LAB 2 REPORT

CMPE 257 MACHINE LEARNING

Utsav Chandresh Savaliya
Utsavchandresh.savaliya@sjsu.edu

Task 1 Artificial Neural Network

1. Load the dataset (CIFAR – 10) from the given link:

Data = cifar10.load_data( )

2. Explain in your own words:

a. Gradient Descent

Gradient descent is an optimization algorithm which is commonly-used to train machine learning models and neural networks. Training data helps these models learn over time, and the cost function within gradient descent specifically acts as a barometer, gauging its accuracy with each iteration of parameter updates.

b. Drop out

The term "dropout" refers to dropping out units (both hidden and visible) in a neural network. Dropout refers to ignoring units (i.e. neurons) during the training phase of certain set of neurons which is chosen at random. By "ignoring", I mean these units are not considered during a particular forward or backward pass.

c. Activation functions

Activation function decides, whether a neuron should be activated or not by calculating weighted sum and further adding bias with it. The purpose of the activation function is to introduce non-linearity into the output of a neuron.

d. Back Propagation

When training a neural network by gradient descent, a loss function is calculated, which represents how far the network's predictions are from the true labels. Backpropagation allows us to calculate the gradient of the loss function with respect to each of the weights of the network. This enables every weight to be updated individually to gradually reduce the loss function over many training iterations.

e. Epochs, Iterations and Batch size

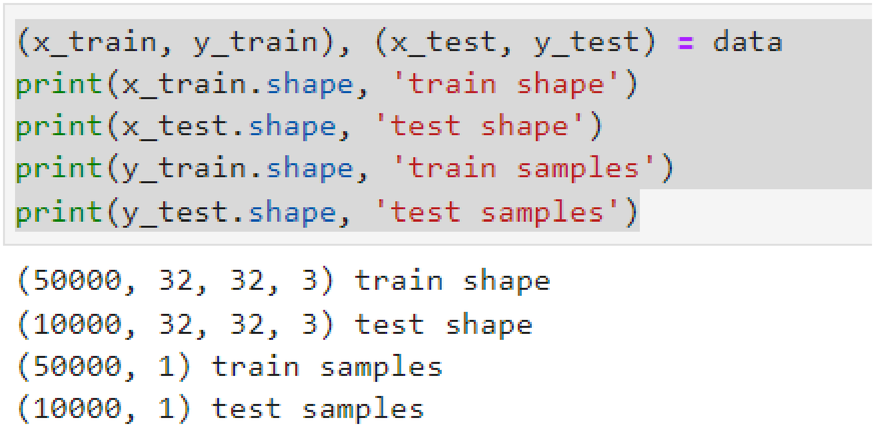
An **epoch** is a term used in machine learning that refers to the number of passes the machine learning algorithm has made over the entire training dataset.

The **batch size** is the number of samples processed before updating the model. The number of epochs represents the total number of passes through the training dataset.

Iterations is the number of batches needed to complete one epoch.

3. Visualize / summarize the data:

a. Number of entities in training and testing set and number of classes in target variable



```
(x_train, y_train), (x_test, y_test) = data
print(x_train.shape, 'train shape')
print(x_test.shape, 'test shape')
print(y_train.shape, 'train samples')
print(y_test.shape, 'test samples')
```

```
(50000, 32, 32, 3) train shape
(10000, 32, 32, 3) test shape
(50000, 1) train samples
(10000, 1) test samples
```

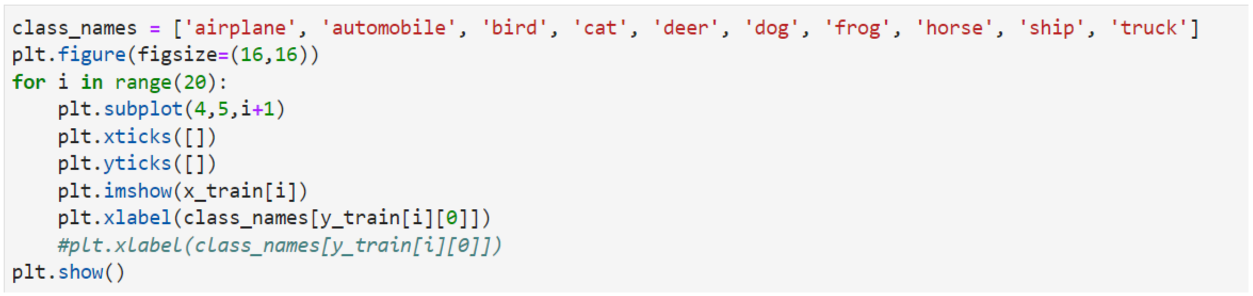
b. Number of pixels in the image (Height and width individually)

Rows = 32 and Columns=32. Therefore number of pixels= 32*32

c. Number of images per class

There are 6000 images per class which makes a total of 60,000 images in the whole Dataset

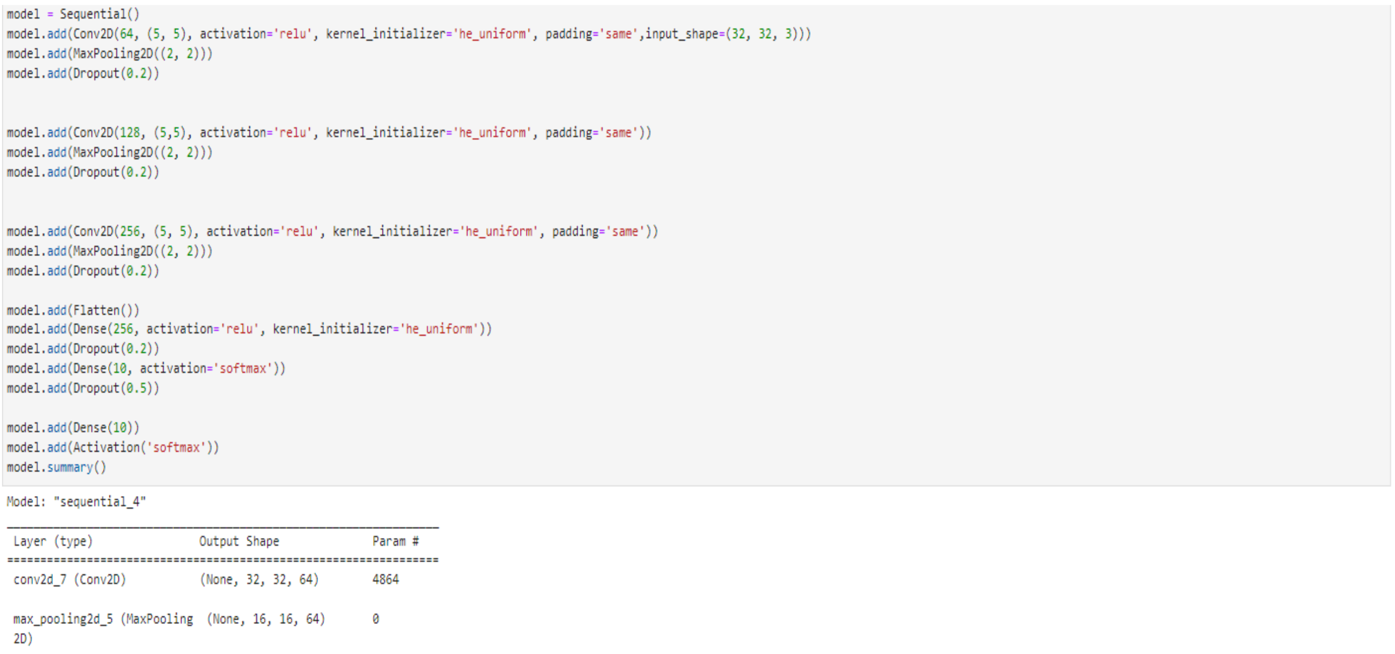
d. Display at least 2 images of each class



```
class_names = ['airplane', 'automobile', 'bird', 'cat', 'deer', 'dog', 'frog', 'horse', 'ship', 'truck']
plt.figure(figsize=(16,16))
for i in range(20):
    plt.subplot(4,5,i+1)
    plt.xticks([])
    plt.yticks([])
    plt.imshow(x_train[i])
    plt.xlabel(class_names[y_train[i][0]])
    #plt.xlabel(class_names[y_train[i][0]])
plt.show()
```

