Artificial intelligence
final project: sentiment classification of user review dataset

Tian Xiaoyang
26001904581

***Abstract*—Sentiment analysis, as a sub-field of AI research, is rising in popularity throughout recent years. Tech companies are competing to build faster and stronger AI models/algorithms to learn about their customers better so they can provide better services. Application of sentiment analysis ranges from online media websites like Netflix, YouTube, social media like Twitter and Reddit to e-shopping sites like Amazon and Rakuten.**

This is the report of the final project for the artificial intelligence course. In this project, 2 different machine learning algorithms will be discussed, Naïve Bayes and BERT, and they will each be used to build a sentiment classifier based on a dataset of text reviews of various items.

The 2 models will be used to classify the sentiments from the dataset. The dataset contains 4000 entries of reviews and ratings on various items, there are 2 columns, one for the text reviews and the other for numerical ratings, 1 for positive reviews and 0 for negative reviews. The 2 sentiment classifiers will be compared to see the performances of these 2 different models.

Introduction

1. Overview of sentiment analysis

Sentiment classification, also known as sentiment analysis, is the use of natural language processing, text analysis, computational linguistics, etc. to identify and take useful information from text documents systematically and efficiently. It is widely used by modern day websites like Amazon, YouTube, Netflix on their "voice by customer" materials such as reviews and surveys, to build a better recommender system and provide better services and products.

Sentiment analysis involves natural language processing (NLP), text analysis, etc. Natural language processing is a field of study that focuses on computational interactions with human languages. The end goal of the study is for computer programs to understand human languages and take actions accordingly to assist human tasks in other ways. Text analysis, also known as text mining, is the process of acquiring data which possess superior information,

just like the process of purifying a mineral. Computer programs are developed and used to "mine" and "filter" useful data with more information from documents.

There are currently 3 main categories of methods for solving sentiment classification problems, knowledge-based methods, statistical methods, and hybrid methods.

Knowledge based method classifies texts by affect categories from words with direct, obvious sentimental presence in different polarities. E.g., "pleased" is obviously a word that expresses positive emotions while words like "painful" or "angry" apparently deliver negative emotions.

Statistical methods utilize machine learning models such as support vector machines, latent semantic analysis, and deep learning algorithms. Latent semantic analysis is the technique in NLP when a program is created and used to infer a "latent" relationship between a document and the words it contains. It builds up a logical connection between different words based on their properties, so it could predict the semantics based on the relationship between words.

Hybrid approaches, as the name indicates, use both knowledge based and statistical methods to gain better results.

2. Naïve Bayes classifier

The 2 different models used to build sentiment classifiers in this project are Naïve Bayes and BERT. Naïve Bayes is a classical probabilistic classifier model. It utilizes the famous Bayes' theorem named after the English mathematician Thomas Bayes'. It calculates the probability of a current event with the knowledge of a prior event/condition that might be related to the current event.

Bayes' theorem

$$P(A|B) = \frac{P(B|A)P(A)}{P(B)}$$

Naive Bayes builds upon Bayes' theorem and makes the "naive" assumption that the features of a dataset are all independent from each other.

For this project, a multinomial naive Bayes classifier model is used. Multinomial naive bayes classifier is a form of naive bayes model that specializes in discrete feature classification, like word classification, which makes it popular in NLP and highly suitable for the project. In multinomial naive bayes, feature vectors represent the frequencies of certain events generated by a multinomial distribution. It usually requires integer feature counts. In multinomial distribution, events and their probability can be likened to a k-sided die rolled for n times. In the dataset's case, it's a binomial distribution, since k = 2 (2 classifications, 1 or 0 for positive or negative reviews), and n=4000(total number of reviews).

3. BERT

Bidirectional Encoder Representations from Transformers. It's a machine learning natural language processing model based on a transformer model.

Transformer is a deep learning model that incorporates attention mechanisms. Attention in computer science is the method of giving different data different importance by assigning them different weights. With the attention technique transformers can handle sequential data like normal RNNs but not necessarily in order, since now the input data has different importance/weights.

"Bidirectional" indicates that when BERT learns and retrieves information from the dtaa, it processes the data from both left-to-right and right-to-left order.

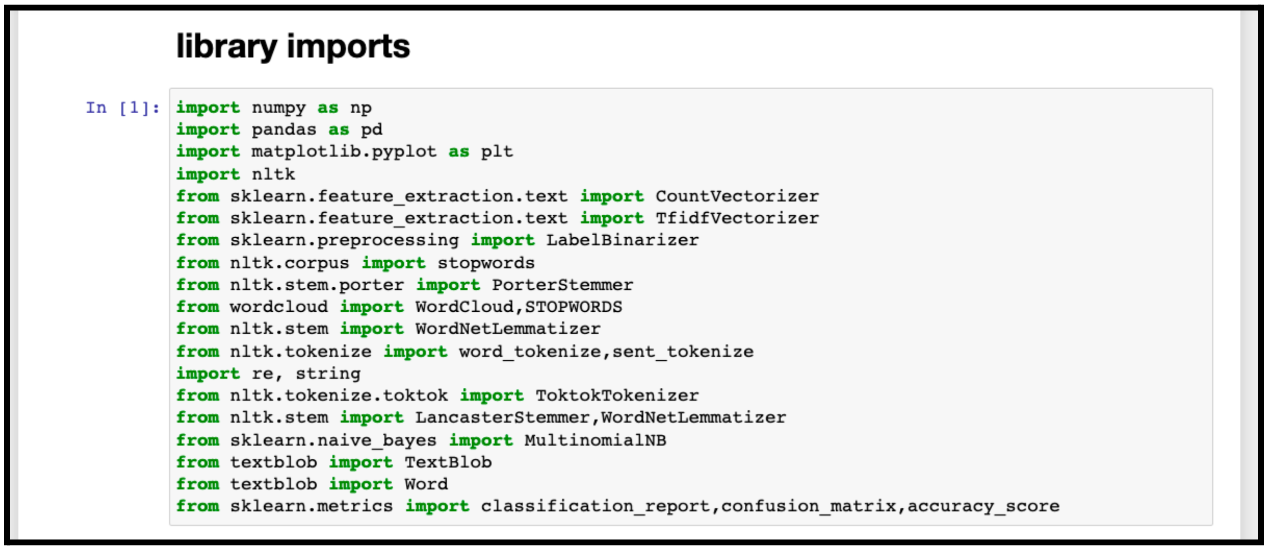
BERT is special for the fact that there are 2 important steps in its architecture, pre-training and fine-tuning. Developers pre-trained the model using unlabeled data on various tasks, then fine-tuned it with labeled data.

There are 2 tasks done in the pre-training step. First, in the masked language model (MLM) some percentage(15%) of the data is masked(replace the actual data with random info) at random, and then predict the masked portion. The next step is the next sentence prediction, which predicts what the sentence after a specific one is based on sentence relationships constructed.

In the fine-tuning step, BERT combines the encoding of text pairs and bidirectional cross self-attention together, by encoding a concatenated text pair so that the encoded text pair includes bidirectional cross self-attention.

Methods & experiments**(code segments display and explanation)**

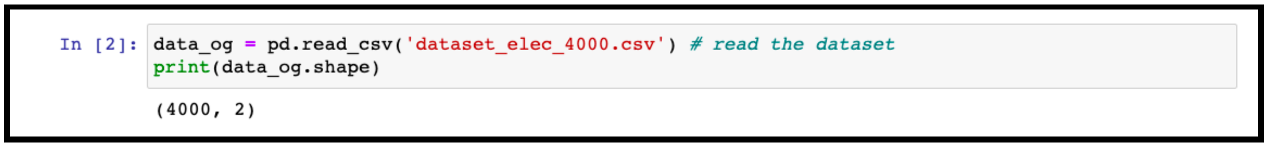
1. <u>Multinomial naive bayes model</u>



library imports

```
In [1]: import numpy as np
        import pandas as pd
        import matplotlib.pyplot as plt
        import nltk
        from sklearn.feature_extraction.text import CountVectorizer
        from sklearn.feature_extraction.text import TfidfVectorizer
        from sklearn.preprocessing import LabelBinarizer
        from nltk.corpus import stopwords
        from nltk.stem.porter import PorterStemmer
        from wordcloud import WordCloud,STOPWORDS
        from nltk.stem import WordNetLemmatizer
        from nltk.tokenize import word_tokenize,sent_tokenize
        import re, string
        from nltk.tokenize.toktok import ToktokTokenizer
        from nltk.stem import LancasterStemmer,WordNetLemmatizer
        from sklearn.naive_bayes import MultinomialNB
        from textblob import TextBlob
        from textblob import Word
        from sklearn.metrics import classification_report,confusion_matrix,accuracy_score
```

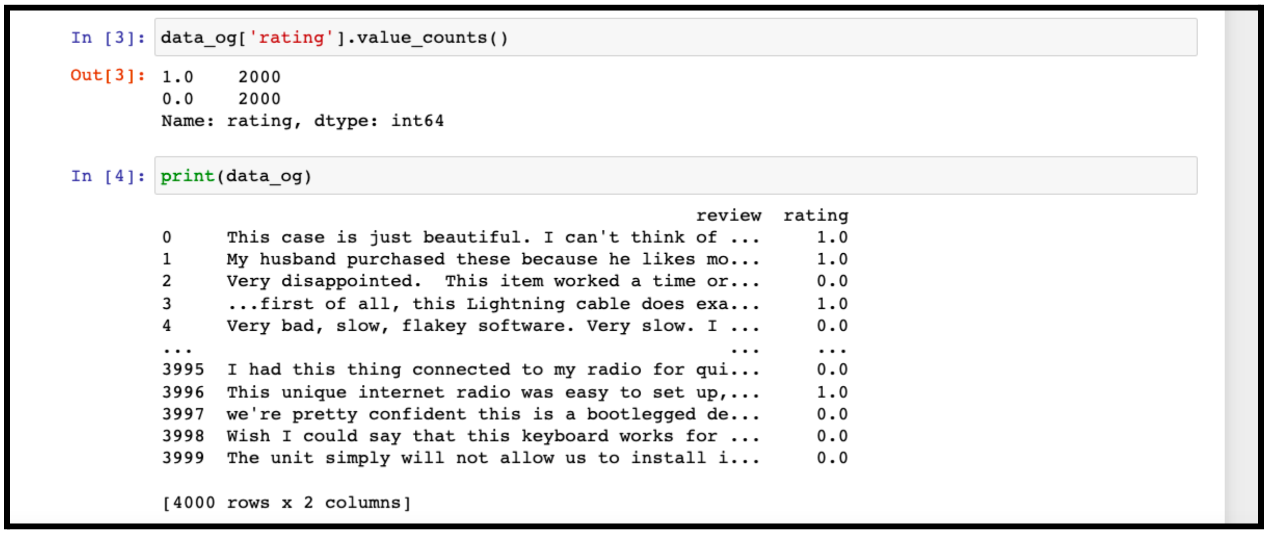
Importing necessary libraries into the file



```
In [2]: data_og = pd.read_csv('dataset_elec_4000.csv') # read the dataset
        print(data_og.shape)

(4000, 2)
```

Use pandas to read the csv file that contains the dataset and assign it to a variable.

Then show the size and dimensions of the dataset.



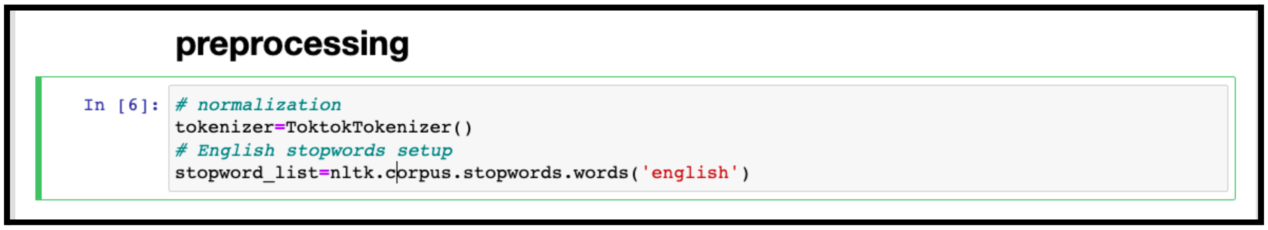
```
In [3]: data_og['rating'].value_counts()

Out[3]: 1.0    2000
        0.0    2000
        Name: rating, dtype: int64

In [4]: print(data_og)

                                                   review  rating
        0     This case is just beautiful. I can't think of ...     1.0
        1     My husband purchased these because he likes mo...     1.0
        2     Very disappointed.  This item worked a time or...     0.0
        3     ...first of all, this Lightning cable does exa...     1.0
        4     Very bad, slow, flakey software. Very slow. I ...     0.0
        ...                                                 ...     ...
        3995  I had this thing connected to my radio for qui...     0.0
        3996  This unique internet radio was easy to set up,...     1.0
        3997  we're pretty confident this is a bootlegged de...     0.0
        3998  Wish I could say that this keyboard works for ...     0.0
        3999  The unit simply will not allow us to install i...     0.0

        [4000 rows x 2 columns]
```

Print some general info about the dataset

As shown, the 2 columns are text reviews and numerical ratings.

Numbering of entries starts from 0 and ends at 3999.

Count the number of different values in the "rating" column of the data. We can see there are 2 values, 0 for negative reviews and 1 for positive ones. The dataset is very balanced as are 2000 of each value.



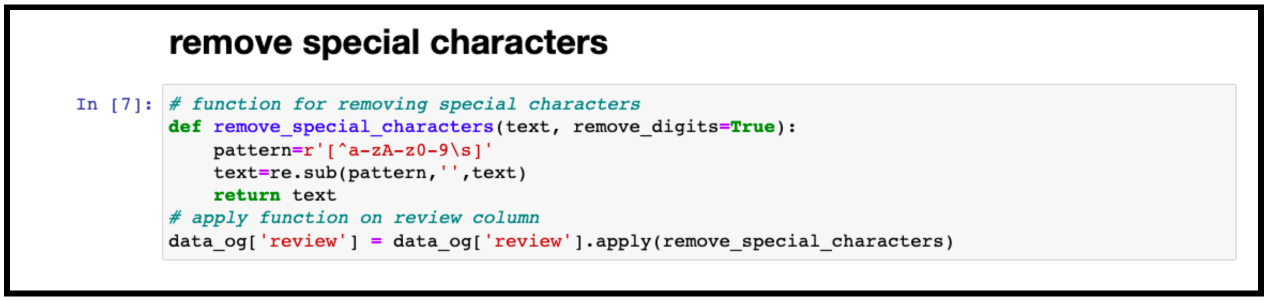
preprocessing

```
In [6]: # normalization
        tokenizer=ToktokTokenizer()
        # English stopwords setup
        stopword_list=nltk.corpus.stopwords.words('english')
```

Next step is the prep crossing.

