Project 3
Harshavardhan Reddy Mallannagari 11603389

Introduction

This project uses a dataset of labeled tweets to train a Neural Network model in PyTorch for sentiment analysis. The model takes 12 different features, including sentiment scores and word count, and outputs a probability of the tweet being positive. The model is trained on a training set and evaluated using accuracy and F1-score on a test set.

Implementation:

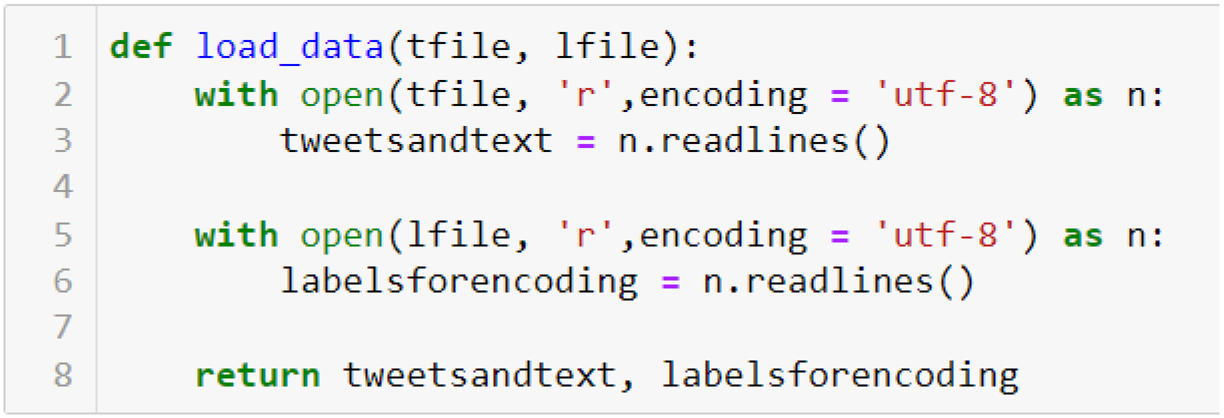
```python
import math
import re
import torch
import torch.nn as nn
import torch.optim as optim
from torch.utils.data import Dataset as DT
from torch.utils.data import DataLoader as DL
from sklearn.metrics import confusion_matrix, f1_score
from sklearn.preprocessing import LabelEncoder
from sklearn.model_selection import KFold
import warnings
import numpy as np
import nltk
from nltk.tokenize import word_tokenize
import string
from nltk.corpus import stopwords
from nltk.stem import PorterStemmer
from gensim.models import Word2Vec
import matplotlib.pyplot as plt

warnings.filterwarnings('ignore')
```

Importing the required libraries for running the code and analyzing it.

```python
def read_lexicons_files(lex_files):
    with open(lex_files, "r", encoding = 'utf-8') as ff:
        data = ff.readlines()
#     basic cleaning of data
    data = [dt.strip().split('\t') for dt in data]
    lexicon_dict = {}
    for v in data:
        if len(v) == 3:
            key = v[0]
            value = {'-ve': float(v[1]), '+ve': float(v[2])}
            lexicon_dict[key] = value

    return lexicon_dict
```

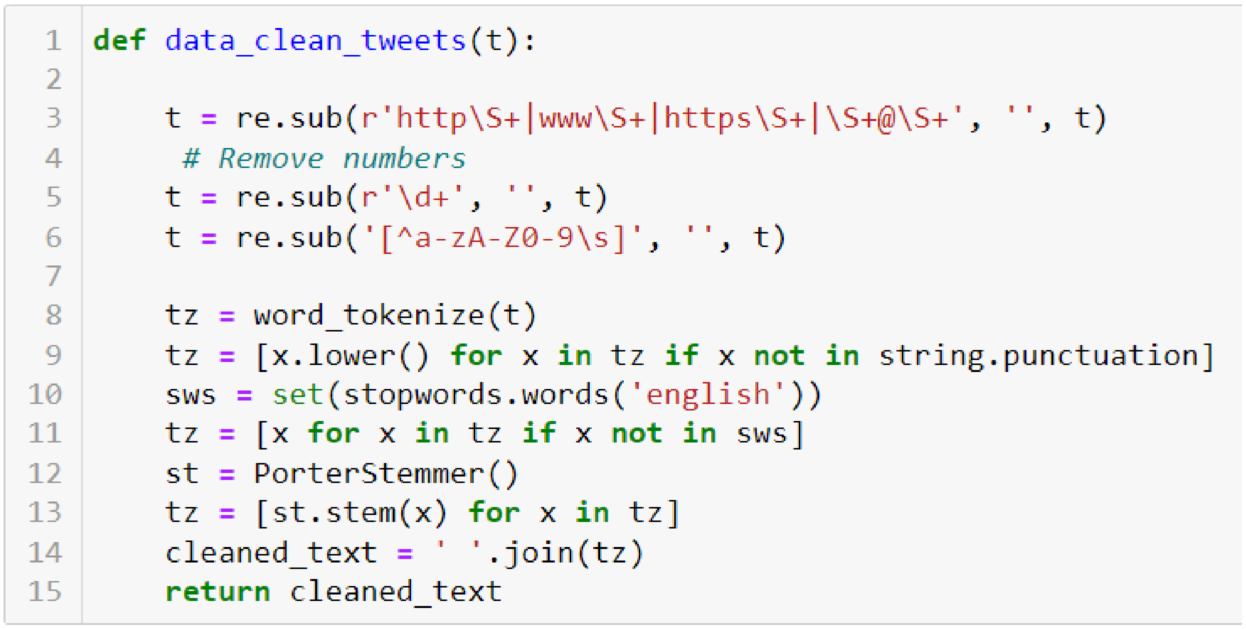
This function definition is used to read the lexicon files using the encoding utf-8, this code reads the data ,cleans the data and assigns positive and negative values to the key.

After parsing all the values the function returns the dictionary of lexicons.



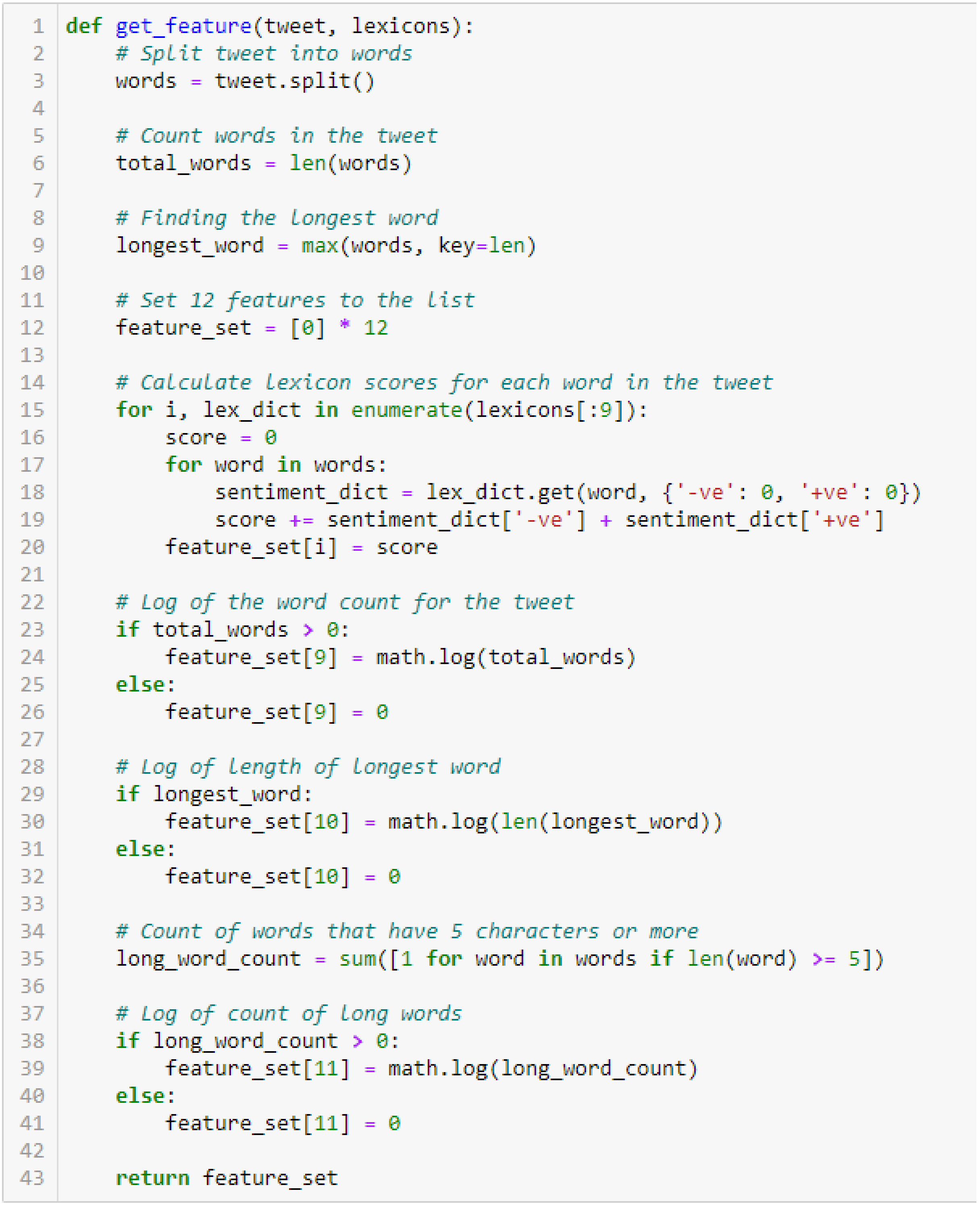
```python
def load_data(tfile, lfile):
    with open(tfile, 'r',encoding = 'utf-8') as n:
        tweetsandtext = n.readlines()

    with open(lfile, 'r',encoding = 'utf-8') as n:
        labelsforencoding = n.readlines()

    return tweetsandtext, labelsforencoding
```