4. Train a neural network

a. Decide the number of layers and neurons in each layer

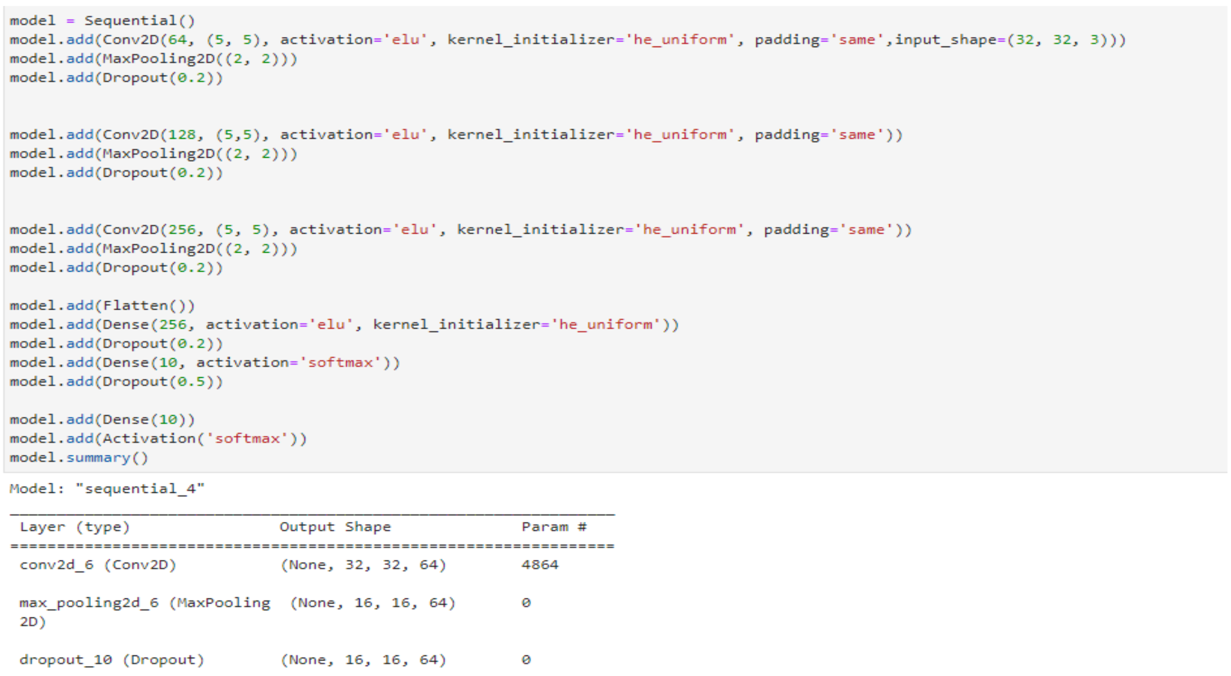


```
model = Sequential()
model.add(Conv2D(64, (5, 5), activation='relu', kernel_initializer='he_uniform', padding='same',input_shape=(32, 32, 3)))
model.add(MaxPooling2D((2, 2)))
model.add(Dropout(0.2))


model.add(Conv2D(128, (5,5), activation='relu', kernel_initializer='he_uniform', padding='same'))
model.add(MaxPooling2D((2, 2)))
model.add(Dropout(0.2))


model.add(Conv2D(256, (5, 5), activation='relu', kernel_initializer='he_uniform', padding='same'))
model.add(MaxPooling2D((2, 2)))
model.add(Dropout(0.2))

model.add(Flatten())
model.add(Dense(256, activation='relu', kernel_initializer='he_uniform'))
model.add(Dropout(0.2))
model.add(Dense(10, activation='softmax'))
model.add(Dropout(0.5))

model.add(Dense(10))
model.add(Activation('softmax'))
model.summary()
```

```
Model: "sequential_4"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d_7 (Conv2D)           (None, 32, 32, 64)        4864

 max_pooling2d_5 (MaxPooling  (None, 16, 16, 64)       0
 2D)
```

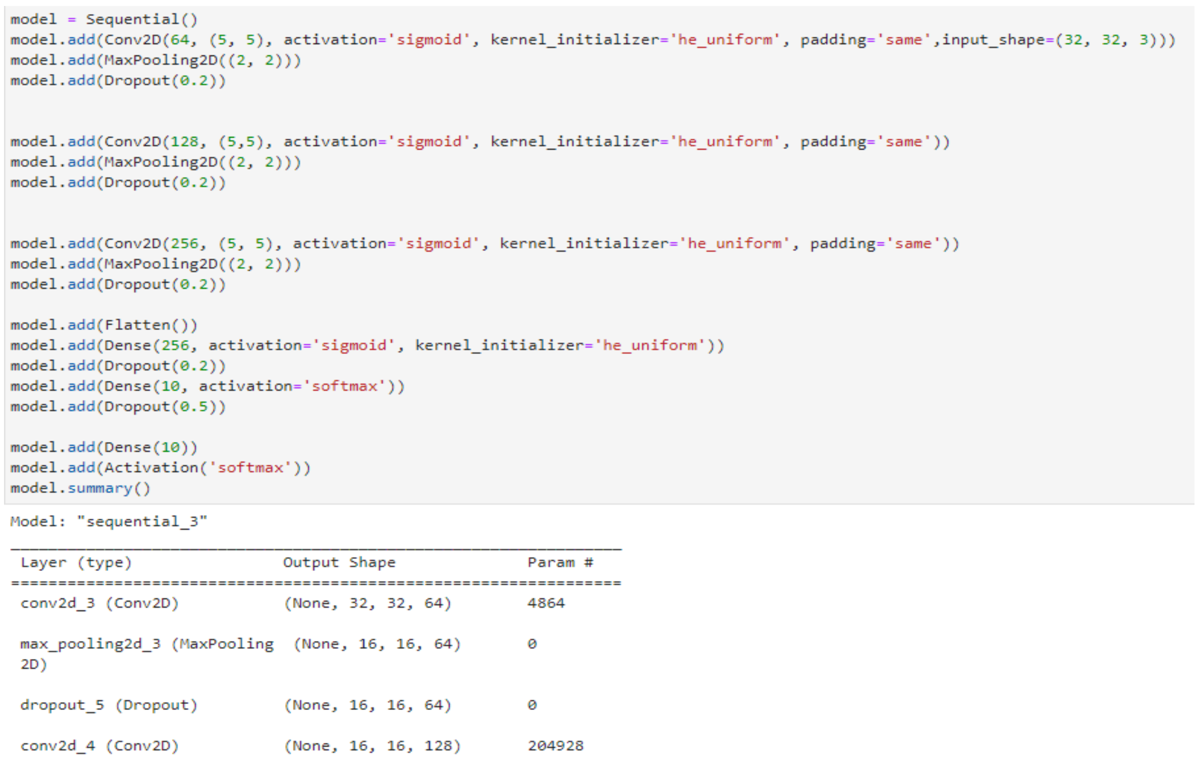
b. Try out different activation functions

