Lab Assignment 2

Member 1

Name : Sireesha Keesara

Class Id : 14

Member 2

Name : Kavin Kumar Arumugam

Class Id : 3

Objectives

- To implement the Linear Regression.Show the graph in TensorBoard and plot the loss.Report the performance by changing hyper parameters 1. Learning Rate, 2. Batch size, 3. Optimizer and 4. activation function.
- To implement the Logistic Regression.Show the graph in TensorBoard and plot the loss. use score=model.evaluate(x_text,y_test)and then print('test accuracy', score[1])to print the accuracy.Change the hyper parameters to see how the performance changes.
- To implement the text classification with CNN model on Spam text classification data set or Reuters data set.
- To implement the text classification with LSTM model on Spam text classification data set or Reuters data set.
- Compare the results of CNN and LSTM models, for the text classification and describe, which model is best for the text classification based on results.
- To implement the image classification with CNN model.

Linear Regression

- Import Sequential model, KerasRegressor from keras library for creating the model and evaluating the model.
- Import cross_val_score, KFold from sklearn.
- Read the CSV file.
- split the data into input (X) and output (Y) attributes.
- Create a model.
- No activation function is used for the output layer because it is a linear regression problem and we are interested in predicting numerical values directly without transform.
- Give the mean_squared_error as the loss function.
- Create an instance of KerasRegressor and pass it along to the model.fit()
- Evaluate the model.
- Plot the loss and graph in TensorBoard.
- Change the hyper parameters and see how performance changes.

Source code Using KerasRegressor

```
df=pd.read_csv("./weatherHistory.csv")
print(df.info)
X=df.iloc[:,5].values
Y=df.iloc[:,3].values
x=X.reshape(-1,1)
y=Y.reshape(-1,1)
def base_model():
# create model
 model = Sequential()
 model.add(Dense(11, input_dim=1, kernel_initializer='normal', activation='relu'))
 model.add(Dense(1, kernel_initializer='normal'))
# Compile model
 model.compile(loss='mean_squared_error', optimizer='adam')

 return model
numpy.random.seed(10)
estimators = []
estimators.append(('standardize', StandardScaler()))
estimators.append(('mlp', KerasRegressor(build_fn=base_model, epochs=2,batch_size=5, verbose=0)))
pipeline = Pipeline(estimators)
kfold = KFold(n_splits=10, random_state=10)
results = cross_val_score(pipeline, x, y, cv=kfold)
print("Standardized: %.2f (%.2f) MSE" % (results.mean(), results.std()))
```

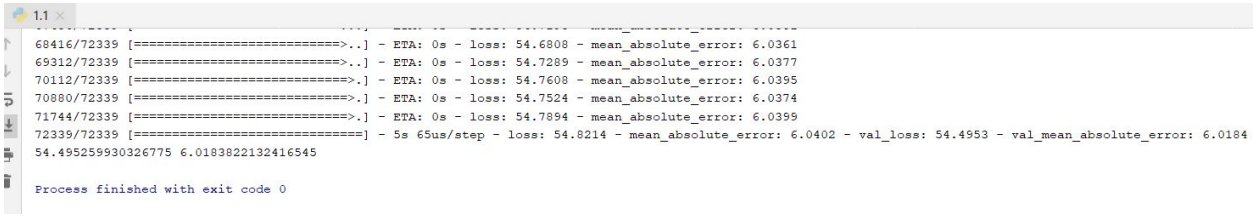
Output

```
W0722 20:47:45.741377  8212 deprecation_wrapper.py:119] From D:\lib\site-packages\keras\backend\ten

W0722 20:47:45.775978  8212 deprecation_wrapper.py:119] From D:\lib\site-packages\keras\optimizers.

W0722 20:47:45.923421  8212 deprecation_wrapper.py:119] From D:\lib\site-packages\keras\backend\ten

W0722 20:47:45.993701  8212 deprecation_wrapper.py:119] From D:\lib\site-packages\keras\backend\ten

2019-07-22 20:47:46.060207: I tensorflow/core/platform/cpu_feature_guard.cc:142] Your CPU supports
Standardized: -54.84 (10.58) MSE

Process finished with exit code 0
```

Source code without KerasRegressor

```
learning_rate = 0.3
epochs = 5
b_size = 32
decay_rate = learning_rate / epochs

adam= Adam(lr=learning_rate, decay=decay_rate)
#sgd = SGD(lr=learning_rate, decay=decay_rate)

# Define the model
model = Sequential()
model.add(Dense(11, input_dim=1, activation='relu'))
model.add(Dense(7, activation='relu'))
model.add(Dense(5, activation='relu'))
model.add(Dense(1))

model.compile(optimizer="adam", loss='mean_squared_error', metrics=[metrics.mae])
tb = TensorBoard(log_dir='./Graph', histogram_freq=0, write_graph=True, write_images=True)
hist = model.fit(X_train, Y_train, validation_data=(X_test, Y_test), epochs=epochs, batch_size=b_size,
                 callbacks=[tb])

# Final evaluation of the model
mae, loss = model.evaluate(X_test, Y_test, verbose=0)
print(mae, loss)
```

