SIT744 Practical Machine Learning for Data Science

Assignment Two: Deep Neural Networks, Representation Learning, and Text Analytics

Due: 24:00pm 24 September 2019 (Tuesday)

Important note: This is an individual assignment. It contributes 40% to your final mark. Read the assignment instruction carefully.

This notebook has been prepared for your to complete Assignment 2. The theme of this assignment is about practical machine learning knowledge and skills in deep neural networks, word embedding and text analytics. Some sections have been partially completed to help you get started. **The total marks for this notebook are 80 marks, which will be re-scaled to 40 marks in the grade.**.

- Before you start, read the entire notebook carefully once to understand what you need to do.
- For each cell marked with **#YOU ARE REQUIRED TO INSERT YOUR CODES IN THIS CELL**, there will be places where you **must** supply your own codes when instructed.

Instruction

This assignment contains **two** parts

- Part 1: Deep Feedforward Neural Network **[45 points]**
- Part 2: Word2Vec, text analytics and application **[35 points]**

Hint: This assignment was essentially designed based on the lectures and practical lab sessions covered from Week 5 to 9. You are strongly recommended to go through these contents thoroughly which might help you to complete this assignment.

What to submit

This assignment is to be completed individually and submitted to CloudDeakin. **By the due date, you are required to submit the following files to the corresponding Assignment in CloudDeakin**:

1. **[YourID]_assignment2_solution.ipynp**: **this is your Python notebook solution source file**.
2. **[YourID]_assingment2_output.html**: **this is the output of your Python notebook solution *exported* in html format**.
3. Any extra files needed to complete your assignment (e.g., images used in your answers).

For example, if your student ID is: 123456, you will then need to submit the following files:

- 123456_assignment2_solution.ipynp

- 123456_assignment2_output.html
- any extra files or subfolder you might have (this can be named according to your preference).

Please proceed to the content below to complete your assignment!

Part 1: Deep Feedforward Neural Network

[Total mark for this part: 45 points]

The first part of this assignment is for you to demonstrate the knowledge in deep learning that you have acquired from the lectures and practical lab materials. Most of the contents in this assignment are drawn from the practical materials in week 5, 6 and 7 for deep neural networks. Going through these materials before attempting this assignment is highly recommended.

*Run the following cell to create necessary subfolders for this assignment. You must **not** modify these codes and **must** run it first*.

In [1]:

```
# Create necessary subfolders to store immediate files for this assignment.

import os
if not os.path.exists("./models/dnn0"):
    os.makedirs("models/dnn0")

#custom function for pretty printing

def printDash(n=50):
    print('='*n)
```

The first part of this assignment is to apply DNN to recognize letters from A-Z. You have played with MNIST dataset in your pracs and this should have given a good sense of how to apply DNN on images for recognition task.

In this assignment, you are going to work with the **notMNIST** dataset for *letter recognition task*. The dataset contains 10 classes of letters A-J taken from different fonts. You will see some examples at the visualization task in the next part. A short blog about the data can be found here (http://yaroslavvb.blogspot.com.au/2011/09/notmnist-dataset.html).

Here we only consider a small subset which can be found at this link (http://yaroslavvb.com/upload/notMNIST/notMNIST_small.mat). This file has been already downloaded and stored in subfolder `datasets` of this assignment folder. The file is in *Matlab* format, thus our first task is to:

Question 1.1. Load the data into *`numpy array`* format of two variables:

- x : storing features with dimension `[num_samples, width, height]` (`num_samples` : number of samples, `width` : image width, `height` : image height), and
- y : storing labels with dimension `num_samples` .

[3 points]

Enter the missing codes in the following cell to complete this question.