In the above code we are taking 2 input files, one is the text file and the other is the label file which is related to the text file.



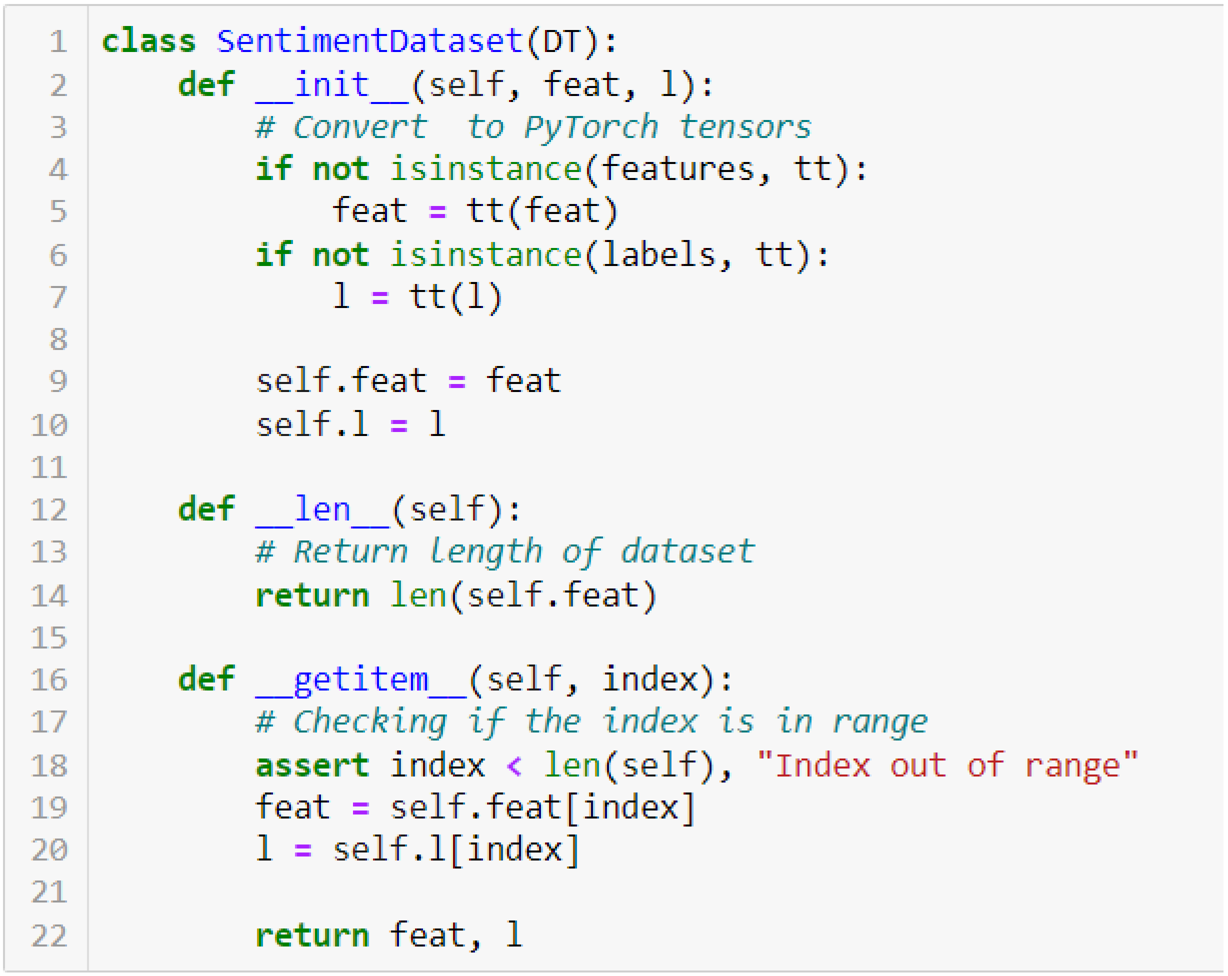
```python
def data_clean_tweets(t):

    t = re.sub(r'http\S+|www\S+|https\S+|\S+@\S+', '', t)
     # Remove numbers
    t = re.sub(r'\d+', '', t)
    t = re.sub('[^a-zA-Z0-9\s]', '', t)

    tz = word_tokenize(t)
    tz = [x.lower() for x in tz if x not in string.punctuation]
    sws = set(stopwords.words('english'))
    tz = [x for x in tz if x not in sws]
    st = PorterStemmer()
    tz = [st.stem(x) for x in tz]
    cleaned_text = ' '.join(tz)
    return cleaned_text
```

In the above function, the data is cleaned by using regular expressions, by using stop-words from nltk module. We are also using porter stemmer which applies stemming to each word.



```python
def get_feature(tweet, lexicons):
    # Split tweet into words
    words = tweet.split()

    # Count words in the tweet
    total_words = len(words)

    # Finding the longest word
    longest_word = max(words, key=len)

    # Set 12 features to the list
    feature_set = [0] * 12

    # Calculate lexicon scores for each word in the tweet
    for i, lex_dict in enumerate(lexicons[:9]):
        score = 0
        for word in words:
            sentiment_dict = lex_dict.get(word, {'-ve': 0, '+ve': 0})
            score += sentiment_dict['-ve'] + sentiment_dict['+ve']
        feature_set[i] = score

    # Log of the word count for the tweet
    if total_words > 0:
        feature_set[9] = math.log(total_words)
    else:
        feature_set[9] = 0

    # Log of length of longest word
    if longest_word:
        feature_set[10] = math.log(len(longest_word))
    else:
        feature_set[10] = 0

    # Count of words that have 5 characters or more
    long_word_count = sum([1 for word in words if len(word) >= 5])

    # Log of count of long words
    if long_word_count > 0:
        feature_set[11] = math.log(long_word_count)
    else:
        feature_set[11] = 0

    return feature_set
```

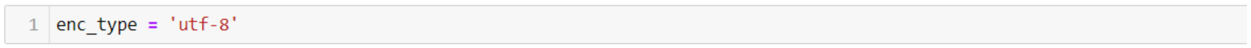
The above function creates the features for passing. This is being used from project 2. This generates 12 features and returns the features list.



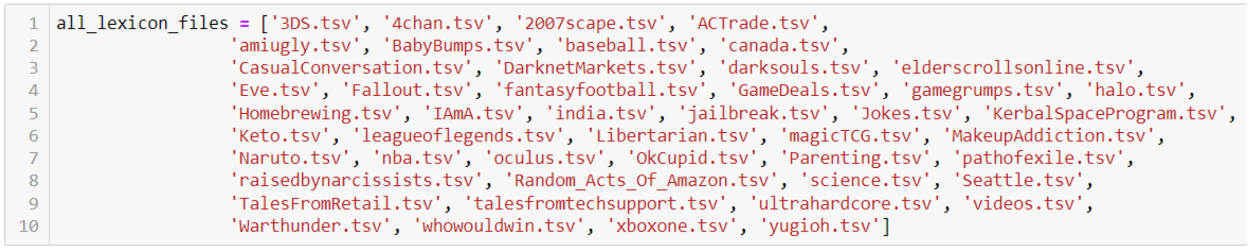
```python
class SentimentDataset(DT):
    def __init__(self, feat, l):
        # Convert  to PyTorch tensors
        if not isinstance(features, tt):
            feat = tt(feat)
        if not isinstance(labels, tt):
            l = tt(l)

        self.feat = feat
        self.l = l

    def __len__(self):
        # Return length of dataset
        return len(self.feat)

    def __getitem__(self, index):
        # Checking if the index is in range
        assert index < len(self), "Index out of range"
        feat = self.feat[index]
        l = self.l[index]

        return feat, l
```

Sentiment data set is a class that is extended from Data set of torch library. Here in this we convert the features to the tensors.

