THE UNIVERSITY OF SYDNEY

School of Information Technologies
Faculty of Engineering & IT

ASSIGNMENT/PROJECT COVERSHEET - INDIVIDUAL ASSESSMENT

Unit of Study: **COMP5318**

Assignment name: **Assignment One**

Tutorial time: **T17C**

Tutor name: **Chen Chen**

DECLARATION

I declare that I have read and understood the *University of Sydney Academic Dishonesty and Plagiarism in Coursework Policy*, and except where specifically acknowledged, the work contained in this assignment/project is my own work, and has not been copied from other sources or been previously submitted for award or assessment.

I understand that failure to comply with the the *Academic Dishonesty and Plagiarism in Coursework Policy*, can lead to severe penalties as outlined under Chapter 8 of the *University of Sydney By-Law 1999* (as amended). These penalties may be imposed in cases where any significant portion of my submitted work has been copied without proper acknowledgement from other sources, including published works, the internet, existing programs, the work of other students, or work previously submitted for other awards or assessments.

I realise that I may be asked to identify those portions of the work contributed by me and required to demonstrate my knowledge of the relevant material by answering oral questions or by undertaking supplementary work, either written or in the laboratory, in order to arrive at the final assessment mark.

Student ID: **470022177**

Student name: **Yichun Wang**

Signed Yichun Wang **Date** **21/10/2020**

SIT Building, J12
The University of Sydney
NSW 2006 Australia

T +61 2 9351 3423
F +61 2 9351 3838
E sit.info@sydney.edu.au
sydney.edu.au/it

ABN 15 211 513 464
CRICOS 00026A

Introduction

This assignment requires to establish a kind of image classifier, mainly to classify a large group of grayscale images with size 28 x 28. Using suitable data preprocessing methods and correct classification algorithms are necessary, which could help reduce the amount of calculation and get reasonable results. Because of the large size of the initial dataset, it is also important to reduce the dimension, in order to decrease computing time.

Based on the tasks needed to be completed in this assignment, after finishing these, we could have a good grasp of the basic knowledge of classifiers, be familiar with the algorithm foundation and logic of various classifiers and master appropriate data preprocessing methods. These knowledge are extremely important, as they could help us cultivate good logical thinking skills, which is useful for laying a solid foundation for subsequent learning. This is the most important meaning of this assignment.

Methods

- **Principal Components Analysis (PCA)**

  As the main data preprocessing technique, PCA plays a vital role in the realization of complete functions. In view of the fact that each initial image provided by the dataset has a large dimension (28 x 28 = 784), while these dimensions also include a large number of noises, which means the number of really useful features is quite small.

  In order to remove these inutile noises, we have to use PCA to reduce the dimension. For example, the total 784 dimensions might only have 100 useful features, then we use PCA to reduce the dimension from 784 to 100. This not only helps us remove the noises but also reduces the calculation time.

  For the coding implementation, based on the mathematical principles of PCA, there are a few basic steps to realize the technique:
  - *Normalize the data (subtract the mean)*
  - *Calculate the covariance matrix of the normalized dataset*
  - *Calculate the eigenvalues and eigenvectors of the covariance matrix*
  - *Keep the most important k features (usually k less than n)*
  - *Find the eigenvectors corresponding to k eigenvalues*
  - *Multiply the m * n data set by the eigenvector (n * k) of k n-dimensional eigenvectors to obtain the final dimensionality reduction data*

In [35]:
```
## Data preprocessing by PCA/SVD
def PCAtrain(conponent, data):
    mean = np.mean(data, axis = 0)
    new_data = data - mean
    covMatrix = np.mat(np.cov(new_data, rowvar = 0))
  #covMatrix = np.dot(new_data.T, new_data) / (new_data.shape[0] - 1)
    eig_vals, eig_vecs = np.linalg.eig(covMatrix)
    index = np.argsort(-eig_vals)
    eig_vecs = eig_vecs[:,index][:,:conponent]
    eig_vals = -np.sort(-eig_vals)[:conponent]
  #pca_data = new_data.T.dot(eig_vecs.T)
    pca_data = np.dot(new_data, eig_vecs)
  #pca_data = new_data.dot(eig_vecs)

    recon = pca_data.dot(eig_vecs.T) + mean

    return pca_data, eig_vecs, recon
```