In [2]:

```
# YOU ARE REQUIRED TO INSERT YOUR CODES IN THIS CELL
from prettytable import PrettyTable
import numpy as np
import scipy.io as sio
import pandas as pd
from pprint import pprint

data = sio.matlab.loadmat("datasets/notMNIST_small.mat")
#print(type(data))
#print(data.images)
#print(data['images'])
print(data.keys())
x, y = data['images'],data['labels']
x = np.rollaxis(x, axis=2)
print(y.shape)
```

```
dict_keys(['__header__', '__version__', '__globals__', 'images', 'labe
ls'])
(18724,)
```

Question 1.2. Print out the total number of data points, and the *unique* labels in this dataset.

[3 points]

In [3]:

```
#YOU ARE REQUIRED TO INSERT YOUR CODES IN THIS CELL
print("Data points = ",format(x.shape[0]))
print("Unique labels = ",format(np.unique(y)))
```

```
Data points =  18724
Unique labels =  [0. 1. 2. 3. 4. 5. 6. 7. 8. 9.]
```

Question 1.3. Display 100 images in the form of `10x10` matrix, each row showing 10 *random* images of a label. You might decide to use the function `display_images` provided at the beginning of this assignment, or you can write your own codes.

[4 points]

In [4]:

```
# this function is a utility to display images from the dataset
import numpy as np
import matplotlib.pyplot as plt
%matplotlib inline

def display_images(images, shape):
    fig = plt.figure(figsize=shape)
    fig.subplots_adjust(left=0, right=1, bottom=0, top=1, hspace=0.05, wspace=0.05)
    for i in range(np.prod(shape)):
        p = fig.add_subplot(shape[0], shape[1], i+1, xticks=[], yticks=[])
        p.imshow(images[i], cmap=plt.cm.bone)
```

In [5]:

```
# YOU ARE REQUIRED TO INSERT YOUR CODES IN THIS CELL

unique_labels = np.unique(y)
images = []
for l in unique_labels:
    idx = np.where(y == l)[0]
    idx = idx[np.random.permutation(len(idx))[:10]]
    for i in idx:
        images += [x[i]]

display_images(images, shape=(10, 10))
```

Question 1.4. Use the *deep feedforward neural network* as the classifier to perform images classification task in a *single split training and testing*.

The total marks for this question is **[35 points]**, with the following detailed breakdown sub-questions:

(a) Write your code to reshape the variable `x` storing features from `[num_samples, width, height]` dimension to `[num_samples, num_features]` with `num_features = width x height`. * (Hint*: you might want to use the `reshape()` function)

[3 points]

In [6]:

```
# YOU ARE REQUIRED TO INSERT YOUR CODES IN THIS CELL
r,c = x[0].shape
x = np.reshape(x,(y.shape[0],r*c))
print(x.shape)
```

```
(18724, 784)
```

In training the DNN, scaling data is important. The pixel intensities of images are in the range of [0, 255], which makes the neural network difficult to learn.

Rescale the input data into the range of [0, 1]

[2 points]

In [28]:

```
# YOU ARE REQUIRED TO INSERT YOUR CODES IN THIS CELL

from sklearn.preprocessing import MinMaxScaler

x_scale = MinMaxScaler().fit(x).transform(x)
print("After scaling:")
print("Min =",format(x_scale.min()))
print("Max =",format(x_scale.max()))
```

```
After scaling:
Min = 0.0
Max = 1.0
```

(b) Split the data into two subsets: 70% for training and 30% for testing. Note that you must use *Stratified-Shuffle-Split* (http://scikit-learn.org/stable/modules/generated/sklearn.model_selection.StratifiedShuffleSplit.html) to make sure training and testing are balanced and randomly shuffled before learning the model.

[5 points]

In [8]:

```
# YOU ARE REQUIRED TO INSERT YOUR CODES IN THIS CELL
from sklearn.model_selection import StratifiedShuffleSplit
ss_split=StratifiedShuffleSplit(n_splits=1,test_size=0.3,random_state=11795)

for train_index, test_index in ss_split.split(x_scale,y):
#     print("TRAIN:", train_index, "TEST:", test_index)
    print(train_index.shape[0]+test_index.shape[0])
    X_train, X_test = x_scale[train_index], x_scale[test_index]
    y_train, y_test = y[train_index], y[test_index]
```

```
18724
```

(c) Construct a deep feedforward neural network with the following architecture:

- An input layer followed by *two* hidden layers, each with *500* hidden units, and an *output* layer;
- *ReLU* activations for neurons in each hidden layer;
- Training with gradient descent optimizer with learning rate **0.0011**, batch size 128 and 50 epochs.