The length returns the length of feat variable which is the features that is converted to tensors. Get Item returns the feat and label.

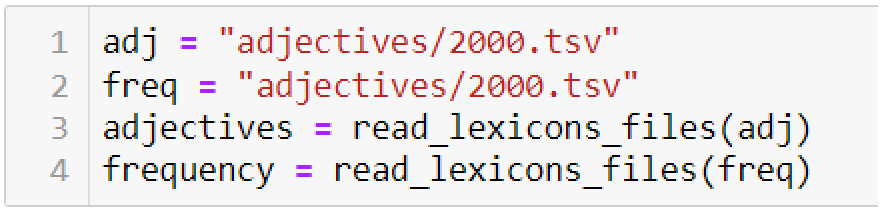


```python
enc_type = 'utf-8'
```



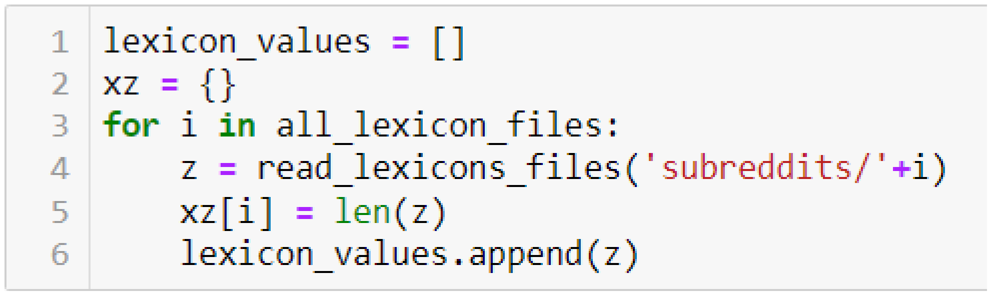
```python
all_lexicon_files = ['3DS.tsv', '4chan.tsv', '2007scape.tsv', 'ACTrade.tsv',
                     'amiugly.tsv', 'BabyBumps.tsv', 'baseball.tsv', 'canada.tsv',
                     'CasualConversation.tsv', 'DarknetMarkets.tsv', 'darksouls.tsv', 'elderscrollsonline.tsv',
                     'Eve.tsv', 'Fallout.tsv', 'fantasyfootball.tsv', 'GameDeals.tsv', 'gamegrumps.tsv', 'halo.tsv',
                     'Homebrewing.tsv', 'IAmA.tsv', 'india.tsv', 'jailbreak.tsv', 'Jokes.tsv', 'KerbalSpaceProgram.tsv',
                     'Keto.tsv', 'leagueoflegends.tsv', 'Libertarian.tsv', 'magicTCG.tsv', 'MakeupAddiction.tsv',
                     'Naruto.tsv', 'nba.tsv', 'oculus.tsv', 'OkCupid.tsv', 'Parenting.tsv', 'pathofexile.tsv',
                     'raisedbynarcissists.tsv', 'Random_Acts_Of_Amazon.tsv', 'science.tsv', 'Seattle.tsv',
                     'TalesFromRetail.tsv', 'talesfromtechsupport.tsv', 'ultrahardcore.tsv', 'videos.tsv',
                     'Warthunder.tsv', 'whowouldwin.tsv', 'xboxone.tsv', 'yugioh.tsv']
```

Setting the encoding type as utf-8 and assigning all the names of lexicon files to the list to read.



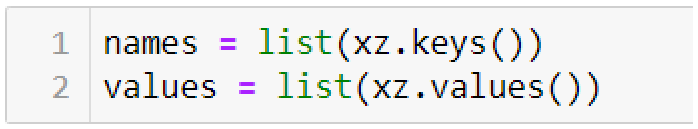
```python
adj = "adjectives/2000.tsv"
freq = "adjectives/2000.tsv"
adjectives = read_lexicons_files(adj)
frequency = read_lexicons_files(freq)
```

This reads the adjective file.



```python
lexicon_values = []
xz = {}
for i in all_lexicon_files:
    z = read_lexicons_files('subreddits/'+i)
    xz[i] = len(z)
    lexicon_values.append(z)
```

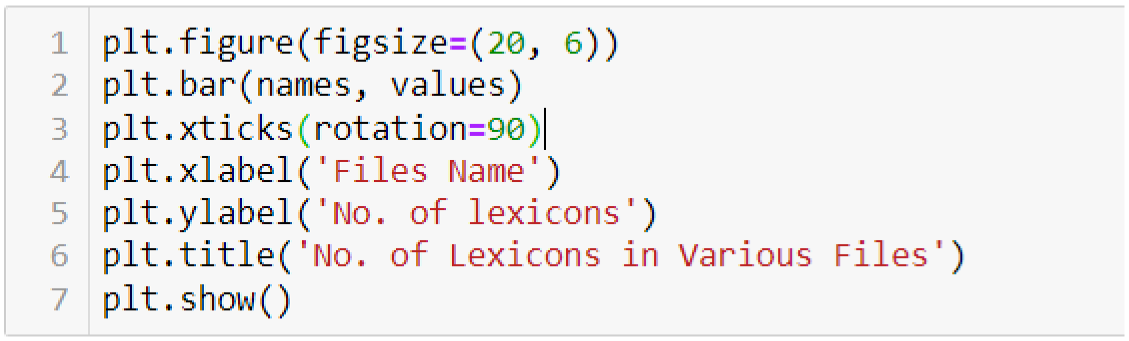
Reading all the lexicon files using a loop and storing in lexicon_values variable.



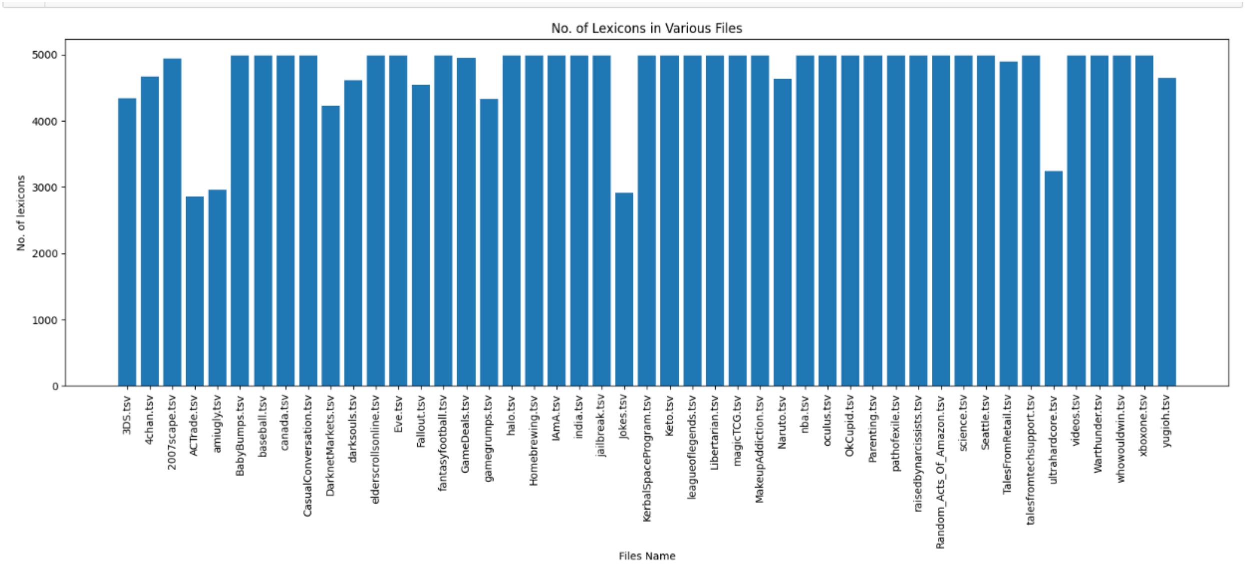
```python
names = list(xz.keys())
values = list(xz.values())
```

Stores keys and values in the variables name ad values.