In [30]:
```
def PCAtest(data):
    mean = np.mean(data, axis = 0)
    new_data = data - mean
    pca_data = np.dot(new_data, eig_vecs_train)

    return pca_data
```

After these steps, the PCA technique has been established and could be used to reduce the dimensionality of initial training dataset and test dataset. One very important thing is, when we use PCA to reduce the dimensionality of test dataset, we have to use the eigenvector obtained from training dataset's PCA. That is because we have to make sure both datasets have to do the conversion in the same space, otherwise, the PCA's results are going to fail.

For the choice of the best dimensionality, maximum likelihood estimation (MLE) is one of the suitable methods, while it needs to be used under the sklearn package which is not allowed to use in this assignment. So the parameter would be found artificially.

- **K-Nearest Neighbors (KNN)**

  KNN is one of the two classifier algorithms used in this assignment, the core idea is to use the classification of the closest k sample data to represent the classification of the target data.

  In order to realize the KNN algorithm, some steps have to be followed as well:
  - *Calculate the distance between the test data and each sample data in the training sample set*
  - *Sort by increasing distance*
  - *Select the nearest k points*
  - *Determine the frequency of classification information in these k points*
  - *Return the category with the highest frequency among the first k points as the category of the current test data*

In [8]:
```
def getLabels(sortedList, label, k):
    record_label = []

    for m in range(k):
        labels = label[sortedList[m]]
        record_label.append(labels)

    maxLabel = max(record_label, key=record_label.count)

    return maxLabel
```

In [9]:
```
def KNN(train, test, label, k):
    result_list = []

    for j in range(test.shape[0]):
        current = test[j]
        dist_nearest = []
        for i in range(train.shape[0]):
            dist = np.linalg.norm(current - train[i])
            dist_nearest.append(dist)
            #sortedDist = getNeibours(train[i], test[j])
        sortedDist = np.argsort(dist_nearest)

        maxLabel = getLabels(sortedDist, label, k)

        result_list.append(maxLabel)

    return result_list
```

As the most important part of the KNN algorithm, the set of the hyperparameter K has a significant impact on the results. At the same time, the K value is usually not more than 20. In order to find the suitable K value, a loop function has been taken to find the best value:

As the diagram shows, for this assignment, the best k value should be 12.

- **Naive Bayes Algorithm (NB)**

  Naive Bayes Algorithm is a kind of most important algorithm in machine learning, especially famous for classification. There are a number of concepts have to be known before designing the algorithm:
  - **Priori probability P(X):** Probability based on past experience and analysis
  - **Posterior probability P(Y|X):** Indicates the probability of event Y occurring under the premise that event X has occurred
  - **Naive:** Assuming that each feature is independent of each other

  Naive Bayes formula:

$$P(Y|X) = \frac{P(x_1|Y)P(x_2|Y)\cdots P(x_n|Y)P(Y)}{P(X)}$$

  In order to realize the Naive Bayes algorithm, we have to follow these steps:
  - *Divide samples by category*
  - *Calculate prior probability*
  - *Extract attribute features by category and counts the times*
  - *Calculate the posterior probability*
  - *Prediction sample (find the class with the largest probability value)*

  As the formula shows, so as to calculate the probability of the final prediction, we have to find the Priori probability and Posterior probability for the final prediction. For the simpler calculation and in order to avoid underflow caused by continuous multiplication operations, log-likelihood is used in this assignment, like this way: log(P(x1|Y)*P(x2|Y)*...*P(xn|Y)) = log(P(x1|Y)) +log(P(x2|Y))+...+log(P(xn|Y)), which means the coding realization could be much easier while the result is going to be the same.