(*Hint*: *this question heavily relies on the knowledge you've learned from lab session in week 5 and 6. You are encouraged to revise these materials for this question*)

[20 points]

In [9]:

```
# YOU ARE REQUIRED TO INSERT YOUR CODES IN THIS CELL [5 marks]

import tensorflow as tf

tf.reset_default_graph()

learn_rate = 0.0011
num_inputs = r*c
num_hidden1 = 500
num_hidden2 = 500
num_outputs = len(np.unique(y))

inputs =  tf.placeholder(tf.float32,shape=[None, num_inputs],name='inputs')
labels = tf.placeholder(tf.int64,shape=[None],name='labels')

W = tf.Variable(tf.truncated_normal([num_inputs, num_outputs], stddev=0.02), name='v
b = tf.Variable(tf.zeros([num_outputs]), name='biases')
logits = tf.add(tf.matmul(inputs, W),b, name='logits')
with tf.name_scope('evaluation'):
    xentropy = tf.nn.sparse_softmax_cross_entropy_with_logits(labels=labels, logits=
    loss = tf.reduce_mean(xentropy, name='loss')
    correct = tf.nn.in_top_k(logits, labels, 1)
    accuracy = tf.reduce_mean(tf.cast(correct, tf.float32), name='accuracy')
with tf.name_scope("train"):
    optimizer=tf.train.GradientDescentOptimizer(learn_rate)
    training_op=optimizer.minimize(loss)

init = tf.global_variables_initializer()
```

In [10]:

```
# YOU ARE REQUIRED TO INSERT YOUR CODES IN THIS CELL [3 marks]

def neuron_layer(x, num_neurons, name, activation=None):
    with tf.name_scope(name):
        num_inputs=int(inputs.get_shape()[1])
        std_dev=2/np.sqrt(num_inputs)
        init=tf.truncated_normal([num_inputs,num_neurons],stddev=std_dev)
        b = tf.Variable(tf.zeros([num_neurons]), name='biases')
        W = tf.Variable(init,name='weights')
        z = tf.matmul(inputs,W)+b
    if activation == "sigmoid":
        return tf.nn.sigmoid(z)
    elif activation == "relu":
        return tf.nn.relu(z)
    else:
        return z
```

In [11]:

```
# YOU ARE REQUIRED TO INSERT YOUR CODES IN THIS CELL [7 marks]

with tf.name_scope("dnn"):
    hidden1 = neuron_layer(num_inputs,num_hidden1, "hidden1", activation="relu")
    hidden2 = neuron_layer(hidden1, num_hidden2, "hidden2", activation="relu")
    logits = neuron_layer(hidden2, num_outputs, "output")
with tf.name_scope("loss"):
    xentropy = tf.nn.sparse_softmax_cross_entropy_with_logits(labels=labels, logits=
    loss = tf.reduce_mean(xentropy, name="loss")

with tf.name_scope("evaluation"):
    xentropy = tf.nn.sparse_softmax_cross_entropy_with_logits(labels=labels, logits=
    correct = tf.nn.in_top_k(logits, labels, 1)
    accuracy = tf.reduce_mean(tf.cast(correct, tf.float32), name='accuracy')

with tf.name_scope("train"):
    optimizer=tf.train.GradientDescentOptimizer(learn_rate)
    grads=optimizer.compute_gradients(loss)
    training_op=optimizer.apply_gradients(grads)
    init = tf.global_variables_initializer()
    saver=tf.train.Saver()

    for var in tf.trainable_variables():
        tf.summary.histogram(var.op.name + "/values", var)

    for grad, var in grads:
        if grad is not None:
            tf.summary.histogram(var.op.name + "/gradients", grad)

# summary
accuracy_summary = tf.summary.scalar('accuracy',accuracy)
```