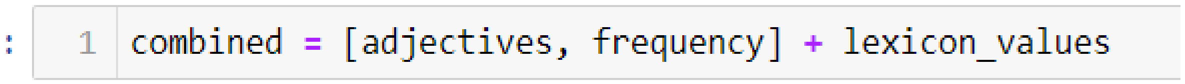
These are used in visualization.



```python
plt.figure(figsize=(20, 6))
plt.bar(names, values)
plt.xticks(rotation=90)
plt.xlabel('Files Name')
plt.ylabel('No. of lexicons')
plt.title('No. of Lexicons in Various Files')
plt.show()
```

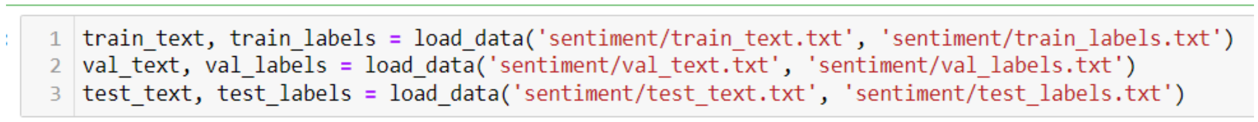


This graph shows how many lexicons are there in which file.



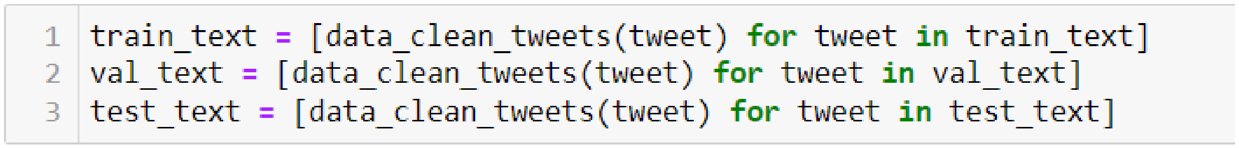
```
combined = [adjectives, frequency] + lexicon_values
```

Combining lexicon and adjective and frequency values.



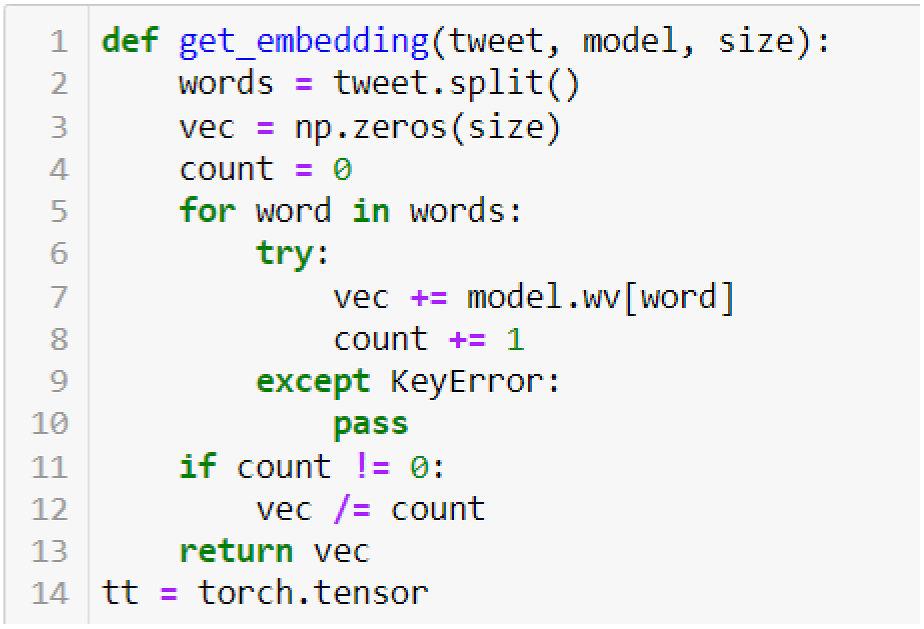
```
train_text, train_labels = load_data('sentiment/train_text.txt', 'sentiment/train_labels.txt')
val_text, val_labels = load_data('sentiment/val_text.txt', 'sentiment/val_labels.txt')
test_text, test_labels = load_data('sentiment/test_text.txt', 'sentiment/test_labels.txt')
```

Reading the values from the files.



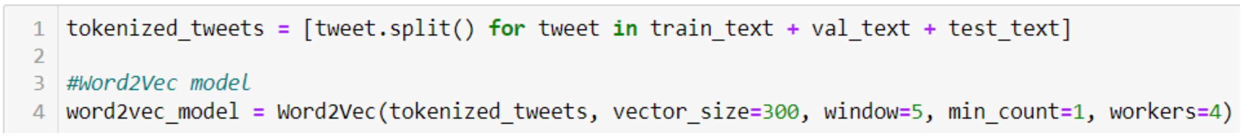
```
train_text = [data_clean_tweets(tweet) for tweet in train_text]
val_text = [data_clean_tweets(tweet) for tweet in val_text]
test_text = [data_clean_tweets(tweet) for tweet in test_text]
```

Cleaning the data.



```python
def get_embedding(tweet, model, size):
    words = tweet.split()
    vec = np.zeros(size)
    count = 0
    for word in words:
        try:
            vec += model.wv[word]
            count += 1
        except KeyError:
            pass
    if count != 0:
        vec /= count
    return vec
tt = torch.tensor
```

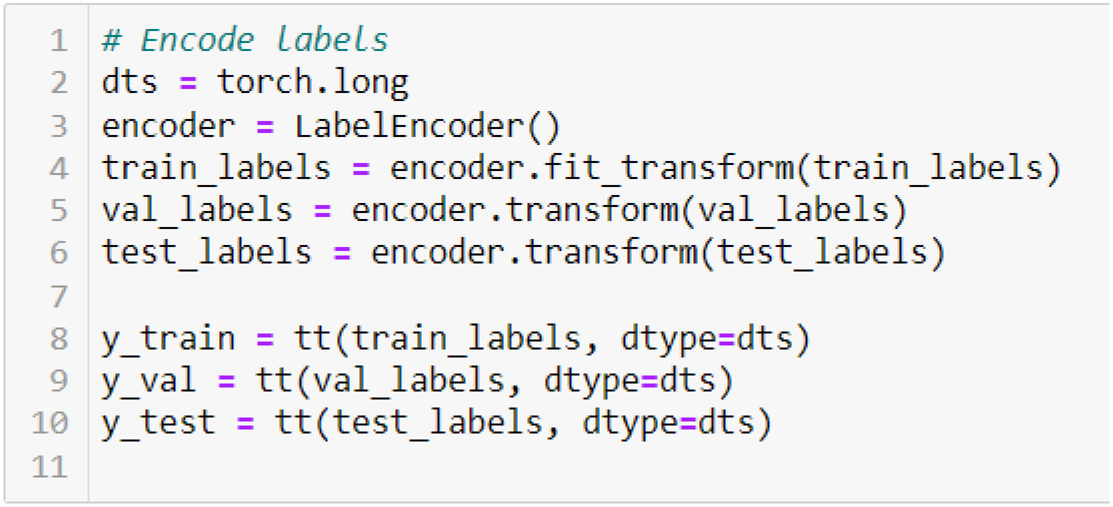
This is used to get the embedding vector by passing the tweet , model and the size.



```python
tokenized_tweets = [tweet.split() for tweet in train_text + val_text + test_text]

#Word2Vec model
word2vec_model = Word2Vec(tokenized_tweets, vector_size=300, window=5, min_count=1, workers=4)
```

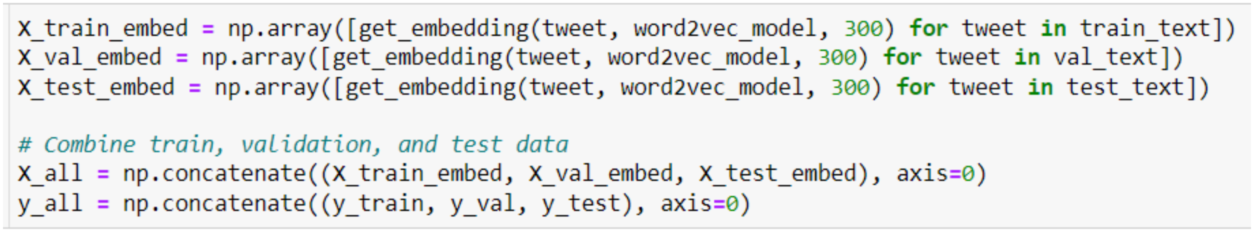
Creating tokens by combining all the data.

Then passing all the tokenized to the word 2 vector model.