First set up a tokenizer for later use.

Import all English stop words (words that contribute barely any real meaning to the sentences) and put them in a variable.



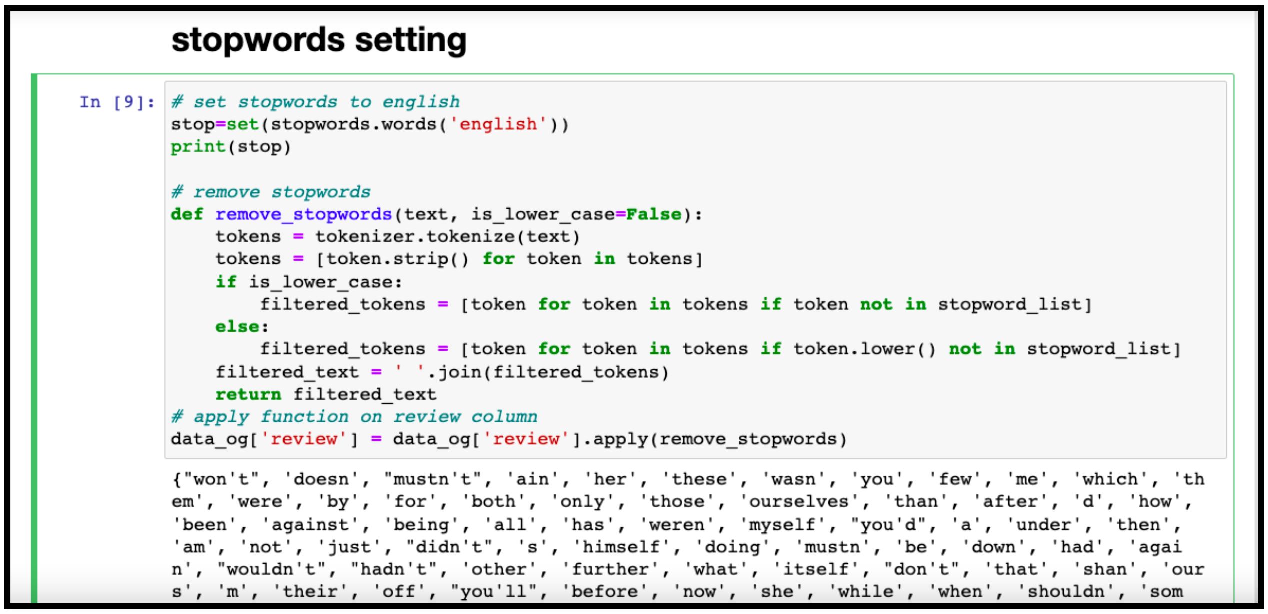
remove special characters

```
In [7]: # function for removing special characters
        def remove_special_characters(text, remove_digits=True):
            pattern=r'[^a-zA-z0-9\s]'
            text=re.sub(pattern,'',text)
            return text
        # apply function on review column
        data_og['review'] = data_og['review'].apply(remove_special_characters)
```

Remove special characters from the texts.

In this case, digits, and

Re.sub function is used to replace the characters in text with the text itself

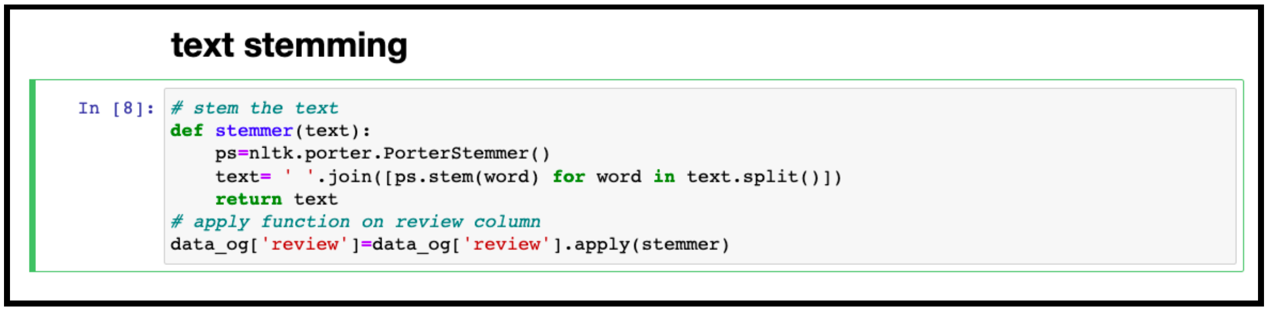


stopwords setting

```
In [9]: # set stopwords to english
        stop=set(stopwords.words('english'))
        print(stop)

        # remove stopwords
        def remove_stopwords(text, is_lower_case=False):
            tokens = tokenizer.tokenize(text)
            tokens = [token.strip() for token in tokens]
            if is_lower_case:
                filtered_tokens = [token for token in tokens if token not in stopword_list]
            else:
                filtered_tokens = [token for token in tokens if token.lower() not in stopword_list]
            filtered_text = ' '.join(filtered_tokens)
            return filtered_text
        # apply function on review column
        data_og['review'] = data_og['review'].apply(remove_stopwords)

        {"won't", 'doesn', "mustn't", 'ain', 'her', 'these', 'wasn', 'you', 'few', 'me', 'which', 'th
        em', 'were', 'by', 'for', 'both', 'only', 'those', 'ourselves', 'than', 'after', 'd', 'how',
        'been', 'against', 'being', 'all', 'has', 'weren', 'myself', "you'd", 'a', 'under', 'then',
        'am', 'not', 'just', "didn't", 's', 'himself', 'doing', 'mustn', 'be', 'down', 'had', 'agai
        n', "wouldn't", "hadn't", 'other', 'further', 'what', 'itself', "don't", 'that', 'shan', 'our
        s', 'm', 'their', 'off', "you'll", 'before', 'now', 'she', 'while', 'when', 'shouldn', 'som
```

Now the stopwords in the dataset will be cleaned

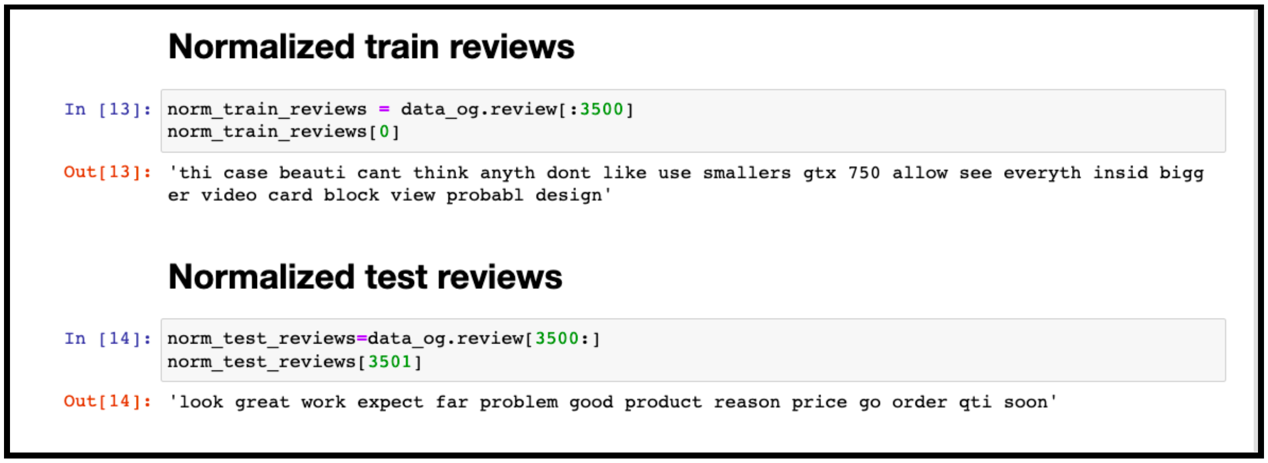
Use the “tokenize” set up earlier to tokenize the text. Tokenization is the process of breaking down large text chunks (paragraphs, sentences, phrases) into small units (words, single letters).



text stemming

```
In [8]: # stem the text
        def stemmer(text):
            ps=nltk.porter.PorterStemmer()
            text= ' '.join([ps.stem(word) for word in text.split()])
            return text
        # apply function on review column
        data_og['review']=data_og['review'].apply(stemmer)
```

Text stemming is the processing of turning inflected words to their simplest original form. E.g., Dogs, doggy, dog-like will be turned into "dog", verbs like "talked", "talking", "talks" will be turned into "talk".



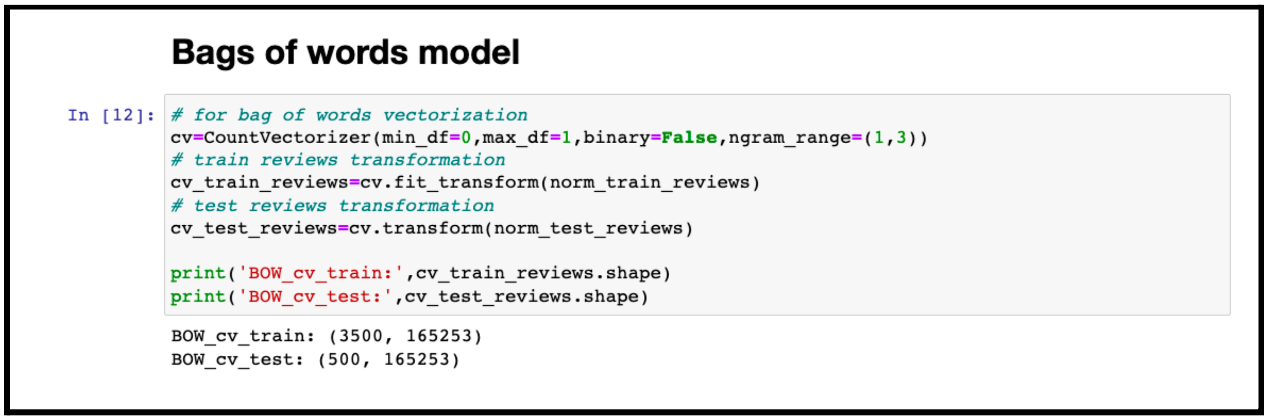
Normalized train reviews

```
In [13]: norm_train_reviews = data_og.review[:3500]
         norm_train_reviews[0]

Out[13]: 'thi case beauti cant think anyth dont like use smallers gtx 750 allow see everyth insid bigg
         er video card block view probabl design'
```

Normalized test reviews

```
In [14]: norm_test_reviews=data_og.review[3500:]
         norm_test_reviews[3501]

Out[14]: 'look great work expect far problem good product reason price go order qti soon'
```

Assign the normalized datasets(datasets that went through stemming, special character cleaning and stopword cleaning) to new variables.
Set up text review data for training and test
3500 entries will be used to train the mode while the rest of the 500 entries are used to test the model.
Training set contains items from row 0 to row 3499, the test set contains rows 3500 to 3999.



Bags of words model

```
In [12]: # for bag of words vectorization
         cv=CountVectorizer(min_df=0,max_df=1,binary=False,ngram_range=(1,3))
         # train reviews transformation
         cv_train_reviews=cv.fit_transform(norm_train_reviews)
         # test reviews transformation
         cv_test_reviews=cv.transform(norm_test_reviews)

         print('BOW_cv_train:',cv_train_reviews.shape)
         print('BOW_cv_test:',cv_test_reviews.shape)

         BOW_cv_train: (3500, 165253)
         BOW_cv_test: (500, 165253)
```

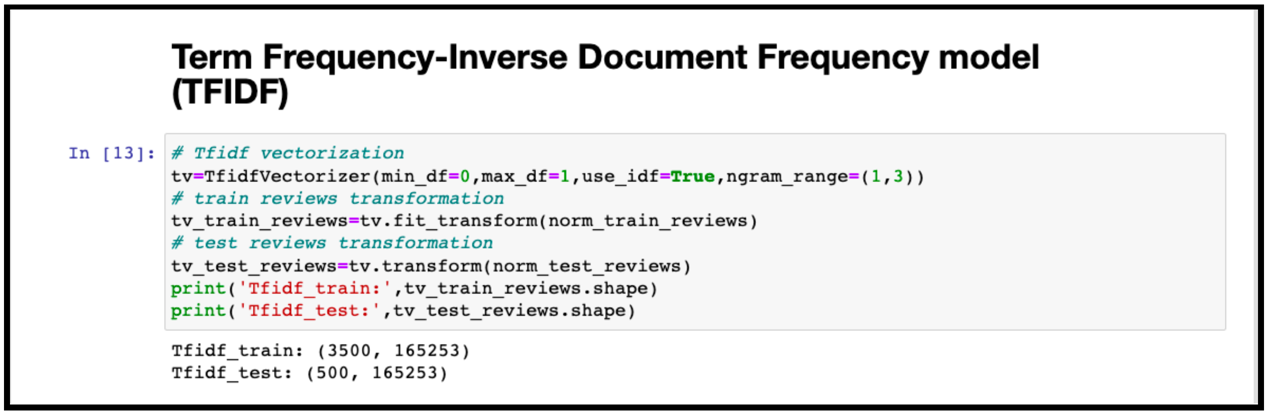
Use bags-of-words model to turn the dataset into BOW vectors.
Bags of words is a simple representation in NLP. documents (texts) are represented as "bags" of the words they contain, leaving out grammatical structures.
Use fit_traform to fit the features to the training dataset.
Use transform to fit the features to the test dataset.

fit_transform() method is used on the training data, it can scale to data of different dimensions. It also learns the scaling parameters. It calculates mean and variance for all features in the data and transforms those features using the mean and variance.
The transform() method is exactly the same as fit_trasfortm without the scale fitting "fit" method.
Bags of words explanation
E.g.
Sentence 1: The dog has the yellow hat
Sentence 2: The hat is yellow and the dog has it
Sentence 3: The dog loves the hat

	the	dog	loves	hat	is	it	yellow	has
Sentence 1	**2**	**1**	**0**	**1**	**0**	**0**	**1**	**1**
Sentence 2	**2**	**1**	**0**	**1**	**1**	**1**	**1**	**1**
Sentence 3	**2**	**1**	**1**	**1**	**0**	**0**	**0**	**0**

Now the vocabulary is established and word frequencies are counted, and they are shown in the table above.
Use the length 7 vector constructed (8 different words appeared) to represent the sentences
Sentence 1: The dog has a yellow hat $[2, 1, 0, 1, 0, 0, 1, 1]$
Sentence 2: The hat is yellow and the dog has it $[2, 1, 0, 1, 1, 1, 1, 1]$
Sentence 3: The dog loves the hat $[2, 1, 1, 1, 0, 0, 0, 0]$

The countVectorizer class sets up the features for BOW
Min_df sets the lower boundary for word frequency; words whose frequencies are lower than min_df are ignored.
Max_df sets up the upper boundary, words whose frequencies are higher than max_df are ignored.
Ngram_range sets up the minimum and maximum number of grams to include in features.
Grams are number of words or vocabulary units



Term Frequency-Inverse Document Frequency model (TFIDF)

```
In [13]: # Tfidf vectorization
         tv=TfidfVectorizer(min_df=0,max_df=1,use_idf=True,ngram_range=(1,3))
         # train reviews transformation
         tv_train_reviews=tv.fit_transform(norm_train_reviews)
         # test reviews transformation
         tv_test_reviews=tv.transform(norm_test_reviews)
         print('Tfidf_train:',tv_train_reviews.shape)
         print('Tfidf_test:',tv_test_reviews.shape)

         Tfidf_train: (3500, 165253)
         Tfidf_test: (500, 165253)
```

Use term frequency–inverse document frequency(TFIDF) model to turn the dataset into BOW vectors.