In [12]:

```
# YOU ARE REQUIRED TO INSERT YOUR CODES IN THIS CELL [5 marks]

# merge all summary
tf.summary.histogram('hidden1/activations', hidden1)
tf.summary.histogram('hidden2/activations', hidden2)
merged = tf.summary.merge_all()

init = tf.global_variables_initializer()
saver = tf.train.Saver()

train_writer = tf.summary.FileWriter("models/dnn0/train", tf.get_default_graph())
test_writer = tf.summary.FileWriter("models/dnn0/test", tf.get_default_graph())

num_epochs = 50
batch_size = 128
```

(d) You are now required write code to train the DNN. Write codes in the following cell. **[5 points]**

In [13]:

```
# YOU ARE REQUIRED TO INSERT YOUR CODES IN THIS CELL

with tf.Session() as sess:
    init.run()

    tbl = PrettyTable()
    tbl.field_names = ["Epoch", "Training Accuracy", "Testing Accuracy"]

    for epoch in range(num_epochs):
        for idx_start in range(0, X_train.shape[0], batch_size):
            idx_end = (idx_start+batch_size)
            x_batch, y_batch = X_train[idx_start:idx_end,:],y_train[idx_start:idx_er
            sess.run(training_op, feed_dict={inputs: x_batch, labels: y_batch})

        summary_train, acc_train = sess.run([merged,accuracy],feed_dict={inputs: x_b
        summary_test, acc_test = sess.run([merged,accuracy],feed_dict={inputs: X_tes

        train_writer.add_summary(summary_train, epoch)
        test_writer.add_summary(summary_test, epoch)
        tbl.add_row([epoch+1, acc_train,acc_test])
    print(tbl)
    save_path = saver.save(sess, "models/dnn0.ckpt")
```

```
+-------+-------------------+------------------+
| Epoch | Training Accuracy | Testing Accuracy |
+-------+-------------------+------------------+
|   1   |        0.1        |    0.25275898    |
|   2   |        0.28       |    0.4013884     |
|   3   |        0.48       |    0.51281595    |
|   4   |        0.6        |    0.59202564    |
|   5   |        0.7        |    0.6571734     |
|   6   |        0.74       |    0.7013172     |
|   7   |        0.78       |    0.73300105    |
|   8   |        0.78       |    0.75934494    |
|   9   |        0.78       |    0.7789249     |
|   10  |        0.8        |    0.7903168     |
|   11  |        0.8        |    0.7986828     |
|   12  |        0.8        |    0.8054468     |
|   13  |        0.8        |    0.81060874    |
|   14  |        0.8        |    0.8170167     |
|   15  |        0.8        |    0.81986475    |
|   16  |        0.82       |    0.8250267     |
|   17  |        0.82       |    0.8285867     |
|   18  |        0.82       |    0.8301887     |
|   19  |        0.82       |    0.83374864    |
|   20  |        0.86       |    0.83588463    |
|   21  |        0.88       |    0.83784264    |
|   22  |        0.88       |    0.84051263    |
|   23  |        0.88       |    0.84247065    |
|   24  |        0.88       |    0.8438946     |
|   25  |        0.88       |    0.8449626     |
|   26  |        0.88       |    0.8463866     |
|   27  |        0.88       |    0.8476326     |
|   28  |        0.88       |    0.8485226     |
|   29  |        0.88       |    0.8501246     |
|   30  |        0.9        |    0.8510146     |
|   31  |        0.9        |    0.8522606     |
|   32  |        0.9        |    0.85297257    |
|   33  |        0.9        |    0.85457456    |
```

```
|    34   |          0.9          |     0.8552866      |
|    35   |          0.9          |     0.85617656     |
|    36   |          0.9          |     0.8568886      |
|    37   |          0.9          |     0.8579566      |
|    38   |          0.9          |     0.8584906      |
|    39   |          0.9          |     0.85884655     |
|    40   |          0.9          |     0.8595586      |
|    41   |          0.9          |     0.85991454     |
|    42   |          0.9          |     0.8600926      |
|    43   |          0.9          |     0.8606266      |
|    44   |          0.9          |     0.86080456     |
|    45   |          0.9          |     0.8611606      |
|    46   |          0.9          |     0.8622286      |
|    47   |          0.9          |     0.86258453     |
|    48   |          0.9          |     0.8627626      |
|    49   |          0.9          |     0.8636525      |
|    50   |          0.9          |     0.8641865      |
+-------+-----------------------+--------------------+
```

