In [ ]:

```
# !wget http://cs.stanford.edu/people/alecmgo/trainingandtestdata.zip
# !unzip trainingandtestdata.zip
```

These code snippets are for importing a data set from a file, reading it into a pandas DataFrame, and creating some summary statistics about the data.

The data set is in a file called 'training.1600000.processed.noemoticon.csv', and it has six columns with the following names: 'target', 'ids', 'date', 'flag', 'user', and 'text'. The 'target' column contains the label for each tweet, with a value of 0 indicating a negative tweet and a value of 4 indicating a positive tweet.

The code reads the data from the file into a pandas DataFrame, and then selects a subset of the data by slicing the DataFrame with the .iloc[] method. The selected subset is the rows with indices between 790000 and 810000 (inclusive).

In [ ]:

```
import numpy as np
import pandas as pd
from sklearn.metrics import confusion_matrix
from sklearn.metrics import accuracy_score
from sklearn.metrics import classification_report
from sklearn.metrics import accuracy_score
from sklearn.feature_extraction.text import CountVectorizer
import pickle
# import feather


cols = ['target', 'ids', 'date', 'flag', 'user', 'text']
df = pd.read_csv('training.1600000.processed.noemoticon.csv', sep=',',names=cols,encoding='la

df.loc[df['target'] == 4, 'target'] = 1

target = df.target
text = df.text
```

Explore the data by printing some sample rows and examining the structure of the DataFrame. I used the head to get an overview of the data

In [ ]:

```
df.head()
```

Out[ ]:

	target	ids	date	flag	user	text
500000	0	2186710671	Mon Jun 15 19:13:35 PDT 2009	NO_QUERY	xxLOVExxPEACE	i cant sleep
500001	0	2186710748	Mon Jun 15 19:13:35 PDT 2009	NO_QUERY	dvanulya	@alba17 Sorry about kid situation. Good luck w...
500002	0	2186710897	Mon Jun 15 19:13:36 PDT 2009	NO_QUERY	byhuy	nhá» nhÃ quÃ¡!!! cá»© má» i láº§n nghe bÃ i ...
500003	0	2186711047	Mon Jun 15 19:13:37 PDT 2009	NO_QUERY	Chi_lanta	Missing Him!! Twitter Me RED?? What the heck i...
500004	0	2186711267	Mon Jun 15 19:13:38 PDT 2009	NO_QUERY	IBEChillin	#musicmonday i got the blues today ***sad ...

The .value_counts() method was used to compute the number of occurrences of each unique value in the 'target' column of the DataFrame. This produces a count of the number of positive and negative tweets in the selected subset of the data.

In [ ]:
```
pd.isnull(df).sum()
```

Out[ ]:
```
target    0
ids       0
date      0
flag      0
user      0
text      0
dtype: int64
```

Q1

Explore and prepare the data (Tokenization, Stemming, Stopwords, visualization, etc.)

To remove URLs from tweets while preprocessing the text, Regular expression used to identify and remove any string that matches the pattern of a URL. re module was used to perform regular expression matching

This function uses the re.sub() function to search for any strings that match the pattern http\S+ and replace them with an empty string. The \S character class matches any non-whitespace character, and the + quantifier indicates that one or more of these characters should be matched. This will remove the URL from the tweet and leave the rest of the text unchanged.

In [ ]:
```
# remove urls
import re
def remove_url(text):
    url = re.compile(r'https?://\S+|www\.\S+|\d+')
    return url.sub(r'',text.lower())

text = text.apply(lambda x : remove_url(x))
```

Tokenize the text of the tweets using a suitable tokenization method. I can also consider using regular expressions or a custom tokenization function if necessary.

The first step in the preprocessing is tokenization, which involves splitting the text into individual words or "tokens". This is done using the word_tokenize() function from the nltk.tokenize module, and the resulting list of tokens is stored in the DataFrame.

The nltk.download() function is used to download two resources from the Natural Language Toolkit (nltk) library: the punkt tokenizer model and the stopwords list. The punkt model is used by the word_tokenize() function to identify the boundaries between tokens, and the stopwords list contains a list of common words that are usually removed from text data as part of the preprocessing step.