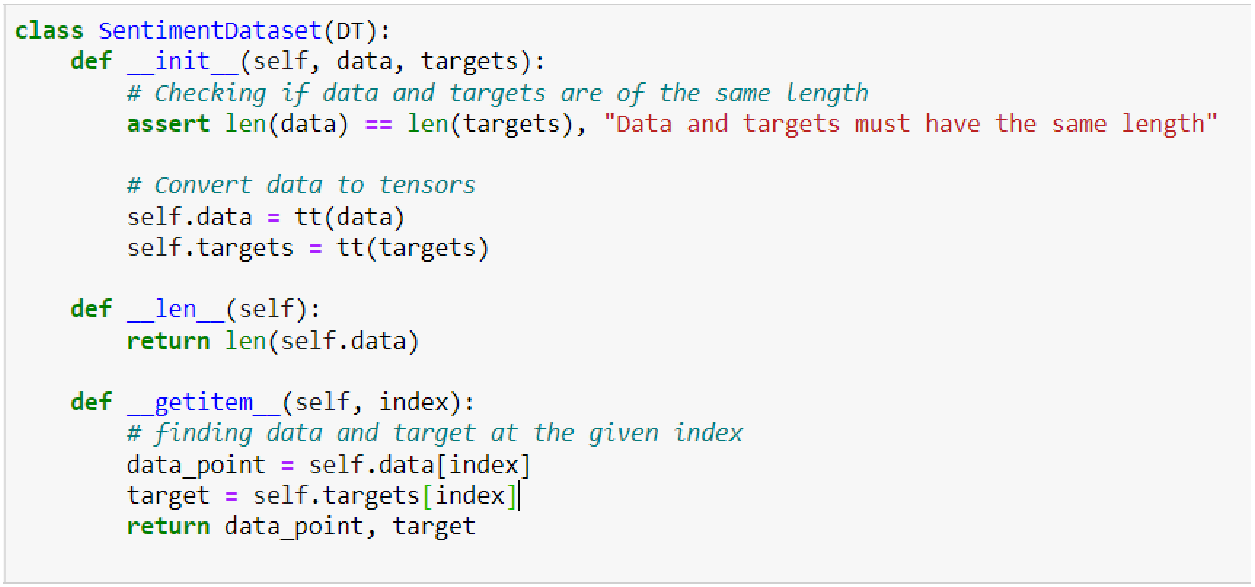
```python
# Encode labels
dts = torch.long
encoder = LabelEncoder()
train_labels = encoder.fit_transform(train_labels)
val_labels = encoder.transform(val_labels)
test_labels = encoder.transform(test_labels)

y_train = tt(train_labels, dtype=dts)
y_val = tt(val_labels, dtype=dts)
y_test = tt(test_labels, dtype=dts)

```

Encoding the labels of the files. The labels are encoded to long variable of torch type.



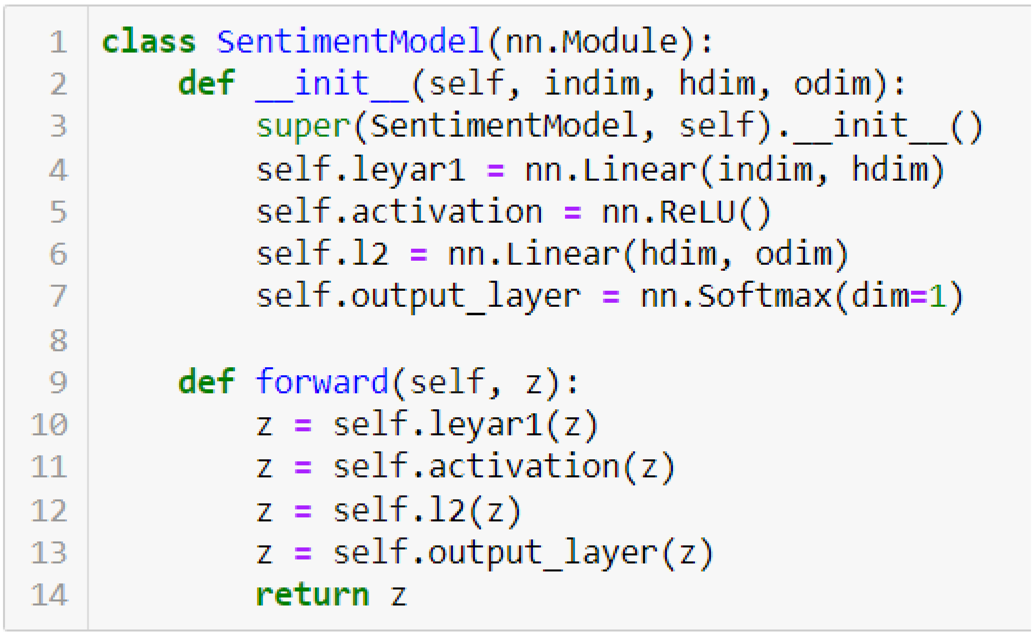
```
X_train_embed = np.array([get_embedding(tweet, word2vec_model, 300) for tweet in train_text])
X_val_embed = np.array([get_embedding(tweet, word2vec_model, 300) for tweet in val_text])
X_test_embed = np.array([get_embedding(tweet, word2vec_model, 300) for tweet in test_text])

# Combine train, validation, and test data
X_all = np.concatenate((X_train_embed, X_val_embed, X_test_embed), axis=0)
y_all = np.concatenate((y_train, y_val, y_test), axis=0)
```

Getting the train validation, testing set of size 300 . After that adding all the values to the x_all variable as a numpy library.



```
class SentimentDataset(DT):
    def __init__(self, data, targets):
        # Checking if data and targets are of the same length
        assert len(data) == len(targets), "Data and targets must have the same length"

        # Convert data to tensors
        self.data = tt(data)
        self.targets = tt(targets)

    def __len__(self):
        return len(self.data)

    def __getitem__(self, index):
        # finding data and target at the given index
        data_point = self.data[index]
        target = self.targets[index]
        return data_point, target
```

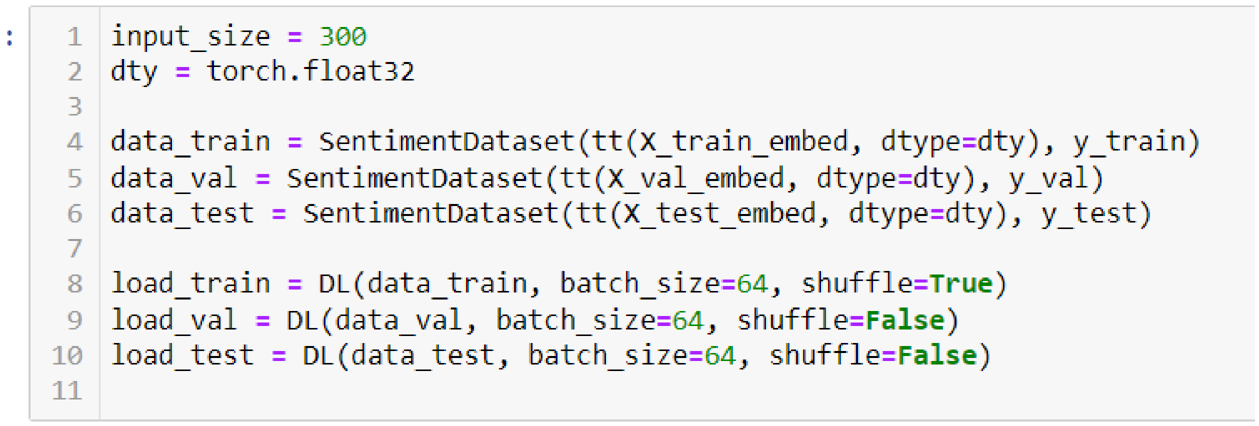
This is extended from the Dataset of the torch module. This checks the input values are of same length or not. It converts the data into torch tensors for running the code.



```
class SentimentModel(nn.Module):
    def __init__(self, indim, hdim, odim):
        super(SentimentModel, self).__init__()
        self.leyar1 = nn.Linear(indim, hdim)
        self.activation = nn.ReLU()
        self.l2 = nn.Linear(hdim, odim)
        self.output_layer = nn.Softmax(dim=1)

    def forward(self, z):
        z = self.leyar1(z)
        z = self.activation(z)
        z = self.l2(z)
        z = self.output_layer(z)
        return z
```

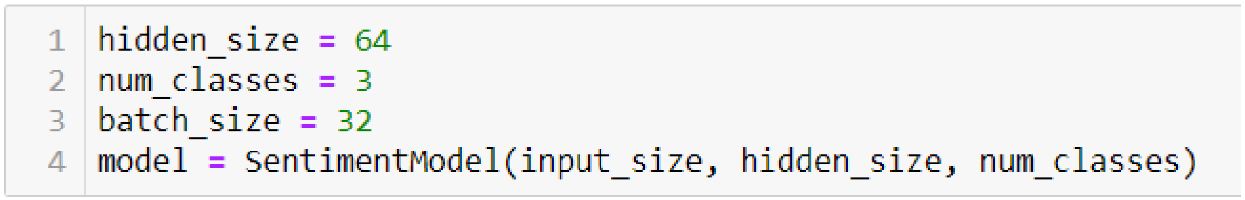
Project 3
Harshavardhan Reddy Mallannagari 11603389

This is extended from the model nn module of torch library. This model uses the linear function in nn module(layers) , relu activation, softmax. The forward function returns the output of the output layer after performing the activation functions on the data.