Part 2: Word2Vec, Text Analytics and Application

[Total mark for this part: 35 points]

In this part, you are going to use Word2Vec for document classification on 20 Newsgroups (http://www.cs.cmu.edu/afs/cs.cmu.edu/project/theo-20/www/data/news20.html) dataset. This dataset is a collection of messages collected from 20 different netnews newsgroups. One thousand messages from each of the twenty newsgroups were chosen at random and partitioned by newsgroup name. The list of newsgroups from which the messages were chosen is as follows:

```
 alt.atheism
talk.politics.guns
talk.politics.mideast
talk.politics.misc
talk.religion.misc
soc.religion.christian
comp.sys.ibm.pc.hardware
comp.graphics
comp.os.ms-windows.misc
comp.sys.mac.hardware
comp.windows.x
rec.autos
rec.motorcycles
rec.sport.baseball
rec.sport.hockey
sci.crypt
sci.electronics
sci.space
sci.med
misc.forsale
```

Download the dataset and data pre-processing

Question 2.1 Your first task is to run the following code to download the dataset.

[1 point]

In [14]:

```
from sklearn.datasets import fetch_20newsgroups
newsgroups_all = fetch_20newsgroups(subset='all', remove=('headers'))
```

Question 2.2. Print out the total number of documents, and the *unique* labels in this dataset.

[1 point]

In [15]:

```
# YOU ARE REQUIRED TO INSERT YOUR CODES IN THIS CELL
print(newsgroups_all.keys())
printDash(75)
print("Total number of documents = ",format(len(newsgroups_all.data)))
printDash(75)
y = newsgroups_all.target
print("Unique labels are : ",format(np.unique(y)))
```

```
dict_keys(['data', 'filenames', 'target_names', 'target', 'DESCR'])
===========================================================================
=====
Total number of documents =  18846
===========================================================================
=====
Unique labels are :  [ 0  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15
16 17 18 19]
```

Question 2.3. Convert the documents into a list of tokens using the function `gensim.utils.tokenize`.

[3 point]

In [16]:

```
import gensim as gs

corpus_token=[]
for i in range(len(newsgroups_all.data)):
    #,lowercase=True,deacc=True
    corpus_token.append(list(gs.utils.tokenize(newsgroups_all.data[i],lowercase=True
```

Train the model

Question 2.4. Train gensim's word2vec model.

[5 points]

- Train gensim's word2vec model with the settings of:
  - The dimensionality of the feature vectors: `size=100` ,
  - The maximum distance between the current and predicted word within a sentence: `window=5` ,
  - Minimum frequence (ignore all words with total frequency lower than this): `min_count=5` ,

In [17]:

```
from gensim.models import Word2Vec

w2v_model = Word2Vec(corpus_token, size=100, window=5, min_count=5, workers=8,sg=1)
words = list(w2v_model.wv.vocab.keys())
```

- Save the trained model to a file named "20_newsgroups.gensim"

In [18]:

```
# YOU ARE REQUIRED TO INSERT YOUR CODES IN THIS CELL
model_name = 'models/20_newsgroups.gensim'
w2v_model.save(model_name)
```

```
//anaconda3/lib/python3.7/site-packages/smart_open/smart_open_lib.py:3
98: UserWarning: This function is deprecated, use smart_open.open inst
ead. See the migration notes for details: https://github.com/RaRe-Tech
nologies/smart_open/blob/master/README.rst#migrating-to-the-new-open-f
unction (https://github.com/RaRe-Technologies/smart_open/blob/master/R
EADME.rst#migrating-to-the-new-open-function)
  'See the migration notes for details: %s' % _MIGRATION_NOTES_URL
```

Question 2.5. Print out the vocabulary size (number of words in vocabulary).