The tokenized text is then saved to a file called 'tokenized.pkl' using the to_pickle() method.

In [ ]:
```
#Tokenization

import nltk
from nltk.tokenize import word_tokenize

nltk.download('punkt')
nltk.download('stopwords')
```

```
text = text.apply(word_tokenize)

!mkdir data
text.to_pickle('data/tokenized.pkl')
```

```
[nltk_data] Downloading package punkt to
[nltk_data]     C:\Users\darklane\AppData\Roaming\nltk_data...
[nltk_data]   Package punkt is already up-to-date!
[nltk_data] Downloading package stopwords to
[nltk_data]     C:\Users\darklane\AppData\Roaming\nltk_data...
[nltk_data]   Package stopwords is already up-to-date!
A subdirectory or file data already exists.
```

The next step in the preprocessing is to remove stopwords and punctuations from the text. The list of stopwords is obtained from the nltk library, and a list of punctuation symbols is created using the string.punctuation attribute.

The text is then stemmed using the Porter stemmer algorithm, which converts each word to its base form by removing common suffixes. The stemmed words are then stored in a file called 'stemmed.pkl' using the to_pickle() method.

In [ ]:
```
#remove stopwords and punctuations
text = pd.read_pickle('data/tokenized.pkl')
from nltk.corpus import stopwords
stop_words = set(stopwords.words('english'))

from nltk.stem import PorterStemmer
ps = PorterStemmer()

#Stemming
import string
punctuations = list(string.punctuation)

text = text.apply(lambda x: [ps.stem(item) for item in x if item not in stop_words and item n

text.to_pickle('data/stemmed.pkl')
```

Visualize the data by plotting word frequencies or creating word clouds. This gives a sense of the most common words in the dataset and help to identify patterns and trends.

In [ ]:
```
# visualization

import matplotlib.pyplot as plt
import seaborn as sns

#wordcloud
# %pip install wordcloud
from wordcloud import WordCloud

all_words = ' '.join([text for text in df['text']])
wordcloud = WordCloud(width=800, height=500, random_state=21, max_font_size=110).generate(all_

plt.figure(figsize=(10, 7))
plt.imshow(wordcloud, interpolation="bilinear")
plt.axis('off')
plt.show()
```


```
In [ ]: text = pd.read_pickle('data/stemmed.pkl')
        text = text.apply(lambda x: " ".join([word for word in x]))
        text.to_pickle('data/cleaned.pkl')
        text.head(5)
```

```
Out[ ]: 500000                                           cant sleep
        500001    alba sorri kid situat good luck vid sorri 's g...
        500002    nhá »   nhã quã¡ cá » © má »  i láº§n nghe bã ...
        500003              miss twitter red heck girl todo sad face
        500004                        musicmonday got blue today sad
        Name: text, dtype: object
```

Q2

Build a BOW and train a KNN, Decision Tree, and SVM model

This is creating a bag of words (BOW) representation of text data stored in a pandas DataFrame called 'text'. A BOW representation is a way of encoding text data as numerical values, which can be used as input to machine learning models.

To create the BOW representation, the code first imports the CountVectorizer class from the feature_extraction.text module of the scikit-learn library. It then creates an instance of the CountVectorizer class and fits it to the 'text' DataFrame using the fit_transform method. The resulting BOW representation is stored in a variable called 'Text'.

The CountVectorizer class has several parameters that can be adjusted to control how the BOW representation is created. In this case, the 'max_features' parameter is set to 1100, which limits the number of features (i.e., unique words or n-grams) to consider in the BOW representation. The 'ngram_range' parameter is set to (1, 3), which indicates that the BOW representation should include 1-grams (individual words), 2-grams (pairs of words), and 3-grams (triplets of words).

Finally, the BOW representation is converted to an array using the toarray method and then cast to the 'int8' data type using the astype method.

In [ ]:
```
#Building a BOW

# Vectorize the text data into a bag of words representation
vectorizer = CountVectorizer(max_features=1100, ngram_range=(1,3))
Text = vectorizer.fit_transform(text)
Text = Text.toarray().astype('bool')
```

In [ ]:
```
# feather.write_dataframe(pd.DataFrame(Text), 'data/Text.feather')
```

It's worth noting that when working with unbalanced data, it can be helpful to use stratified sampling to ensure that the training and test sets have a similar class distribution to the original dataset. To do this, stratify parameter was passed to train_test_split().