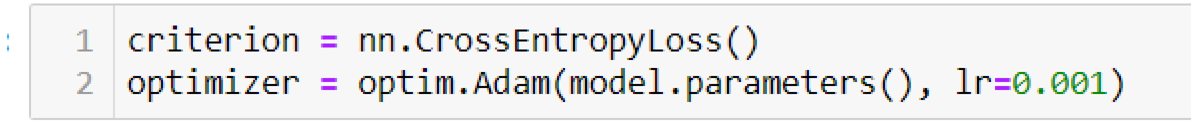
```
input_size = 300
dty = torch.float32

data_train = SentimentDataset(tt(X_train_embed, dtype=dty), y_train)
data_val = SentimentDataset(tt(X_val_embed, dtype=dty), y_val)
data_test = SentimentDataset(tt(X_test_embed, dtype=dty), y_test)

load_train = DL(data_train, batch_size=64, shuffle=True)
load_val = DL(data_val, batch_size=64, shuffle=False)
load_test = DL(data_test, batch_size=64, shuffle=False)

```

This has variables which holds the data, inputsize. These variables holds the data that is passed to the DataLoader of the torch library and holds those values in the provided variables.



```
hidden_size = 64
num_classes = 3
batch_size = 32
model = SentimentModel(input_size, hidden_size, num_classes)
```

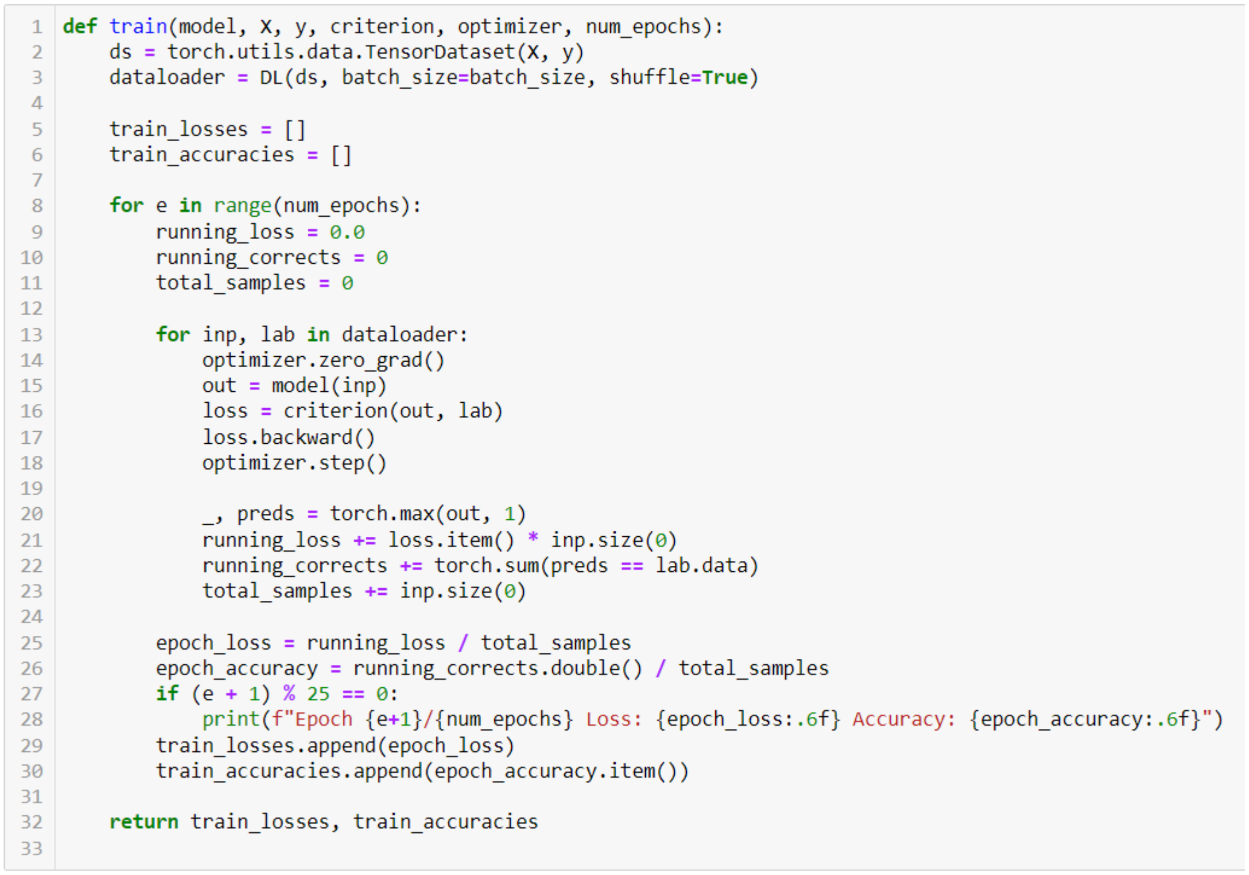
Setting the hidden size of layers as 64, classes division as positive, negative , neutral, batch size as input to neural network as 32 and model as sentimentModel. This is the class object created for the sentiment Model.



```
criterion = nn.CrossEntropyLoss()
optimizer = optim.Adam(model.parameters(), lr=0.001)
```

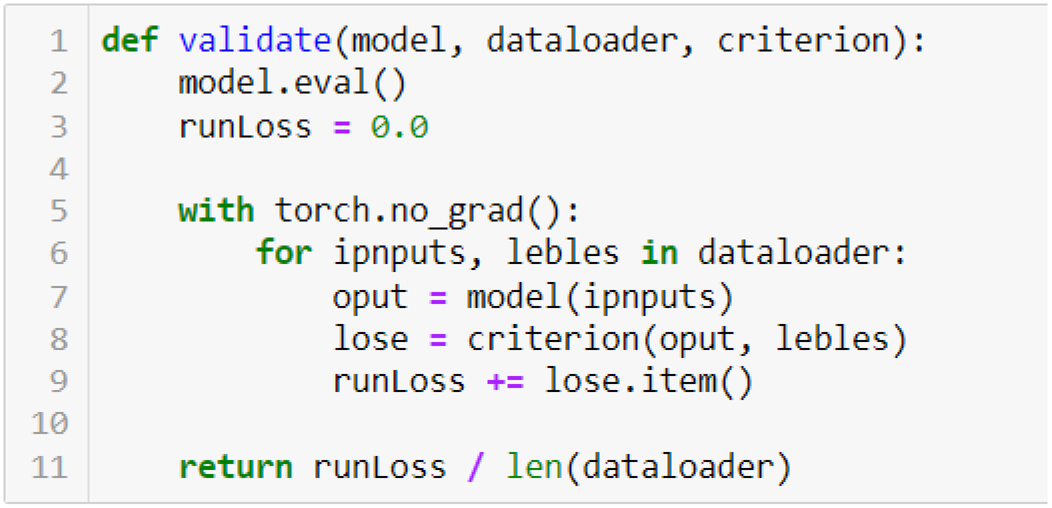
using the crossEntropyLoss as the criterion.

Using the adam optimizer for the model.



```python
def train(model, X, y, criterion, optimizer, num_epochs):
    ds = torch.utils.data.TensorDataset(X, y)
    dataloader = DL(ds, batch_size=batch_size, shuffle=True)

    train_losses = []
    train_accuracies = []

    for e in range(num_epochs):
        running_loss = 0.0
        running_corrects = 0
        total_samples = 0

        for inp, lab in dataloader:
            optimizer.zero_grad()
            out = model(inp)
            loss = criterion(out, lab)
            loss.backward()
            optimizer.step()

            _, preds = torch.max(out, 1)
            running_loss += loss.item() * inp.size(0)
            running_corrects += torch.sum(preds == lab.data)
            total_samples += inp.size(0)

        epoch_loss = running_loss / total_samples
        epoch_accuracy = running_corrects.double() / total_samples
        if (e + 1) % 25 == 0:
            print(f"Epoch {e+1}/{num_epochs} Loss: {epoch_loss:.6f} Accuracy: {epoch_accuracy:.6f}")
        train_losses.append(epoch_loss)
        train_accuracies.append(epoch_accuracy.item())

    return train_losses, train_accuracies
```

This is the train model that calculates the losses and stores the values for every fold and prints the loss for every epoch. Finally it returns the appened values of losses and accuracies



```python
def validate(model, dataloader, criterion):
    model.eval()
    runLoss = 0.0

    with torch.no_grad():
        for ipnputs, lebles in dataloader:
            oput = model(ipnputs)
            lose = criterion(oput, lebles)
            runLoss += lose.item()

    return runLoss / len(dataloader)
```

This function sets the model in evaluation This turns off training-specific layers like dropout and batch normalization which should not be active during evaluation.