[2 points]

In [19]:

```
# YOU ARE REQUIRED TO INSERT YOUR CODES IN THIS CELL
print("The number of words = ",format(len(words)))

print("The first 10 words in the vocabularies = ",format(words[0:10]))
```

```
The number of words =  34983
The first 10 words in the vocabularies =  ['i', 'am', 'sure', 'some',
'bashers', 'of', 'pens', 'fans', 'are', 'pretty']
```

Question 2.6. Using the embedding results, calculate and print out the ten most similar words to word 'law' and their corresponding similarity scores.

[3 points]

In [20]:

```
# YOU ARE REQUIRED TO INSERT YOUR CODES IN THIS CELL


similar_words_list = w2v_model.wv.most_similar(positive = ['law'],topn=10)
tbl = PrettyTable()
tbl.field_names = ["Words(similar to *Law*)", "Similarity score"]

for row in similar_words_list:
    tbl.add_row([row[0],row[1]])

print(tbl)
```

```
+-------------------------+--------------------+
| Words(similar to *Law*) |  Similarity score  |
+-------------------------+--------------------+
|       enforcement       | 0.8403921127319336 |
|         abiding         | 0.7467506527900696 |
|          giveth         | 0.6896644830703735 |
|     unconstitutional    | 0.6697934865951538 |
|           fca           | 0.6657330393791199 |
|         menacing        | 0.6591196060180664 |
|          laws           | 0.6582514047622681 |
|          uphold         | 0.6499788761138916 |
|          felony         | 0.6490593552589417 |
|           ccw           | 0.6488798260688782 |
+-------------------------+--------------------+
```

Evaluate the embeddings using classification

Now we investigate the quality of embedded vectors via document classification task. We have learned the embeddings for words, but not for documents yet, thus we need to find a way to extract the document embeddings from word embeddings. We are going to try two approaches:

- Taking the **sum** of vectors of all words in the document; or
- Taking the **average** of vectors of all words in the document.

Question 2.7. Extract document vectors using `sum`.

[5 points]

- Remove all ***empty*** documents. A document is empty if it does not contain any word in the vocabulary;
- Extract document vectors and save to variable `x`;
- Save the corresponding labels to variable `y`.

In [21]:

```
# YOU ARE REQUIRED TO INSERT YOUR CODES IN THIS CELL
def extract_doc2vec(doc, w2v,operator='sum'):
    vecs = [w2v[word] for word in doc if word in w2v]
    if len(vecs) > 0:
        if (operator == 'avg'): #modified for question 2.10
            vecs = np.asarray(vecs).mean(axis=0)
        else:
            vecs = np.asarray(vecs).sum(axis=0)

    return vecs


#returns the index of empty documents so that it can be removed

def remove_empty_docs(corpus,labels):
    idx = []
    for ix, i in enumerate(corpus):
        if len(i) == 0:
            #print(ix)
            idx.append(ix)

    corpus = np.delete(corpus,idx,0)
    labels = np.delete(labels,idx,0)

    return corpus,labels


vec_x_sum=[extract_doc2vec(t,w2v_model) for t in corpus_token]


vec_x_sum, y_filtered = remove_empty_docs(vec_x_sum,y)
print(vec_x_sum.shape)
print(y_filtered.shape)
```

```
//anaconda3/lib/python3.7/site-packages/ipykernel_launcher.py:3: Depre
cationWarning: Call to deprecated `__contains__` (Method will be remov
ed in 4.0.0, use self.wv.__contains__() instead).
  This is separate from the ipykernel package so we can avoid doing im
ports until
//anaconda3/lib/python3.7/site-packages/ipykernel_launcher.py:3: Depre
cationWarning: Call to deprecated `__getitem__` (Method will be remove
d in 4.0.0, use self.wv.__getitem__() instead).
  This is separate from the ipykernel package so we can avoid doing im
ports until

(18809,)
(18809,)
```

Question 2.8. Print out the number of documents retained after removing empty documents.