This will ensure that the training and test sets have a similar class distribution to the original dataset, which can be especially important when working with unbalanced data.

In [ ]:
```
# import feather
# Text = feather.read_dataframe('data/Text.feather').to_numpy()
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(Text, target, test_size=0.1, random_state
```

Training Models

The KNN model is trained using the KNeighborsClassifier class from the neighbors module of the scikit-learn library. The classifier is initialized with the parameter 'n_neighbors' set to 2, which specifies the number of nearest neighbors to consider when making predictions. The fit method is then used to train the model on the training data stored in the 'X_train' and 'y_train' variables. The 'X_train' variable contains the feature data, while the 'y_train' variable contains the labels.

In [ ]:
```
# KNN

from sklearn.neighbors import KNeighborsClassifier

knn = KNeighborsClassifier(n_neighbors = 2)
knn.fit(X_train, y_train)
pickle.dump(knn, open('data/knn.pkl', 'wb'))
```

The SVM model is trained using the LinearSVC class from the svm module of the scikit-learn library. The classifier is initialized with the parameter 'C' set to 0.9, which controls the regularization strength of the model. The fit method is then used to train the model on the training data stored in the 'X_train' and 'y_train' variables.

In [ ]:
```
from sklearn.svm import LinearSVC

# Train a linear SVM classifier
svm = LinearSVC(C=0.9, random_state=0)
svm.fit(X_train, y_train)
pickle.dump(svm, open('data/svm.pkl', 'wb'))
```

The XGBoost model is trained using the XGBClassifier class from the xgboost library. The classifier is initialized with default parameters and the fit method is used to train the model on the training data stored in the 'X_train' and 'y_train' variables.

In [ ]:
```
from xgboost import XGBClassifier
# model = XGBClassifier(tree_method='gpu_hist')
model = XGBClassifier()
model.fit(X_train,y_train)
pickle.dump(knn, open('data/dt.pkl', 'wb'))
```

Q3

Evaluate the above models (confusion matrix, accuracy, classification report, etc.)

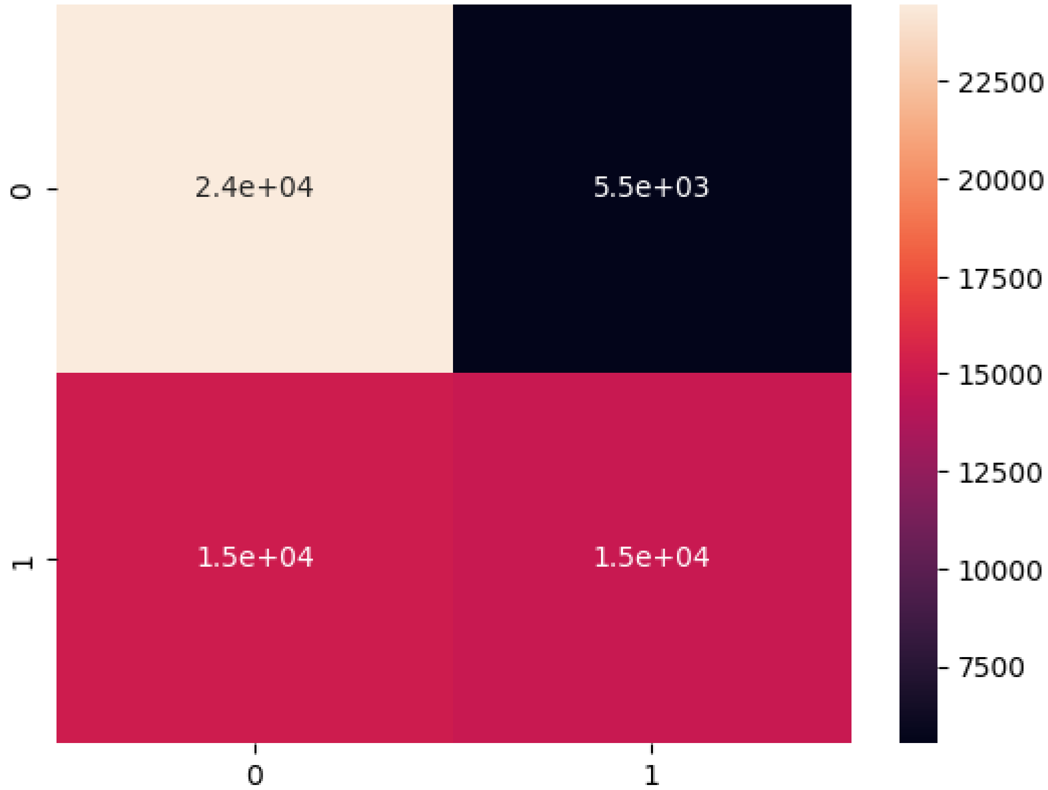
Below code evaluate the performance of models on the test dataset. It first calculates and prints the accuracy of the model using the accuracy_score function from the metrics module of the scikit-learn library. The accuracy is calculated by comparing the predicted labels stored in the 'y_pred_knn','y_pred_svm' and 'y_pred_dt' variable to the true labels stored in the 'y_test' variable.

