model

October 6, 2023

Name: Kaushik Ilango
CWID: 20011241
Course: Natural Language Processing
Notes: I have used the data as is. That is I am going to predict the values in the range of(1-5) instead of binary classification as was in the assignment as a secondary suggestion.
I have tried to provide comments or reasons for writing certain logic but if I do miss anything please let me know during grading my assignment. Thanks.

First obviously we import some basic but necessary libraries for preprocessing and array manipulation.

[1]:
```
import numpy as np
import pandas as pd
from tqdm import tqdm
```

I am gonna use the pandas library to import the provide dataset into this program variable.

[2]:
```
amz_rev = pd.read_csv('amazon_reviews.csv')
```

Let's do some analysis of the product that was reviewed using some simple commands on the amz_rev dataset

[3]:
```
amz_rev.head()
```

[3]:
```
   overall                                         reviewText
0        4                                         No issues.
1        5  Purchased this for my device, it worked as adv…
2        4  it works as expected. I should have sprung for…
3        5  This think has worked out great.Had a diff. br…
4        5  Bought it with Retail Packaging, arrived legit…
```

As we can see there are just two columns (attributes) for each row so just the review statement and the rating. Now let's see the average rating of the product.

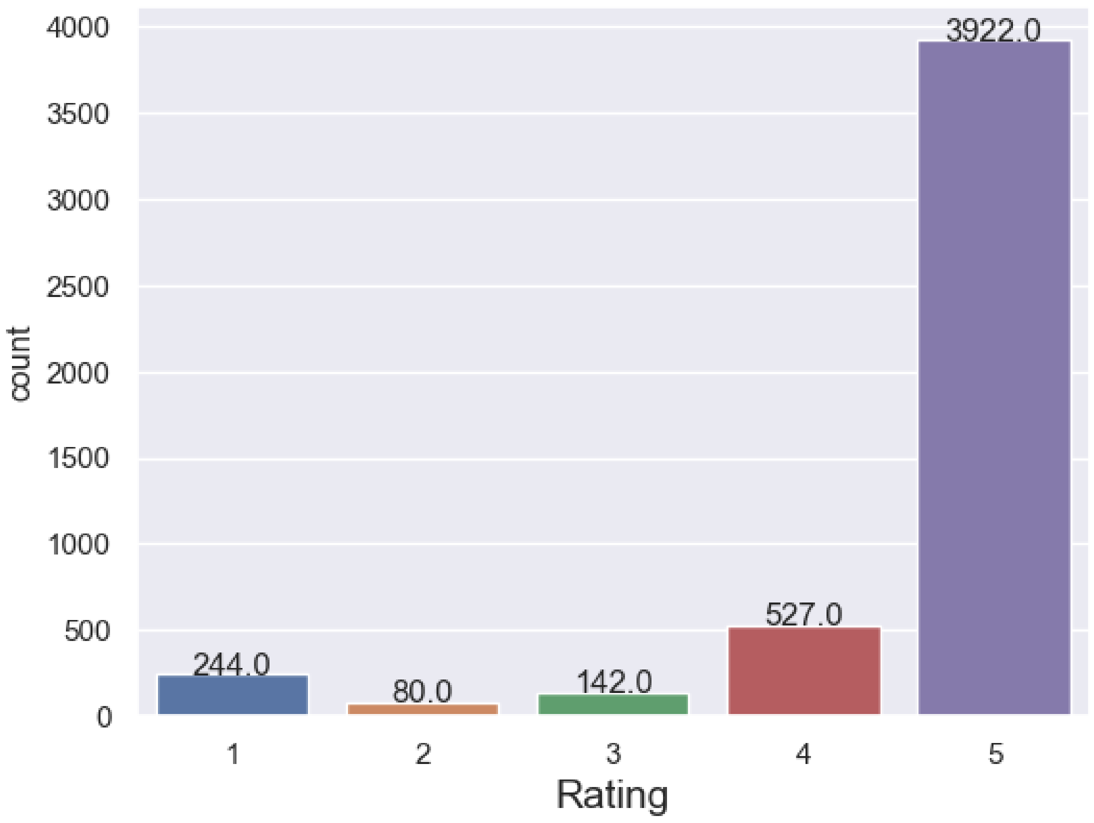
[4]:
```
sum(amz_rev['overall'])/len(amz_rev['overall'])
```

[4]:
```
4.587589013224822
```

4.5875 (approx 4.6). Which means that more than 90% liked product and provided positive reviews. Let's see what he number of each review rating.

[5]:
```
import seaborn as sns
import matplotlib.pyplot as plt
sns.set_theme(style="darkgrid")
def plot_count(y,xlabel):
    ax = sns.countplot(x = y)
    for p in ax.patches:
        ax.annotate(f'{p.get_height()}', (p.get_x() + p.get_width() / 2., p.
 ↪get_height()), ha='center', va='baseline')
    ax.set_xlabel(xlabel,fontsize=15)
```

[6]:
```
plot_count(amz_rev['overall'],'Rating')
plt.show()
```



Before we can retrieve more data let's tokenize the review statements so we can gather more insights. For this I am going to directly use the nltk.tokenize module and more specifically the RegexpTokenizer to make it more processable by the model.

[7]:
```
from nltk.tokenize import RegexpTokenizer
def setup_data(corpus):
    print("Cleaning data ... ")
    cleaned_corpus = []
    count = 0
    for doc in tqdm(corpus):
        tokenizer = RegexpTokenizer(r'\w+')
        cleaned_corpus.append(tokenizer.tokenize(doc.lower()))
        count = count + 1
    print(f"Done cleaning {count} data")
    return cleaned_corpus
```

Let's input our whole dataset statements into the new function and look at the output after which we can try to derive some insights. But before we do that we need to replace any missing value in the review statements to "NA" empty or space string. I already ran this once and got an error while trying to convert it to lower since **nan** is a float. I am not replacing it with " " since those will be removed and it will again become null string

[8]:
```
amz_rev['reviewText'] = amz_rev['reviewText'].replace(np.nan, 'NA', regex=True)
tokenized_data = setup_data(np.array(amz_rev['reviewText']))
```

```
Cleaning data …

100%|      | 4915/4915 [00:00<00:00, 30214.61it/s]

Done cleaning 4915 data
```

Now we can see the most max used words and least used words to draw some insights.