Use fit_traform to fit the features to the training dataset.

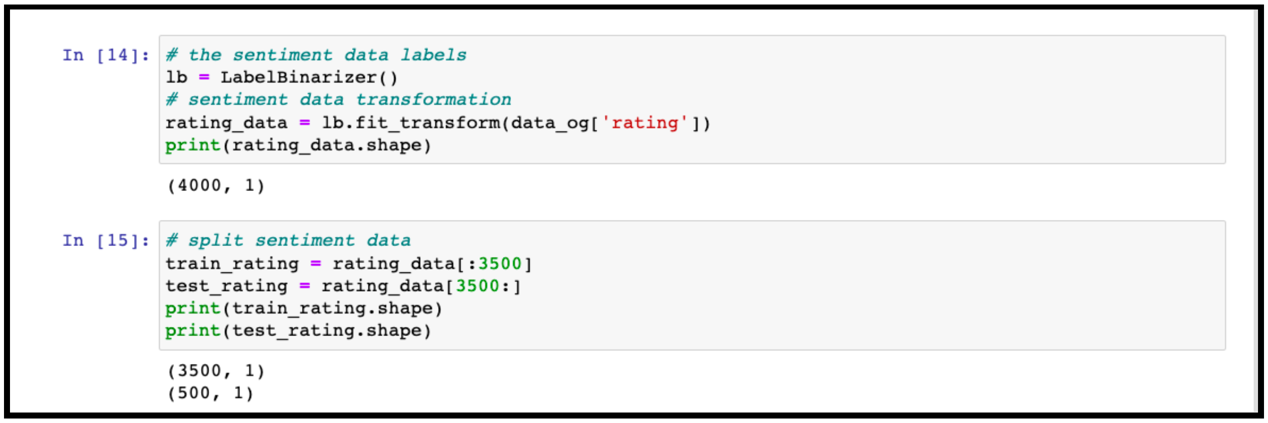
Use transform to fit the features to the test dataset.

TFIDF shows the importance of certains words to documents in a collection of corpus.

Term frequency refers to the times a certain term (word) appears in a document. To reduce the effect of unimportant words with high frequencies, like word “the”, inverse document frequency is used. By lowering the weights of words that appear too often and increasing weights for very rare words.

$$tf(t,d) = \frac{ft,d}{\sum_{t' \epsilon d} f_{t',d}}$$

$tf(t,d)$**: times term “t” occurs in a document “d”**



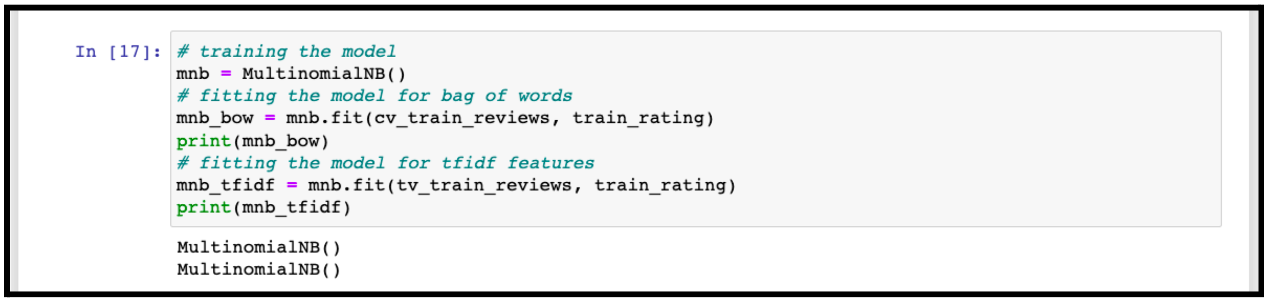
```
In [14]: # the sentiment data labels
         lb = LabelBinarizer()
         # sentiment data transformation
         rating_data = lb.fit_transform(data_og['rating'])
         print(rating_data.shape)

         (4000, 1)

In [15]: # split sentiment data
         train_rating = rating_data[:3500]
         test_rating = rating_data[3500:]
         print(train_rating.shape)
         print(test_rating.shape)

         (3500, 1)
         (500, 1)
```

Fit labels to the ratings in the original dataset using labelbinarizer, then split it into the training and test sets.

Set up numerical rating data for training and test

3500 entries will be used to train the mode while the rest of the 500 entries are used to test the model.

Training set contains items from row 0 to row 3499, the test set contains rows 3500 to 4000.

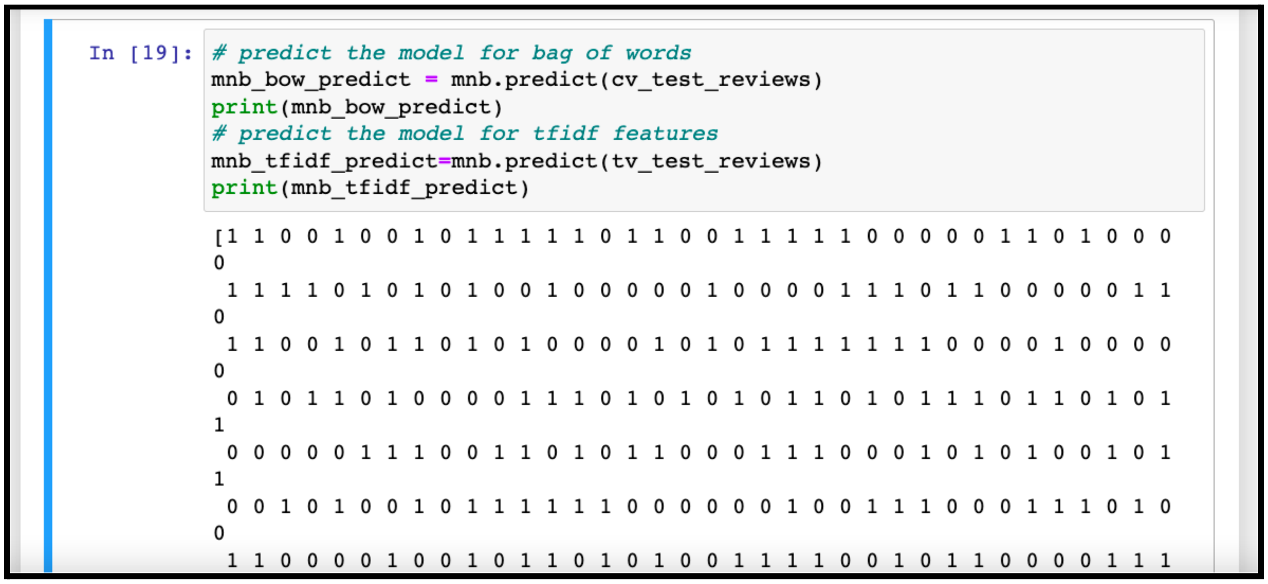


```
In [17]: # training the model
         mnb = MultinomialNB()
         # fitting the model for bag of words
         mnb_bow = mnb.fit(cv_train_reviews, train_rating)
         print(mnb_bow)
         # fitting the model for tfidf features
         mnb_tfidf = mnb.fit(tv_train_reviews, train_rating)
         print(mnb_tfidf)

         MultinomialNB()
         MultinomialNB()
```

Use the multinomial Naive Bayes classifier model loaded from the sklearn . Train the multinomial naive bayes model by fitting it to both BOW and TF IDF features, using the vectorized text review data and the labeled numerical data

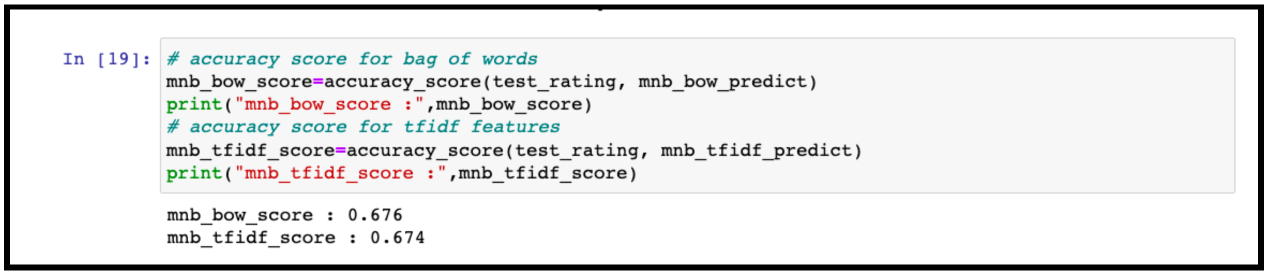
Extra:

Sklearn.naive_bayes contains 5 different naive bayes classifier models, BernoulliNB for multivariate classifier, CategoricalNB for categorical feature classification, ComplemeNB which complements the normal multinomialNB, suitable for unbalanced datasets, GaussianNB and MultinomialNB for multinomial models used in this project.



```
In [19]: # predict the model for bag of words
         mnb_bow_predict = mnb.predict(cv_test_reviews)
         print(mnb_bow_predict)
         # predict the model for tfidf features
         mnb_tfidf_predict=mnb.predict(tv_test_reviews)
         print(mnb_tfidf_predict)

         [1 1 0 0 1 0 0 1 0 1 1 1 1 1 0 1 1 0 0 1 1 1 1 1 0 0 0 0 0 1 1 0 1 0 0 0
         0
          1 1 1 1 0 1 0 1 0 1 0 0 1 0 0 0 0 0 1 0 0 0 0 1 1 1 0 1 1 0 0 0 0 0 1 1
         0
          1 1 0 0 1 0 1 1 0 1 0 1 0 0 0 0 1 0 1 0 1 1 1 1 1 1 1 0 0 0 0 1 0 0 0 0
         0
          0 1 0 1 1 0 1 0 0 0 0 1 1 1 0 1 0 1 0 1 0 1 1 0 1 0 1 1 1 0 1 1 0 1 0 1
         1
          0 0 0 0 0 1 1 1 0 0 1 1 0 1 0 1 1 0 0 0 1 1 1 0 0 0 1 0 1 0 1 0 0 1 0 1
         1
          0 0 1 0 1 0 0 1 0 1 1 1 1 1 1 0 0 0 0 0 0 1 0 0 1 1 1 0 0 0 1 1 1 0 1 0
         0
          1 1 0 0 0 0 1 0 0 1 0 1 1 0 1 0 1 0 0 1 1 1 1 0 0 1 0 1 1 0 0 0 0 1 1 1
```

Use the trained model to predict the numerical ratings for the test dataset text reviews, based on 2 different features.