In [80]:
```
def NavieBayes(train, label):
    total_count = train.shape[0]

    separated = [] # Combine same type items into one list
    prob_list = [] # Calculate prior probability
    sum_list = []
    predict_list = []

    for classes in np.unique(label):
        same_list = []
        for sample, label_sample in zip(train, label):
            if label_sample == classes:
                same_list.append(sample)

        separated.append(same_list)

    for i in separated:
        prob = np.log(len(i) / total_count)
        prob_list.append(prob)

    for count in separated:
        sum_count = np.array(count).sum(axis=0)
        sum_list.append(sum_count)

        counts = np.array(sum_list) + 1

    for j in train:
        feature_prob = np.log(counts / counts.sum(axis=1)[np.newaxis].T)
        predict_prob = (feature_prob * j).sum(axis=1) + prob
        predict_list.append(predict_prob)

    final_predict = np.argmax(predict_list, axis=1)

    return final_predict
```

When it comes to the part of extract attribute features by category, the number of occurrences of a feature might be zero, while the domain of logarithm must be greater than 0. In order to avoid this situation, when calculating the counts, we have to add a constant 1 which could make sure the count is larger than 0 (Laplacian correction).

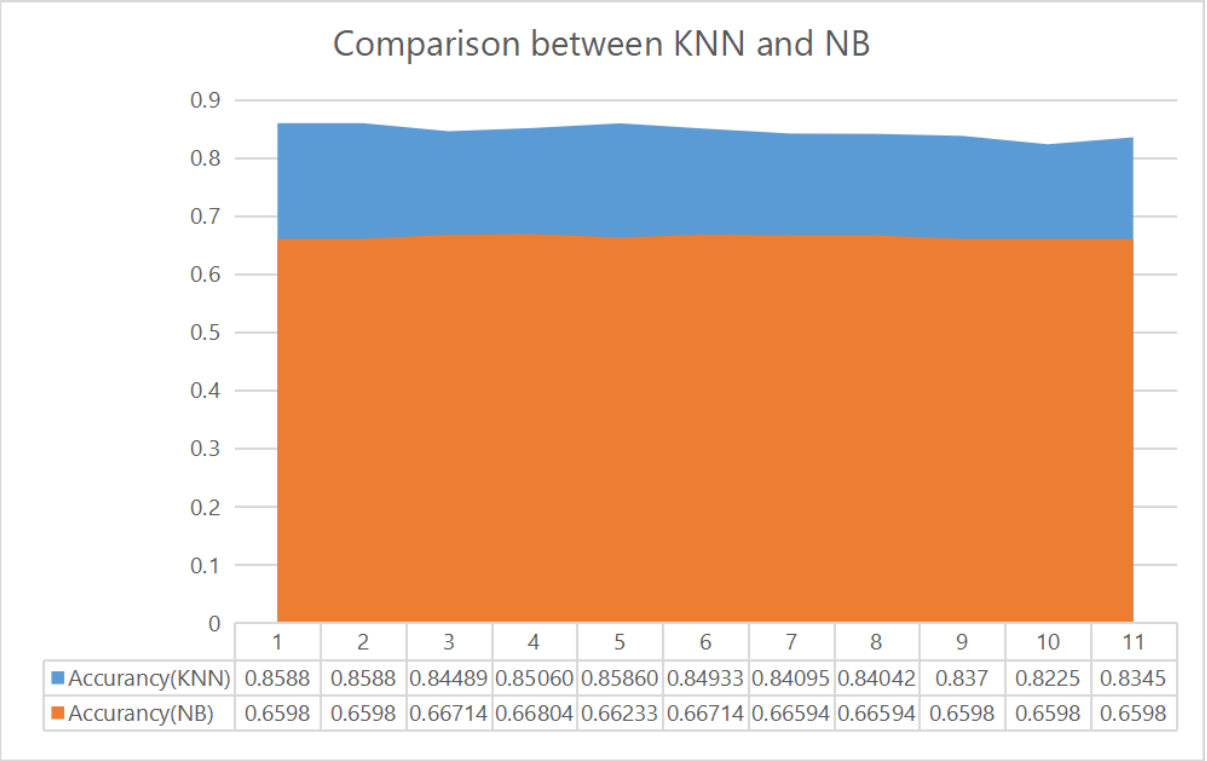
Experiments and Results

Regarding the choice between the KNN algorithm and the Naive Bayes algorithm, we have to use the same training set and test set to verify the accuracy of each algorithm. At the same time, in order to avoid directly using the provided final test set to adjust the hyperparameters, for example, using PCA to explain how many dimensions the data is reduced and how many nearest neighbors should be selected in KNN, the validation based on the training set has been used for receiving reasonable results, especially for finding the best hyperparameters.

In [27]:
```
def my_train_test_split(X, train_size=0.8, shuffle=True):
    border = int(train_size*len(X))
    X_train, X_test = X[:border], X[border:]
    return X_train, X_test
```

```
In [87]: split_train, split_test = my_train_test_split(data_train)
         split_label_train, split_label_test = my_train_test_split(label_train)
```

By realizing the data split, the initial training dataset has been separated randomly into two different datasets, which are considered as training dataset and test dataset, while the initial training label dataset has been split as well. In the following coding, all the predictions are based on these new datasets. Also, as the final test dataset and test label dataset are all in size 5000, so as to make sure the result could be reasonable, the setting of the size of the new test dataset is at least 5000, which means the parameter(train_size) in the split function is at least %.