[9]:
```
def draw_insights(corpus):
    min = 0
    max = 0
    total = 0
    for doc in corpus:
        if len(doc) < min:
            min = len(doc)
        if len(doc) > max:
            max = len(doc)
        total = total + len(doc)
    avg = total/len(corpus)
    print(f"Min: {min}, Max: {max}, Avg: {avg}")
```

[10]:
```
draw_insights(tokenized_data)
```

```
Min: 0, Max: 1599, Avg: 52.11088504577823
```

The above results show that the users are dedicated to give a proper review of the product since the avg is at around 52 words which approximately is around 5 lines. Now let's split the data into three parts.

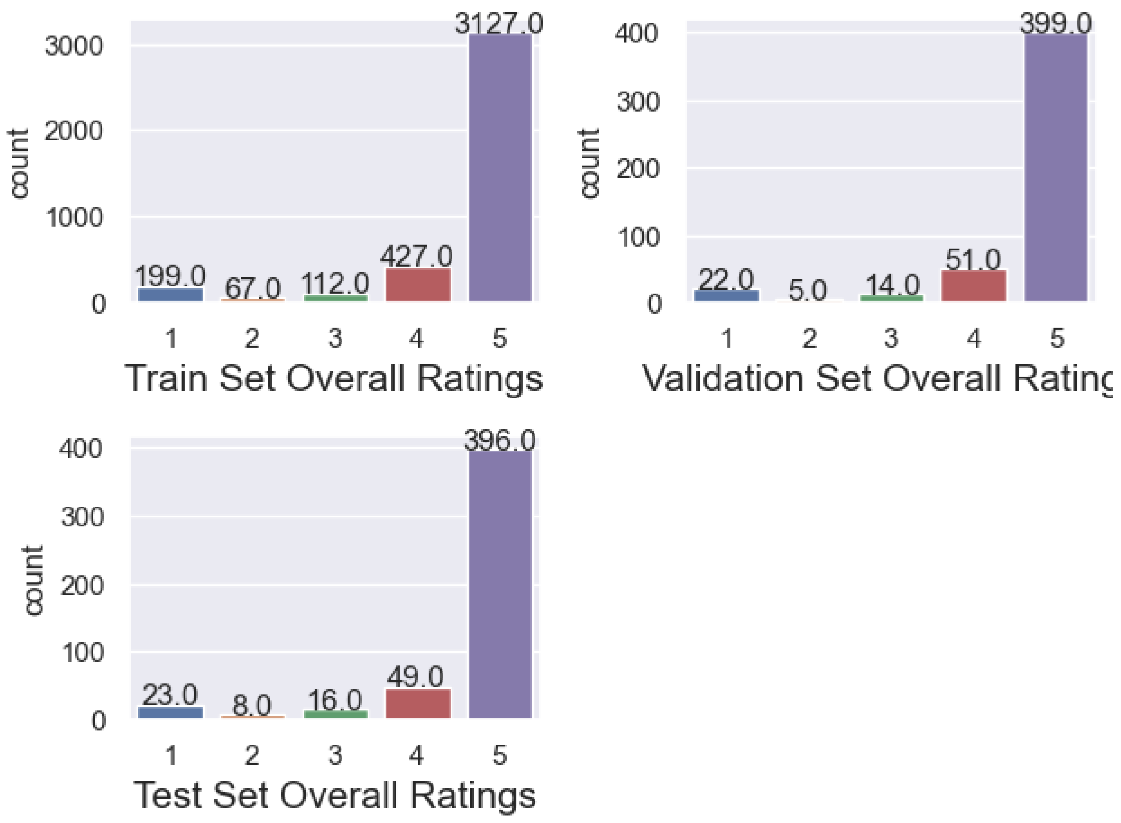
[11]:
```
from sklearn.model_selection import train_test_split
def train_test_val_split(X,y, test_size=0.1, val_size = 0.1,random_state=16):
    total_test_size = test_size + val_size
    X_train, X_t, y_train, y_t = train_test_split(X, y,␣
 ↪test_size=total_test_size, random_state=random_state)
    X_val, X_test, y_val, y_test = train_test_split(X_t, y_t,␣
 ↪test_size=test_size/total_test_size, random_state=random_state)
    return X_train, X_val, X_test, y_train, y_val, y_test
```

The above function is just a modified function of the usual train_test_split method in model_selection module which I have developed to provide the validation data set as well.

[12]:
```
X_train, X_val, X_test, y_train, y_val, y_test =␣
 ↪train_test_val_split(tokenized_data, np.array(amz_rev['overall']))
```

Let's run the analysis for the same Train, Test and Validation Data sets

[13]:
```
plt.subplot(2,2,1)
plot_count(y_train,xlabel="Train Set Overall Ratings")
plt.subplot(2,2,2)
plot_count(y_val,xlabel="Validation Set Overall Ratings")
plt.subplot(2,2,3)
plot_count(y_test,xlabel="Test Set Overall Ratings")
plt.tight_layout()
plt.show()
```



Now let's jump into the preprocessing and setting up our embeddings to develop a model. As we have already preprocessed our data using the `setup_data` funtion we can use the output to develop a model. Now let's develop a function for retrievinf the vocabulary. There are many ways to do it but I am following the assignment guidelines and using a list comprehension to complete the task.

[14]:
```
def get_vocab(corpus):
    vocab = []
    vocab = [x for line in corpus for x in line]
    vocab = list(set(vocab))
    vocab = sorted(vocab)
    return vocab
```

I have also sorted the words since it was asked to do so which I guess would help in word2index and embedding.

[15]:
```
vocab = get_vocab(tokenized_data)
```

Now let's further dive in and do the complicated part of computing the co-occurence matrix of the whole dataset which would allow us to embed the data and find similarities.