Relu, sigmoid, elu etc activation function is being tried out



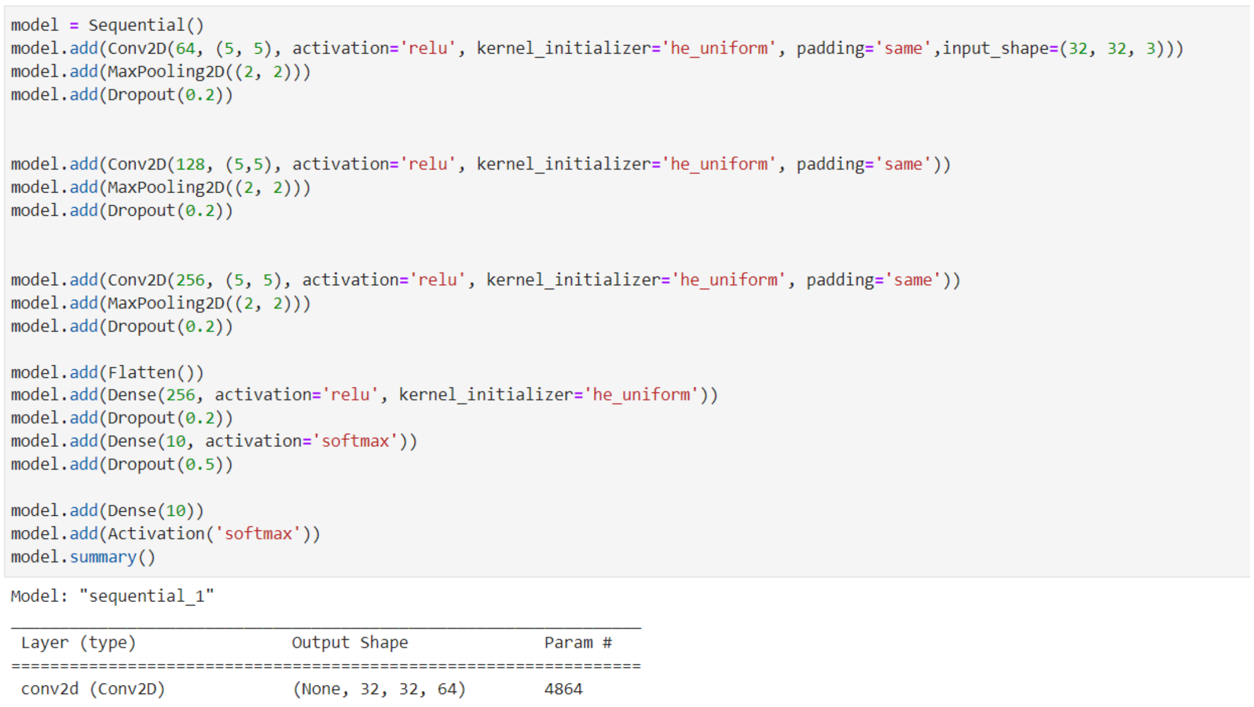
```
model = Sequential()
model.add(Conv2D(64, (5, 5), activation='elu', kernel_initializer='he_uniform', padding='same',input_shape=(32, 32, 3)))
model.add(MaxPooling2D((2, 2)))
model.add(Dropout(0.2))


model.add(Conv2D(128, (5,5), activation='elu', kernel_initializer='he_uniform', padding='same'))
model.add(MaxPooling2D((2, 2)))
model.add(Dropout(0.2))


model.add(Conv2D(256, (5, 5), activation='elu', kernel_initializer='he_uniform', padding='same'))
model.add(MaxPooling2D((2, 2)))
model.add(Dropout(0.2))

model.add(Flatten())
model.add(Dense(256, activation='elu', kernel_initializer='he_uniform'))
model.add(Dropout(0.2))
model.add(Dense(10, activation='softmax'))
model.add(Dropout(0.5))

model.add(Dense(10))
model.add(Activation('softmax'))
model.summary()
```

```
Model: "sequential_4"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d_6 (Conv2D)           (None, 32, 32, 64)        4864

 max_pooling2d_6 (MaxPooling  (None, 16, 16, 64)       0
 2D)

 dropout_10 (Dropout)        (None, 16, 16, 64)        0
```



```
model = Sequential()
model.add(Conv2D(64, (5, 5), activation='sigmoid', kernel_initializer='he_uniform', padding='same',input_shape=(32, 32, 3)))
model.add(MaxPooling2D((2, 2)))
model.add(Dropout(0.2))


model.add(Conv2D(128, (5,5), activation='sigmoid', kernel_initializer='he_uniform', padding='same'))
model.add(MaxPooling2D((2, 2)))
model.add(Dropout(0.2))


model.add(Conv2D(256, (5, 5), activation='sigmoid', kernel_initializer='he_uniform', padding='same'))
model.add(MaxPooling2D((2, 2)))
model.add(Dropout(0.2))

model.add(Flatten())
model.add(Dense(256, activation='sigmoid', kernel_initializer='he_uniform'))
model.add(Dropout(0.2))
model.add(Dense(10, activation='softmax'))
model.add(Dropout(0.5))

model.add(Dense(10))
model.add(Activation('softmax'))
model.summary()
```

```
Model: "sequential_3"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d_3 (Conv2D)           (None, 32, 32, 64)        4864

 max_pooling2d_3 (MaxPooling  (None, 16, 16, 64)       0
 2D)

 dropout_5 (Dropout)         (None, 16, 16, 64)        0

 conv2d_4 (Conv2D)           (None, 16, 16, 128)       204928
```



```
model = Sequential()
model.add(Conv2D(64, (5, 5), activation='relu', kernel_initializer='he_uniform', padding='same',input_shape=(32, 32, 3)))
model.add(MaxPooling2D((2, 2)))
model.add(Dropout(0.2))


model.add(Conv2D(128, (5,5), activation='relu', kernel_initializer='he_uniform', padding='same'))
model.add(MaxPooling2D((2, 2)))
model.add(Dropout(0.2))


model.add(Conv2D(256, (5, 5), activation='relu', kernel_initializer='he_uniform', padding='same'))
model.add(MaxPooling2D((2, 2)))
model.add(Dropout(0.2))

model.add(Flatten())
model.add(Dense(256, activation='relu', kernel_initializer='he_uniform'))
model.add(Dropout(0.2))
model.add(Dense(10, activation='softmax'))
model.add(Dropout(0.5))

model.add(Dense(10))
model.add(Activation('softmax'))
model.summary()
```

```
Model: "sequential_1"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d (Conv2D)             (None, 32, 32, 64)        4864
```

c. Try with and without using drop out

Here the Dropout is 0.3

```
model.compile(optimizer='adam',loss='sparse_categorical_crossentropy' , metrics = ['accuracy'])
result = model.fit(x_train,y_train,validation_data=(x_test,y_test),epochs=5)
```

```
Epoch 1/5
1563/1563 [==============================] - 316s 202ms/step - loss: 2.3109 - accuracy: 0.1009 - val_loss: 2.3027 - val_accuracy: 0.1000
Epoch 2/5
1563/1563 [==============================] - 346s 221ms/step - loss: 2.3042 - accuracy: 0.0987 - val_loss: 2.3028 - val_accuracy: 0.1000
Epoch 3/5
1563/1563 [==============================] - 337s 216ms/step - loss: 2.3033 - accuracy: 0.0988 - val_loss: 2.3028 - val_accuracy: 0.1000
Epoch 4/5
1563/1563 [==============================] - 355s 227ms/step - loss: 2.3043 - accuracy: 0.1001 - val_loss: 2.3027 - val_accuracy: 0.1000
Epoch 5/5
1563/1563 [==============================] - 416s 266ms/step - loss: 2.3030 - accuracy: 0.0982 - val_loss: 2.3027 - val_accuracy: 0.1000
```

Dropout as 0.5

```
model.compile(optimizer='adam',loss='sparse_categorical_crossentropy' , metrics = ['accuracy'])
result = model.fit(x_train,y_train,validation_data=(x_test,y_test),epochs=5)
```

```
Epoch 1/5
1563/1563 [==============================] - 363s 232ms/step - loss: 2.3157 - accuracy: 0.0991 - val_loss: 2.3029 - val_accuracy: 0.1000
Epoch 2/5
1563/1563 [==============================] - 388s 248ms/step - loss: 2.3036 - accuracy: 0.0983 - val_loss: 2.3027 - val_accuracy: 0.1000
Epoch 3/5
1563/1563 [==============================] - 383s 245ms/step - loss: 2.3028 - accuracy: 0.0991 - val_loss: 2.3028 - val_accuracy: 0.1000
Epoch 4/5
1563/1563 [==============================] - 381s 244ms/step - loss: 2.3089 - accuracy: 0.0990 - val_loss: 2.3028 - val_accuracy: 0.1000
Epoch 5/5
1411/1563 [==========================>...] - ETA: 35s - loss: 2.3078 - accuracy: 0.0989
```

Dropout is not used

```
model.compile(optimizer='adam',loss='sparse_categorical_crossentropy' , metrics = ['accuracy'])
result = model.fit(x_train,y_train,validation_data=(x_test,y_test),epochs=5)
```

```
Epoch 1/5
1563/1563 [==============================] - 382s 244ms/step - loss: 2.3097 - accuracy: 0.1004 - val_loss: 2.3027 - val_accuracy: 0.1000
Epoch 2/5
1563/1563 [==============================] - 337s 216ms/step - loss: 2.3029 - accuracy: 0.0985 - val_loss: 2.3028 - val_accuracy: 0.1000
Epoch 3/5
1563/1563 [==============================] - 337s 216ms/step - loss: 2.3029 - accuracy: 0.1003 - val_loss: 2.3027 - val_accuracy: 0.1000
Epoch 4/5
1563/1563 [==============================] - 327s 209ms/step - loss: 2.3029 - accuracy: 0.0979 - val_loss: 2.3027 - val_accuracy: 0.1000
Epoch 5/5
1563/1563 [==============================] - 332s 213ms/step - loss: 2.3029 - accuracy: 0.0994 - val_loss: 2.3028 - val_accuracy: 0.1000
```

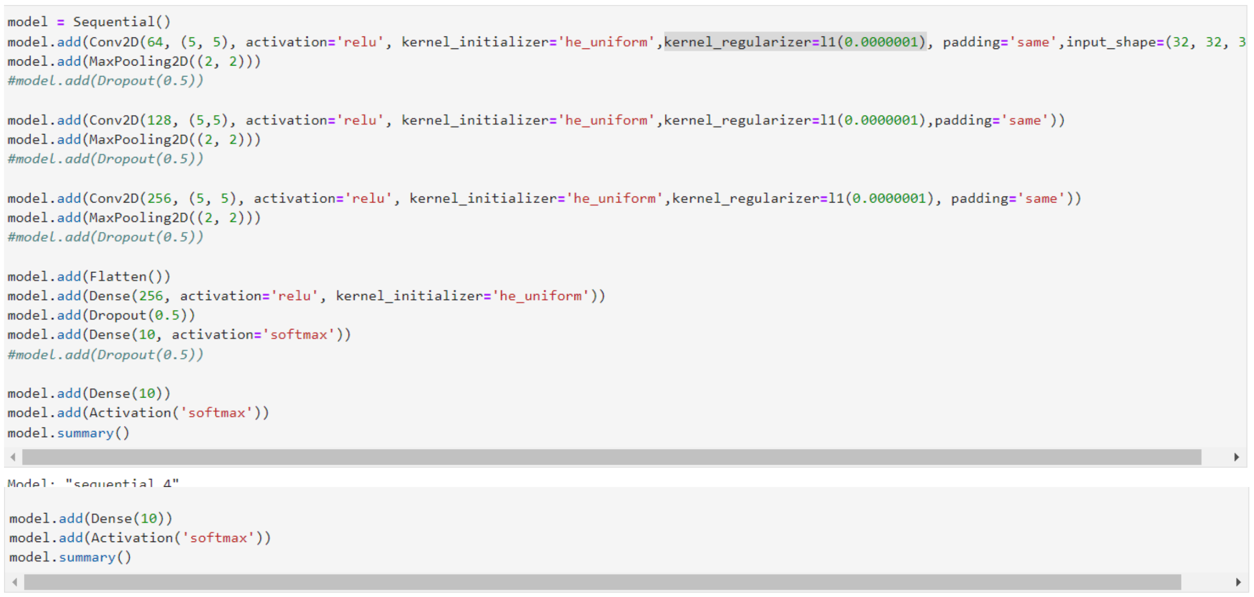
d. Try different regularizations apart from dropout:

kernel reguliser = l1(0.0000001)

```
model = Sequential()
model.add(Conv2D(64, (5, 5), activation='relu', kernel_initializer='he_uniform',kernel_regularizer=l1(0.0000001), padding='same',input_shape=(32, 32, 3
model.add(MaxPooling2D((2, 2)))
#model.add(Dropout(0.5))

model.add(Conv2D(128, (5,5), activation='relu', kernel_initializer='he_uniform',kernel_regularizer=l1(0.0000001),padding='same'))
model.add(MaxPooling2D((2, 2)))
#model.add(Dropout(0.5))

model.add(Conv2D(256, (5, 5), activation='relu', kernel_initializer='he_uniform',kernel_regularizer=l1(0.0000001), padding='same'))
model.add(MaxPooling2D((2, 2)))
#model.add(Dropout(0.5))

model.add(Flatten())
model.add(Dense(256, activation='relu', kernel_initializer='he_uniform'))
model.add(Dropout(0.5))
model.add(Dense(10, activation='softmax'))
#model.add(Dropout(0.5))

model.add(Dense(10))
model.add(Activation('softmax'))
model.summary()
```

```
Model: "sequential_4"
```

kernel reguliser= l2(0.0000001)



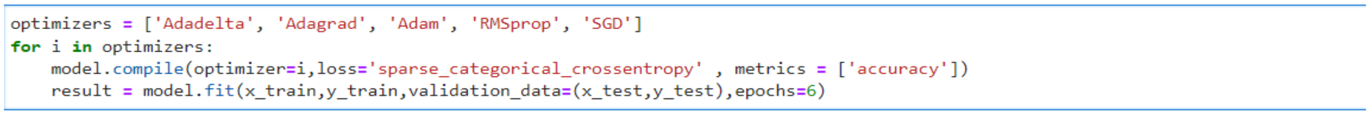
```
model = Sequential()
model.add(Conv2D(64, (5, 5), activation='relu', kernel_initializer='he_uniform',kernel_regularizer=l1(0.0000001), padding='same',input_shape=(32, 32, 3
model.add(MaxPooling2D((2, 2)))
#model.add(Dropout(0.5))

model.add(Conv2D(128, (5,5), activation='relu', kernel_initializer='he_uniform',kernel_regularizer=l1(0.0000001),padding='same'))
model.add(MaxPooling2D((2, 2)))
#model.add(Dropout(0.5))

model.add(Conv2D(256, (5, 5), activation='relu', kernel_initializer='he_uniform',kernel_regularizer=l1(0.0000001), padding='same'))
model.add(MaxPooling2D((2, 2)))
#model.add(Dropout(0.5))

model.add(Flatten())
model.add(Dense(256, activation='relu', kernel_initializer='he_uniform'))
model.add(Dropout(0.5))
model.add(Dense(10, activation='softmax'))
#model.add(Dropout(0.5))

model.add(Dense(10))
model.add(Activation('softmax'))
model.summary()
```

```
Model: "sequential_4"
```

```
model.add(Dense(10))
model.add(Activation('softmax'))
model.summary()
```

e. Try different optimization algorithms (such as Gradient Descent, Adam etc.)

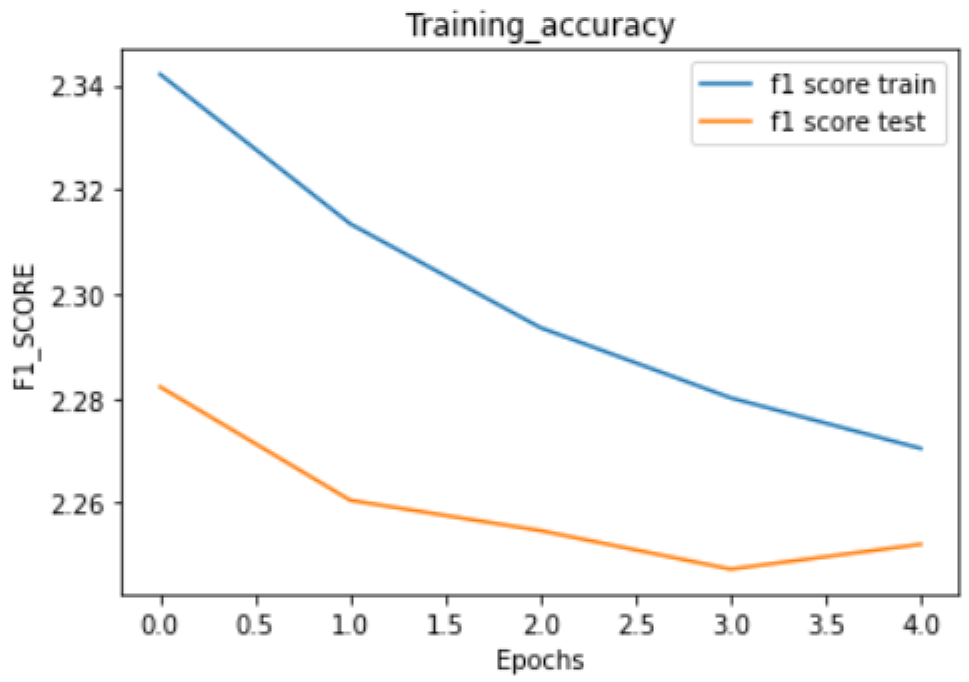
Adadelta, Adagrad, Adam, RMSprop, SGD etc are used here.



```
optimizers = ['Adadelta', 'Adagrad', 'Adam', 'RMSprop', 'SGD']
for i in optimizers:
    model.compile(optimizer=i,loss='sparse_categorical_crossentropy' , metrics = ['accuracy'])
    result = model.fit(x_train,y_train,validation_data=(x_test,y_test),epochs=6)
```

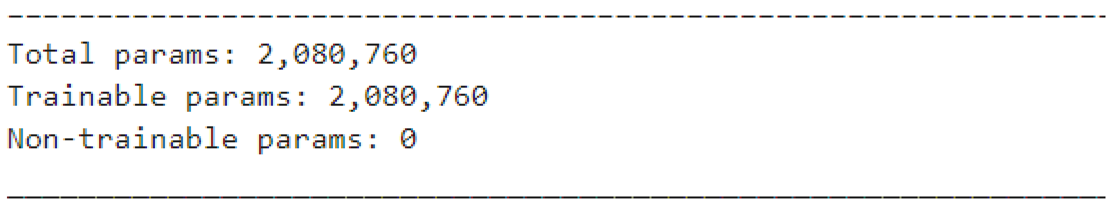
f. Create a graph of f1 score vs epochs for training and validation set.

```
loss = result.history['loss']
val_loss =result.history['val_loss']
precision = [1-x for x in myhistory['prec']]
recall = [1-x for x in myhistory['reca']]
f_one = [1-x for x in myhistory['f1']]


epochs = range(1, len(loss) + 1)
plt.plot(loss,label='f1 score train')
plt.plot(val_loss,label='f1 score test')
plt.title("Training_accuracy")
plt.ylabel('F1_SCORE')
plt.xlabel('Epochs')
plt.legend()
plt.show()
```



I have used only 5 epochs here as the running is very high due to some limitations.

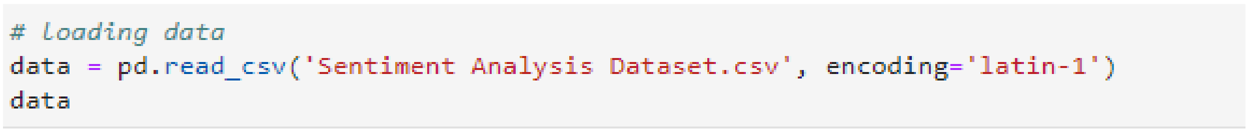
g. Calculate the number of trainable parameters in your final model.



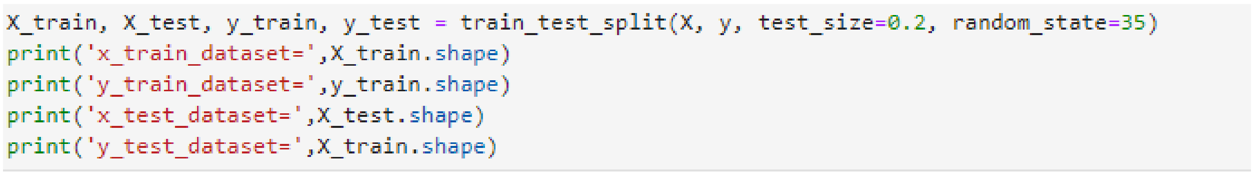
```
________________________________________________________________
Total params: 2,080,760
Trainable params: 2,080,760
Non-trainable params: 0
________________________________________________________________
```

Task 2 Natural Language Processing

1. Load the movie reviews sentiment analysis dataset and split into 80:20 ratio for training and test data



```python
# Loading data
data = pd.read_csv('Sentiment Analysis Dataset.csv', encoding='latin-1')
data
```



```python
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=35)
print('x_train_dataset=',X_train.shape)
print('y_train_dataset=',y_train.shape)
print('x_test_dataset=',X_test.shape)
print('y_test_dataset=',X_train.shape)
```

2. What is lexical vs semantic text analysis

Lexical analysis is based on smaller token but on the other side semantic analysis focuses on larger chunks**.** Semantic Analysis captures the meaning of the given text considering into account context, of sentences grammar roles and logical structuring. Its aim is to find the meaning from the given text.