	1	2	3	4	5	6	7	8	9	10	11
Accurancy(KNN)	0.8588	0.8588	0.84489	0.85060	0.85860	0.84933	0.84095	0.84042	0.837	0.8225	0.8345
Accurancy(NB)	0.6598	0.6598	0.66714	0.66804	0.66233	0.66714	0.66594	0.66594	0.6598	0.6598	0.6598

Putting the newly divided datasets into the two algorithms, after a period of calculation, we can clearly see the difference in the results of the two. To put it simply, when the content of the datasets is the same and the data dimension is the same, the overall performance of KNN is better than NB, and the average prediction accuracy is nearly 20% higher.

Since KNN's hyperparameters are selected independently and have no direct effect on the overall comparison, this report is going to focus on the PCA dimension and the size of splitted datasets. Here are some tables includes some results by changing parameters:

Validation	Accuracy(KNN)	Accuracy(NB)	Component(PCA)
5/6	85.88%	65.98%	40
0.8	85.86%	66.23%	40

0.7	84.93%	66.71%	40
5/6	85.88%	65.98%	20
0.8	85.06%	66.80%	20
0.7	84.86%	66.59%	20

As the table above shows, there are only a few results by using different parameters while the hyperparameter K for KNN algorithm is unchanging 12. When the size of datasets changes, the larger the test dataset is, the less accuracy of KNN and larder accuracy of NB are. When the component of PCA changes, there are not clear differences among them, but overall, the component 40 is better than 20.

Obviously, for the results, there is no doubt that the KNN algorithm is much better than the Naive Bayes algorithm. Nevertheless, for the training and inference time consumption, NB is much better than KNN. The complete process of KNN algorithm requires a lot of time, while at the same time, the Naive Bayes algorithm only needs a short time, which might only be 1 or 2 seconds. This could help NB to calculate the result without reducing dimensionality.

The reason why KNN needs such a long time to do the calculation is that it has a huge amount of calculation which leads to the slow analysis speed, also, high complexity of calculation and space increased computing time. However, KNN does not need to estimate parameters and it could provide high precision. That is the biggest reason for the final choice. For the Naive Bayes algorithm, in spite of it has stable classification efficiency and the principle is simple for understanding, but since we determine the posterior probability through the prior probability and data to determine the classification, there is a certain error rate in the classification decision, that is the reason for the low accuracy.