It then uses the confusion_matrix function from the metrics module to calculate a confusion matrix, which is a table that shows the number of correct and incorrect predictions made by the model. The confusion matrix is visualized using the heatmap function from the seaborn library. The heatmap function also displays the annotation of the matrix values.

Finally, it a classification report using the classification_report function from the metrics module. The classification report includes several evaluation metrics, such as precision, recall, and f1-score, for each class in the dataset. These metrics can provide a more detailed understanding of the model's performance

In [ ]:
```
# KNN
knn = pickle.load(open('data/knn.pkl', 'rb'))
y_pred_knn = knn.predict(X_test)

print("\nAccuracy-",accuracy_score(y_test, y_pred_knn),'\n')
cm = confusion_matrix(y_test, y_pred_knn)
sns.heatmap(cm, annot=True)
print(classification_report(y_test,y_pred_knn))
```

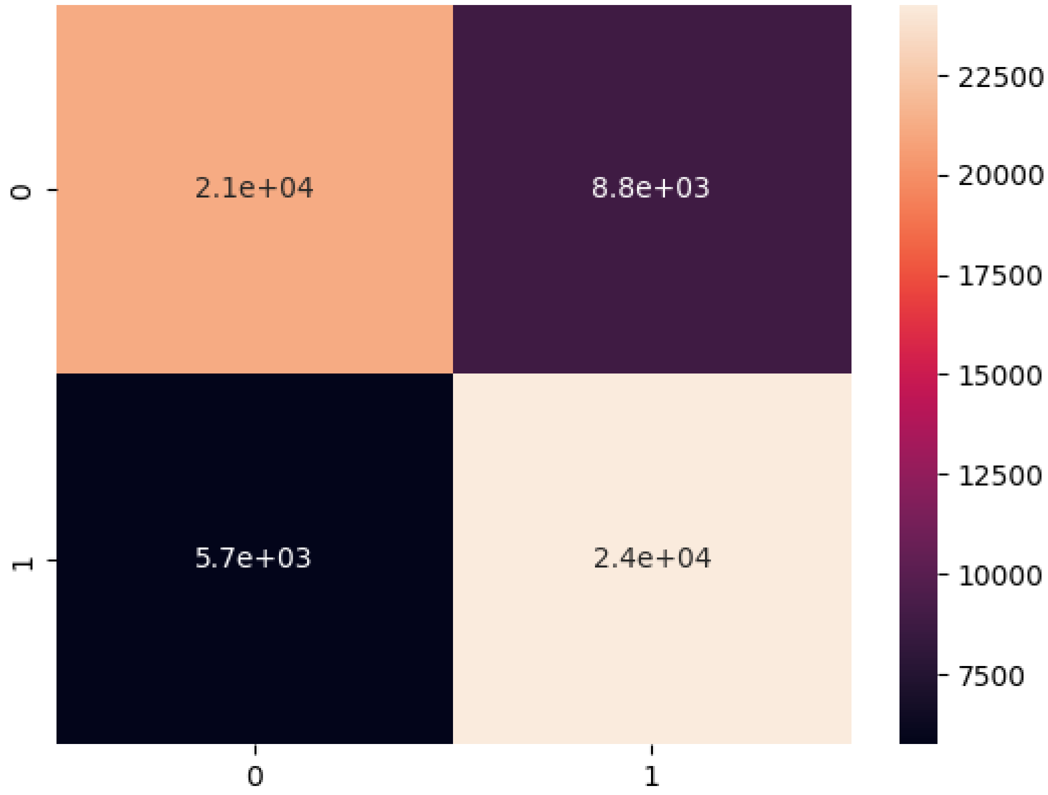
```
Accuracy- 0.6559

              precision    recall  f1-score   support

           0       0.62      0.82      0.70     30000
           1       0.73      0.50      0.59     30000

    accuracy                           0.66     60000
   macro avg       0.67      0.66      0.65     60000
weighted avg       0.67      0.66      0.65     60000
```