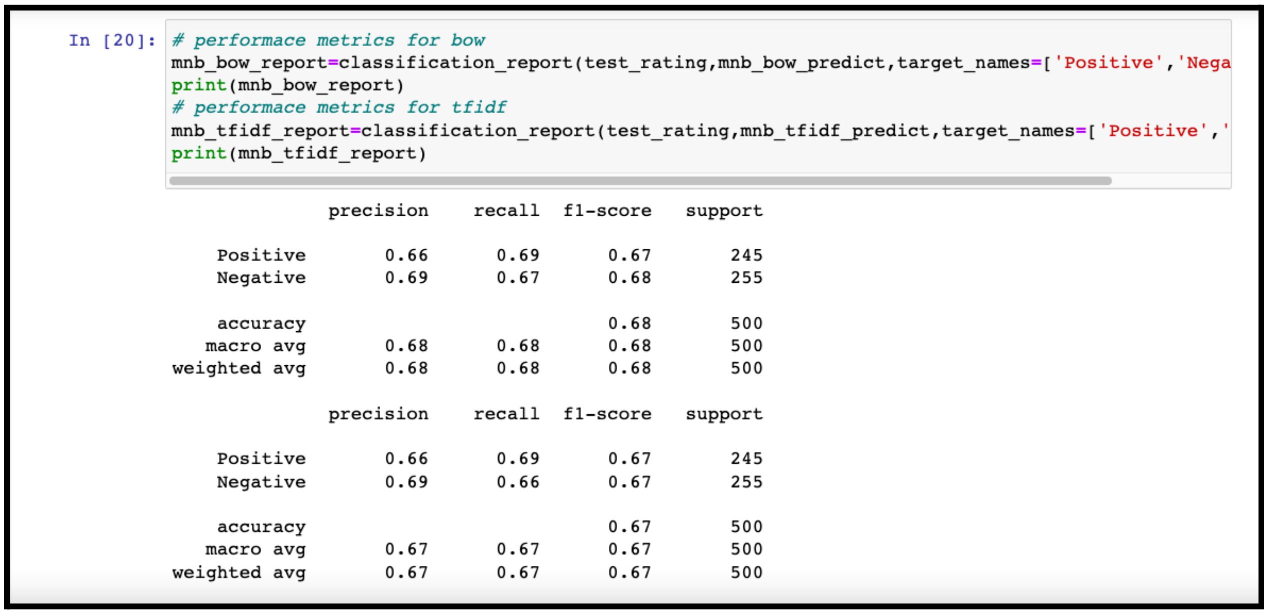
```
In [19]: # accuracy score for bag of words
         mnb_bow_score=accuracy_score(test_rating, mnb_bow_predict)
         print("mnb_bow_score :",mnb_bow_score)
         # accuracy score for tfidf features
         mnb_tfidf_score=accuracy_score(test_rating, mnb_tfidf_predict)
         print("mnb_tfidf_score :",mnb_tfidf_score)

         mnb_bow_score : 0.676
         mnb_tfidf_score : 0.674
```

Check the prediction accuracy of the trained model for both BOW and TFidf by comparing the predicted numerical trainings to the actual ratings

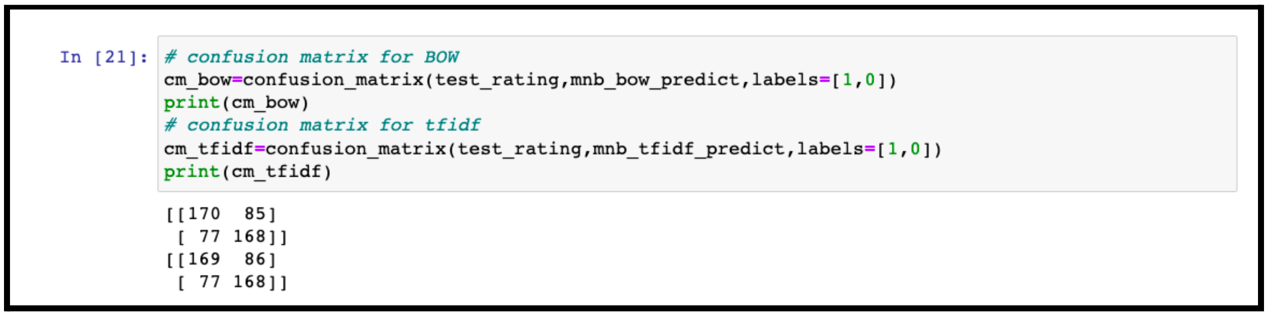
Multinomial naive bayes on the BOW model have a slightly higher accuracy compared to it on the TF IDF model.

BOW accuracy score: 67.6%

TFIDF accuracy score: 67.4%

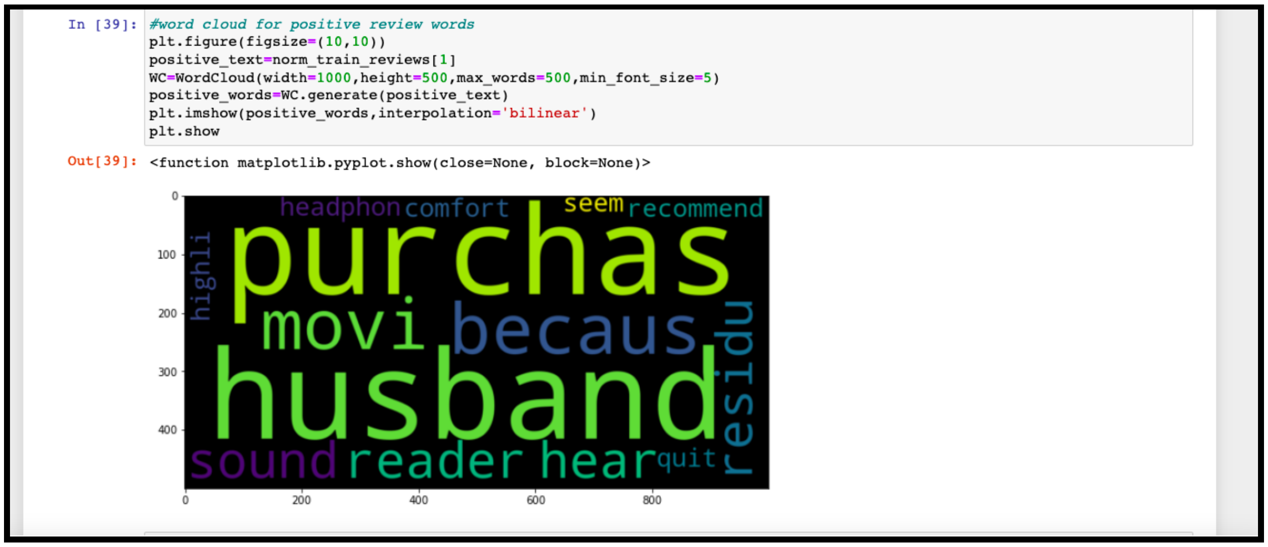


```
In [20]: # performace metrics for bow
         mnb_bow_report=classification_report(test_rating,mnb_bow_predict,target_names=['Positive','Nega
         print(mnb_bow_report)
         # performace metrics for tfidf
         mnb_tfidf_report=classification_report(test_rating,mnb_tfidf_predict,target_names=['Positive','
         print(mnb_tfidf_report)

                       precision    recall  f1-score   support

             Positive       0.66      0.69      0.67       245
             Negative       0.69      0.67      0.68       255

             accuracy                           0.68       500
            macro avg       0.68      0.68      0.68       500
         weighted avg       0.68      0.68      0.68       500

                       precision    recall  f1-score   support

             Positive       0.66      0.69      0.67       245
             Negative       0.69      0.66      0.67       255

             accuracy                           0.67       500
            macro avg       0.67      0.67      0.67       500
         weighted avg       0.67      0.67      0.67       500
```



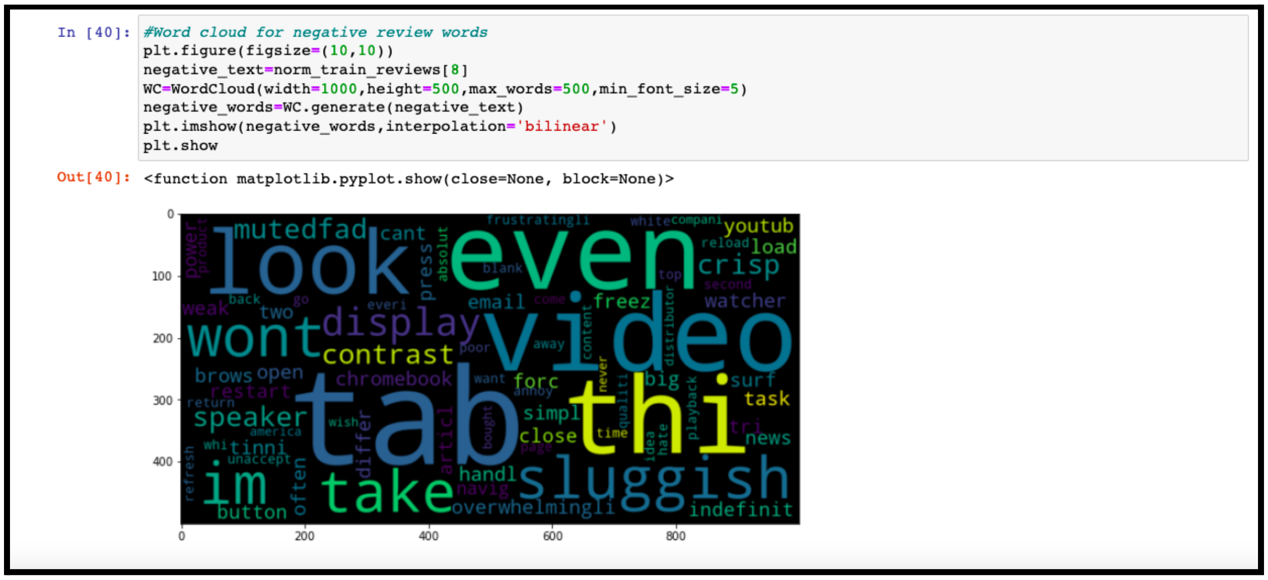
```
In [21]: # confusion matrix for BOW
         cm_bow=confusion_matrix(test_rating,mnb_bow_predict,labels=[1,0])
         print(cm_bow)
         # confusion matrix for tfidf
         cm_tfidf=confusion_matrix(test_rating,mnb_tfidf_predict,labels=[1,0])
         print(cm_tfidf)

         [[170  85]
          [ 77 168]]
         [[169  86]
          [ 77 168]]
```

Print the performance metrics and the confusion matrices for the model's prediction results based on the 2 features.



```
In [39]: #word cloud for positive review words
         plt.figure(figsize=(10,10))
         positive_text=norm_train_reviews[1]
         WC=WordCloud(width=1000,height=500,max_words=500,min_font_size=5)
         positive_words=WC.generate(positive_text)
         plt.imshow(positive_words,interpolation='bilinear')
         plt.show

Out[39]: <function matplotlib.pyplot.show(close=None, block=None)>
```

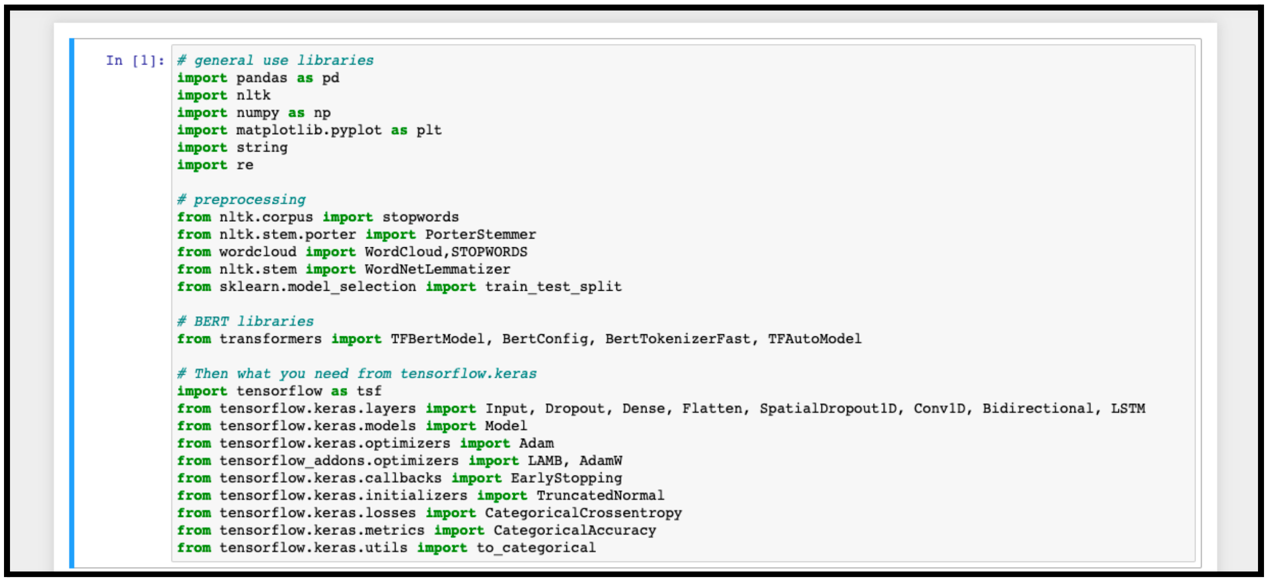
Use matplot library and wordcloud to visualize the words in the dataset that are considered positive by the model.



```
In [40]: #Word cloud for negative review words
         plt.figure(figsize=(10,10))
         negative_text=norm_train_reviews[8]
         WC=WordCloud(width=1000,height=500,max_words=500,min_font_size=5)
         negative_words=WC.generate(negative_text)
         plt.imshow(negative_words,interpolation='bilinear')
         plt.show

Out[40]: <function matplotlib.pyplot.show(close=None, block=None)>
```

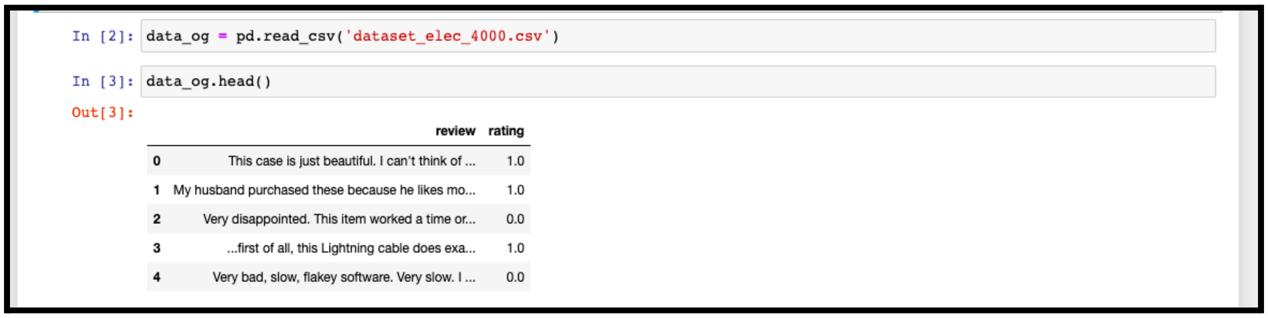
Use matplot library and wordcloud to visualize the words in the dataset that are considered negative by the model.