Output



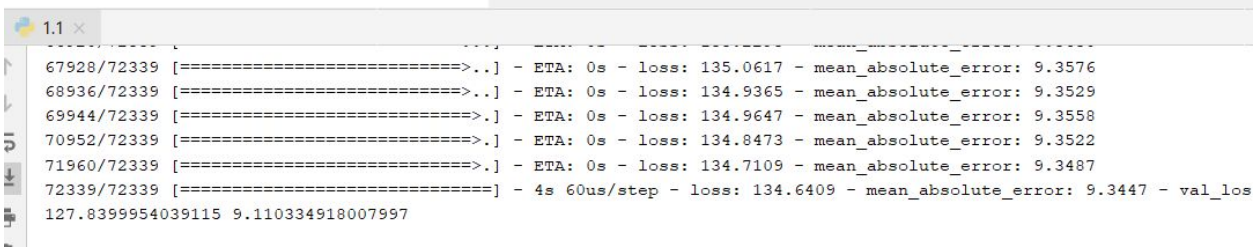
Changing hyper parameter learning rate from 0.3 to 0.7

```
70560/72339 [==============================>.] - ETA: 0s - loss: 54.4834 - mean_abs
71136/72339 [==============================>.] - ETA: 0s - loss: 54.4607 - mean_abs
71744/72339 [==============================>.] - ETA: 0s - loss: 54.4685 - mean_abs
72288/72339 [==============================>.] - ETA: 0s - loss: 54.4239 - mean_abs
72339/72339 [==============================] - 7s 103us/step - loss: 54.4254 - me
54.0583828181545 6.009937176748306
```

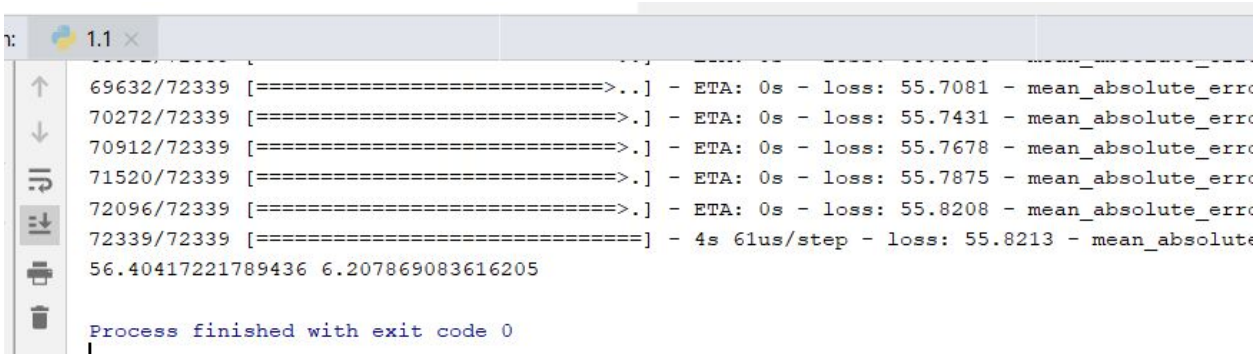
Changing hyper parameter batch size from 32 to 56

```
model = Sequential()
model.add(Dense(11, input_dim=1, activation='tanh'))
model.add(Dense(7, activation='tanh'))
model.add(Dense(5, activation='relu'))
```