In [ ]:

```
#SVM

# Predict the labels for new data
modelSvm = pickle.load(open('data/svm.pkl', 'rb'))
y_pred_svm = modelSvm.predict(X_test)

print("\nAccuracy-",accuracy_score(y_test, y_pred_svm),'\n')
cm = confusion_matrix(y_test, y_pred_svm)
sns.heatmap(cm, annot=True)
print(classification_report(y_test,y_pred_svm))
```

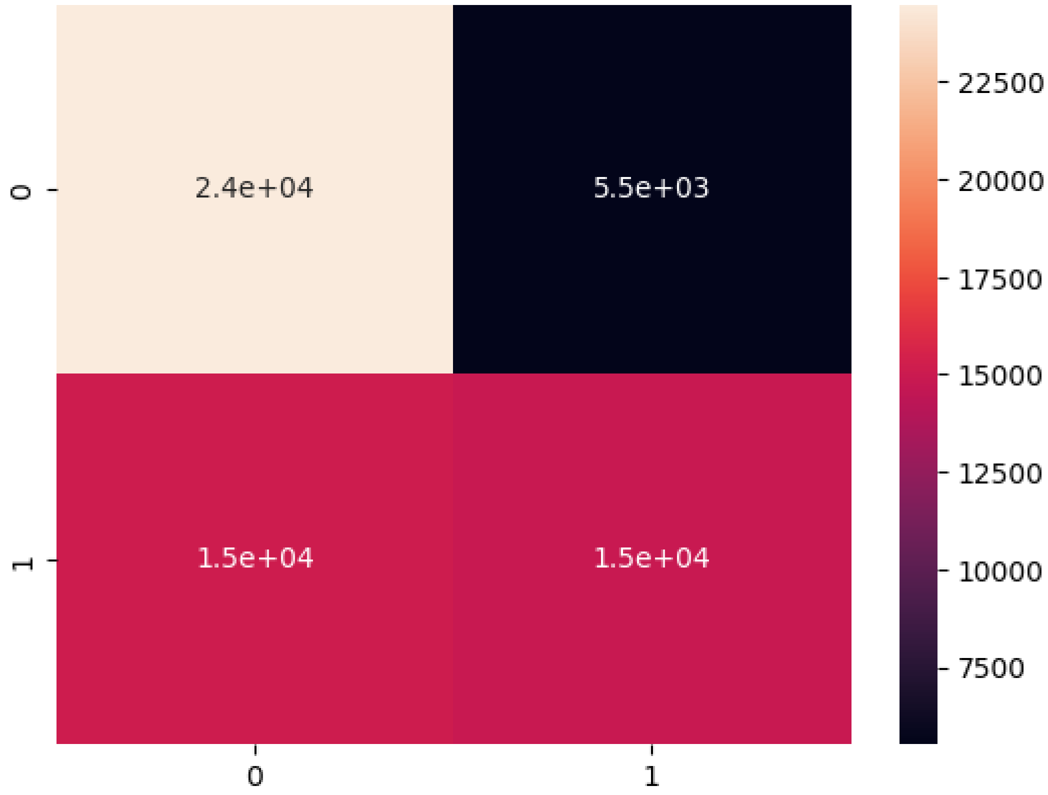
```
Accuracy- 0.75775

              precision    recall  f1-score   support

           0       0.79      0.71      0.74     30000
           1       0.73      0.81      0.77     30000

    accuracy                           0.76     60000
   macro avg       0.76      0.76      0.76     60000
weighted avg       0.76      0.76      0.76     60000
```