[16]:
```
def compute_co_occurence_matrix(corpus,window_size =4):
    word2index = {}
    v = get_vocab(corpus)
    for i in v:
        word2index[i] = v.index(i)
    print(f"Number of unique words {len(v)}")
    x = len(v)
    M = np.zeros((x,x),dtype = 'float32')
    print(M.shape)
    print("Computing pairs...")
    for review in tqdm(corpus):
        for i in range(len(review)):
            for j in range(max(0,i-window_size),min(i+window_size,len(review))):
                if i!=j:
                    M[word2index[review[i]],word2index[review[j]]] += 1
    return M,word2index
```

[17]:
```
M,word2index = compute_co_occurence_matrix(tokenized_data)
```

```
Number of unique words 8516
(8516, 8516)
Computing pairs…

100%|      | 4915/4915 [00:08<00:00, 577.25it/s]
```

Now we have successfully implemented the co-occurence matrix. Now let's try to reduce the dimensions using SVD in specific as mentioned in the assignment we have to use TruncatedSVD.

FYI, I have used float in the Matrix because its easier for numpy to process and decompose it to lower dimensions than working with integers

```
[18]: from sklearn.decomposition import TruncatedSVD
      def reduce_to_k_dim(M,k=2):
          print(f"Reducing to {k} dimensions...")
          svd = TruncatedSVD(n_components=k, n_iter=10, random_state=16)
          M_reduced = svd.fit_transform(M)
          return M_reduced
```

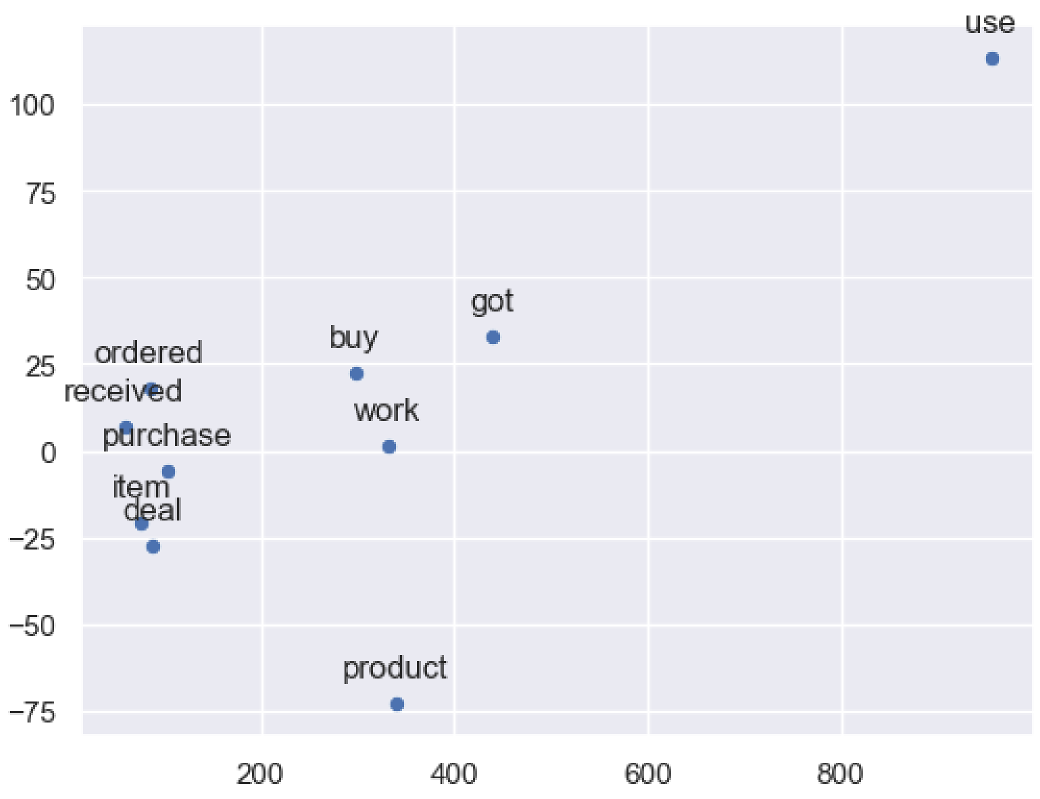
```
[19]: M_reduced = reduce_to_k_dim(M)
```

```
Reducing to 2 dimensions…
```

As mentioned in the assignment we can plot the words using the indices an the 2-d matrix which will provide us with the (x,y) co-ordinates.

```
[20]: def plot_embeddings(M_reduced,word2index,words_to_plot):
          x = []
          y = []
          for word in words_to_plot:
              index = word2index[word]
              x.append(M_reduced[index,0])
              y.append(M_reduced[index,1])
          labels = words_to_plot
          sns.scatterplot(x =x,y = y)
          plt.grid(True)
          for i, label in enumerate(labels):
              plt.annotate(label, (x[i], y[i]), textcoords="offset points",␣
       ↪xytext=(0,10), ha='center')
```

```
[21]: plot_embeddings(M_reduced,word2index,words_to_plot=["purchase", "buy", "work",␣
       ↪"got", "ordered", "received", "product", "item", "deal", "use"])
```



Now let's use the pretrained GloVe embeddings to plot the same above graph. For this we use th predefined definitions as mentioned in the assignment. I am just going to copy paste them in the following cells

[22]:
```
import gensim.downloader as api
def load_embedding_model():
    wv_from_bin = api.load("glove-wiki-gigaword-200")
    print("Loaded vocab size %i" % len(list(wv_from_bin.index_to_key)))
    return wv_from_bin
```

[23]:
```
wv_from_bin = load_embedding_model()
```

```
Loaded vocab size 400000
```

[24]:
```
import random
def get_matrix_of_vectors(wv_from_bin, required_words):

    words = list(wv_from_bin.index_to_key)
    print("Shuffling words ...")
    random.seed(225)
    random.shuffle(words)
    words = words[:10000]
    print("Putting %i words into word2ind and matrix M..." % len(words))
    word2ind = {}
    M = []
    curInd = 0
    for w in words:
        try:
            M.append(wv_from_bin.get_vector(w))
            word2ind[w] = curInd
            curInd += 1
        except KeyError:
            continue
    for w in required_words:
        if w in words:
            continue
        try:
            M.append(wv_from_bin.get_vector(w))
            word2ind[w] = curInd
            curInd += 1
        except KeyError:
            continue
    M = np.stack(M)
    print("Done.")
    return M, word2ind
```

Now let's get the matrix and the word2index for this model in `M2` and `word2index2` variables