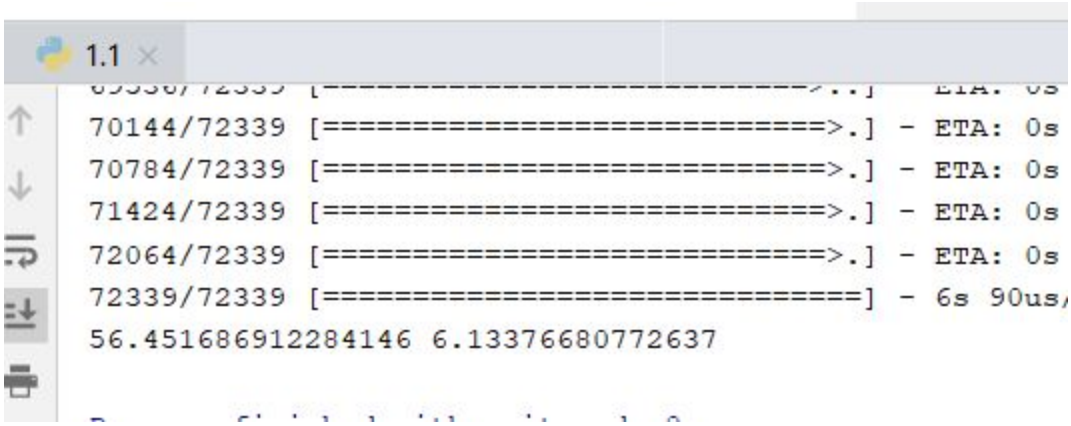
Output



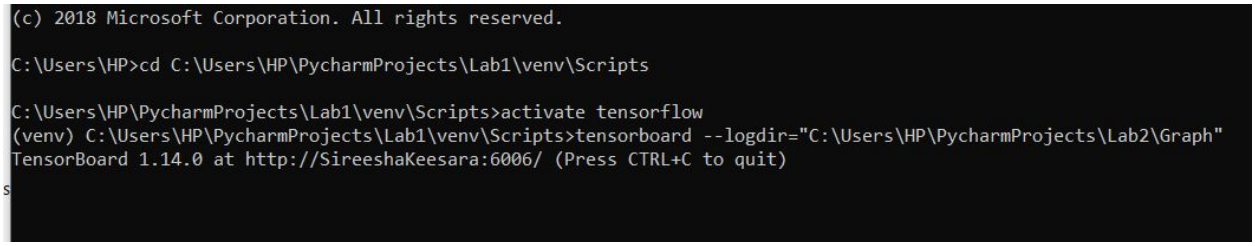
Changing hyper parameter optimizer to sgd.



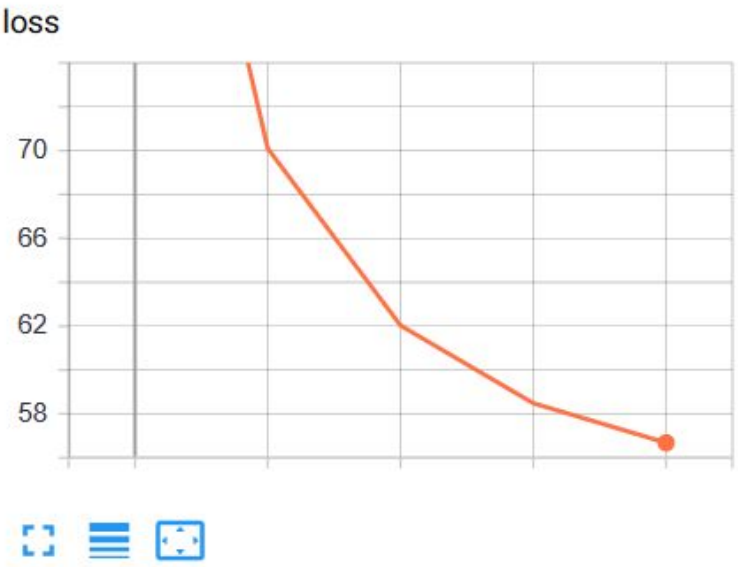
Changing hyper parameter activation function to tanh.