2. BERT deep learning model



```
In [1]: # general use libraries
        import pandas as pd
        import nltk
        import numpy as np
        import matplotlib.pyplot as plt
        import string
        import re

        # preprocessing
        from nltk.corpus import stopwords
        from nltk.stem.porter import PorterStemmer
        from wordcloud import WordCloud,STOPWORDS
        from nltk.stem import WordNetLemmatizer
        from sklearn.model_selection import train_test_split

        # BERT libraries
        from transformers import TFBertModel, BertConfig, BertTokenizerFast, TFAutoModel

        # Then what you need from tensorflow.keras
        import tensorflow as tsf
        from tensorflow.keras.layers import Input, Dropout, Dense, Flatten, SpatialDropout1D, Conv1D, Bidirectional, LSTM
        from tensorflow.keras.models import Model
        from tensorflow.keras.optimizers import Adam
        from tensorflow_addons.optimizers import LAMB, AdamW
        from tensorflow.keras.callbacks import EarlyStopping
        from tensorflow.keras.initializers import TruncatedNormal
        from tensorflow.keras.losses import CategoricalCrossentropy
        from tensorflow.keras.metrics import CategoricalAccuracy
        from tensorflow.keras.utils import to_categorical
```

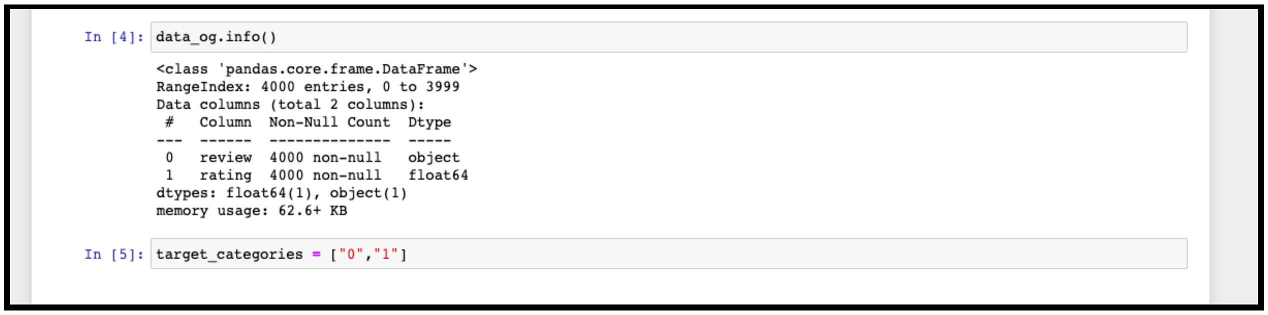
Import the necessary libraries like we did with the Naive Bayes model.



```
In [2]: data_og = pd.read_csv('dataset_elec_4000.csv')

In [3]: data_og.head()
Out[3]:
```

	review	rating
0	This case is just beautiful. I can't think of ...	1.0
1	My husband purchased these because he likes mo...	1.0
2	Very disappointed. This item worked a time or...	0.0
3	...first of all, this Lightning cable does exa...	1.0
4	Very bad, slow, flakey software. Very slow. I ...	0.0

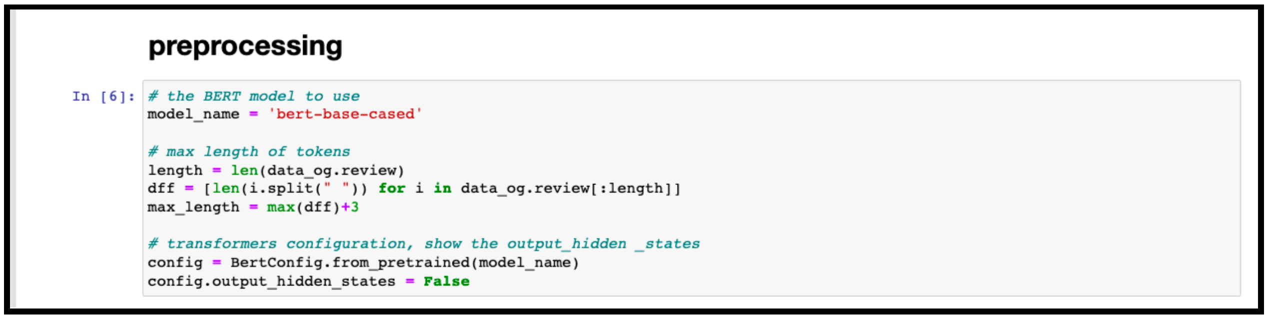
Use pandas to read the csv file and show the top 5 rows for view.



```
In [4]: data_og.info()

        <class 'pandas.core.frame.DataFrame'>
        RangeIndex: 4000 entries, 0 to 3999
        Data columns (total 2 columns):
         #   Column  Non-Null Count  Dtype
        ---  ------  --------------  -----
         0   review  4000 non-null   object
         1   rating  4000 non-null   float64
        dtypes: float64(1), object(1)
        memory usage: 62.6+ KB

In [5]: target_categories = ["0","1"]
```

Print out data information for the dataset.

Information types include data type, review is object, rating is float, counts of non-null entries.



preprocessing

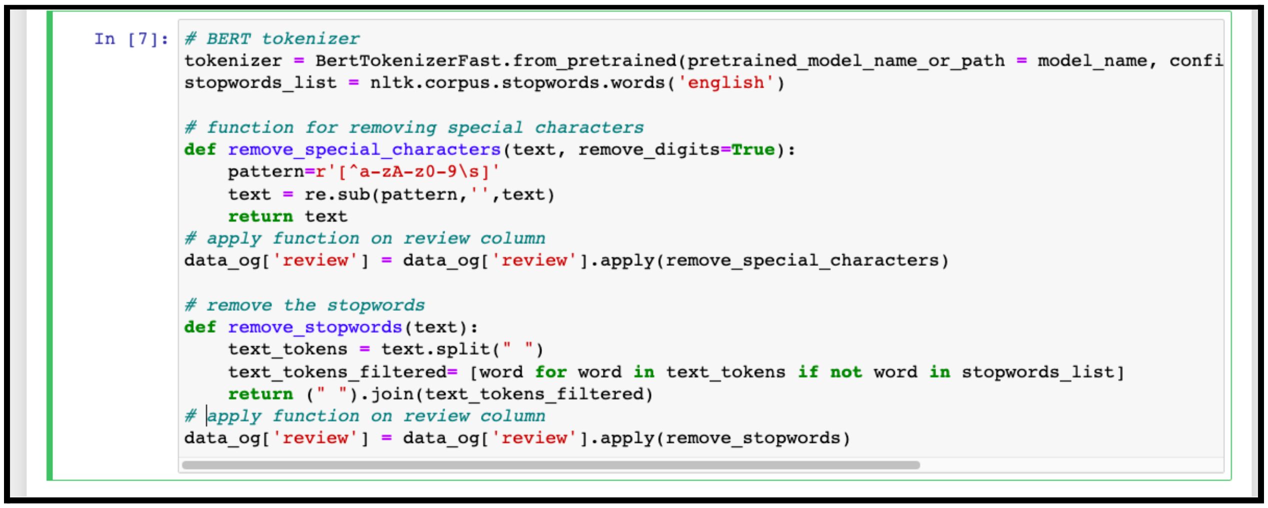
```
In [6]: # the BERT model to use
        model_name = 'bert-base-cased'

        # max length of tokens
        length = len(data_og.review)
        dff = [len(i.split(" ")) for i in data_og.review[:length]]
        max_length = max(dff)+3

        # transformers configuration, show the output_hidden _states
        config = BertConfig.from_pretrained(model_name)
        config.output_hidden_states = False
```

Set up the model which will be used, imported from the transformers library.

Set up a max length of tokens, equal to the number of rows in the dataset.

.split() method is used to pick single words from the review column apart, separating them by the spaces between them.

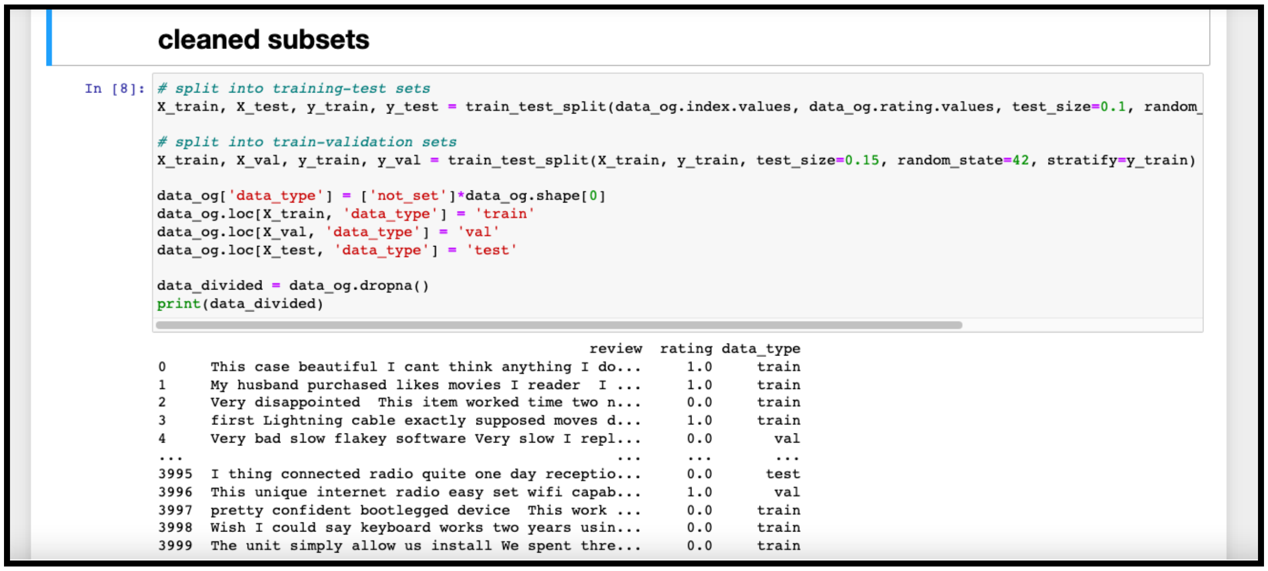
Loads the pretrained model configuration for 'bert-case-incased'



```
In [7]: # BERT tokenizer
        tokenizer = BertTokenizerFast.from_pretrained(pretrained_model_name_or_path = model_name, confi
        stopwords_list = nltk.corpus.stopwords.words('english')

        # function for removing special characters
        def remove_special_characters(text, remove_digits=True):
            pattern=r'[^a-zA-z0-9\s]'
            text = re.sub(pattern,'',text)
            return text
        # apply function on review column
        data_og['review'] = data_og['review'].apply(remove_special_characters)

        # remove the stopwords
        def remove_stopwords(text):
            text_tokens = text.split(" ")
            text_tokens_filtered= [word for word in text_tokens if not word in stopwords_list]
            return (" ").join(text_tokens_filtered)
        # apply function on review column
        data_og['review'] = data_og['review'].apply(remove_stopwords)
```

Set up the BERT tokenizer.

Clean the dataset by removing special characters, stopwords.



cleaned subsets

```
In [8]: # split into training-test sets
        X_train, X_test, y_train, y_test = train_test_split(data_og.index.values, data_og.rating.values, test_size=0.1, random_

        # split into train-validation sets
        X_train, X_val, y_train, y_val = train_test_split(X_train, y_train, test_size=0.15, random_state=42, stratify=y_train)

        data_og['data_type'] = ['not_set']*data_og.shape[0]
        data_og.loc[X_train, 'data_type'] = 'train'
        data_og.loc[X_val, 'data_type'] = 'val'
        data_og.loc[X_test, 'data_type'] = 'test'

        data_divided = data_og.dropna()
        print(data_divided)
```

```
                                                 review  rating data_type
0     This case beautiful I cant think anything I do...     1.0     train
1     My husband purchased likes movies I reader  I ...     1.0     train
2     Very disappointed  This item worked time two n...     0.0     train
3     first Lightning cable exactly supposed moves d...     1.0     train
4     Very bad slow flakey software Very slow I repl...     0.0       val
...                                                 ...     ...       ...
3995  I thing connected radio quite one day receptio...     0.0      test
3996  This unique internet radio easy set wifi capab...     1.0       val
3997  pretty confident bootlegged device  This work ...     0.0     train
3998  Wish I could say keyboard works two years usin...     0.0     train
3999  The unit simply allow us install We spent thre...     0.0     train
```

Now put the cleaned dataset into subsets.

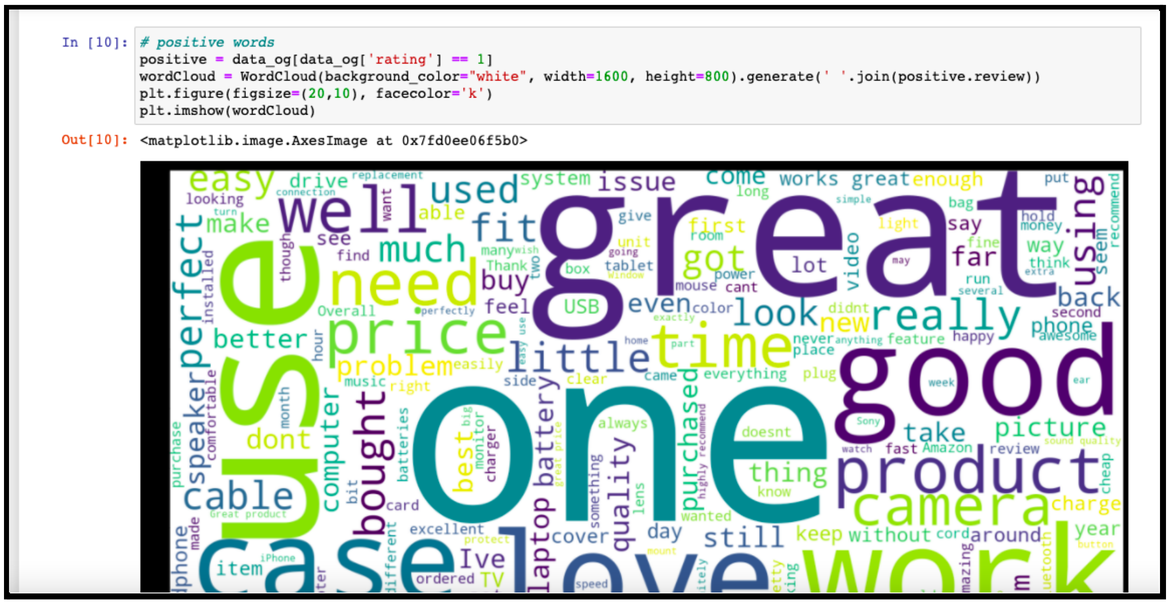
Use the train_test_split function to split the dataset into training and test sets.

.values returns the view object of the dictionary in the dataset as a list. View objects are the values in a dictionary, in this case 0s and 1s.

