [100, 200]
}
# Create a based model
rf = RandomForestClassifier()
# Instantiate the grid search model
grid_search = GridSearchCV(estimator = rf, param_grid = param_grid,
                          cv = 2, verbose = 1, n_jobs = -1)
```

In [683]:

```
# Fit the grid search to the data
grid_search.fit(train_features, y_train)
```

```
Fitting 2 folds for each of 36 candidates, totalling 72 fits

[Parallel(n_jobs=-1)]: Using backend LokyBackend with 4 concurrent w
orkers.
[Parallel(n_jobs=-1)]: Done  42 tasks      | elapsed: 15.6min
[Parallel(n_jobs=-1)]: Done  72 out of  72 | elapsed: 44.9min finish
ed
```

Out[683]:

```
GridSearchCV(cv=2, error_score='raise-deprecating',
       estimator=RandomForestClassifier(bootstrap=True, class_weight
=None, criterion='gini',
            max_depth=None, max_features='auto', max_leaf_nodes=Non
e,
            min_impurity_decrease=0.0, min_impurity_split=None,
            min_samples_leaf=1, min_samples_split=2,
            min_weight_fraction_leaf=0.0, n_estimators='warn', n_job
s=None,
            oob_score=False, random_state=None, verbose=0,
            warm_start=False),
       fit_params=None, iid='warn', n_jobs=-1,
       param_grid={'max_depth': [4, 15, None], 'min_samples_leaf':
[1, 2], 'min_samples_split': [2, 5, 10], 'n_estimators': [100, 20
0]},
       pre_dispatch='2*n_jobs', refit=True, return_train_score='war
n',
       scoring=None, verbose=1)
```

In [685]:

```
grid_search.best_params_
```

Out[685]:

```
{'max_depth': None,
 'min_samples_leaf': 1,
 'min_samples_split': 2,
 'n_estimators': 200}
```

Fitting a Random Forest model using the best parameters from Grid Search

In [691]:

```
# We didn't check for n_estimators > 200 in the grid search.
#But, increasing the n_estimators to #1000 further improved the performance
rf_final = RandomForestClassifier(n_estimators=1000, min_samples_split=2,\
                                  min_samples_leaf=1,
                                  max_depth=None, random_state=10)
```

In [692]:

```
rf_final.fit(train_features, y_train)
```

Out[692]:

```
RandomForestClassifier(bootstrap=True, class_weight=None, criterion
='gini',
            max_depth=None, max_features='auto', max_leaf_nodes=Non
e,
            min_impurity_decrease=0.0, min_impurity_split=None,
            min_samples_leaf=1, min_samples_split=2,
            min_weight_fraction_leaf=0.0, n_estimators=1000, n_jobs=
None,
            oob_score=False, random_state=10, verbose=0, warm_start=
False)
```

Prediction on test dataset

In [693]:

```
predicted_numbers = rf_final.predict(test_features)
```

In [694]:

```
predicted_numbers
```

Out[694]:

```
array([7, 2, 1, ..., 4, 5, 6])
```

Accuracy

In [698]:

```
#1000 trees
accuracy = sum(y_test.values == predicted_numbers) / len(y_test)
print("The accuracy of the model is: ", accuracy*100, "%")
```

```
The accuracy of the model is:  95.47 %
```