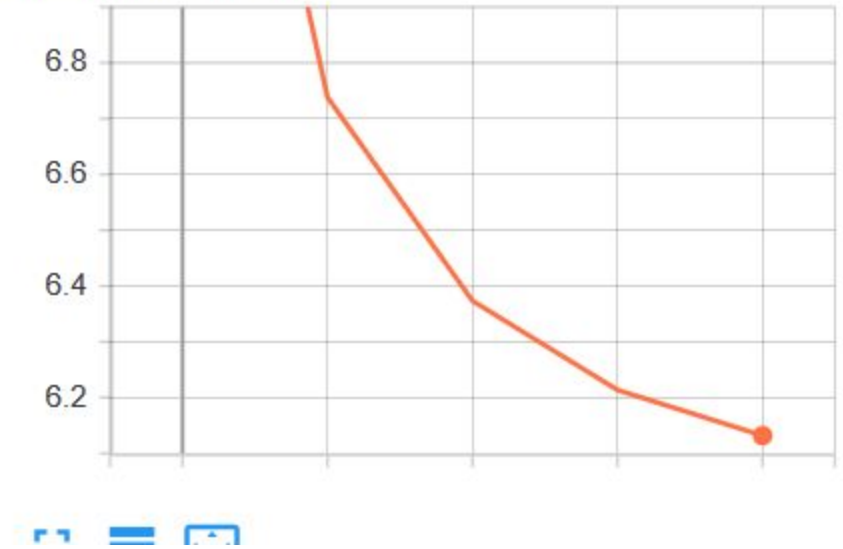
TensorBoard Visualizations



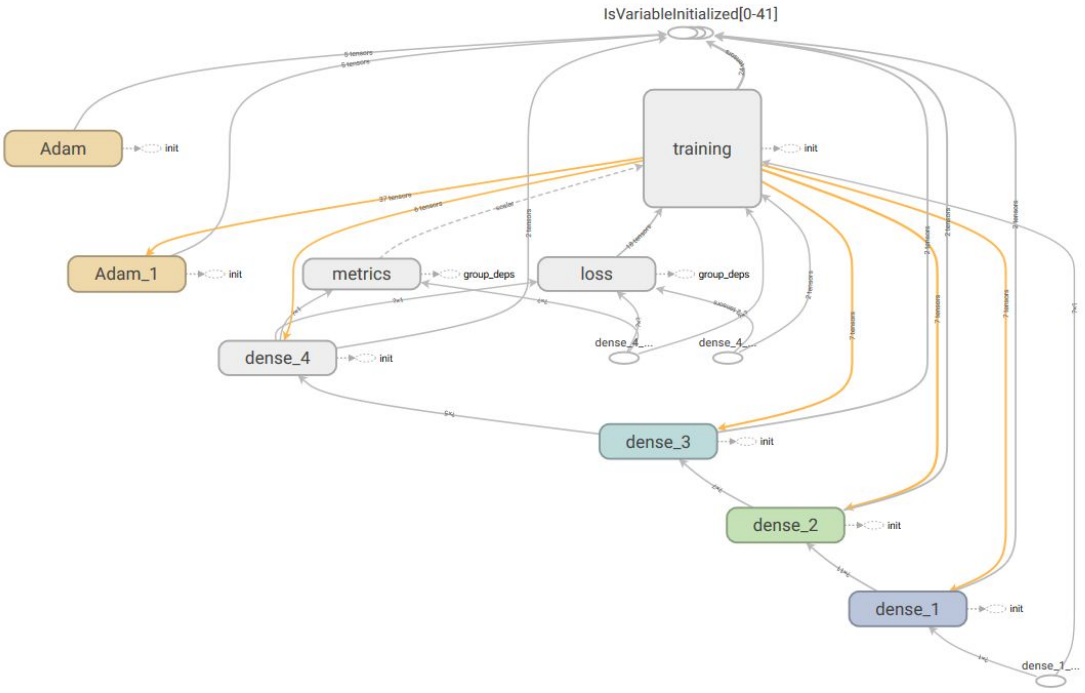
loss



mean_absolute_error



Main Graph



Logistic Regression

- Read the CSV file.
- split the data into input (X) and output (Y) attributes.
- Further divide the data into test and train.
- Create a model.
- Output layer activation function sigmoid suits best for Logistic regression.
- Give the binary_crossentropy as the loss function.
- Run the model on train data.
- Evaluate the model on test data.
- Plot the loss and graph in TensorBoard.
- Change the hyper parameters and see how performance changes.

Source code

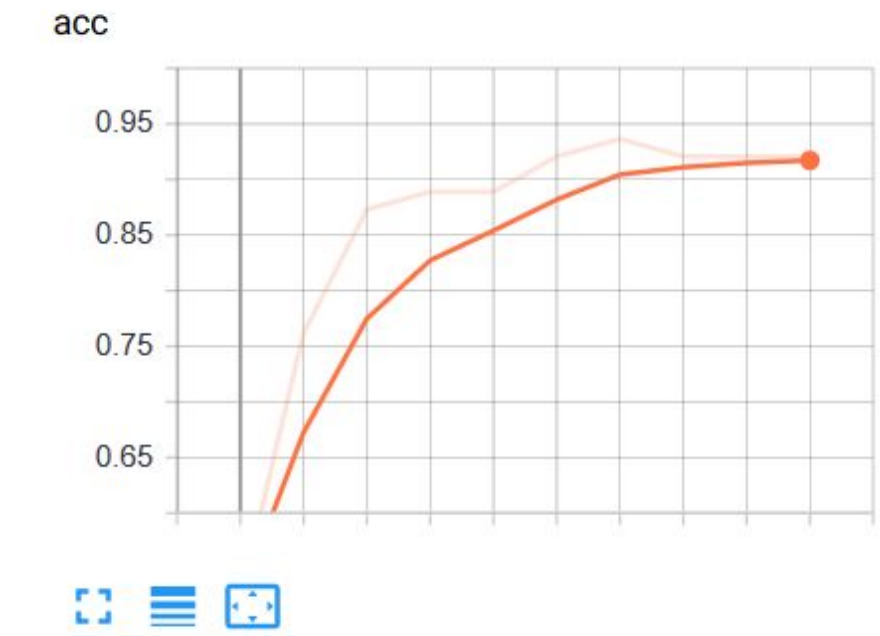
```
df=pd.read_csv('./candy-data.csv')
df.info()
df.drop("competitorname", inplace = True, axis=1)
y = df.chocolate.values
x_data= df.drop(["chocolate"], axis = 1)
x = (x_data-np.min(x_data))/(np.max(x_data)-np.min(x_data))
x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=.25, random_state=45)
learning_rate=0.1
epochs=10
b_size=64
decay_rate= learning_rate / epochs
adam= Adam(lr=learning_rate, decay=decay_rate)

# Create Model
model = Sequential()
model.add(Dense(12, activation='tanh', input_dim=11))
model.add(Dense(6, activation='tanh'))
model.add(Dense(1, activation='sigmoid'))
model.compile(optimizer = adam, loss = 'binary_crossentropy', metrics = ["accuracy"])
tb = TensorBoard(log_dir='./logs', histogram_freq=0, write_graph=True, write_images=True)
hist = model.fit(x_train, y_train, validation_data=(x_test, y_test), epochs=epochs, batch_size=b_size,callbacks=[tb])

# Final evaluation of the model
scores = model.evaluate(x_test, y_test, verbose=0)
print("Accuracy: %.2f%%" % (scores[1]*100))
```