In [ ]:

```
#DecisionTree
modelKnn = pickle.load(open('data/dt.pkl', 'rb'))
y_pred_dt = modelKnn.predict(X_test)

print("\nAccuracy-",accuracy_score(y_test, y_pred_dt),'\n')
cm = confusion_matrix(y_test, y_pred_dt)
sns.heatmap(cm, annot=True)
print(classification_report(y_test,y_pred_dt))
```

```
Accuracy- 0.6559

              precision    recall  f1-score   support

           0       0.62      0.82      0.70     30000
           1       0.73      0.50      0.59     30000

    accuracy                           0.66     60000
   macro avg       0.67      0.66      0.65     60000
weighted avg       0.67      0.66      0.65     60000
```



Text Classification Using Convolutional Neural Networks

In [ ]:
```
# !wget http://nlp.stanford.edu/data/glove.twitter.27B.zip
# !wget http://cs.stanford.edu/people/alecmgo/trainingandtestdata.zip
# !unzip trainingandtestdata.zip
# !unzip glove.twitter.27B.zip
```

Q4

Use one of the word embeddings (word2vec, Glove, fasText) and build a CNN model

The above code imports TensorFlow and sets the number of threads used for inter-op parallelism to 8. It also imports various functions and classes from the keras library, including Sequential, Dense, Dropout, Activation, Embedding, Conv1D, MaxPooling1D, and Flatten. It also imports sequence and text from keras.preprocessing.

In [ ]:
```
import tensorflow as tf
tf.config.threading.set_inter_op_parallelism_threads(8)
from keras.preprocessing import sequence, text
from keras.models import Sequential
from keras.layers import Dense, Dropout, Activation, Embedding, Conv1D, MaxPooling1D, Flatten
from keras_preprocessing.sequence import pad_sequences
import numpy as np
import pandas as pd
```

The code then reads a CSV file called 'training.1600000.processed.noemoticon.csv' using pandas, and stores the resulting DataFrame in a variable called df. The cols list specifies the names of the columns in the CSV file, which are used as the column labels in the resulting DataFrame. The tweets variable is assigned the values in the 'text' column of the DataFrame, and the target variable is assigned the values in the 'target' column.

In [ ]:
```
cols = ['target', 'ids', 'date', 'flag', 'user', 'text']
df = pd.read_csv('training.1600000.processed.noemoticon.csv', sep=',',names=cols,encoding='la
# df= df.iloc[790000:810000]

tweets = df['text']
target = df['target']
del df
```

The target values are then mapped to 0 or 1 using a dictionary, where the value 0 is mapped to 0 and the value 4 is mapped to 1. Finally, the target.value_counts() function is used to print the counts of the unique values in the target column.

In [ ]:
```
print(target.value_counts())
target = target.map({0:0, 4:1}).astype('int8')
```

```
0    800000
4    800000
Name: target, dtype: int64
```

The below code imports the gensim library and applies the simple_preprocess function to the tweets series, which tokenizes the tweets and lowercases them. It then uses the train_test_split function from scikit-learn to split the tweets and target variables into training and testing sets, with a test size of 10%.

In [ ]:
```
import gensim
tweets = tweets.apply(gensim.utils.simple_preprocess)
tweets.head

from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(tweets, target, test_size=0.1, random_sta
```

The code then sets several hyperparameters for the model, including the vocabulary size, maximum length of the input sequences, embedding dimension, batch size, and number of epochs. It also sets the number of filters, kernel size, and hidden dimensions for the Convolutional Neural Network (CNN) that will be used in the model.

In [ ]:
```
# set parameters:
vocab_size = 1000
max_length = 1000 # optimal 1000
embedding_dim = 100
batch_size = 32
epochs = 6 # optimal 10
filters = 16
kernel_size = 3
hidden_dims = 250
```

The code then initializes a Tokenizer object with the specified vocabulary size and fits it on the training data. It then converts the training data into sequences of integers using the texts_to_sequences method and pads the sequences to the maximum length using the pad_sequences function. The padded sequences are then converted to int16 type.