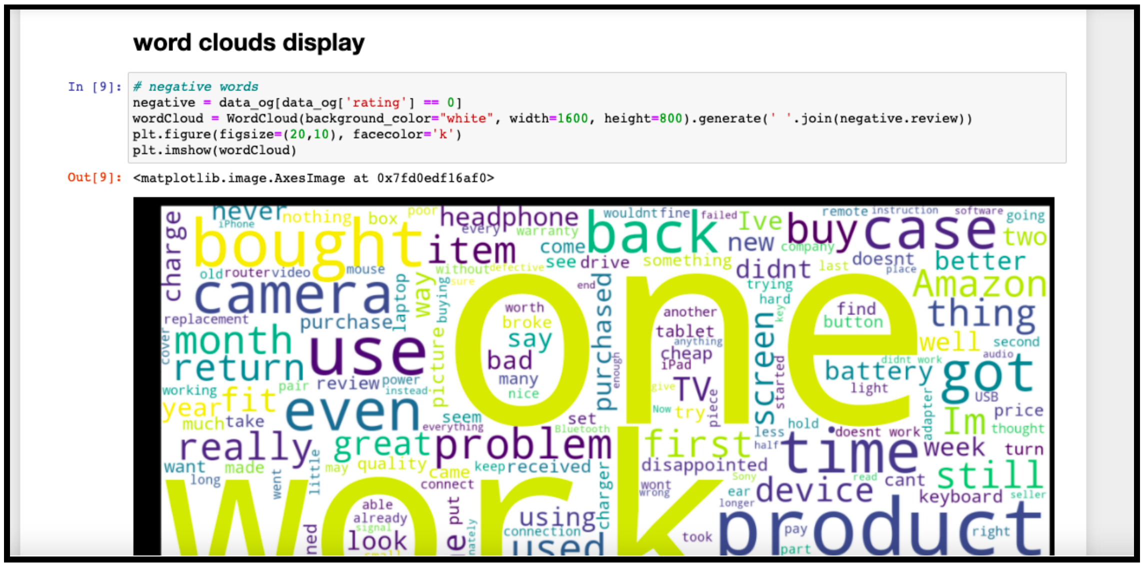
Test_size is a float, 0.1 means the test set is 0.1 portion of the original dataset.

Random_state passes a random int for data shuffling, so that reproducible results can be achieved across function calls.

Then add a new column to the dataset containing new info, labels for entries' set info. Which are in the training set, which in test and which in validation.



```
In [10]: # positive words
         positive = data_og[data_og['rating'] == 1]
         wordCloud = WordCloud(background_color="white", width=1600, height=800).generate(' '.join(positive.review))
         plt.figure(figsize=(20,10), facecolor='k')
         plt.imshow(wordCloud)

Out[10]: <matplotlib.image.AxesImage at 0x7fd0ee06f5b0>
```



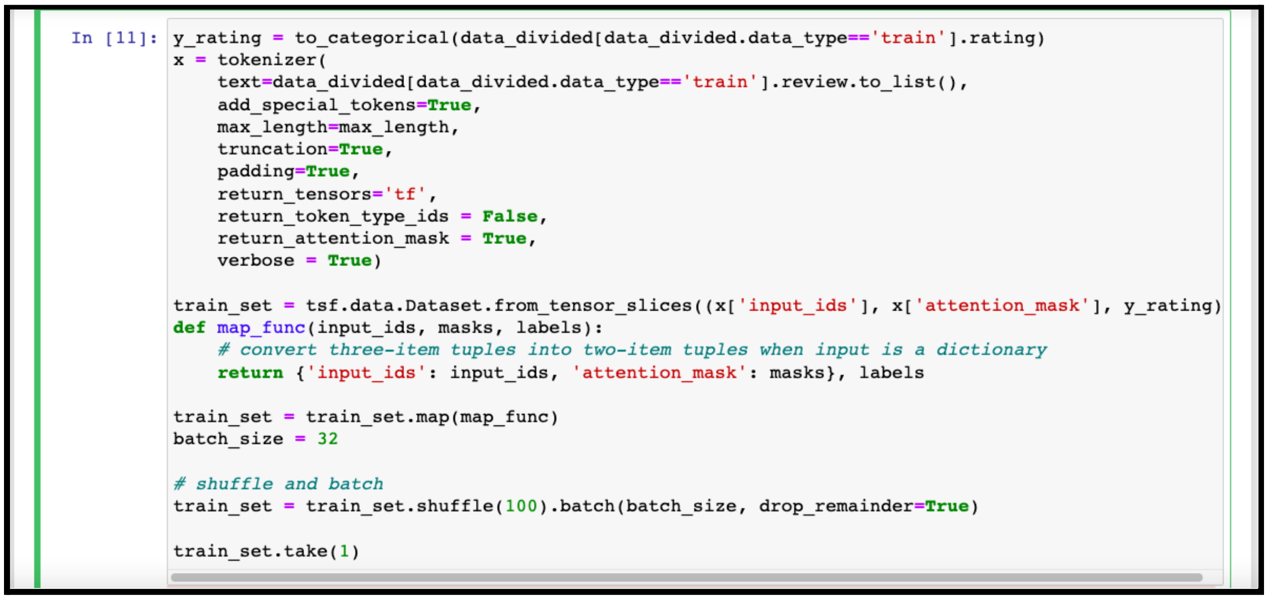
word clouds display

```
In [9]: # negative words
        negative = data_og[data_og['rating'] == 0]
        wordCloud = WordCloud(background_color="white", width=1600, height=800).generate(' '.join(negative.review))
        plt.figure(figsize=(20,10), facecolor='k')
        plt.imshow(wordCloud)

Out[9]: <matplotlib.image.AxesImage at 0x7fd0edf16af0>
```

Use word cloud to show positive and negative words. A rough visualization.

Positive word cloud contains words from reviews with “0” numerical ratings

Negative word cloud contains words from the reviews with “1” numerical ratings.



```
In [11]: y_rating = to_categorical(data_divided[data_divided.data_type=='train'].rating)
         x = tokenizer(
             text=data_divided[data_divided.data_type=='train'].review.to_list(),
             add_special_tokens=True,
             max_length=max_length,
             truncation=True,
             padding=True,
             return_tensors='tf',
             return_token_type_ids = False,
             return_attention_mask = True,
             verbose = True)

         train_set = tsf.data.Dataset.from_tensor_slices((x['input_ids'], x['attention_mask'], y_rating)
         def map_func(input_ids, masks, labels):
             # convert three-item tuples into two-item tuples when input is a dictionary
             return {'input_ids': input_ids, 'attention_mask': masks}, labels

         train_set = train_set.map(map_func)
         batch_size = 32

         # shuffle and batch
         train_set = train_set.shuffle(100).batch(batch_size, drop_remainder=True)

         train_set.take(1)
```

Use the to_categorical function to turn a matrix of binary values for the original vector values in the dataset.

In the tokenizer, text collects the text review data from the dataset.

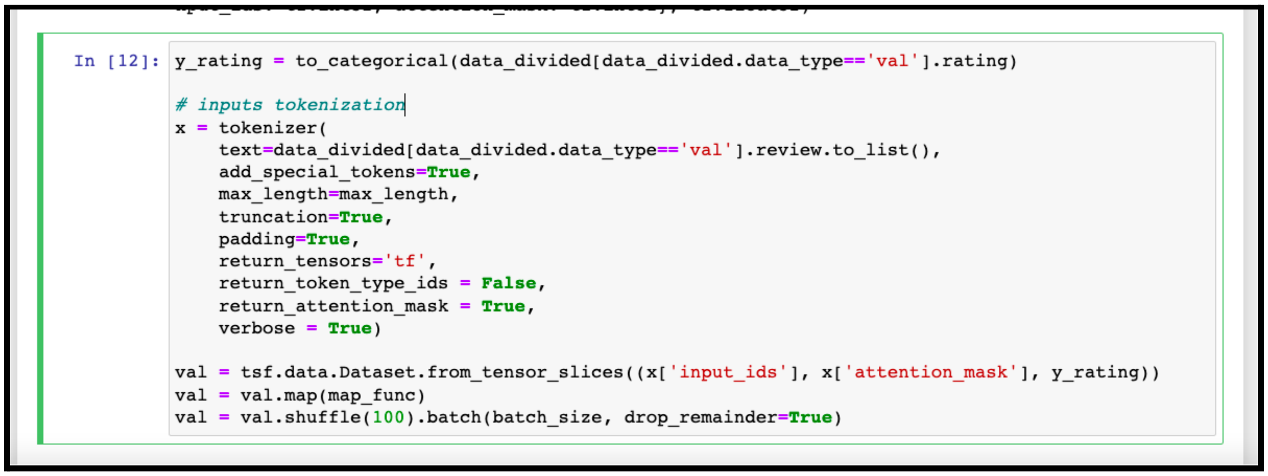
Add_special_tokens =True adds special token encoding.

Truncation truncates tokens to the specified max length.

return_token_type_ids = False will not return token type info.

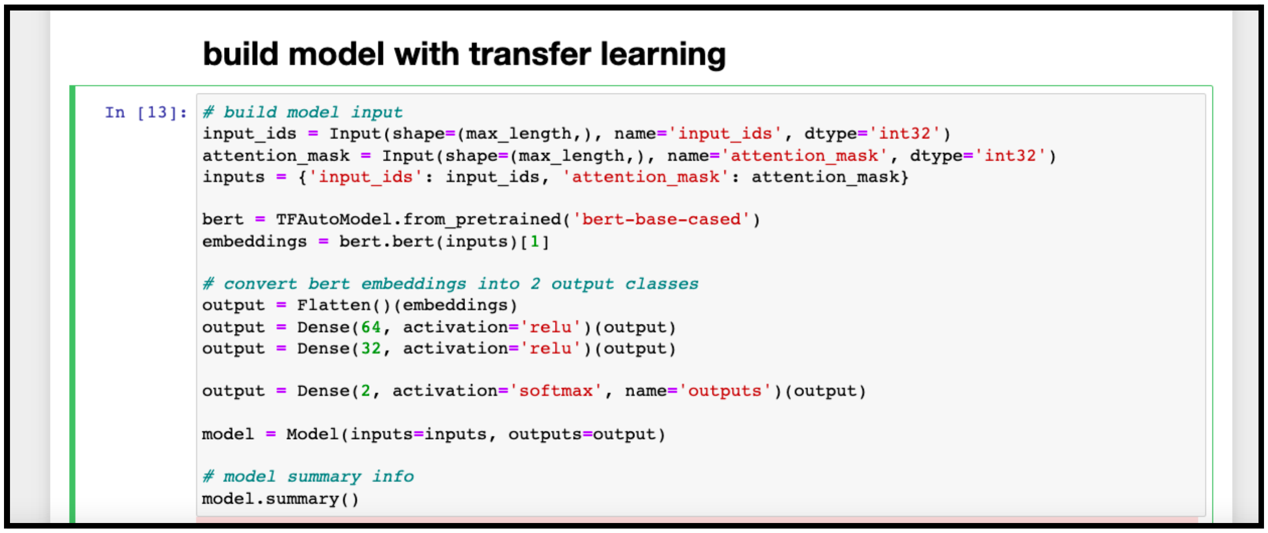
return_attention_mask = True returns the attention mask.

Use the tuple converting map_func to turn train_set into 2 item tuples.
.shuffle(100) randomize the order of 100 tokens.
.batch() to represent the batch_sized set.



```
In [12]: y_rating = to_categorical(data_divided[data_divided.data_type=='val'].rating)

         # inputs tokenization
         x = tokenizer(
             text=data_divided[data_divided.data_type=='val'].review.to_list(),
             add_special_tokens=True,
             max_length=max_length,
             truncation=True,
             padding=True,
             return_tensors='tf',
             return_token_type_ids = False,
             return_attention_mask = True,
             verbose = True)

         val = tsf.data.Dataset.from_tensor_slices((x['input_ids'], x['attention_mask'], y_rating))
         val = val.map(map_func)
         val = val.shuffle(100).batch(batch_size, drop_remainder=True)
```

Repeat the process shown above again, but this time with the validation subset of the data.

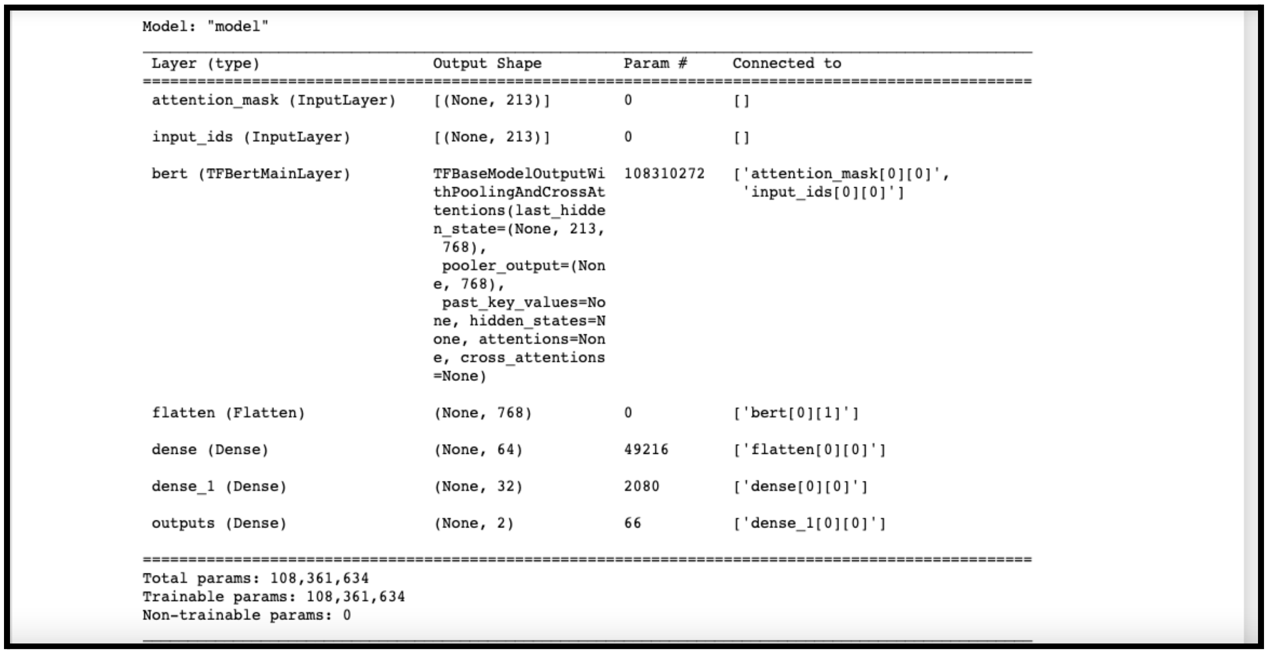


build model with transfer learning

```
In [13]: # build model input
         input_ids = Input(shape=(max_length,), name='input_ids', dtype='int32')
         attention_mask = Input(shape=(max_length,), name='attention_mask', dtype='int32')
         inputs = {'input_ids': input_ids, 'attention_mask': attention_mask}

         bert = TFAutoModel.from_pretrained('bert-base-cased')
         embeddings = bert.bert(inputs)[1]

         # convert bert embeddings into 2 output classes
         output = Flatten()(embeddings)
         output = Dense(64, activation='relu')(output)
         output = Dense(32, activation='relu')(output)

         output = Dense(2, activation='softmax', name='outputs')(output)

         model = Model(inputs=inputs, outputs=output)

         # model summary info
         model.summary()
```

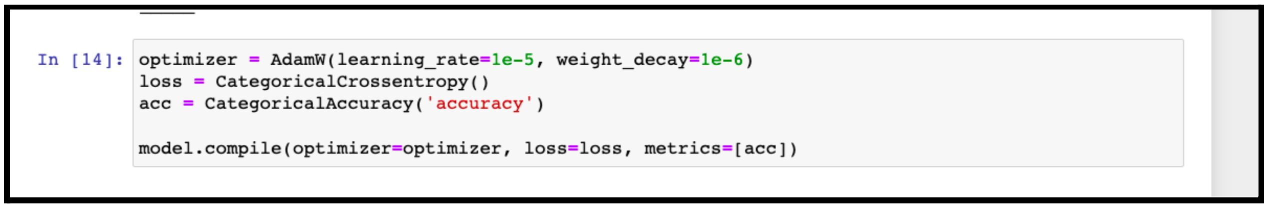
Now build our model with transfer learning.
Settle input ids and attention mask.
Use the ‘bert-ase-uncased’ pretrained model.
Inputs include input ids and attention masks.
Use BERT embedding to transform input data into real value vectors.
flatten() function used to “flatten” out the inputs, turning high dimension data into 1 dimension data. E.g. turn a multiple row matrix into a vector.