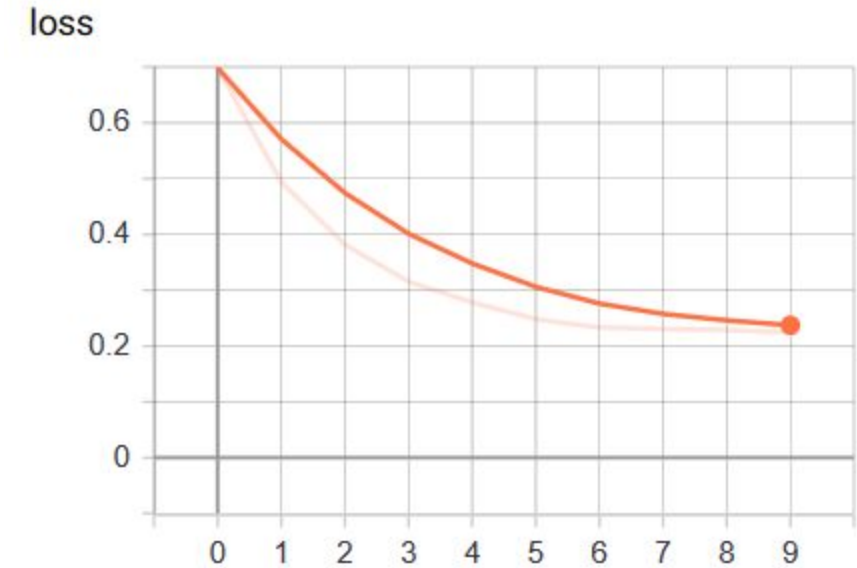
Output

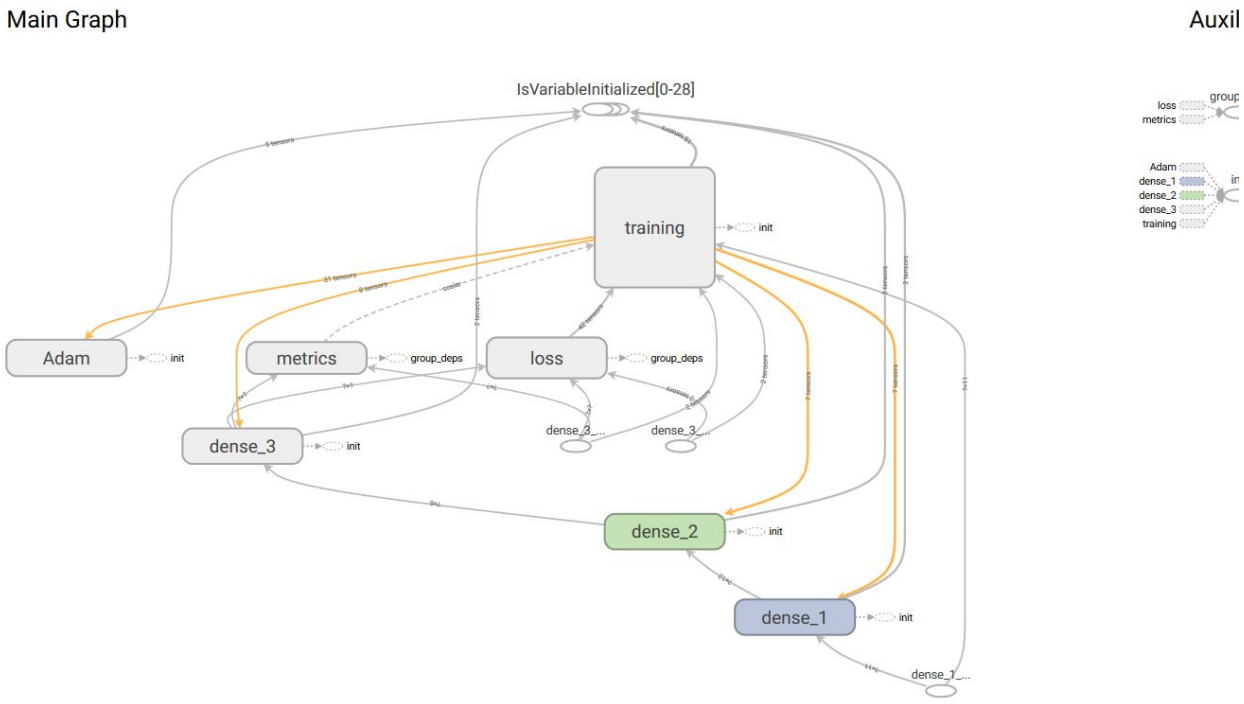
```
Epoch 10/10

63/63 [==============================] - 0s 48us/step - loss:
Accuracy: 90.91%
```