In [ ]:
```
tokenizer = text.Tokenizer(num_words=vocab_size)
tokenizer.fit_on_texts(X_train)
```

```
X_train = tokenizer.texts_to_matrix(X_train)
X_test = tokenizer.texts_to_matrix(X_test)
```

In [ ]:
```
X_train = pad_sequences(X_train, maxlen=max_length).astype('int16')
X_test = sequence.pad_sequences(X_test, maxlen=max_length).astype('int16')
```

The below code initializes an empty dictionary called Embedding_index and opens the file 'glove.twitter.27B.100d.txt', which is a file containing pre-trained GloVe word embeddings. The code then reads the file line by line, splits each line into a list of values using the split method, and assigns the first element of the list (the word) to the word variable and the rest of the elements (the word embeddings) to the coefs variable. The coefs variable is then converted to a NumPy array using the asarray function. The Embedding_index dictionary is then updated with the word and its corresponding word embeddings. The file is then closed.

In [ ]:
```
Embedding_index = {}
f = open('glove.twitter.27B.100d.txt', encoding='utf-8')
for line in f:
    values = line.split()
    word = values[0]
    coefs = np.asarray(values[1:], dtype='float32')
    Embedding_index[word] = coefs
f.close()
```

The code then initializes a NumPy array called Embedding_matrix with dimensions vocab_size by embedding_dim and fills it with zeros. It then iterates over the items in the tokenizer.word_index dictionary, which maps words to their indices in the vocabulary. For each word, the code checks if its index is less than the vocabulary size. If it is, it looks up the word in the Embedding_index dictionary to get its word embeddings, and assigns these embeddings to the corresponding row in the Embedding_matrix array. If the index is greater than or equal to the vocabulary size, the loop breaks.

In [ ]:
```
Embedding_matrix = np.zeros((vocab_size, embedding_dim))
for word, i in tokenizer.word_index.items():
    if i > vocab_size - 1:
        break
    else:
        embedding_vector = Embedding_index.get(word)
        if embedding_vector is not None:
            Embedding_matrix[i] = embedding_vector
```

In [ ]:
```
del Embedding_index, tokenizer
del   tweets, target
```

The above code defines a CNN model using the Sequential class from Keras. The model consists of an Embedding layer, followed by two Conv1D layers, two MaxPooling1D layers, a Flatten layer, a Dense layer with hidden_dims units and a ReLU activation function, a Dropout layer with a rate of 0.5, and a final Dense layer with a single unit and a sigmoid activation function. The Embedding layer takes the vocabulary size, embedding dimension, and input length as arguments, and is initialized with the Embedding_matrix array and set to not be trainable. The Conv1D layers have a specified number of filters and kernel size, and use a 'valid' padding. They use a 'relu' activation function. The MaxPooling1D layers have no arguments. The Dense layers have the specified number of units and use a 'relu' or 'sigmoid' activation function. The Dropout layer has a rate of 0.5.

In [ ]:
```
model = Sequential()
```

```
# model.add(Embedding(vocab_size, embedding_dim, input_length=max_length ))
# model.add(Dropout(0.5))


model.add(Embedding(vocab_size, embedding_dim, input_length=max_length, trainable=False, weig
model.add(Conv1D(filters, kernel_size, padding='valid', activation='relu'))
model.add(MaxPooling1D())
model.add(Conv1D(filters, kernel_size, padding='valid', activation='relu'))
model.add(MaxPooling1D())
model.add(Flatten())
model.add(Dense(hidden_dims, activation='relu'))
model.add(Dropout(0.5))
model.add(Dense(1, activation='sigmoid'))
```

The model is then compiled with a 'binary_crossentropy' loss function, the 'adam' optimizer, and the 'accuracy' metric. It is then trained on the training data using the fit method, with the specified number of epochs and batch size, and the validation data is passed as an argument.