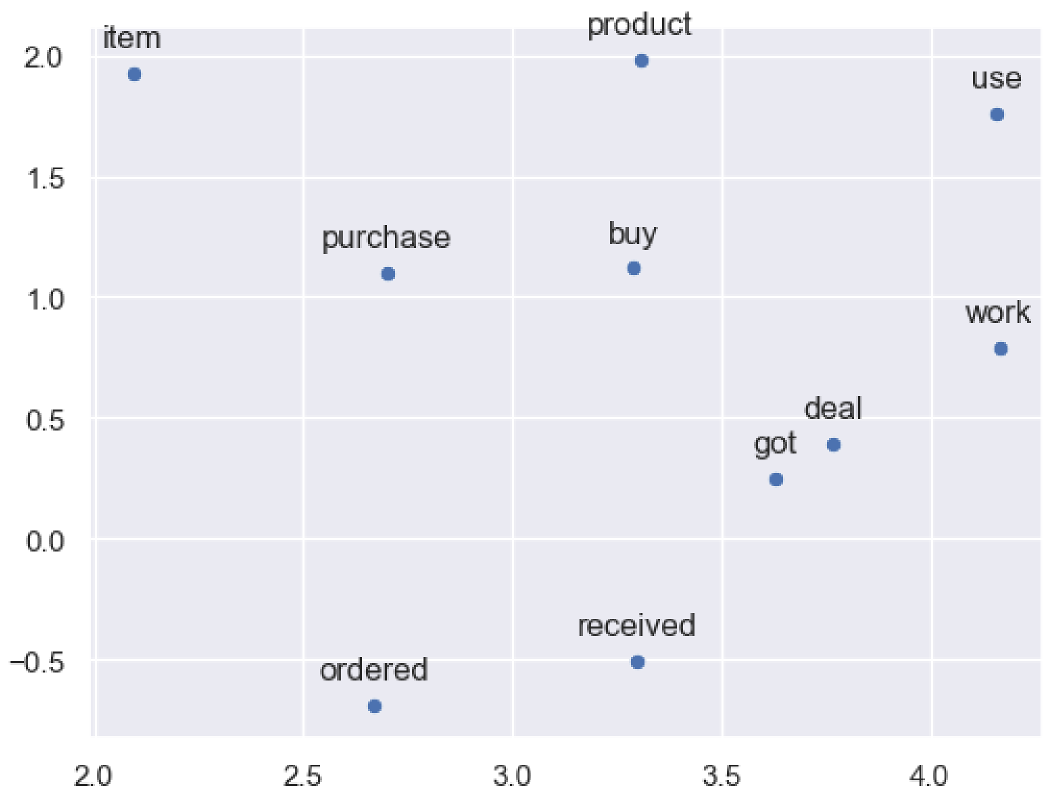
[25]:
```
M2,word2index2 = get_matrix_of_vectors(wv_from_bin, vocab)
```

```
Shuffling words …
Putting 10000 words into word2ind and matrix M…
Done.
```

Let's do the same action of reducing the dimensions using our functions and plotting them like we did earlier

[26]:
```
M2_reduced = reduce_to_k_dim(M2)
plot_embeddings(M2_reduced,word2index2,words_to_plot=["purchase", "buy",␣
 ↪"work", "got", "ordered", "received", "product", "item", "deal", "use"])
```

```
Reducing to 2 dimensions…
```



As we can see that this model is not context based whereas our earlier embeddings is context based that is the model was developed on the specific data such as in our case the review dataset so the words had some context before they were plotted and hence it showed up in the plot. Here only the most similar words close to each other even if you consider them close otherwise they are scattered all over the place in this plot. For example 'buy' and 'purchase' have to b as close as possible but they are close but not as closely related as in our context basewd embeddings.

Now let's write a definition for computing the average of each word in the review to get a review embedding

[27]:
```
def review_embedding(M,word2index,tokenized_reviews):
    print(M_128.shape)
    review_embeddings = []
    embeddings_shape = []
    print("Computing review embeddings...")
    for review in tqdm(tokenized_reviews):
        embedding = np.mean([M[word2index[w]] for w in review],axis = 0)
        embeddings_shape.append(embedding.shape)
        review_embeddings.append(embedding)
    f = [x for x in embeddings_shape if x != (128,)]
    if f:
        print(f"Warning: {len(f)} review embeddings have shape other than␣
↪(128,)")
    else:
        print("All review embeddings have shape (128,)")
    print("Done.")
    return review_embeddings
```

Now let's use our embeddings model (co-occurence matrix) to develop a predictive model for the given data set. Let's reduce the dimensions of the matrix M to 128 dimensions to provide more information while still optimizing our run time. I am printing out the shapes just for deubbing since I was getting some errors. But it shows that the each word of shape 128 has been averaged in the review.

[28]:
```
M_128 = reduce_to_k_dim(M,k=128)
review_embeddings = review_embedding(M_128,word2index,tokenized_data)
```

```
Reducing to 128 dimensions…
(8516, 128)
Computing review embeddings…

100%|      | 4915/4915 [00:00<00:00, 13117.32it/s]

All review embeddings have shape (128,)
Done.
```

Converting it into a numpy array to make it faster for computation.

[29]:
```
review_embeddings = np.array(review_embeddings)
review_embeddings.shape
```

[29]:
```
(4915, 128)
```

Now this is our feature dataset which we can use to develop models

[30]:
```
X_train, X_val, X_test, y_train, y_val, y_test =␣
↪train_test_val_split(review_embeddings, np.array(amz_rev['overall']))
```

Let's first build a logistic regression model using scikit library. Since this is multilabel (1-5) we need to set some parameters in the regressor.

[31]:
```
from sklearn.preprocessing import StandardScaler

scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_val = scaler.transform(X_val)
X_test = scaler.transform(X_test)
```

[32]:
```
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score

regressor = LogisticRegression(multi_class =␣
 ↪'multinomial',random_state=16,max_iter=1000,penalty = 'l2',C = 1.0)
regressor.fit(X_train,y_train)
```

[32]:
```
LogisticRegression(max_iter=1000, multi_class='multinomial', random_state=16)
```

In the above code I set some max_iter values because I was getting an warning that the convergence failed due to less iterations, but that still didnt reduce the warnings so I used standard scaler to scale the values and standardize the data set.