Dense is a neural network layer from tf.keras.layers, units is the dimension of a specific layer's output, activation implements the relu activation function.



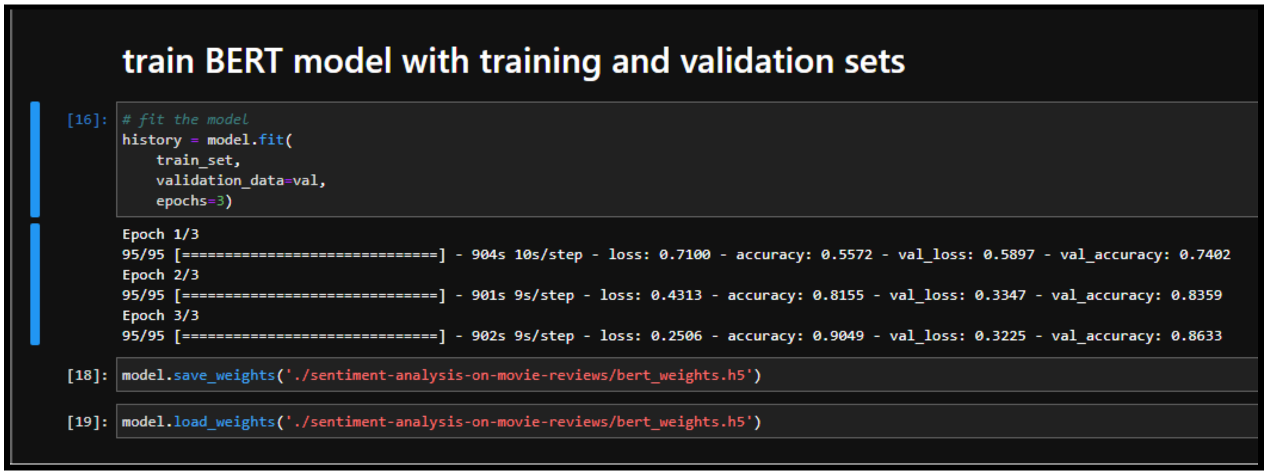
```
Model: "model"
__________________________________________________________________________________________________
 Layer (type)                   Output Shape         Param #     Connected to                     
==================================================================================================
 attention_mask (InputLayer)    [(None, 213)]        0           []                               
                                                                                                  
 input_ids (InputLayer)         [(None, 213)]        0           []                               
                                                                                                  
 bert (TFBertMainLayer)         TFBaseModelOutputWi  108310272   ['attention_mask[0][0]',         
                                thPoolingAndCrossAt               'input_ids[0][0]']              
                                tentions(last_hidde                                               
                                n_state=(None, 213,                                               
                                 768),                                                            
                                 pooler_output=(Non                                               
                                e, 768),                                                          
                                 past_key_values=No                                               
                                ne, hidden_states=N                                               
                                one, attentions=Non                                               
                                e, cross_attentions                                               
                                =None)                                                            
                                                                                                  
 flatten (Flatten)              (None, 768)          0           ['bert[0][1]']                   
                                                                                                  
 dense (Dense)                  (None, 64)           49216       ['flatten[0][0]']                
                                                                                                  
 dense_1 (Dense)                (None, 32)           2080        ['dense[0][0]']                  
                                                                                                  
 outputs (Dense)                (None, 2)            66          ['dense_1[0][0]']                
                                                                                                  
==================================================================================================
Total params: 108,361,634
Trainable params: 108,361,634
Non-trainable params: 0
__________________________________________________________________________________________________
```

Summary of the model. Information like layer type, output and output size can be seen.



```
In [14]: optimizer = AdamW(learning_rate=1e-5, weight_decay=1e-6)
         loss = CategoricalCrossentropy()
         acc = CategoricalAccuracy('accuracy')

         model.compile(optimizer=optimizer, loss=loss, metrics=[acc])
```

Use Adam algorithm with weight decay for optimizer.
Adam algorithm is an optimization algorithm specifically trained for deep learning models.
Weight decay is a regularization technique. In NN, weight decay adds a penalty to the cost function, which causes the weights to "decay" (decrease).
Assign model accuracy and loss to respective variables.
Compile the model.



train BERT model with training and validation sets

```
[16]: # fit the model
      history = model.fit(
          train_set,
          validation_data=val,
          epochs=3)

      Epoch 1/3
      95/95 [==============================] - 904s 10s/step - loss: 0.7100 - accuracy: 0.5572 - val_loss: 0.5897 - val_accuracy: 0.7402
      Epoch 2/3
      95/95 [==============================] - 901s 9s/step - loss: 0.4313 - accuracy: 0.8155 - val_loss: 0.3347 - val_accuracy: 0.8359
      Epoch 3/3
      95/95 [==============================] - 902s 9s/step - loss: 0.2506 - accuracy: 0.9049 - val_loss: 0.3225 - val_accuracy: 0.8633

[18]: model.save_weights('./sentiment-analysis-on-movie-reviews/bert_weights.h5')

[19]: model.load_weights('./sentiment-analysis-on-movie-reviews/bert_weights.h5')
```

Training of the model.

The model is run for 3 epochs (a complete run-through of the model's processing), the time spent can be seen.

Loss, a numerical representation of wrong predictions.

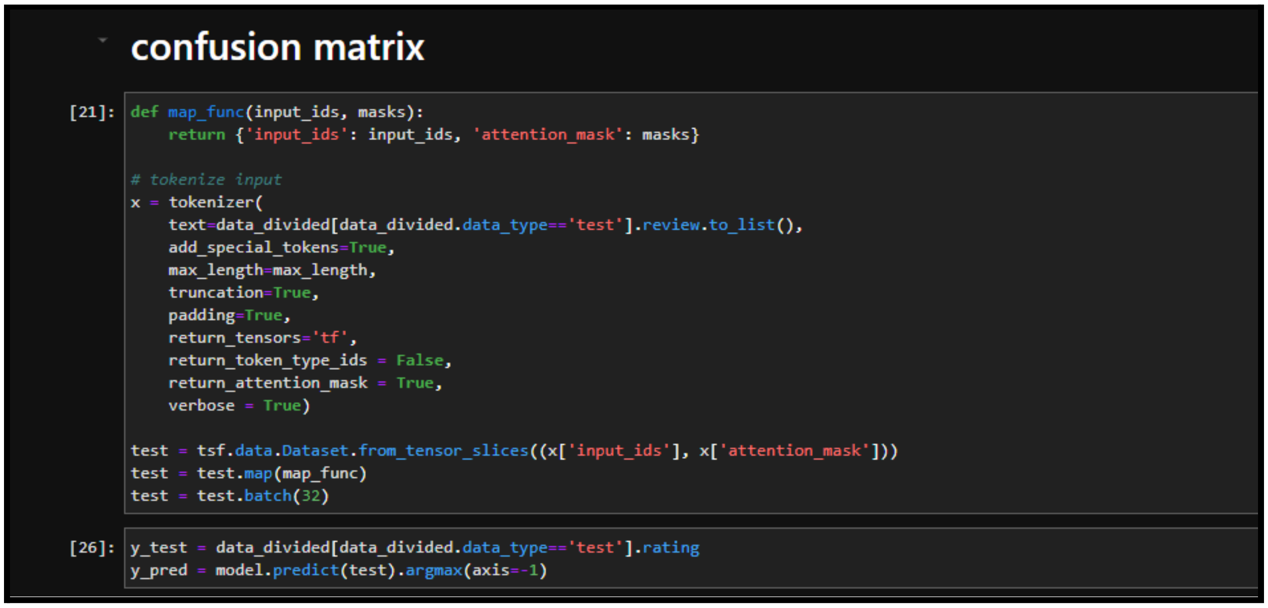
Accuracy increases from 55.72% in the first epoch to an amazing 90.49% in the third epoch.

Loss in validation keeps decreasing from 58.97% to 32.25%.

Validation accuracy goes from 74.02% to 86.33%.

Validation set is a set used in the training step to test how well the model performs on new data. A good way to detect overfitting.

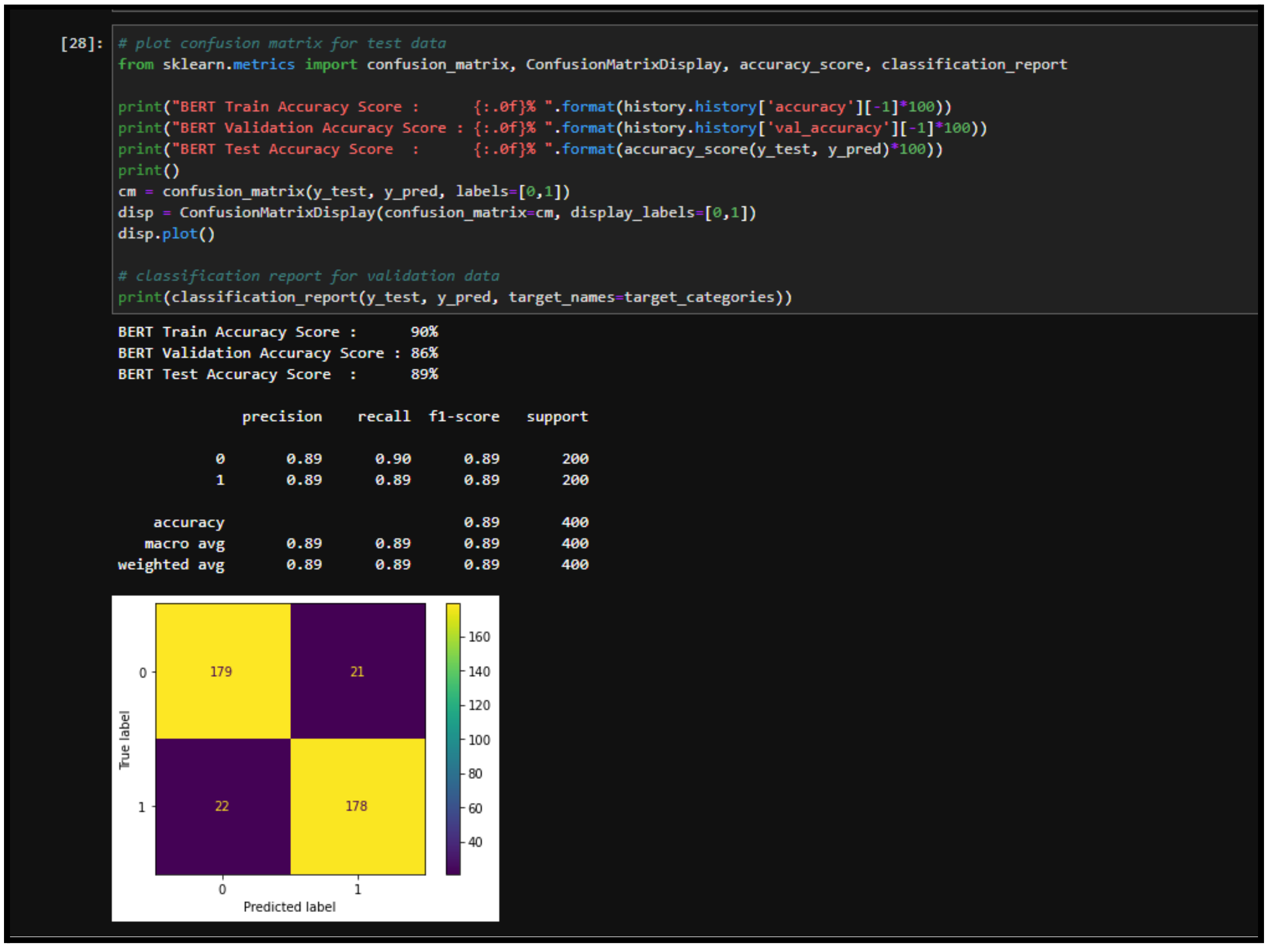
Save weights and export the files, but since we are doing transfer learning or model reroll, they are not used here.



confusion matrix

```
[21]: def map_func(input_ids, masks):
          return {'input_ids': input_ids, 'attention_mask': masks}

      # tokenize input
      x = tokenizer(
          text=data_divided[data_divided.data_type=='test'].review.to_list(),
          add_special_tokens=True,
          max_length=max_length,
          truncation=True,
          padding=True,
          return_tensors='tf',
          return_token_type_ids = False,
          return_attention_mask = True,
          verbose = True)

      test = tsf.data.Dataset.from_tensor_slices((x['input_ids'], x['attention_mask']))
      test = test.map(map_func)
      test = test.batch(32)

[26]: y_test = data_divided[data_divided.data_type=='test'].rating
      y_pred = model.predict(test).argmax(axis=-1)
```

Tokenize the test dataset for plotting the confusion matrix.