In other words the purpose of semantic analysis is to draw exact meaning, or you can say dictionary meaning from the text. Lexical analysis is the process of trying to understand what words mean, intuit their context, and note the relationship of one word to others. It is often the entry point to many NLP data pipelines.

The first step of compilation, called lexical analysis, is to convert the input from a simple sequence of characters into a list of tokens of different kinds, such as numerical and string constants, variable identifiers, and programming language keywords.

3. Perform required preprocessing on the dataset (removing stop words, vectorization)

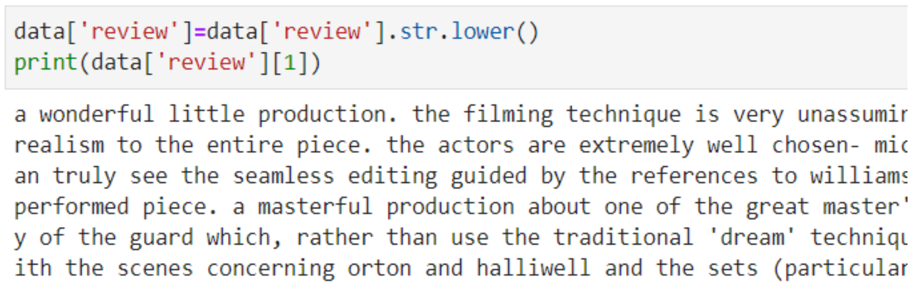
```
data['review']=data['review'].replace(r'<.*?>','',regex=True)
data['review']
```

```
0        One of the other reviewers has mentioned that ...
1        A wonderful little production. The filming tec...
2        I thought this was a wonderful way to spend ti...
3        Basically there's a family where a little boy ...
4        Petter Mattei's "Love in the Time of Money" is...
                                ...
49995    I thought this movie did a down right good job...
49996    Bad plot, bad dialogue, bad acting, idiotic di...
49997    I am a Catholic taught in parochial elementary...
49998    I'm going to have to disagree with the previou...
49999    No one expects the Star Trek movies to be high...
Name: review, Length: 50000, dtype: object
```

```
data['sentiment'].replace('negative',0,inplace=True)
data['sentiment'].replace('positive',1,inplace=True)
data
```

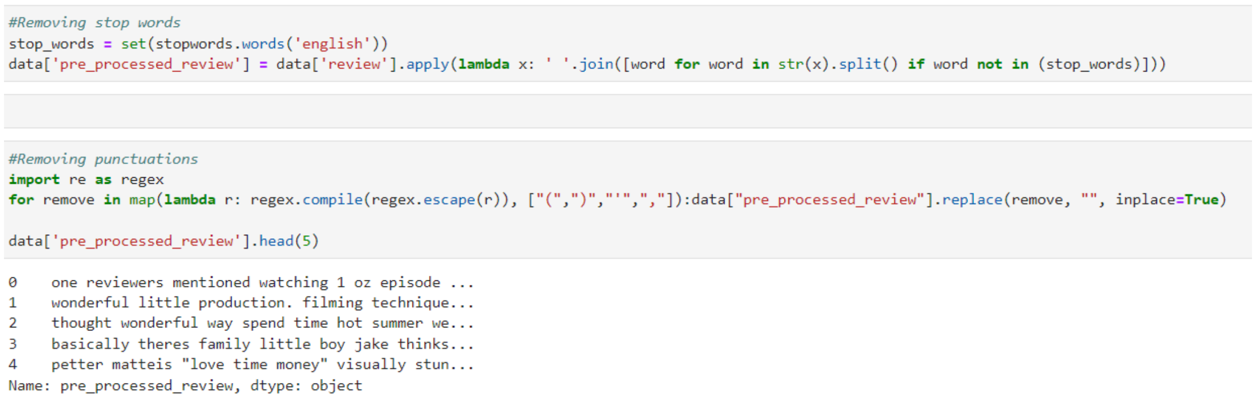
	review	sentiment
0	One of the other reviewers has mentioned that ...	1
1	A wonderful little production. The filming tec...	1
2	I thought this was a wonderful way to spend ti...	1
3	Basically there's a family where a little boy ...	0
4	Petter Mattei's "Love in the Time of Money" is...	1
...	...	...
49995	I thought this movie did a down right good job...	1
49996	Bad plot, bad dialogue, bad acting, idiotic di...	0
49997	I am a Catholic taught in parochial elementary...	0
49998	I'm going to have to disagree with the previou...	0
49999	No one expects the Star Trek movies to be high...	0

50000 rows × 2 columns



```python
data['review']=data['review'].str.lower()
print(data['review'][1])
```

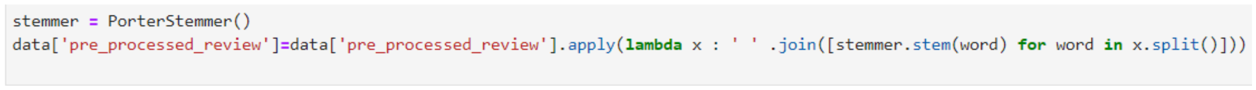
```
a wonderful little production. the filming technique is very unassumir
realism to the entire piece. the actors are extremely well chosen- mic
an truly see the seamless editing guided by the references to williams
performed piece. a masterful production about one of the great master'
y of the guard which, rather than use the traditional 'dream' techniqu
ith the scenes concerning orton and halliwell and the sets (particular
```



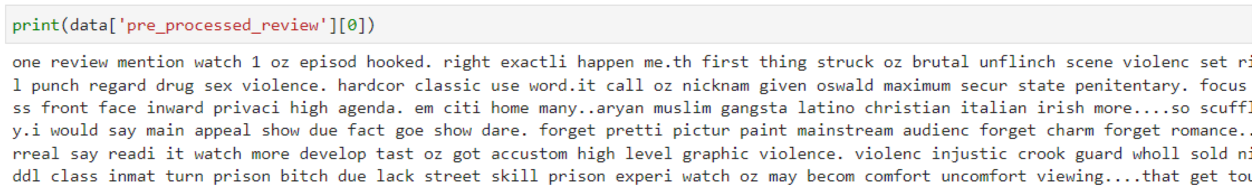
```python
#Removing stop words
stop_words = set(stopwords.words('english'))
data['pre_processed_review'] = data['review'].apply(lambda x: ' '.join([word for word in str(x).split() if word not in (stop_words)]))
```

```python
#Removing punctuations
import re as regex
for remove in map(lambda r: regex.compile(regex.escape(r)), ["(",")","'",","]):data["pre_processed_review"].replace(remove, "", inplace=True)

data['pre_processed_review'].head(5)
```

```
0    one reviewers mentioned watching 1 oz episode ...
1    wonderful little production. filming technique...
2    thought wonderful way spend time hot summer we...
3    basically theres family little boy jake thinks...
4    petter matteis "love time money" visually stun...
Name: pre_processed_review, dtype: object
```