[1 point]

In [22]:

```
# YOU ARE REQUIRED TO INSERT YOUR CODES IN THIS CELL

print("Number of documents after removing empty = ",format(len(vec_x_sum)))


y_filtered = np.array(y_filtered)
vec_x_sum = np.array(vec_x_sum)
```

```
Number of documents after removing empty =  18809
```

Question 2.9. Split the data into two subsets: 70% for training and 30% for testing. Note that you must use *Stratified-Shuffle-Split* (http://scikit-learn.org/stable/modules/generated/sklearn.model_selection.StratifiedShuffleSplit.html) to make sure training and testing are balanced and randomly shuffled before learning the model.

[2 points]

In [23]:

```
# YOU ARE REQUIRED TO INSERT YOUR CODES IN THIS CELL

ssts = StratifiedShuffleSplit(n_splits=1,test_size=0.3, random_state=999)

for train_index, test_index in ssts.split(vec_x_sum,y_filtered):
    y_train, y_test = y_filtered[train_index], y_filtered[test_index]
    X_train, X_test = vec_x_sum[train_index], vec_x_sum[test_index]

print(X_train.shape,y_test.shape)
pprint(X_train[1].shape)
```

```
(13166,) (5643,)
(100,)
```

Question 2.10. Use Logistic Regression (http://scikit-learn.org/stable/modules/generated/sklearn.linear_model.LogisticRegression.html) as the classifier, train and test the model using the training and test datasets from the previous step. Report the training accuracy and testing accuracy.

[2 points]

In [24]:

```
#cleaning data

x_train = pd.DataFrame(list(X_train))
x_test  = pd.DataFrame(list(X_test))
null_data = x_train[x_train.isnull().any(axis=1)]

x_train.fillna(x_train.mean(),inplace=True)

print("Shape of null/empty data",format(null_data.shape))
```

```
Shape of null/empty data (0, 100)
```

In [25]:

```
##### YOU ARE REQUIRED TO INSERT YOUR CODES IN THIS CELL
from sklearn.linear_model import LogisticRegression #as LogReg
from sklearn.metrics import accuracy_score
from sklearn.model_selection import cross_val_score


mx_iter = 99^999
logReg = LogisticRegression(solver='lbfgs',max_iter=mx_iter,multi_class='multinomial
logReg.fit(x_train,y_train)

tbl = PrettyTable()
tbl.field_names = ["Accuracy", "Value"]


y_pred=logReg.predict(x_train)
train_accuracy_sum=accuracy_score(y_train,y_pred)
tbl.add_row(['Training',train_accuracy_sum])
#print("Train Accuracy =",format(train_accuracy))

y_pred=logReg.predict(x_test)
test_accuracy_sum=accuracy_score(y_test,y_pred)
tbl.add_row(['Testing',test_accuracy_sum])
#print ("Test Accuracy =",format(test_accuracy))
'''
train_accuracy=cross_val_score(logReg,x_train,y_train,scoring='accuracy', verbose=1,
#print("Training accuracy =",format(np.mean(train_accuracy)))
tbl.add_row(['Training - CV',np.mean(train_accuracy)])

test_accuracy=cross_val_score(logReg,x_test,y_test, scoring='accuracy', verbose=1, n
#print("Testing accuracy =",format(np.mean(test_accuracy)))
tbl.add_row(['Testing - CV',np.mean(test_accuracy)])
'''

print(tbl)
```

```
+----------+--------------------+
| Accuracy |       Value        |
+----------+--------------------+
| Training | 0.7662919641500835 |
| Testing  | 0.7185894027999291 |
+----------+--------------------+

//anaconda3/lib/python3.7/site-packages/sklearn/linear_model/logistic.
py:947: ConvergenceWarning: lbfgs failed to converge. Increase the num
ber of iterations.
  "of iterations.", ConvergenceWarning)
```

Question 2.11. Now modify the `extract_doc2vec` function above to extract document vectors using `average`, instead of `sum`, and repeat the experiment: split the data, train and test using Logistic Regression.