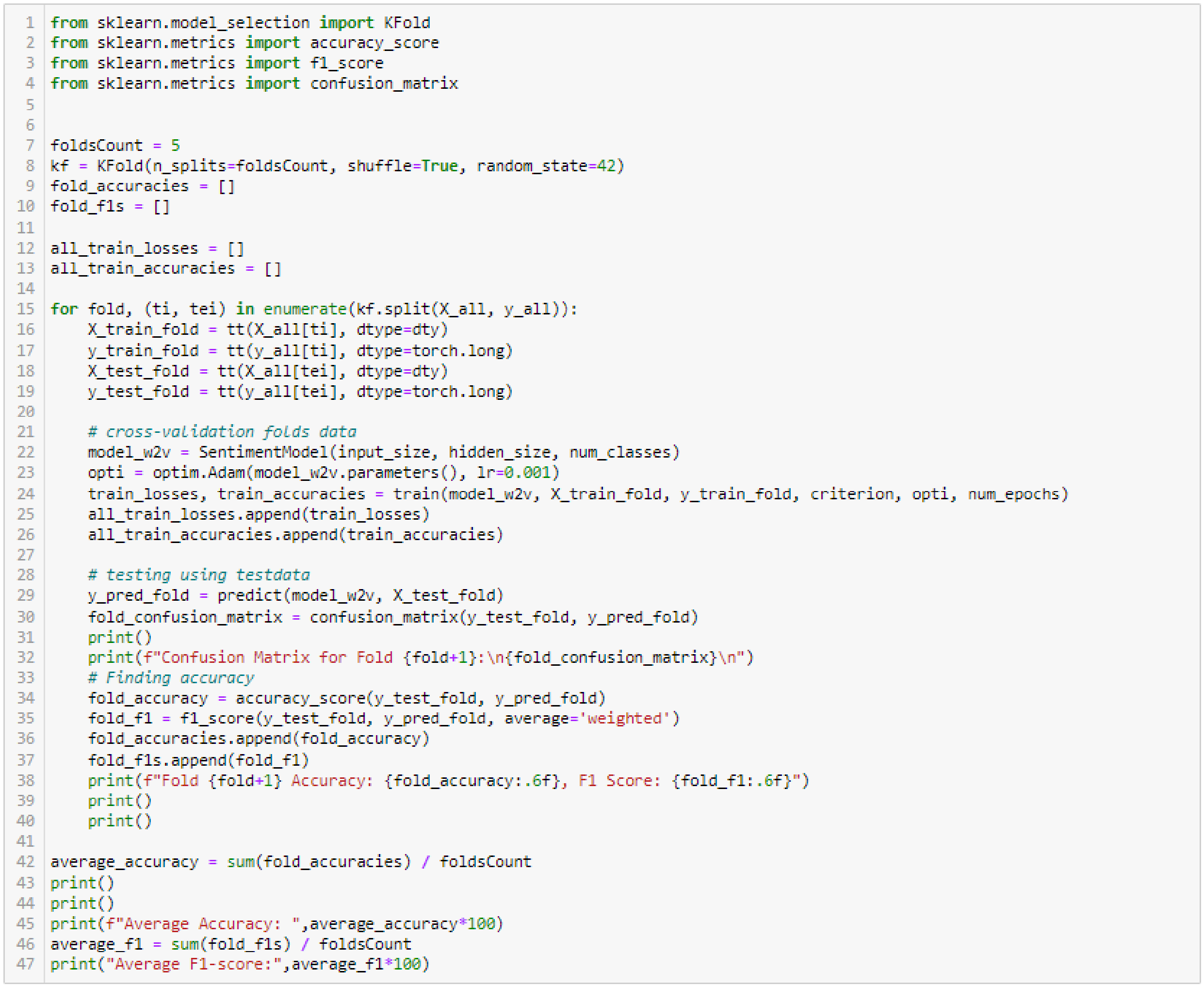
Runloss is used to keep track of the overall loss. This uses the criterion. This returns the average loss of the batch that is sent.

```python
def predict(model, X):
    model.eval()
    ds = torch.utils.data.TensorDataset(X)
    dloader = DL(ds, batch_size=batch_size, shuffle=False)

    pre = []
    with torch.no_grad():
        for inputs, in dloader:
            outputs = model(inputs)
            _, preds = torch.max(outputs, 1)
            pre.extend(preds.tolist())
    return pre
```

This method predict takes the model and tensor. After all the work this function sends the predicted values from the pytorch model.

```python
num_epochs = 100
```

Setting number of epochs for each fold as 100.



```python
from sklearn.model_selection import KFold
from sklearn.metrics import accuracy_score
from sklearn.metrics import f1_score
from sklearn.metrics import confusion_matrix


foldsCount = 5
kf = KFold(n_splits=foldsCount, shuffle=True, random_state=42)
fold_accuracies = []
fold_f1s = []

all_train_losses = []
all_train_accuracies = []

for fold, (ti, tei) in enumerate(kf.split(X_all, y_all)):
    X_train_fold = tt(X_all[ti], dtype=dty)
    y_train_fold = tt(y_all[ti], dtype=torch.long)
    X_test_fold = tt(X_all[tei], dtype=dty)
    y_test_fold = tt(y_all[tei], dtype=torch.long)

    # cross-validation folds data
    model_w2v = SentimentModel(input_size, hidden_size, num_classes)
    opti = optim.Adam(model_w2v.parameters(), lr=0.001)
    train_losses, train_accuracies = train(model_w2v, X_train_fold, y_train_fold, criterion, opti, num_epochs)
    all_train_losses.append(train_losses)
    all_train_accuracies.append(train_accuracies)

    # testing using testdata
    y_pred_fold = predict(model_w2v, X_test_fold)
    fold_confusion_matrix = confusion_matrix(y_test_fold, y_pred_fold)
    print()
    print(f"Confusion Matrix for Fold {fold+1}:\n{fold_confusion_matrix}\n")
    # Finding accuracy
    fold_accuracy = accuracy_score(y_test_fold, y_pred_fold)
    fold_f1 = f1_score(y_test_fold, y_pred_fold, average='weighted')
    fold_accuracies.append(fold_accuracy)
    fold_f1s.append(fold_f1)
    print(f"Fold {fold+1} Accuracy: {fold_accuracy:.6f}, F1 Score: {fold_f1:.6f}")
    print()
    print()

average_accuracy = sum(fold_accuracies) / foldsCount
print()
print()
print(f"Average Accuracy: ",average_accuracy*100)
average_f1 = sum(fold_f1s) / foldsCount
print("Average F1-score:",average_f1*100)
```