TensorBoard Visualizations

acc







Question - 3

Import libraries:

pandas library - which is mainly used for data manipulation and analysis.

sklearn library - helps us to implement machine learning techniques.

keras library - consists of functions and programs related to neural network.

re library - operations related to regular expression.

```
import pandas as pd

from sklearn.preprocessing import LabelEncoder
from sklearn.model_selection import train_test_split

from keras.preprocessing.text import Tokenizer
from keras.preprocessing.sequence import pad_sequences
from keras.models import Sequential
from keras.layers import Dense, Embedding, SpatialDropout1D
from keras.utils.np_utils import to_categorical
from keras.layers.convolutional import Conv1D, MaxPooling1D
from keras.layers import Flatten
from keras import optimizers
from keras.constraints import maxnorm


import re
```

Read Dataset:

Reading csv file using pandas library.

Pass filename and encoding (encoding which is used for UTF while reading/writing example: 'utf-8') as parameter.

```
dataset = pd.read_csv('spam.csv', encoding='latin-1')
```

View Dataset:

.head and .info displays the dataset and information about the dataset respectively.

```
dataset.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 5572 entries, 0 to 5571
Data columns (total 5 columns):
spam_or_ham    5572 non-null object
message        5572 non-null object
Unnamed: 2     50 non-null object
Unnamed: 3     12 non-null object
Unnamed: 4     6 non-null object
dtypes: object(5)
memory usage: 217.7+ KB
```

```
dataset.head()
```

	spam_or_ham	message
0	ham	Go until jurong point, crazy.. Available only ...
1	ham	Ok lar... Joking wif u oni...
2	spam	Free entry in 2 a wkly comp to win FA Cup fina...
3	ham	U dun say so early hor... U c already then say...
4	ham	Nah I don't think he goes to usf, he lives aro...

Drop unecessary columns:

Unecessary columns are dropped using .drop of method.

Pass list of column names to be dropped, index (0 to drop from index or 1 to drop from columns), inplace (true, for doing operation inplace or else return none).

```
dataset['spam_or_ham']=dataset.spam_or_ham.str.strip()
dataset['message']=dataset.message.str.strip()
dataset['message']=dataset.message.str.lower()
dataset.head()
```

	spam_or_ham	message
0	ham	go until jurong point, crazy.. available only ...
1	ham	ok lar... joking wif u oni...
2	spam	free entry in 2 a wkly comp to win fa cup fina...
3	ham	u dun say so early hor... u c already then say...
4	ham	nah i don't think he goes to usf, he lives aro...

Check drop action:

Check whether the unecessary features are dropped from the dataframe y viewing the dataset again.

Data Pre-processing:

Use .strip to remove empty charecter from both the sides.

Use .lower function to change all the string to lower case.

Count categories:

Use .value_counts function to count number of datapoints in each of the classes.

```
dataset.spam_or_ham.value_counts()
```

```
ham     4825
spam     747
Name: spam_or_ham, dtype: int64
```

Tokenization:

Tokenization chops the string into pieces called tokens.

For example: input : "Hi python, this is my last assignment" output : ["hi","python","this","is","my","last","assignment"]

Create tokenizer model with 1500 as num_word and split the string by the charecter ' '.

fit on the dataset.

make the texts to sequences.

```
maximum_number_of_features = 1500
tokenizer = Tokenizer(num_words=maximum_number_of_features, split=' ')
tokenizer.fit_on_texts(dataset['message'].values)
X = tokenizer.texts_to_sequences(dataset['message'].values)
```

Change the sequence to 2D Numpy array:

Use pad_sequences from keras to change the sequence to 2D Numpy array.

```
print(X)
X = pad_sequences(X)
print("after applying pad_sequences")
print(X)
```

Make categorical variable:

Create a LabelEncoder model.

Use fit_transform to fit the categorical feature 'spam_or_ham to chenge it to categorical variable.