[33]:
```
y_val_pred = regressor.predict(X_val)
accuracy_score(y_val,y_val_pred)
```

[33]:
```
0.8105906313645621
```

[34]:
```
y_test_pred = regressor.predict(X_test)
accuracy_score(y_test,y_test_pred)
```

[34]:
```
0.8130081300813008
```

Getting a pretty good score of 81.3% using logistic regression. Let's try to implement using the Neural Network model.

[35]:
```
import tensorflow as tf
from tensorflow import keras
from tensorflow.keras import layers
from tensorflow.keras import Sequential

model = Sequential(layers = [
    layers.Input(shape = (128,)),
    layers.Dense(64,activation = 'relu'),
    layers.Dense(32,activation = 'relu'),
    layers.Dense(16,activation = 'relu'),
    layers.Dense(6,activation = 'softmax')

])

model.compile(optimizer = 'sgd',loss =␣
 ↪'sparse_categorical_crossentropy',metrics = ['accuracy'])
model.summary()
```

```
Model: "sequential"

_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense (Dense)               (None, 64)                8256

 dense_1 (Dense)             (None, 32)                2080

 dense_2 (Dense)             (None, 16)                528

 dense_3 (Dense)             (None, 6)                 102

=================================================================
Total params: 10966 (42.84 KB)
Trainable params: 10966 (42.84 KB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

[36]:
```
history = model.fit(X_train,y_train,epochs = 35,validation_data = (X_val,y_val))
```

```
Epoch 1/35
123/123 [==============================] - 1s 4ms/step - loss: 0.9496 -
accuracy: 0.7510 - val_loss: 0.7457 - val_accuracy: 0.8126
Epoch 2/35
123/123 [==============================] - 0s 2ms/step - loss: 0.7466 -
accuracy: 0.7953 - val_loss: 0.6961 - val_accuracy: 0.8126
Epoch 3/35
123/123 [==============================] - 0s 2ms/step - loss: 0.7191 -
accuracy: 0.7953 - val_loss: 0.6795 - val_accuracy: 0.8126
Epoch 4/35
123/123 [==============================] - 0s 2ms/step - loss: 0.7032 -
accuracy: 0.7953 - val_loss: 0.6646 - val_accuracy: 0.8126
Epoch 5/35
123/123 [==============================] - 0s 2ms/step - loss: 0.6907 -
accuracy: 0.7953 - val_loss: 0.6546 - val_accuracy: 0.8126
Epoch 6/35
123/123 [==============================] - 0s 2ms/step - loss: 0.6798 -
accuracy: 0.7953 - val_loss: 0.6485 - val_accuracy: 0.8126
Epoch 7/35
123/123 [==============================] - 0s 1ms/step - loss: 0.6697 -
accuracy: 0.7953 - val_loss: 0.6425 - val_accuracy: 0.8126
Epoch 8/35
123/123 [==============================] - 0s 1ms/step - loss: 0.6604 -
accuracy: 0.7953 - val_loss: 0.6369 - val_accuracy: 0.8126
Epoch 9/35
123/123 [==============================] - 0s 2ms/step - loss: 0.6509 -
accuracy: 0.7955 - val_loss: 0.6312 - val_accuracy: 0.8126
Epoch 10/35
123/123 [==============================] - 0s 2ms/step - loss: 0.6404 -
accuracy: 0.7958 - val_loss: 0.6252 - val_accuracy: 0.8126
Epoch 11/35
123/123 [==============================] - 0s 1ms/step - loss: 0.6322 -
```

```
accuracy: 0.7965 - val_loss: 0.6214 - val_accuracy: 0.8126
Epoch 12/35
123/123 [==============================] - 0s 2ms/step - loss: 0.6236 -
accuracy: 0.7973 - val_loss: 0.6166 - val_accuracy: 0.8126
Epoch 13/35
123/123 [==============================] - 0s 2ms/step - loss: 0.6149 -
accuracy: 0.7973 - val_loss: 0.6119 - val_accuracy: 0.8126
Epoch 14/35
123/123 [==============================] - 0s 2ms/step - loss: 0.6066 -
accuracy: 0.7986 - val_loss: 0.6088 - val_accuracy: 0.8147
Epoch 15/35
123/123 [==============================] - 0s 2ms/step - loss: 0.5989 -
accuracy: 0.7996 - val_loss: 0.6052 - val_accuracy: 0.8167
Epoch 16/35
123/123 [==============================] - 0s 2ms/step - loss: 0.5916 -
accuracy: 0.8001 - val_loss: 0.6021 - val_accuracy: 0.8187
Epoch 17/35
123/123 [==============================] - 0s 2ms/step - loss: 0.5843 -
accuracy: 0.8021 - val_loss: 0.5991 - val_accuracy: 0.8187
Epoch 18/35
123/123 [==============================] - 0s 1ms/step - loss: 0.5768 -
accuracy: 0.8047 - val_loss: 0.5972 - val_accuracy: 0.8228
Epoch 19/35
123/123 [==============================] - 0s 1ms/step - loss: 0.5694 -
accuracy: 0.8060 - val_loss: 0.5949 - val_accuracy: 0.8248
Epoch 20/35
123/123 [==============================] - 0s 1ms/step - loss: 0.5625 -
accuracy: 0.8080 - val_loss: 0.5925 - val_accuracy: 0.8228
Epoch 21/35
123/123 [==============================] - 0s 1ms/step - loss: 0.5552 -
accuracy: 0.8090 - val_loss: 0.5921 - val_accuracy: 0.8248
Epoch 22/35
123/123 [==============================] - 0s 2ms/step - loss: 0.5483 -
accuracy: 0.8103 - val_loss: 0.5906 - val_accuracy: 0.8167
Epoch 23/35
123/123 [==============================] - 0s 1ms/step - loss: 0.5414 -
accuracy: 0.8136 - val_loss: 0.5884 - val_accuracy: 0.8167
Epoch 24/35
123/123 [==============================] - 0s 2ms/step - loss: 0.5334 -
accuracy: 0.8136 - val_loss: 0.5918 - val_accuracy: 0.8228
Epoch 25/35
123/123 [==============================] - 0s 1ms/step - loss: 0.5277 -
accuracy: 0.8149 - val_loss: 0.5883 - val_accuracy: 0.8147
Epoch 26/35
123/123 [==============================] - 0s 1ms/step - loss: 0.5204 -
accuracy: 0.8169 - val_loss: 0.5875 - val_accuracy: 0.8167
Epoch 27/35
123/123 [==============================] - 0s 1ms/step - loss: 0.5138 -
```