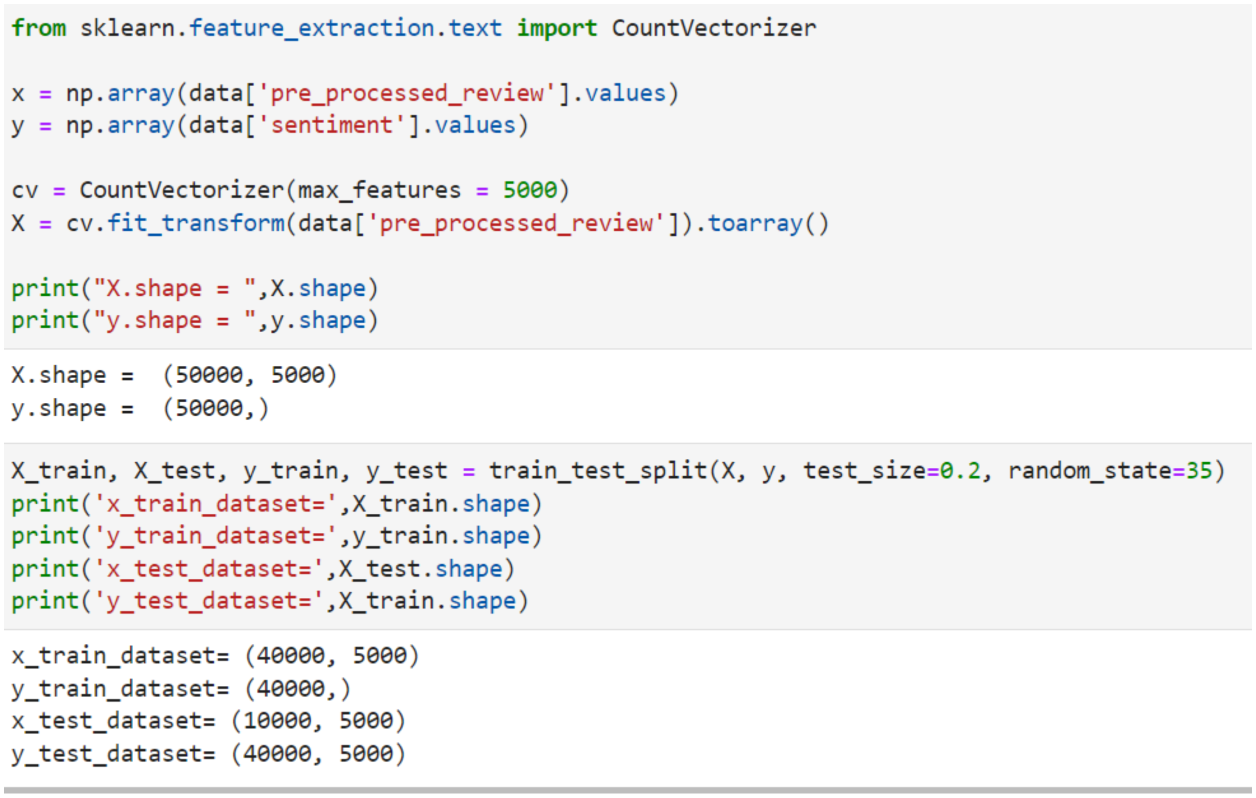
```python
stemmer = PorterStemmer()
data['pre_processed_review']=data['pre_processed_review'].apply(lambda x : ' ' .join([stemmer.stem(word) for word in x.split()]))
```



```python
print(data['pre_processed_review'][0])
```

```
one review mention watch 1 oz episod hooked. right exactli happen me.th first thing struck oz brutal unflinch scene violenc set ri
l punch regard drug sex violence. hardcor classic use word.it call oz nicknam given oswald maximum secur state penitentary. focus
ss front face inward privaci high agenda. em citi home many..aryan muslim gangsta latino christian italian irish more....so scuffl
y.i would say main appeal show due fact goe show dare. forget pretti pictur paint mainstream audienc forget charm forget romance..
rreal say readi it watch more develop tast oz got accustom high level graphic violence. violenc injustic crook guard wholl sold ni
ddl class inmat turn prison bitch due lack street skill prison experi watch oz may becom comfort uncomfort viewing....that get tou
```

4.Build a model to classify the rows

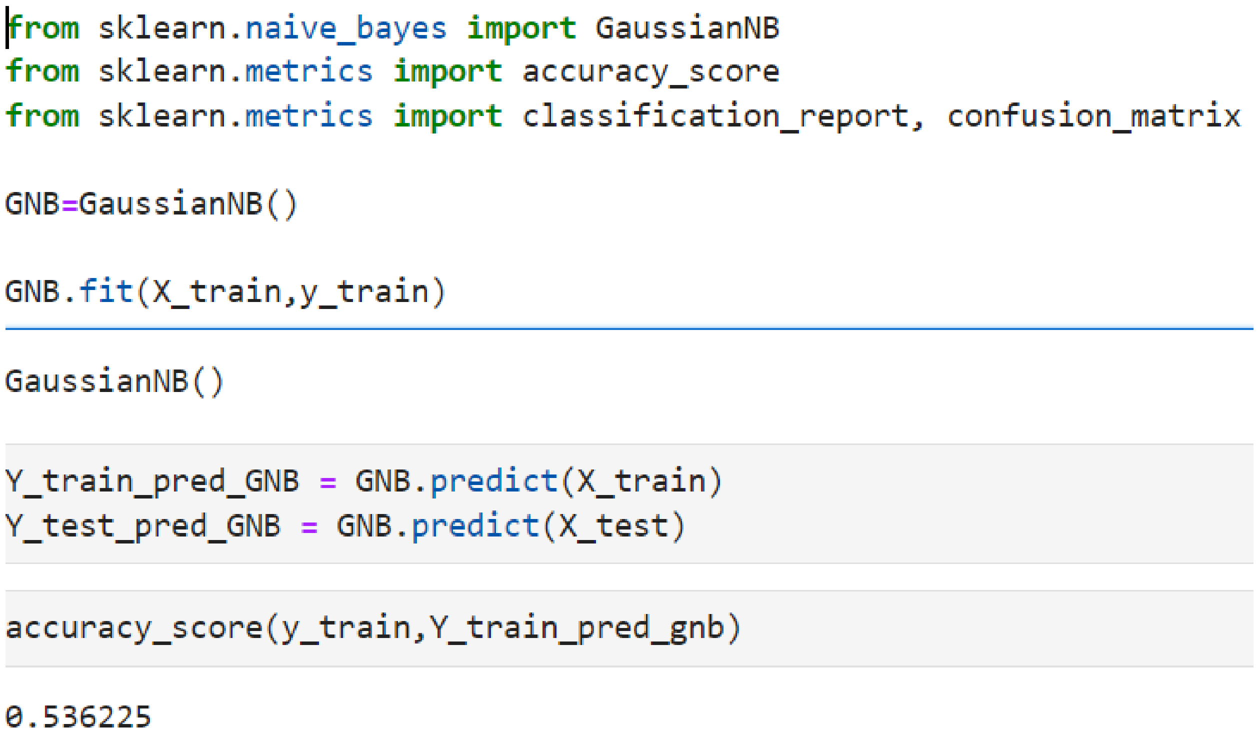


```
from sklearn.feature_extraction.text import CountVectorizer

x = np.array(data['pre_processed_review'].values)
y = np.array(data['sentiment'].values)

cv = CountVectorizer(max_features = 5000)
X = cv.fit_transform(data['pre_processed_review']).toarray()

print("X.shape = ",X.shape)
print("y.shape = ",y.shape)
```

```
X.shape =  (50000, 5000)
y.shape =  (50000,)
```

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=35)
print('x_train_dataset=',X_train.shape)
print('y_train_dataset=',y_train.shape)
print('x_test_dataset=',X_test.shape)
print('y_test_dataset=',X_train.shape)
```

```
x_train_dataset= (40000, 5000)
y_train_dataset= (40000,)
x_test_dataset= (10000, 5000)
y_test_dataset= (40000, 5000)
```



```
from sklearn.naive_bayes import GaussianNB
from sklearn.metrics import accuracy_score
from sklearn.metrics import classification_report, confusion_matrix

GNB=GaussianNB()

GNB.fit(X_train,y_train)
```

```
GaussianNB()
```

```
Y_train_pred_GNB = GNB.predict(X_train)
Y_test_pred_GNB = GNB.predict(X_test)
```

```
accuracy_score(y_train,Y_train_pred_gnb)
```

```
0.536225
```

```
from sklearn.linear_model import LogisticRegression
model = LogisticRegression(solver='liblinear', random_state=0)
```

```
model.fit(X_train,y_train)
```

```
LogisticRegression(random_state=0, solver='liblinear')
```

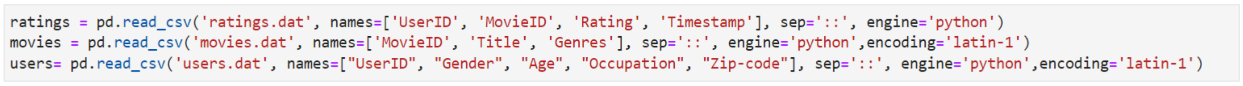
```
Y_train_pred = model.predict(X_train)
Y_test_pred = model.predict(X_test)
```

```
print(classification_report(y_train,Y_train_pred))
```

```
              precision    recall  f1-score   support

           0       1.00      0.10      0.18     19950
           1       0.53      1.00      0.69     20050

    accuracy                           0.55     40000
   macro avg       0.76      0.55      0.44     40000
weighted avg       0.76      0.55      0.44     40000
```

Task 3 Recommender System

1. Download m1.zip file from the link
2. Load the movies and ratings data



```
ratings = pd.read_csv('ratings.dat', names=['UserID', 'MovieID', 'Rating', 'Timestamp'], sep='::', engine='python')
movies = pd.read_csv('movies.dat', names=['MovieID', 'Title', 'Genres'], sep='::', engine='python',encoding='latin-1')
users= pd.read_csv('users.dat', names=["UserID", "Gender", "Age", "Occupation", "Zip-code"], sep='::', engine='python',encoding='latin-1')
```

3. What do you mean by Singular Value Decomposition

In linear algebra, the singular value decomposition (SVD) is a factorization of a real or complex matrix. It generalizes the eigen decomposition of a square normal matrix with an orthonormal eigen basis to any matrix. It is related to the polar decomposition.

- A = U W V^T
- U: m x n matrix of the orthonormal eigenvectors of A A^T .
- W: n x n diagonal matrix of the singular values which are the square roots of the eigenvalues of A^T A
- V^T: transpose of a n x n matrix containing the orthonormal eigenvectors of A^T A.

It is used for : Calculation of Pseudo-inverse , Solving a set of Homogeneous Linear Equations (Mx =b), Curve Fitting Problem

4. What do you mean by Principal Component Analysis ?

- Principal Component Analysis (PCA) is a statistical technique used for data reduction without losing its properties.
- Basically, it describes the composition of variances and covariances through several linear combinations of the primary variables, without missing an important part of the original information.
- In another term, it is about obtaining a unique set of orthogonal axes where the data has the largest variance. Its main aim is to overcome the dimensionality of the problem.
- The reduction of dimensionality should be such that when dropping higher dimensions, the loss of data is minimum.

5. Explain content-based vs collaborative recommendation?

Content-based filtering system: Content-Based recommender system tries to guess the features or behavior of a user given the item's features, he/she reacts positively to.

Collaborative filtering System: Collaborative does not need the features of the items to be given. Every user and item is described by a feature vector or embedding.

6. Create m x u matrix with movies as row and users as column. Normalize the matrix.