In [ ]:

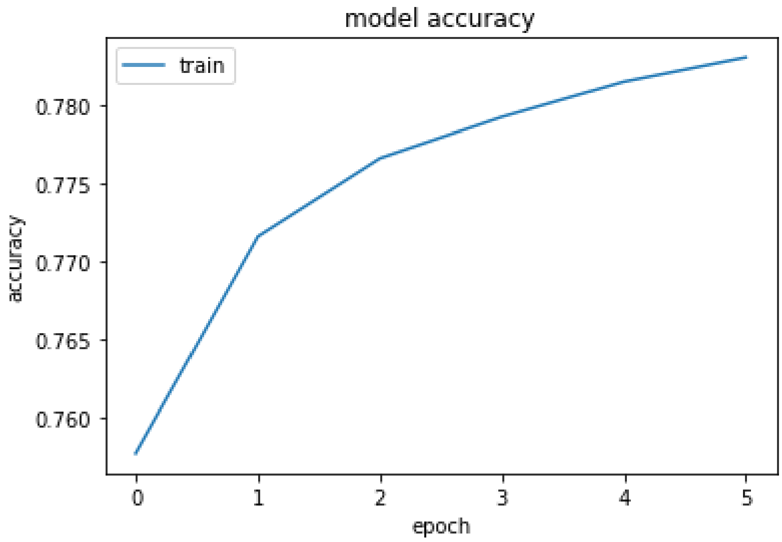
```
model.compile(loss='binary_crossentropy', optimizer='adam', metrics=['accuracy'])
history = model.fit(X_train, y_train, validation_data=(X_test, y_test), epochs=epochs, batch_
```

```
2022-12-23 18:55:02.699826: I tensorflow/compiler/mlir/mlir_graph_optimization_pass.cc:185] N
one of the MLIR Optimization Passes are enabled (registered 2)
Epoch 1/6
45000/45000 [==============================] - 1132s 25ms/step - loss: 0.4960 - accuracy: 0.7
577 - val_loss: 0.4701 - val_accuracy: 0.7733
Epoch 2/6
45000/45000 [==============================] - 1091s 24ms/step - loss: 0.4756 - accuracy: 0.7
716 - val_loss: 0.4650 - val_accuracy: 0.7782
Epoch 3/6
45000/45000 [==============================] - 1114s 25ms/step - loss: 0.4685 - accuracy: 0.7
766 - val_loss: 0.4622 - val_accuracy: 0.7787
Epoch 4/6
45000/45000 [==============================] - 1106s 25ms/step - loss: 0.4638 - accuracy: 0.7
793 - val_loss: 0.4611 - val_accuracy: 0.7800
Epoch 5/6
45000/45000 [==============================] - 1130s 25ms/step - loss: 0.4605 - accuracy: 0.7
815 - val_loss: 0.4619 - val_accuracy: 0.7797
Epoch 6/6
45000/45000 [==============================] - 1101s 24ms/step - loss: 0.4583 - accuracy: 0.7
831 - val_loss: 0.4595 - val_accuracy: 0.7807
```

The below code uses the plot function from the matplotlib library (imported as plt) to generate a line plot of the accuracy of a model as it was trained over a series of epochs. The plot is given a title ('model accuracy') and labels for the y-axis ('accuracy') and x-axis ('epoch'). The history object being plotted is assumed to contain the training and test accuracy for each epoch. The legend function is used to specify that the line for the 'train' data should be labeled as such and the line for the 'test' data should be labeled as such. The show function is then called to display the plot.

In [ ]:

```
import matplotlib.pyplot as plt
plt.plot(history.history['accuracy'])
plt.title('model accuracy')
plt.ylabel('accuracy')
plt.xlabel('epoch')
plt.legend(['train', 'test'], loc='upper left')
plt.show()
```



The model is then used to make predictions on the testing data using the predict method, and the resulting predictions are stored in the y_pred variable.

In [ ]:
```
# prediction
y_pred = model.predict(X_test)
```

In [ ]:
```
# accuracy
from sklearn.metrics import accuracy_score
print(accuracy_score(y_test, y_pred.round()))
```

```
0.7806875
```

KNN - 0.66 SVM - 0.76 Desision Tree - 0.66

CNN - 78

:- By considering accuracy CNN is better than other three.