```
accuracy: 0.8184 - val_loss: 0.5879 - val_accuracy: 0.8187
Epoch 28/35
123/123 [==============================] - 0s 1ms/step - loss: 0.5069 -
accuracy: 0.8215 - val_loss: 0.5881 - val_accuracy: 0.8187
Epoch 29/35
123/123 [==============================] - 0s 1ms/step - loss: 0.4998 -
accuracy: 0.8217 - val_loss: 0.5901 - val_accuracy: 0.8167
Epoch 30/35
123/123 [==============================] - 0s 1ms/step - loss: 0.4931 -
accuracy: 0.8250 - val_loss: 0.5909 - val_accuracy: 0.8167
Epoch 31/35
123/123 [==============================] - 0s 2ms/step - loss: 0.4859 -
accuracy: 0.8268 - val_loss: 0.5926 - val_accuracy: 0.8167
Epoch 32/35
123/123 [==============================] - 0s 2ms/step - loss: 0.4792 -
accuracy: 0.8299 - val_loss: 0.5948 - val_accuracy: 0.8208
Epoch 33/35
123/123 [==============================] - 0s 2ms/step - loss: 0.4723 -
accuracy: 0.8304 - val_loss: 0.5957 - val_accuracy: 0.8167
Epoch 34/35
123/123 [==============================] - 0s 2ms/step - loss: 0.4651 -
accuracy: 0.8324 - val_loss: 0.5983 - val_accuracy: 0.8147
Epoch 35/35
123/123 [==============================] - 0s 2ms/step - loss: 0.4585 -
accuracy: 0.8349 - val_loss: 0.6015 - val_accuracy: 0.8167
```

[37]:
```
model.evaluate(X_test,y_test)
```

```
16/16 [==============================] - 0s 1ms/step - loss: 0.6411 - accuracy:
0.8150
```

[37]:
```
[0.6410635113716125, 0.815040647983551]
```

After tuning the parameters I found that the best accuracy for the test without overfitting the data was 81.5 % which is almost the same as the logistic regression. My analysis is that running the newural network for long number of epcohs leads to overfitting of the validation set and doesnt produce the best result in test cases. This is common among the text datasets since once the model trains on the constant dataset it is difficult to understand the context of new texts and hence the predictions go wrong. The neural networks perform better than the logistic regresiion model, but i believe the nural network ca be modified more to add regularization to get a better score than the current score.

I tried adding dropout layers and regularizations in the model. Just a trial.

[38]:
```
model_1 = Sequential(layers = [
    layers.Input(shape = (128,)),
    layers.Dense(64,activation = 'relu'),
    layers.Dropout(0.2),
    layers.Dense(32,activation = 'relu'),
    layers.Dropout(0.2),
    layers.Dense(16,activation = 'relu'),
    layers.Dense(6,activation = 'softmax')

])
model_1.compile(optimizer = 'adagrad',loss =␣
 ↪'sparse_categorical_crossentropy',metrics = ['accuracy'])
model_1.summary()
```

```
Model: "sequential_1"

_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense_4 (Dense)             (None, 64)                8256

 dropout (Dropout)           (None, 64)                0

 dense_5 (Dense)             (None, 32)                2080

 dropout_1 (Dropout)         (None, 32)                0

 dense_6 (Dense)             (None, 16)                528

 dense_7 (Dense)             (None, 6)                 102

=================================================================
Total params: 10966 (42.84 KB)
Trainable params: 10966 (42.84 KB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