[5 points]

In [26]:

```
##### YOU ARE REQUIRED TO INSERT YOUR CODES IN THIS CELL

vec_x_avg=[extract_doc2vec(t,w2v_model,'avg') for t in corpus_token]
vec_x_avg , y_filtered = remove_empty_docs(vec_x_avg,y)
vec_x_avg = np.array(vec_x_avg)


ssts = StratifiedShuffleSplit(n_splits=1,test_size=0.3, random_state=999)

for train_index, test_index in ssts.split(vec_x_avg,y_filtered):
    y_train, y_test = y_filtered[train_index], y_filtered[test_index]
    X_train, X_test = vec_x_avg[train_index], vec_x_avg[test_index]


x_train = pd.DataFrame(list(X_train))
x_test  = pd.DataFrame(list(X_test))
null_data = x_train[x_train.isnull().any(axis=1)]

x_train.fillna(x_train.mean(),inplace=True)


lr = LogisticRegression(solver='lbfgs',max_iter=mx_iter,multi_class='multinomial')
lr.fit(x_train,y_train)

tbl = PrettyTable()
tbl.field_names = ["Accuracy", "Value"]

y_pred=lr.predict(x_train)
train_accuracy_avg = accuracy_score(y_train,y_pred)
tbl.add_row(['Training',train_accuracy_avg])
#print("Train Accuracy =",format(train_accuracy))

y_pred=lr.predict(x_test)
test_accuracy_avg=accuracy_score(y_test,y_pred)
tbl.add_row(['Testing',test_accuracy_avg])
#print ("Test Accuracy =",format(test_accuracy))

'''
train_accuracy=cross_val_score(lr,x_train,y_train,scoring='accuracy', verbose=1, n_j
#print("Training accuracy =",format(np.mean(train_accuracy)))
tbl.add_row(['Training - CV',np.mean(train_accuracy)])

test_accuracy=cross_val_score(lr,x_test,y_test, scoring='accuracy', verbose=1, n_jo
#print("Testing accuracy =",format(np.mean(test_accuracy)))
tbl.add_row(['Testing - CV',np.mean(test_accuracy)])
'''

print(tbl)
```

```
//anaconda3/lib/python3.7/site-packages/ipykernel_launcher.py:3: Depre
cationWarning: Call to deprecated `__contains__` (Method will be remov
ed in 4.0.0, use self.wv.__contains__() instead).
  This is separate from the ipykernel package so we can avoid doing im
ports until
//anaconda3/lib/python3.7/site-packages/ipykernel_launcher.py:3: Depre
cationWarning: Call to deprecated `__getitem__` (Method will be remove
d in 4.0.0, use self.wv.__getitem__() instead).

  This is separate from the ipykernel package so we can avoid doing im
```

```
ports until

+----------+--------------------+
| Accuracy |       Value        |
+----------+--------------------+
| Training | 0.7304420476986176 |
| Testing  | 0.7136275031011873 |
+----------+--------------------+
```

Question 2.12. Which representation (sum or average doc vector) gives the best performance? Write your observations and any lessons learned.

[5 points]

Vector represented using *SUM* gives the best performance.

Vector	*Train*	*Test*
SUM	*0.7662919641500835*	*0.7185894027999291*
Average	0.7304420476986176	0.7136275031011873

As the vector calculated using the sum, actually sums the whole length and then perform operations unlike, *average* which just considers the average and base the result on that. In our case, the sample size is less and hence the difference cannot be seen much, but it cannot be used in production.

END OF ASSIGNMENT