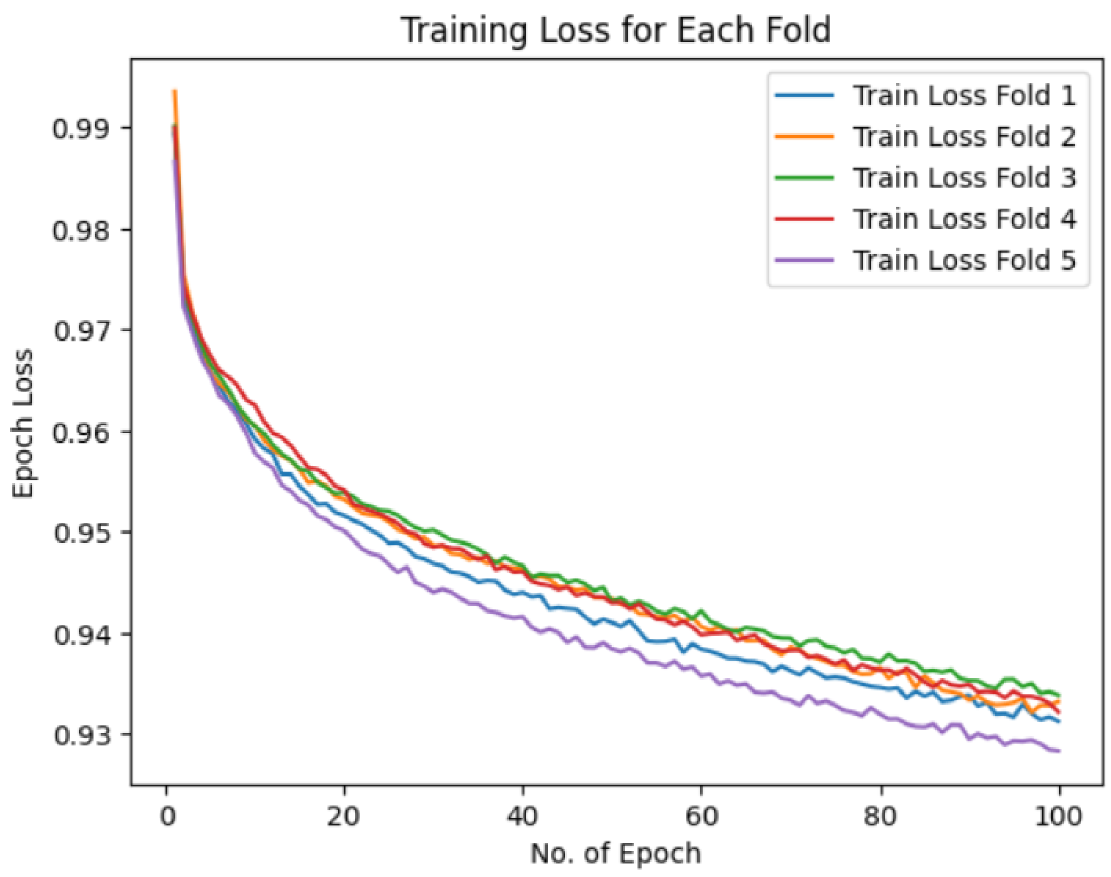
This is the main cell block which calls the train model and calculates the losses and accuracies of the model for each epoch. In this we hold all the losses and accuracies for graph visualization. This prints the visualizations and confusion matrix , accuracy , f1 score.

```
Epoch 25/100 Loss: 0.948883 Accuracy: 0.578622
Epoch 50/100 Loss: 0.941048 Accuracy: 0.589891
Epoch 75/100 Loss: 0.935682 Accuracy: 0.596924
Epoch 100/100 Loss: 0.931273 Accuracy: 0.602663

Confusion Matrix for Fold 1:
[[ 623 1218  428]
 [ 451 3749 1280]
 [ 115 1526 2590]]

Fold 1 Accuracy: 0.581135, F1 Score: 0.569202


Epoch 25/100 Loss: 0.951017 Accuracy: 0.577474
Epoch 50/100 Loss: 0.943405 Accuracy: 0.585864
Epoch 75/100 Loss: 0.936826 Accuracy: 0.594691
Epoch 100/100 Loss: 0.933241 Accuracy: 0.600367

Confusion Matrix for Fold 2:
[[ 610 1280  374]
 [ 434 3968 1089]
 [ 102 1711 2412]]

Fold 2 Accuracy: 0.583472, F1 Score: 0.569812
```

The above is the sample of the output.

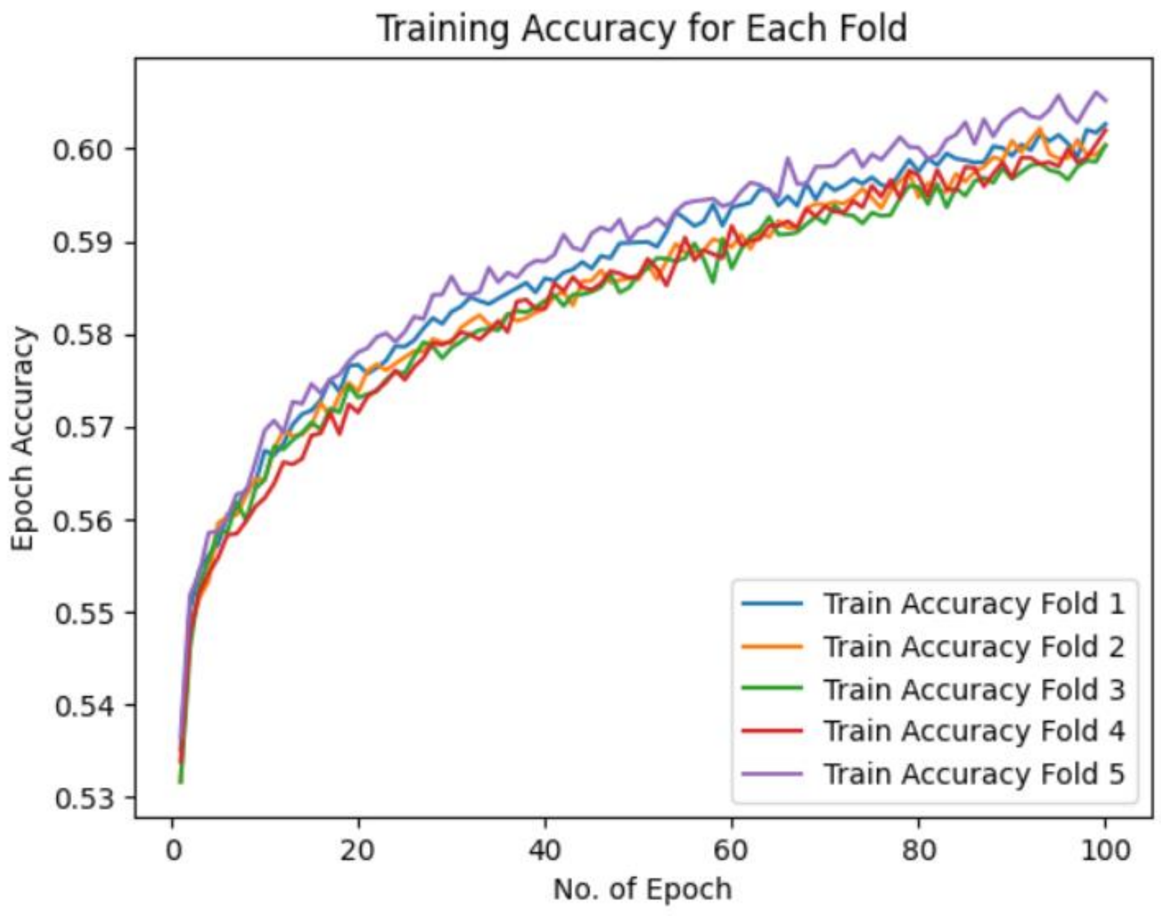
```python
for fold, train_losses in enumerate(all_train_losses):
    plt.plot(range(1, num_epochs+1), train_losses, label=f'Train Loss Fold {fold+1}')

plt.title('Training Loss for Each Fold')
plt.xlabel('No. of Epoch')
plt.ylabel('Epoch Loss')
plt.legend()
plt.show()
```

The above code shows graph for the epoch and epoch loss for every fold.



```python
for fold, train_accuracies in enumerate(all_train_accuracies):
    plt.plot(range(1, num_epochs+1), train_accuracies, label=f'Train Accuracy Fold {fold+1}')

plt.title('Training Accuracy for Each Fold')
plt.xlabel('No. of Epoch')
plt.ylabel('Epoch Accuracy')
plt.legend()
plt.show()
```

The above code shows the epoch number and accuracy of the model.



References:

https://pypi.org/project/nets/0.0.3.1/?
https://discuss.pytorch.org/t/batch-normalization-and-weight-initialization-in-seq2seq/54934
https://numpy.org/
https://pandas.pydata.org/
https://matplotlib.org/
https://scikit-learn.org/stable/
https://pytorch.org/get-started/locally/
https://scikit-learn.org/stable/modules/generated/sklearn.feature_extraction.text.CountVectorizer.html
https://stackoverflow.com/questions/34093264/python-logistic-regression-beginner
https://pytorch.org/docs/stable/optim.html
https://stackoverflow.com/questions/22540449/how-can-i-rotate-a-matplotlib-plot-through-90-degrees
https://stackoverflow.com/questions/53975717/pytorch-connection-between-loss-backward-and-optimizer-step
https://pytorch.org/docs/stable/generated/torch.nn.BCELoss.html
https://scikit-learn.org/stable/modules/generated/sklearn.metrics.f1_score.html
https://www.deeplearningwizard.com/deep_learning/practical_pytorch/pytorch_feedforward_neuralnetwork/

Project 3
Harshavardhan Reddy Mallannagari 11603389

https://www.deeplearningwizard.com/deep_learning/boosting_models_pytorch/weight_initialization_activation_functions/