LabelEncoder makes the classes from 0 to total class-1

```
labelencoder = LabelEncoder()
integer_encoded = labelencoder.fit_transform(dataset['spam_or_ham'])
Y = to_categorical(integer_encoded)
print(Y)
```

```
[[1. 0.]
 [1. 0.]
 [0. 1.]
 ...
 [1. 0.]
 [1. 0.]
 [1. 0.]]
```

Split Dataset:

Use train_test_split() function to split the into four that is to categorical feature for train and test (Y_train and Y_test) and othe features (dependent features) for train and test (X_train and X_test).

Make test size to 33%.

Use random state to randomize the datapoints

Print the shape of the splitted datapoints

```
X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size=0.33, random_state=42)
print(X_train.shape, Y_train.shape)
print(X_test.shape, Y_test.shape)
```

```
(3733, 172) (3733, 2)
(1839, 172) (1839, 2)
```

CNN Model training:

Create a Sequential model where sequential model where sequential model is adding to several layers to the model.

Add several layers like Embedding, Conv1D with MaxPooling1D and SpatialDropout1D

At last flatten the fit.

```
model = Sequential()
model.add(Embedding(1500, 128, dropout=0.2, input_length=X.shape[1]))
model.add(SpatialDropout1D(0.2))

model.add(Conv1D(64, kernel_size=3, padding='same', activation='relu'))
model.add(MaxPooling1D(pool_size=2))

model.add(Conv1D(32, kernel_size=3, padding='same', activation='relu'))
model.add(MaxPooling1D(pool_size=2))

model.add(Flatten())
model.add(Dense(2, activation='sigmoid'))

print(model.summary())
model.compile(loss='categorical_crossentropy', optimizer=optimizers.RMSprop(lr=0.001), metrics=['accuracy'])
```

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
embedding_3 (Embedding)      (None, 172, 128)          192000
_________________________________________________________________
spatial_dropout1d_3 (Spatial (None, 172, 128)          0
_________________________________________________________________
conv1d_7 (Conv1D)            (None, 172, 64)           24640
_________________________________________________________________
max_pooling1d_5 (MaxPooling1 (None, 86, 64)            0
_________________________________________________________________
conv1d_8 (Conv1D)            (None, 86, 32)            6176
_________________________________________________________________
max_pooling1d_6 (MaxPooling1 (None, 43, 32)            0
_________________________________________________________________
flatten_2 (Flatten)          (None, 1376)              0
_________________________________________________________________
dense_2 (Dense)              (None, 2)                 2754
=================================================================
Total params: 225,570
Trainable params: 225,570
Non-trainable params: 0
_________________________________________________________________
None
```

Fit model:

Fit the trained model on the splitted dataset

Fit the model for 16 epochs(number of times), batch_size(number of datapoints taken for each step of fitting), validation_data for doing validation.

```
model.fit(X_train, Y_train, validation_data=(X_test, Y_test), epochs=16, batch_size=100, verbose=2)
```

```
Epoch 6/16
 - 4s - loss: 3.8305e-06 - acc: 1.0000 - val_loss: 0.1756 - val_acc: 0.9831
Epoch 7/16
 - 4s - loss: 1.7783e-05 - acc: 1.0000 - val_loss: 0.1776 - val_acc: 0.9804
Epoch 8/16
 - 4s - loss: 1.6294e-05 - acc: 1.0000 - val_loss: 0.1881 - val_acc: 0.9842
Epoch 9/16
 - 4s - loss: 4.0949e-07 - acc: 1.0000 - val_loss: 0.1794 - val_acc: 0.9826
Epoch 10/16
 - 4s - loss: 2.8796e-07 - acc: 1.0000 - val_loss: 0.2003 - val_acc: 0.9831
Epoch 11/16
 - 4s - loss: 1.4407e-07 - acc: 1.0000 - val_loss: 0.2002 - val_acc: 0.9831
Epoch 12/16
 - 4s - loss: 6.8296e-07 - acc: 1.0000 - val_loss: 0.2253 - val_acc: 0.9837
Epoch 13/16
 - 4s - loss: 8.1902e-07 - acc: 1.0000 - val_loss: 0.1953 - val_acc: 0.9821
Epoch 14/16
 - 5s - loss: 9.7425e-07 - acc: 1.0000 - val_loss: 0.2411 - val_acc: 0.9831
Epoch 15/16
 - 5s - loss: 2.3718e-07 - acc: 1.0000 - val_loss: 0.2219 - val_acc: 0.9826
Epoch 16/16
 - 5s - loss: 1.5943e-07 - acc: 1.0000 - val_loss: 0.2148 - val_acc: 0.9831

<keras.callbacks.History at 0x121dbc722b0>
```

Find Accuracy of the model:

Find the accuracy using model.evaluate()

```
score, acc = model.evaluate(X_test, Y_test, verbose=2, batch_size=100)
print(score)
print(acc)
```

```
0.21483234983786348
0.9831430221005845
```

Question - 4

Repeat steps from question - 3 until model training

LSTM Model training:

Create a Sequential model in which sequential model can add several layers to the model.

Add LSTM Layer.

Add several layers like Embedding with and SpatialDropout1D.