```
new_df['Gender']=new_df['Gender'].replace('M',1,inplace=True)
new_df['Gender']=new_df['Gender'].replace('F',0,inplace=True)
```

```
ratings_as_matrix = np.ndarray(
    shape=(np.max(df.MovieID.values), np.max(df.UserID.values)),
    dtype=np.uint8)

#print(ratings_matrix)
ratings_as_matrix
```

```
array([[0, 0, 0, ..., 0, 0, 0],
       [0, 0, 0, ..., 0, 0, 0],
       [0, 0, 0, ..., 0, 0, 0],
       ...,
       [0, 0, 0, ..., 0, 0, 0],
       [0, 0, 0, ..., 0, 0, 0],
       [0, 0, 0, ..., 0, 0, 0]], dtype=uint8)
```

```
ratings_as_matrix[df.MovieID.values-1, df.UserID.values-1] = df.Rating.values
print(ratings_as_matrix)
```

```
[[5 0 0 ... 0 0 3]
 [0 0 0 ... 0 0 0]
 [0 0 0 ... 0 0 0]
 ...
 [0 0 0 ... 0 0 0]
 [0 0 0 ... 0 0 0]
 [0 0 0 ... 0 0 0]]
```

```
normalised_matrix = ratings_as_matrix - np.asarray([(np.mean(ratings_as_matrix, 1))]).T
print(normalised_matrix)
```

```
[[ 3.57400662 -1.42599338 -1.42599338 ... -1.42599338 -1.42599338
   1.57400662]
 [-0.37152318 -0.37152318 -0.37152318 ... -0.37152318 -0.37152318
  -0.37152318]
 [-0.23874172 -0.23874172 -0.23874172 ... -0.23874172 -0.23874172
  -0.23874172]
 ...
 [-0.03278146 -0.03278146 -0.03278146 ... -0.03278146 -0.03278146
  -0.03278146]
 [-0.02582781 -0.02582781 -0.02582781 ... -0.02582781 -0.02582781
  -0.02582781]
 [-0.24288079 -0.24288079 -0.24288079 ... -0.24288079 -0.24288079
  -0.24288079]]
```

7. Perform SVD to get U, S and V

```
A = normalised_matrix.T / np.sqrt(ratings_as_matrix.shape[0] - 1)
U, S, V = np.linalg.svd(A)
print('U=\n',U)
print('S=\n',S)
print('V=\n',V)
```

```
U=
 [[ 7.13393053e-03  1.64099327e-03  2.14622406e-03 ...  4.52266370e-03
   6.00783778e-03  8.85674875e-03]
 [ 6.40383513e-04 -2.70126226e-03 -2.00478360e-04 ...  2.54577717e-03
   5.40385541e-04 -1.55791869e-02]
 [ 6.72473390e-03 -3.34737240e-03 -3.95617989e-03 ...  1.71114905e-03
   5.91124462e-03  8.44640940e-03]
 ...
 [ 1.13666709e-02  1.80896437e-03 -5.62198713e-04 ...  7.73909103e-01
  -1.48780057e-02 -2.19959777e-03]
 [ 3.49381899e-03  1.87620989e-02 -1.08962191e-02 ... -9.78238175e-03
   3.32703088e-01 -3.14995403e-03]
 [-1.32856412e-02  4.08015550e-02 -3.63311909e-03 ... -1.09584563e-03
   5.44124149e-03  8.83627250e-02]]
S=
 [2.06617808e+01 1.06804930e+01 9.14055972e+00 ... 1.71271486e-15
 1.71271486e-15 3.80173012e-16]
V=
 [[-5.72245537e-02 -2.69349804e-02 -1.44300959e-02 ... -3.50288960e-03
  -8.19971409e-04 -1.25419482e-02]
 [-2.09461200e-02 -2.97843268e-02 -1.66998921e-02 ...  1.87545849e-03
   2.26486119e-03  5.02192377e-03]
 [ 2.86333511e-02 -8.62001932e-03  1.36557569e-02 ...  1.91670702e-03
   3.47436586e-03  2.20808096e-02]
 ...
 [ 0.00000000e+00  1.41748444e-31 -1.46145455e-31 ...  5.94341981e-31
   3.22978452e-31  7.83853487e-32]
 [ 0.00000000e+00 -2.89130233e-32  2.93168384e-32 ...  3.84223024e-32
   1.01111322e-33  2.76912615e-32]
 [ 0.00000000e+00  7.58941521e-17  7.57448665e-17 ... -1.73472348e-18
  -6.76542156e-17 -3.25260652e-19]]
```

8. Select top 50 components from V.T

```
Casper (1995)
Borrowers, The (1997)
```

```
print(VT)
```

```
[[-0.05722455 -0.02094612  0.02863335 ... -0.02620973  0.06051542
   0.01070457]
 [-0.02693498 -0.02978433 -0.00862002 ...  0.01891212  0.04177941
  -0.00106454]
 [-0.0144301  -0.01669989  0.01365576 ... -0.00767895  0.01198254
  -0.03774839]
 ...
 [-0.00350289  0.00187546  0.00191671 ...  0.00615582  0.01056114
   0.00374833]
 [-0.00081997  0.00226486  0.00347437 ...  0.00353482  0.0099851
   0.00452786]
 [-0.01254195  0.00502192  0.02208081 ...  0.00898529  0.02015898
   0.02577483]]
```

9. Calculate the covariance matrix for the entire dataset (from step 6)

```
normalised_matrix = ratings_as_matrix - np.asarray([(np.mean(ratings_as_matrix, 1))]).T
print(normalised_matrix)
```

```
[[ 3.57400662 -1.42599338 -1.42599338 ... -1.42599338 -1.42599338
   1.57400662]
 [-0.37152318 -0.37152318 -0.37152318 ... -0.37152318 -0.37152318
  -0.37152318]
 [-0.23874172 -0.23874172 -0.23874172 ... -0.23874172 -0.23874172
  -0.23874172]
 ...
 [-0.03278146 -0.03278146 -0.03278146 ... -0.03278146 -0.03278146
  -0.03278146]
 [-0.02582781 -0.02582781 -0.02582781 ... -0.02582781 -0.02582781
  -0.02582781]
 [-0.24288079 -0.24288079 -0.24288079 ... -0.24288079 -0.24288079
```

```
covariance_matrix= np.cov(normalised_matrix)
print(covariance_matrix)
```

```
[[ 4.13030618e+00  5.75769310e-01  2.58605324e-01 ...  3.68692352e-02
  -1.89683174e-03  1.01352495e-01]
 [ 5.75769310e-01  1.16348766e+00  1.58844772e-01 ...  2.35864233e-02
   1.72608728e-04  4.73547767e-02]
 [ 2.58605324e-01  1.58844772e-01  7.54054386e-01 ...  1.23744228e-02
  -3.18657205e-03  2.89395420e-02]
 ...
 [ 3.68692352e-02  2.35864233e-02  1.23744228e-02 ...  1.28748016e-01
   2.31637842e-02  7.54942213e-02]
 [-1.89683174e-03  1.72608728e-04 -3.18657205e-03 ...  2.31637842e-02
   1.07297708e-01  5.63190257e-02]
 [ 1.01352495e-01  4.73547767e-02  2.89395420e-02 ...  7.54942213e-02
   5.63190257e-02  9.15498240e-01]]
```

10. Get the eigen vectors from the covariance matrix

```
evals,evecs=np.linalg.eig(covariance_matrix)
print("EIGEN Values:",evals)
print("EIGEN Vectors:",evecs)
```

```
EIGEN Values: [279.30422154+0.j  74.63191754+0.j  54.66225966+0.j ...   0.         +0.j
   0.         +0.j   0.         +0.j]
EIGEN Vectors: [[ 0.05722455+0.j  0.02094612+0.j -0.02863335+0.j ...  0.         +0.j
   0.         +0.j  0.         +0.j]
 [ 0.02693498+0.j  0.02978433+0.j  0.00862002+0.j ...  0.         +0.j
   0.         +0.j  0.         +0.j]
 [ 0.0144301 +0.j  0.01669989+0.j -0.01365576+0.j ...  0.         +0.j
   0.         +0.j  0.         +0.j]
 ...
 [ 0.00350289+0.j -0.00187546+0.j -0.00191671+0.j ...  0.         +0.j
   0.         +0.j  0.         +0.j]
 [ 0.00081997+0.j -0.00226486+0.j -0.00347437+0.j ...  0.         +0.j
   0.         +0.j  0.         +0.j]
 [ 0.01254195+0.j -0.00502192+0.j -0.02208081+0.j ...  0.         +0.j
   0.         +0.j  0.         +0.j]]
```

11. Get the top 50 eigen vectors using eigen values