[39]:
```
history_1 = model_1.fit(X_train,y_train,epochs = 35,validation_data =␣
 ↪(X_val,y_val))
```

```
Epoch 1/35
123/123 [==============================] - 1s 2ms/step - loss: 1.3363 -
accuracy: 0.6473 - val_loss: 1.2007 - val_accuracy: 0.7963
Epoch 2/35
123/123 [==============================] - 0s 2ms/step - loss: 1.1400 -
accuracy: 0.7709 - val_loss: 1.0246 - val_accuracy: 0.8106
Epoch 3/35
123/123 [==============================] - 0s 1ms/step - loss: 1.0292 -
accuracy: 0.7897 - val_loss: 0.9235 - val_accuracy: 0.8126
Epoch 4/35
123/123 [==============================] - 0s 1ms/step - loss: 0.9644 -
accuracy: 0.7930 - val_loss: 0.8632 - val_accuracy: 0.8126
```

```
Epoch 5/35
123/123 [==============================] - 0s 1ms/step - loss: 0.9330 -
accuracy: 0.7935 - val_loss: 0.8264 - val_accuracy: 0.8126
Epoch 6/35
123/123 [==============================] - 0s 1ms/step - loss: 0.9070 -
accuracy: 0.7950 - val_loss: 0.8007 - val_accuracy: 0.8126
Epoch 7/35
123/123 [==============================] - 0s 1ms/step - loss: 0.8942 -
accuracy: 0.7950 - val_loss: 0.7826 - val_accuracy: 0.8126
Epoch 8/35
123/123 [==============================] - 0s 1ms/step - loss: 0.8592 -
accuracy: 0.7953 - val_loss: 0.7676 - val_accuracy: 0.8126
Epoch 9/35
123/123 [==============================] - 0s 2ms/step - loss: 0.8375 -
accuracy: 0.7953 - val_loss: 0.7560 - val_accuracy: 0.8126
Epoch 10/35
123/123 [==============================] - 0s 2ms/step - loss: 0.8475 -
accuracy: 0.7948 - val_loss: 0.7464 - val_accuracy: 0.8126
Epoch 11/35
123/123 [==============================] - 0s 2ms/step - loss: 0.8388 -
accuracy: 0.7950 - val_loss: 0.7391 - val_accuracy: 0.8126
Epoch 12/35
123/123 [==============================] - 0s 1ms/step - loss: 0.8346 -
accuracy: 0.7953 - val_loss: 0.7323 - val_accuracy: 0.8126
Epoch 13/35
123/123 [==============================] - 0s 2ms/step - loss: 0.8221 -
accuracy: 0.7953 - val_loss: 0.7260 - val_accuracy: 0.8126
Epoch 14/35
123/123 [==============================] - 0s 2ms/step - loss: 0.8162 -
accuracy: 0.7953 - val_loss: 0.7206 - val_accuracy: 0.8126
Epoch 15/35
123/123 [==============================] - 0s 2ms/step - loss: 0.8099 -
accuracy: 0.7950 - val_loss: 0.7159 - val_accuracy: 0.8126
Epoch 16/35
123/123 [==============================] - 0s 2ms/step - loss: 0.8106 -
accuracy: 0.7953 - val_loss: 0.7117 - val_accuracy: 0.8126
Epoch 17/35
123/123 [==============================] - 0s 2ms/step - loss: 0.8071 -
accuracy: 0.7950 - val_loss: 0.7076 - val_accuracy: 0.8126
Epoch 18/35
123/123 [==============================] - 0s 2ms/step - loss: 0.8053 -
accuracy: 0.7953 - val_loss: 0.7041 - val_accuracy: 0.8126
Epoch 19/35
123/123 [==============================] - 0s 2ms/step - loss: 0.7905 -
accuracy: 0.7953 - val_loss: 0.7007 - val_accuracy: 0.8126
Epoch 20/35
123/123 [==============================] - 0s 2ms/step - loss: 0.7891 -
accuracy: 0.7953 - val_loss: 0.6975 - val_accuracy: 0.8126
```

```
Epoch 21/35
123/123 [==============================] - 0s 2ms/step - loss: 0.7816 -
accuracy: 0.7953 - val_loss: 0.6946 - val_accuracy: 0.8126
Epoch 22/35
123/123 [==============================] - 0s 2ms/step - loss: 0.7752 -
accuracy: 0.7953 - val_loss: 0.6917 - val_accuracy: 0.8126
Epoch 23/35
123/123 [==============================] - 0s 2ms/step - loss: 0.7785 -
accuracy: 0.7953 - val_loss: 0.6891 - val_accuracy: 0.8126
Epoch 24/35
123/123 [==============================] - 0s 2ms/step - loss: 0.7699 -
accuracy: 0.7953 - val_loss: 0.6866 - val_accuracy: 0.8126
Epoch 25/35
123/123 [==============================] - 0s 2ms/step - loss: 0.7704 -
accuracy: 0.7953 - val_loss: 0.6842 - val_accuracy: 0.8126
Epoch 26/35
123/123 [==============================] - 0s 2ms/step - loss: 0.7682 -
accuracy: 0.7953 - val_loss: 0.6821 - val_accuracy: 0.8126
Epoch 27/35
123/123 [==============================] - 0s 2ms/step - loss: 0.7670 -
accuracy: 0.7953 - val_loss: 0.6802 - val_accuracy: 0.8126
Epoch 28/35
123/123 [==============================] - 0s 2ms/step - loss: 0.7702 -
accuracy: 0.7953 - val_loss: 0.6781 - val_accuracy: 0.8126
Epoch 29/35
123/123 [==============================] - 0s 2ms/step - loss: 0.7528 -
accuracy: 0.7953 - val_loss: 0.6762 - val_accuracy: 0.8126
Epoch 30/35
123/123 [==============================] - 0s 2ms/step - loss: 0.7652 -
accuracy: 0.7953 - val_loss: 0.6747 - val_accuracy: 0.8126
Epoch 31/35
123/123 [==============================] - 0s 2ms/step - loss: 0.7522 -
accuracy: 0.7953 - val_loss: 0.6731 - val_accuracy: 0.8126
Epoch 32/35
123/123 [==============================] - 0s 2ms/step - loss: 0.7579 -
accuracy: 0.7953 - val_loss: 0.6716 - val_accuracy: 0.8126
Epoch 33/35
123/123 [==============================] - 0s 2ms/step - loss: 0.7536 -
accuracy: 0.7953 - val_loss: 0.6703 - val_accuracy: 0.8126
Epoch 34/35
123/123 [==============================] - 0s 2ms/step - loss: 0.7528 -
accuracy: 0.7953 - val_loss: 0.6690 - val_accuracy: 0.8126
Epoch 35/35
123/123 [==============================] - 0s 2ms/step - loss: 0.7445 -
accuracy: 0.7953 - val_loss: 0.6676 - val_accuracy: 0.8126
```

[40]:
```
model_1.evaluate(X_test,y_test)
```

```
16/16 [==============================] - 0s 1ms/step - loss: 0.6815 - accuracy:
0.8049
```