Conclusion

Based on the results of these algorithms, the final choice is the KNN algorithm, though it needs a long time to provide the prediction, while it could give the best accuracy when compared with the NB algorithm. So the output algorithm is KNN.

For the improvement of these two algorithms, the code structure of KNN still needs further optimization, like reducing the internal loop statements and so on. When it comes to the Naive Bayes algorithm, a more rigorous Gaussian distribution can be used to replace the log likelihood, which might have a great influence on the results. In general, the code needs more refined optimization, not only to ensure the meticulous logic, but also to effectively reduce the computing time.

Considering the advantages and disadvantages of these two algorithms, actually, neither of them is the best choice. There must be more efficient classification algorithms, which could keep high accuracy and spend less time than the KNN algorithm. In order to master these high-level algorithms, I also need to learn deeper theoretical knowledge to enhance my logical thinking ability.

References

Xiaolin,H. 1st of September 2018. "机器学习系列之手把手教你实现一个 naiveBayes", availble at: https://developer.ibm.com/zh/articles/machine-learning-hands-on3-naivebayes/

Appendix

Hardware and software spec used for evaluation:
Hardware: Inter(R) Core(TM) i7-6700HQ CPU @ 2.60Ghz
RAM 12.0GB
Software: Python 3 Jupyter Notebook

Steps for running code:

In order to run the code with a final 5000 size test dataset, when you run the code, please replace the validations. In details:

In [105]:
```
## load datasets
### WARNING: check the path of files
with h5py.File('./Assignment1/Input/train/images_training.h5','r') as H:
    data_train = np.copy(H['datatrain'])
with h5py.File('./Assignment1/Input/train/labels_training.h5','r') as H:
    label_train = np.copy(H['labeltrain'])
with h5py.File('./Assignment1/Input/test/images_testing.h5','r') as H:
    data_test = np.copy(H['datatest'])
with h5py.File('./Assignment1/Input/test/labels_testing_2000.h5','r') as H:
    label_test = np.copy(H['labeltest'])
```

Please replace the final 5000 size label_test file instead of 'labels_testing_2000.h5'. And followed by: **Do not use the split function as it only used for getting validations.**

Next directly go to the function PCA:

:
```
## Get the datasets processed by PCA
pca_data_train, eig_vecs_train, recon = PCAtrain(40, split_train) # The component is 40
pca_data_test = PCAtest(split_test)
```

Replace the 'split_train' with 'data_train' (the initial training dataset) and replace the 'split_test' with 'data_test' as well.

Then for the result print out, as shown below:

In [122]:
```
## Get predictions from KNN
knn = KNN(pca_data_train, pca_data_test[0:5000], split_label_train, 12)
```

In [123]:
```
## Reput the output file for recheck (if need)
#with h5py.File('./Assignment1/Output/predicted_labels.h5','r') as H:
    #KNN = np.copy(H['Output'])
```

In [124]:
```
## Calculate the accuacy of KNN
def accurancyKNN(kn, label):
    count = 0

    for i in range(label.shape[0]):
        if kn[i] == label[i]: # if label in kn is same as label list, count +1
            count += 1

    accurancy = count/(label.shape[0]) # Calculate the accurancy
    return accurancy
```

In [125]:
```
## Get the results by percentage
accu = accurancyKNN(knn, split_label_test[0:5000])
print("percent: {:.4%}".format(accu))
```

Please change the ‘split_label_train’ to ‘label_train’ and change the’split_label_test’ to ‘label_test’ as well. Since the final ‘label_test’ file is size (5000,), the range behind ‘label_test’ does not need to be changed or removed.

Finally, directly go to the end of the code file,

```
## Output the KNN result list (5000,)
## Check the file path as well
with h5py.File('./Assignment1/Output/predicted_labels.h5','w') as H:
    H.create_dataset('Output',data=knn)
```

Make sure the output data is ‘knn’ and check the output path. Then you should get a h5 file with (5000,) size, which are labels predicted by KNN.