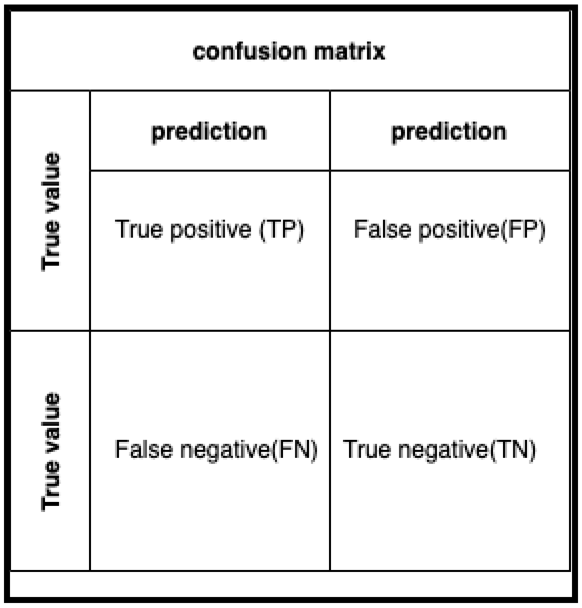
```
[28]: # plot confusion matrix for test data
      from sklearn.metrics import confusion_matrix, ConfusionMatrixDisplay, accuracy_score, classification_report

      print("BERT Train Accuracy Score :      {:.0f}% ".format(history.history['accuracy'][-1]*100))
      print("BERT Validation Accuracy Score : {:.0f}% ".format(history.history['val_accuracy'][-1]*100))
      print("BERT Test Accuracy Score  :      {:.0f}% ".format(accuracy_score(y_test, y_pred)*100))
      print()
      cm = confusion_matrix(y_test, y_pred, labels=[0,1])
      disp = ConfusionMatrixDisplay(confusion_matrix=cm, display_labels=[0,1])
      disp.plot()

      # classification report for validation data
      print(classification_report(y_test, y_pred, target_names=target_categories))
```

```
BERT Train Accuracy Score :      90%
BERT Validation Accuracy Score : 86%
BERT Test Accuracy Score  :      89%

              precision    recall  f1-score   support

           0       0.89      0.90      0.89       200
           1       0.89      0.89      0.89       200

    accuracy                           0.89       400
   macro avg       0.89      0.89      0.89       400
weighted avg       0.89      0.89      0.89       400
```

Model performance results and visualization.
Includes accuracy scores for training, validation and test datasets.

Results & performances comparison

Now we have results from both models, we can compare their performances on the same task. To understand the differences between the 4 metrics, the confusion matrix needs to be discussed.



confusion matrix		
	prediction	prediction
True value	True positive (TP)	False positive(FP)
True value	False negative(FN)	True negative(TN)

Accuracy = $\frac{TN+TP}{TN+TP+FN+FP}$

Accuracy is the most straightforward and brute-force measure for a model's performance. All correct predictions divided by all predictions. However , accuracy is not the best measure, especially when the dataset is unbalanced. Number of entries in each class is different from one another. A small class could be wholly misclassified and the model's accuracy can still be high.

There are some more nuanced and detailed metrics which can help us decide exactly how well the model really performs.

Precision is also called positive predictive value.

Precision =$\frac{TP}{TP+FP}$

Number of correct positive predictions divided by sum of correct positive and wrong positive predictions. Is a classifier is good enough, there will be 0 FP, in that case, Precision = $\frac{TP}{TP} = 1$

Recall is also known as sensitivity or true positive rate.

Recall = $\frac{TP}{TP+FN}$

Number of correct positive predictions divided by sum of correct positive and wrong negative predictions. Similar to precision, if a model is good enough, recall =1, meaning that there's no wrong predictions for the negative class.

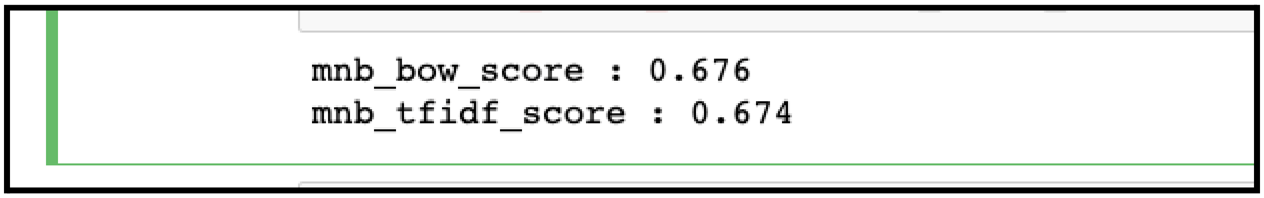
F1 score is a measure that takes both recall and precision into account.

F1 = $2 \times \frac{precision \times recall}{precision+recall}$

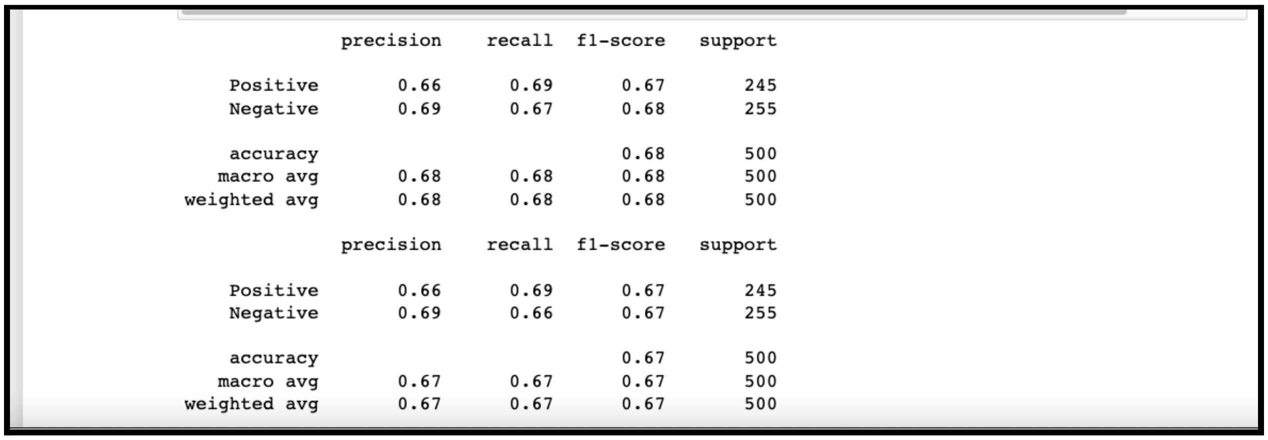
F1 score is a direct reflection of precision and recall, if precision and recall are both 1, F1 score also becomes 1, making the model extraordinarily good. F1 is directly proportional to both recall and precision, if one or both increase, F1 increases, if one or both decrease, F1 decreases. F1 score is better compared to accuracy in cases of unbalanced data, and F1 score tries to find a balance between recall and precision, which are more detailed and accurate measures.

Now let's look at the performance metrics.

Naive Bayes

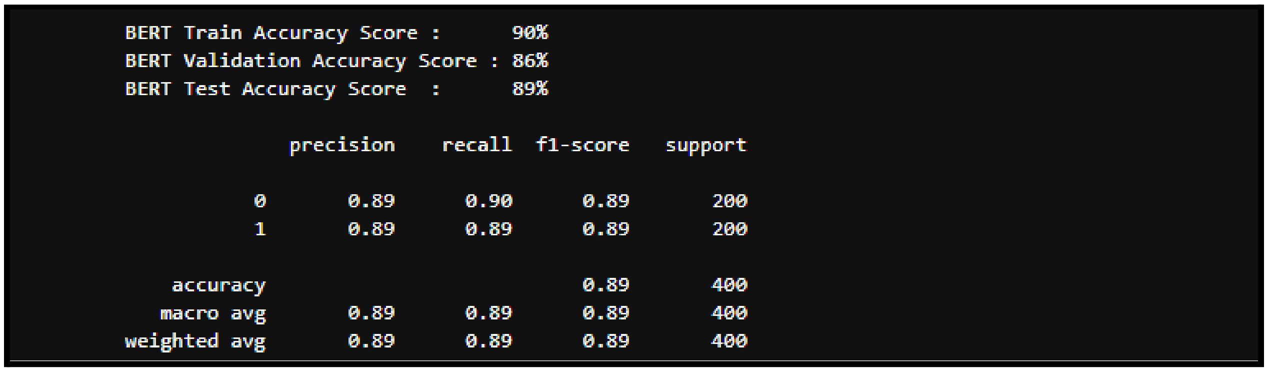


```
mnb_bow_score : 0.676
mnb_tfidf_score : 0.674
```



```
              precision    recall  f1-score   support

    Positive       0.66      0.69      0.67       245
    Negative       0.69      0.67      0.68       255

    accuracy                           0.68       500
   macro avg       0.68      0.68      0.68       500
weighted avg       0.68      0.68      0.68       500
              precision    recall  f1-score   support

    Positive       0.66      0.69      0.67       245
    Negative       0.69      0.66      0.67       255

    accuracy                           0.67       500
   macro avg       0.67      0.67      0.67       500
weighted avg       0.67      0.67      0.67       500
```

BERT



```
BERT Train Accuracy Score :      90%
BERT Validation Accuracy Score : 86%
BERT Test Accuracy Score  :      89%

              precision    recall  f1-score   support

           0       0.89      0.90      0.89       200
           1       0.89      0.89      0.89       200

    accuracy                           0.89       400
   macro avg       0.89      0.89      0.89       400
weighted avg       0.89      0.89      0.89       400
```

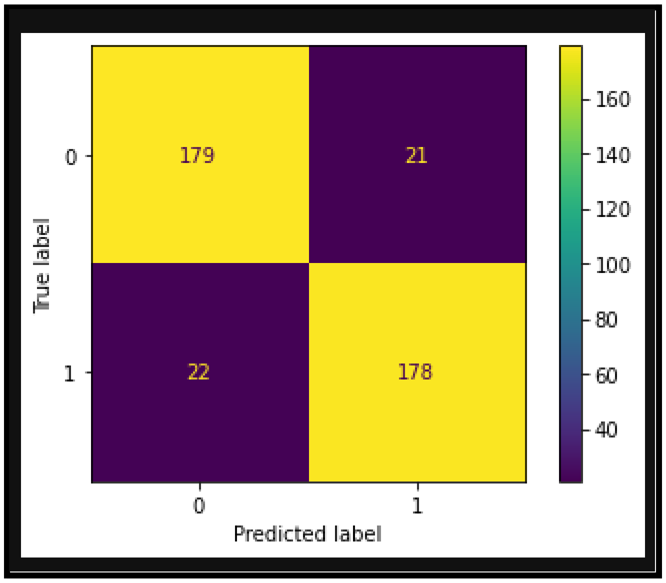
As shown in the performance metrics, BERT has better scores in all metrics.

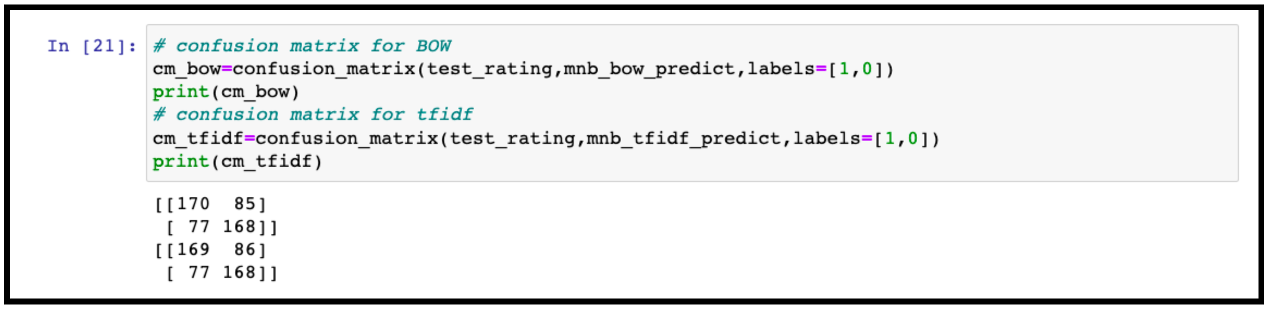
Naive Bayes has better accuracy for its BOW features than the TF IDF features.

Naive bayes has better precision for both features while classifying the negative class. It has a better recall for positive class. Its f1 scores are not very different for both features, and it's also even for both classes.

BERT has the same precision and F1 score for both negative and positive classes. For its recall, it obviously performs better in negative classes.

In both models' confusion matrices , we can see the number of correct and wrong predictions for both negative and positive classes and it gives us a better idea of their performance metrics.





```
In [21]: # confusion matrix for BOW
         cm_bow=confusion_matrix(test_rating,mnb_bow_predict,labels=[1,0])
         print(cm_bow)
         # confusion matrix for tfidf
         cm_tfidf=confusion_matrix(test_rating,mnb_tfidf_predict,labels=[1,0])
         print(cm_tfidf)

         [[170  85]
          [ 77 168]]
         [[169  86]
          [ 77 168]]
```

Conclusion

As shown in the results & performance comparison section, BERT has a far better performance than Multinomial Naive Bayes on this sentiment classification task. It stays true to the general case, as BERT was developed by Google researchers for Google's search engines. And BERT was also pretrained on a huge language dataset, so that naturally gives BERT a substantial edge compared to traditional models. However, that is not to say Naive Bayes is a bad model, it actually has a fairly decent performance while competing with other

traditional models like linear regression, logistic regression, decision tree and KNN. One thing worth noting though, is that being a heavy deep learning model, BERT has very high hardware requirements and a much longer runtime, integrated gpus are out of consideration if one's trying to build and run a BERT model.

In conclusion, BERT is one of the best deep learning models for sentiment analysis tasks, however its heavy hardware requirements and runtime makes it inaccessible in some ways. That's where traditional models like Naive Bayes shine, even though it has a poorer performance on the task, its simplicity and easiness to program and light hardware requirement still makes it popular.

References

[1] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova, "BERT: Pre-training of deep bidirectional Transformers for language understanding," arXiv [cs.CL], 2018.