[40]:
```
[0.6815322041511536, 0.8048780560493469]
```

[41]:
```
!jupyter nbconvert --to pdf model.ipynb
```

```
[NbConvertApp] Converting notebook model.ipynb to pdf
[NbConvertApp] Support files will be in model_files\
[NbConvertApp] Making directory .\model_files
[NbConvertApp] Writing 65307 bytes to notebook.tex
[NbConvertApp] Building PDF
[NbConvertApp] Running xelatex 3 times: ['xelatex', 'notebook.tex', '-quiet']
[NbConvertApp] CRITICAL | x failed: xelatex notebook.tex -quiet
b"This is XeTeX, Version 3.141592653-2.6-0.999995 (TeX Live 2023) (preloaded
format=xelatex)\r\n restricted \\write18 enabled.\r\nentering extended
mode\r\n(./notebook.tex\r\nLaTeX2e <2023-06-01> patch level 1\r\nL3 programming
layer <2023-08-29>\r\n(c:/Users/kilan/AppData/Roaming/TinyTeX/texmf-
dist/tex/latex/base/article.cls\r\nDocument Class: article 2023/05/17 v1.4n
Standard LaTeX document class\r\n(c:/Users/kilan/AppData/Roaming/TinyTeX/texmf-
dist/tex/latex/base/size11.clo)) (c:/Users/kilan/AppData/Roaming/TinyTeX/texmf-
dist/tex/latex/tcolorbox/tcolorbox.sty\r\n\r\n! LaTeX Error: File `pgf.sty' not
found.\r\n\r\nType X to quit or <RETURN> to proceed,\r\nor enter new name.
(Default extension: sty)\r\n\r\nEnter file name: \r\n! Emergency stop.\r\n<read
*> \r\n         \r\nl.24 \\RequirePackage{pgf}[2008/01/15]\r\n
^^M\r\nNo pages of output.\r\nTranscript written on notebook.log.\r\n"
Traceback (most recent call last):
  File "C:\Program Files\WindowsApps\PythonSoftwareFoundation.Python.3.8_3.8.280
0.0_x64__qbz5n2kfra8p0\lib\runpy.py", line 194, in _run_module_as_main
    return _run_code(code, main_globals, None,
  File "C:\Program Files\WindowsApps\PythonSoftwareFoundation.Python.3.8_3.8.280
0.0_x64__qbz5n2kfra8p0\lib\runpy.py", line 87, in _run_code
    exec(code, run_globals)
  File "c:\Users\kilan\AppData\Local\Packages\PythonSoftwareFoundation.Python.3.
8_qbz5n2kfra8p0\LocalCache\local-packages\Python38\Scripts\jupyter-
nbconvert.EXE\__main__.py", line 7, in <module>
  File "C:\Users\kilan\AppData\Local\Packages\PythonSoftwareFoundation.Python.3.
8_qbz5n2kfra8p0\LocalCache\local-packages\Python38\site-
packages\jupyter_core\application.py", line 285, in launch_instance
    return super().launch_instance(argv=argv, **kwargs)
  File "C:\Users\kilan\AppData\Local\Packages\PythonSoftwareFoundation.Python.3.
8_qbz5n2kfra8p0\LocalCache\local-packages\Python38\site-
packages\traitlets\config\application.py", line 1043, in launch_instance
    app.start()
  File "C:\Users\kilan\AppData\Local\Packages\PythonSoftwareFoundation.Python.3.
8_qbz5n2kfra8p0\LocalCache\local-packages\Python38\site-
packages\nbconvert\nbconvertapp.py", line 410, in start
    self.convert_notebooks()
  File "C:\Users\kilan\AppData\Local\Packages\PythonSoftwareFoundation.Python.3.
8_qbz5n2kfra8p0\LocalCache\local-packages\Python38\site-
packages\nbconvert\nbconvertapp.py", line 585, in convert_notebooks
    self.convert_single_notebook(notebook_filename)
  File "C:\Users\kilan\AppData\Local\Packages\PythonSoftwareFoundation.Python.3.
8_qbz5n2kfra8p0\LocalCache\local-packages\Python38\site-
packages\nbconvert\nbconvertapp.py", line 551, in convert_single_notebook
    output, resources = self.export_single_notebook(
  File "C:\Users\kilan\AppData\Local\Packages\PythonSoftwareFoundation.Python.3.
8_qbz5n2kfra8p0\LocalCache\local-packages\Python38\site-
packages\nbconvert\nbconvertapp.py", line 477, in export_single_notebook
    output, resources = self.exporter.from_filename(
  File "C:\Users\kilan\AppData\Local\Packages\PythonSoftwareFoundation.Python.3.
8_qbz5n2kfra8p0\LocalCache\local-packages\Python38\site-
packages\nbconvert\exporters\templateexporter.py", line 389, in from_filename
    return super().from_filename(filename, resources, **kw)  # type:ignore
  File "C:\Users\kilan\AppData\Local\Packages\PythonSoftwareFoundation.Python.3.
8_qbz5n2kfra8p0\LocalCache\local-packages\Python38\site-
packages\nbconvert\exporters\exporter.py", line 201, in from_filename
    return self.from_file(f, resources=resources, **kw)
  File "C:\Users\kilan\AppData\Local\Packages\PythonSoftwareFoundation.Python.3.
8_qbz5n2kfra8p0\LocalCache\local-packages\Python38\site-
packages\nbconvert\exporters\templateexporter.py", line 395, in from_file
    return super().from_file(file_stream, resources, **kw)  # type:ignore
  File "C:\Users\kilan\AppData\Local\Packages\PythonSoftwareFoundation.Python.3.
8_qbz5n2kfra8p0\LocalCache\local-packages\Python38\site-
packages\nbconvert\exporters\exporter.py", line 220, in from_file
    return self.from_notebook_node(
  File "C:\Users\kilan\AppData\Local\Packages\PythonSoftwareFoundation.Python.3.
8_qbz5n2kfra8p0\LocalCache\local-packages\Python38\site-
packages\nbconvert\exporters\pdf.py", line 199, in from_notebook_node
    self.run_latex(tex_file)
  File "C:\Users\kilan\AppData\Local\Packages\PythonSoftwareFoundation.Python.3.
8_qbz5n2kfra8p0\LocalCache\local-packages\Python38\site-
packages\nbconvert\exporters\pdf.py", line 168, in run_latex
    return self.run_command(
  File "C:\Users\kilan\AppData\Local\Packages\PythonSoftwareFoundation.Python.3.
8_qbz5n2kfra8p0\LocalCache\local-packages\Python38\site-
packages\nbconvert\exporters\pdf.py", line 158, in run_command
    raise raise_on_failure(msg)
nbconvert.exporters.pdf.LatexFailed: PDF creating failed, captured latex output:
Failed to run "xelatex notebook.tex -quiet" command:
This is XeTeX, Version 3.141592653-2.6-0.999995 (TeX Live 2023) (preloaded
format=xelatex)

 restricted \write18 enabled.

entering extended mode
```

```
(./notebook.tex

LaTeX2e <2023-06-01> patch level 1

L3 programming layer <2023-08-29>

(c:/Users/kilan/AppData/Roaming/TinyTeX/texmf-dist/tex/latex/base/article.cls

Document Class: article 2023/05/17 v1.4n Standard LaTeX document class

(c:/Users/kilan/AppData/Roaming/TinyTeX/texmf-dist/tex/latex/base/size11.clo))
(c:/Users/kilan/AppData/Roaming/TinyTeX/texmf-
dist/tex/latex/tcolorbox/tcolorbox.sty



! LaTeX Error: File `pgf.sty' not found.



Type X to quit or <RETURN> to proceed,

or enter new name. (Default extension: sty)



Enter file name:

! Emergency stop.

<read *>



l.24 \RequirePackage{pgf}[2008/01/15]

                                     ^^M

No pages of output.

Transcript written on notebook.log.
```