```
print(top50_evalues)
```

```
[[ 0.05722455+0.j  0.02094612+0.j -0.02863335+0.j ...  0.01362589+0.j
  -0.00193866+0.j -0.05997806+0.j]
 [ 0.02693498+0.j  0.02978433+0.j  0.00862002+0.j ...  0.00439231+0.j
   0.01130363+0.j  0.02143723+0.j]
 [ 0.0144301 +0.j  0.01669989+0.j -0.01365576+0.j ...  0.00700238+0.j
  -0.02176034+0.j -0.01457793+0.j]
 ...
 [ 0.00350289+0.j -0.00187546+0.j -0.00191671+0.j ...  0.00069592+0.j
  -0.00668185+0.j  0.006408   +0.j]
 [ 0.00081997+0.j -0.00226486+0.j -0.00347437+0.j ... -0.00036645+0.j
  -0.00401153+0.j  0.00217063+0.j]
 [ 0.01254195+0.j -0.00502192+0.j -0.02208081+0.j ...  0.02609    +0.j
  -0.02060116+0.j  0.01451355+0.j]]
```

12. Using cosine similarity find 10 closest movies using the 50 components from SVD (step 8)

```
def top_cosine_similarity(data, movie_id, top_n):
    index = movie_id - 1
    movie_row = data[index, :]
    magnitude = np.sqrt(np.einsum('ij, ij -> i', data, data))
    similarity = np.dot(movie_row, data.T) / (magnitude[index] * magnitude)
    sort_indexes = np.argsort(-similarity)
    return sort_indexes[:top_n]


def similar_movies(movie_data, movie_id, top_indies):
    print('Recommendations for Movie {0}: \n'.format(
    movie_data[movie_data.MovieID == movie_id].Title.values[0]))
    for id in top_indies + 1:
        print(movie_data[movie_data.MovieID == id].Title.values[0])

k = 50
movie_id = 6
top_n = 10

VT = V.T[:, :k]
movies_indies = top_cosine_similarity(VT, movie_id, top_n)
similar_movies(movies, movie_id, movies_indexes)
```

```
Recommendations for Movie Heat (1995):

Heat (1995)
Ronin (1998)
True Romance (1993)
Out of Sight (1998)
U Turn (1997)
King of New York (1990)
In the Line of Fire (1993)
Professional, The (a.k.a. Leon: The Professional) (1994)
Wild Things (1998)
Menace II Society (1993)
```

13. Using cosine similarity find 10 closest movies using the 50 components from PCA (step 11)

```
k = 50
movie_id = 6
top_n = 10

top50_evalues = evecs[:, :k]
top_indixes = top_cosine_similarity(sliced, movie_id, top_n)
print_similar_movies(movies, movie_id, top_indixes )
```

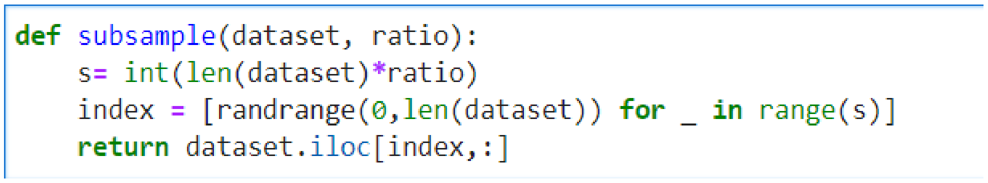
```
Recommendations for Movie Heat (1995):

Heat (1995)
Ronin (1998)
True Romance (1993)
Out of Sight (1998)
Cop Land (1997)
Menace II Society (1993)
Desperado (1995)
Donnie Brasco (1997)
Jackie Brown (1997)
In the Line of Fire (1993)
```

14. Compare the results of the above two methods

- I find that singular value decomposition seems faster than the covariance matrix method
- The eigenvalues are not sorted in ascending order and it needs to be sorted while the singular values from SVD are already sorted
- In results also there is a slight variation in order of the movies which is suggested

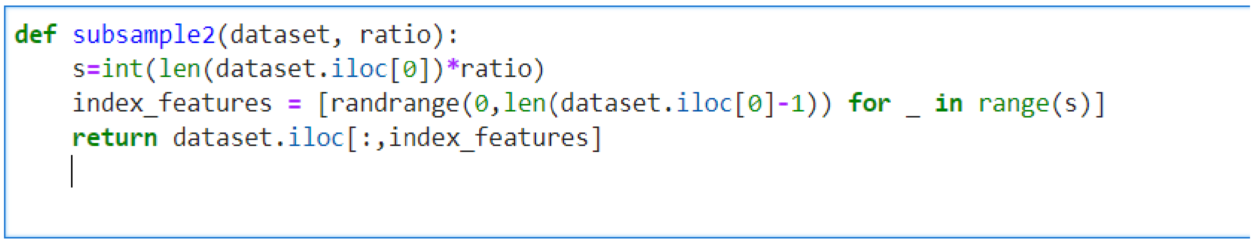
Task 4 Random Forest – Self implementation:

1. **Complete the following function that creates subsample of a dataset with replacement**



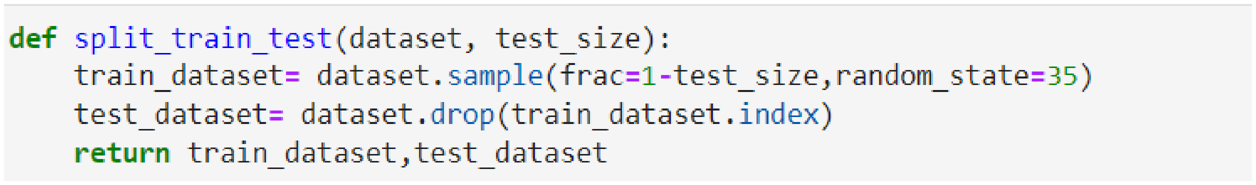
```
def subsample(dataset, ratio):
    s= int(len(dataset)*ratio)
    index = [randrange(0,len(dataset)) for _ in range(s)]
    return dataset.iloc[index,:]
```

2. Complete the following function that creates subsample of the dataset with feature size reduced as per the given ratio



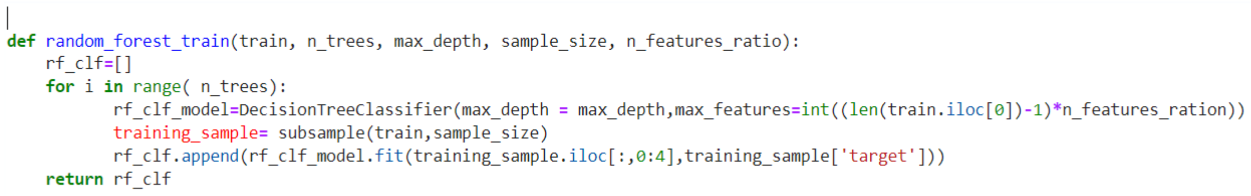
```
def subsample2(dataset, ratio):
    s=int(len(dataset.iloc[0])*ratio)
    index_features = [randrange(0,len(dataset.iloc[0]-1)) for _ in range(s)]
    return dataset.iloc[:,index_features]
```

3. Perform train test split without using sklearn (complete the following function)



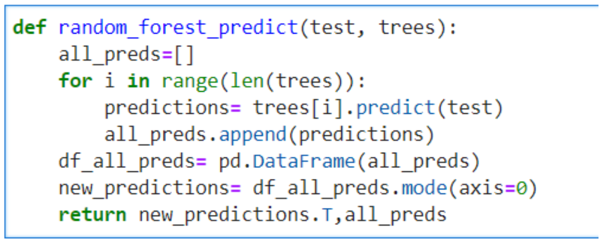
```
def split_train_test(dataset, test_size):
    train_dataset= dataset.sample(frac=1-test_size,random_state=35)
    test_dataset= dataset.drop(train_dataset.index)
    return train_dataset,test_dataset
```

4. Perform training using random forest algorithm by completing the function below



```
def random_forest_train(train, n_trees, max_depth, sample_size, n_features_ratio):
    rf_clf=[]
    for i in range( n_trees):
          rf_clf_model=DecisionTreeClassifier(max_depth = max_depth,max_features=int((len(train.iloc[0])-1)*n_features_ration))
          training_sample= subsample(train,sample_size)
          rf_clf.append(rf_clf_model.fit(training_sample.iloc[:,0:4],training_sample['target']))
    return rf_clf
```

5. Complete the following prediction function of random forest



```
def random_forest_predict(test, trees):
    all_preds=[]
    for i in range(len(trees)):
        predictions= trees[i].predict(test)
        all_preds.append(predictions)
    df_all_preds= pd.DataFrame(all_preds)
    new_predictions= df_all_preds.mode(axis=0)
    return new_predictions.T,all_preds
```

Note about my learnings:

- The task given for this lab is very useful covers the broad segment of topics which are useful in order to learn ML or DL course.
- For task1 the training time was taking too much time to hardware constraints but I learned how to figure alternate options if there are those issues.
- For task2, the preprocessing part is main part which taught me that for any ML algorithm to run if the dataset is cleaned and in proper format than it is much easier to run our model.