```
model = Sequential()
model.add(Embedding(1500, 128,input_length = X.shape[1]))
model.add(SpatialDropout1D(0.4))
model.add(LSTM(196, dropout=0.2, recurrent_dropout=0.2))
model.add(Dense(2,activation='sigmoid'))
model.compile(loss = 'categorical_crossentropy', optimizer='adam',metrics = ['accuracy'])

print(model.summary())
```

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
embedding_6 (Embedding)      (None, 172, 128)          192000
_________________________________________________________________
spatial_dropout1d_3 (Spatial (None, 172, 128)          0
_________________________________________________________________
lstm_4 (LSTM)                (None, 196)               254800
_________________________________________________________________
dense_4 (Dense)              (None, 2)                 394
=================================================================
Total params: 447,194
Trainable params: 447,194
Non-trainable params: 0
_________________________________________________________________
None
```

```
model.fit(X_train, Y_train, validation_data=(X_test, Y_test), epochs = 16, batch_size=100, verbose = 2)
```

```
Epoch 9/16
 - 46s - loss: 0.0048 - acc: 0.9989 - val_loss: 0.0760 - val_acc: 0.9848
Epoch 10/16
 - 44s - loss: 0.0038 - acc: 0.9992 - val_loss: 0.0880 - val_acc: 0.9815
Epoch 11/16
 - 43s - loss: 0.0126 - acc: 0.9960 - val_loss: 0.0760 - val_acc: 0.9777
Epoch 12/16
 - 47s - loss: 0.0091 - acc: 0.9971 - val_loss: 0.0944 - val_acc: 0.9826
Epoch 13/16
 - 44s - loss: 0.0049 - acc: 0.9989 - val_loss: 0.0798 - val_acc: 0.9837
Epoch 14/16
 - 44s - loss: 0.0030 - acc: 0.9992 - val_loss: 0.0866 - val_acc: 0.9853
Epoch 15/16
 - 45s - loss: 0.0031 - acc: 0.9989 - val_loss: 0.0857 - val_acc: 0.9837
Epoch 16/16
 - 47s - loss: 0.0019 - acc: 0.9995 - val_loss: 0.0906 - val_acc: 0.9821

<keras.callbacks.History at 0x19f3ffdfd68>
```

```
score,acc = model.evaluate(X_test,Y_test,verbose=2,batch_size=40)
print(score)
print(acc)
```

```
0.6916262805299827
0.6943991251650422
```

Question - 5

Accuracy of CCN is much greater than LSTM.

Question - 6

Import Libraries

numpy - to perform operations on large and multi-dimensional arrays.

keras - for doing operations on deep learning.

Import dataset

Import dataset using .load_data() function.

Nomalize datapoints

Change the datatype of the datapoints to 'float32'. Divide the datapoints by 255 to normalize.

Do onehot encoding:

Do onehot encoding to change the target variable to categorical variable.

Create LSTM model

Fit the model

Print Accuracy

```
Train on 50000 samples, validate on 10000 samples

W0722 21:02:03.515216  9064 deprecation_wrapper.py:119] From C:\Users\kaphc\Anaconda3\envs\python3.6\lib\site-packages\keras
\callbacks.py:850: The name tf.summary.merge_all is deprecated. Please use tf.compat.v1.summary.merge_all instead.

W0722 21:02:03.516214  9064 deprecation_wrapper.py:119] From C:\Users\kaphc\Anaconda3\envs\python3.6\lib\site-packages\keras
\callbacks.py:853: The name tf.summary.FileWriter is deprecated. Please use tf.compat.v1.summary.FileWriter instead.


Epoch 1/1
50000/50000 [==============================] - 712s 14ms/step - loss: 4.6059 - acc: 0.0090 - val_loss: 4.6052 - val_acc: 0.0
100

<keras.callbacks.History at 0x1d3c9753ba8>
```

```
4.605185461044312
0.009999999776482582
```

dropout_9 (Dropout)	(None, 128, 8, 8)	0
conv2d_12 (Conv2D)	(None, 128, 8, 8)	147584
max_pooling2d_6 (MaxPooling2	(None, 128, 4, 4)	0
flatten_2 (Flatten)	(None, 2048)	0
dropout_10 (Dropout)	(None, 2048)	0
dense_3 (Dense)	(None, 1024)	2098176
dropout_11 (Dropout)	(None, 1024)	0
dense_4 (Dense)	(None, 512)	524800
dropout_12 (Dropout)	(None, 512)	0
dense_5 (Dense)	(None, 100)	51300

```
Total params: 2,961,284
Trainable params: 2,961,284
Non-trainable params: 0

None
```