Python-Machine-Learning-Neural-Network-SMS-Text-Classifier

Author: Fran Panteli

1. Contents

2. Task Description

Note: You are currently reading this using Google Colaboratory which is a cloud-hosted version of Jupyter Notebook. This is a document containing both text cells for documentation and runnable code cells. If you are unfamiliar with Jupyter Notebook, watch this 3-minute introduction before starting this challenge: https://www.youtube.com/watch?v=inN8seMm7UI

In this challenge, you need to create a machine learning model that will classify SMS messages as either "ham" or "spam". A "ham" message is a normal message sent by a friend. A "spam" message is an advertisement or a message sent by a company.

You should create a function called `predict_message` that takes a message string as an argument and returns a list. The first element in the list should be a number between zero and one that indicates the likeliness of "ham" (0) or "spam" (1). The second element in the list should be the word "ham" or "spam", depending on which is most likely.

For this challenge, you will use the SMS Spam Collection dataset. The dataset has already been grouped into train data and test data.

The first two cells import the libraries and data. The final cell tests your model and function. Add your code in between these cells.

3. Importing the Data & Modules

```
# IMPORT MODULES
# %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

"""
     Project outline:
          -> We are creating a model which classifies messages into
"hams" and "spams"
          -> Spams are messages sent by companies or advertisements
          -> We are creating the `predict_message` function
               -> This takes a string as the argument and returns a
number
               -> The number is between 0 and 1 and tells us how
likely it is that the input string is a ham (0) or a spam (1)
          -> The dataset is the SMS Spam Collection dataset
               -> This has been grouped into train and test data
          -> The first two cells import the libraries and data -> the
last cell runs unit tests to test how good the predictions of our
function are

     Importing the modules:
          -> This cell imports modules / libraries
          -> TensorFlow, pandas (to manipulate the data)
          -> numpy, matplotlib, word cloud
          -> NLTK <- natural language toolkit
               -> Word tokenisation
          -> The string module
          -> Scikit learn <- for machine learning
               -> Random forest / grid search
          -> These modules are for natural language processing
          -> Pandas is for data manipulation
"""

try:
  # %tensorflow_version only exists in Colab.
  !pip install tf-nightly
except Exception:
  pass
import tensorflow as tf
import pandas as pd
from tensorflow import keras
!pip install tensorflow-datasets
import tensorflow_datasets as tfds
import numpy as np
import matplotlib.pyplot as plt
from wordcloud import WordCloud
import nltk
from nltk import word_tokenize
nltk.download('punkt')
```

```
import string
from nltk.corpus import stopwords
nltk.download('stopwords')
from sklearn.feature_extraction.text import CountVectorizer,
TfidfTransformer
from sklearn.model_selection import train_test_split, GridSearchCV
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import classification_report, confusion_matrix,
accuracy_score
from sklearn.svm import SVC
from xgboost import XGBRegressor

print(tf.__version__)
```

```
Requirement already satisfied: tf-nightly in
/usr/local/lib/python3.7/dist-packages (2.9.0.dev20220301)
Requirement already satisfied: gast<=0.4.0,>=0.2.1 in
/usr/local/lib/python3.7/dist-packages (from tf-nightly) (0.4.0)
Requirement already satisfied: grpcio<2.0,>=1.24.3 in
/usr/local/lib/python3.7/dist-packages (from tf-nightly) (1.44.0)
Requirement already satisfied: keras-preprocessing>=1.1.1 in
/usr/local/lib/python3.7/dist-packages (from tf-nightly) (1.1.2)
Requirement already satisfied: h5py>=2.9.0 in
/usr/local/lib/python3.7/dist-packages (from tf-nightly) (3.1.0)
Requirement already satisfied: keras-nightly~=2.9.0.dev in
/usr/local/lib/python3.7/dist-packages (from tf-nightly)
(2.9.0.dev2022030108)
Requirement already satisfied: tf-estimator-nightly~=2.9.0.dev in
/usr/local/lib/python3.7/dist-packages (from tf-nightly)
(2.9.0.dev2022030109)
Requirement already satisfied: flatbuffers>=1.12 in
/usr/local/lib/python3.7/dist-packages (from tf-nightly) (2.0)
Requirement already satisfied: numpy>=1.20 in
/usr/local/lib/python3.7/dist-packages (from tf-nightly) (1.21.5)
Requirement already satisfied: opt-einsum>=2.3.2 in
/usr/local/lib/python3.7/dist-packages (from tf-nightly) (3.3.0)
Requirement already satisfied: six>=1.12.0 in
/usr/local/lib/python3.7/dist-packages (from tf-nightly) (1.15.0)
Requirement already satisfied: wrapt>=1.11.0 in
/usr/local/lib/python3.7/dist-packages (from tf-nightly) (1.13.3)
Requirement already satisfied: typing-extensions>=3.6.6 in
/usr/local/lib/python3.7/dist-packages (from tf-nightly) (3.10.0.2)
Requirement already satisfied: libclang>=13.0.0 in
/usr/local/lib/python3.7/dist-packages (from tf-nightly) (13.0.0)
Requirement already satisfied: packaging in
/usr/local/lib/python3.7/dist-packages (from tf-nightly) (21.3)
Requirement already satisfied: protobuf>=3.9.2 in
/usr/local/lib/python3.7/dist-packages (from tf-nightly) (3.17.3)
Requirement already satisfied: google-pasta>=0.1.1 in
/usr/local/lib/python3.7/dist-packages (from tf-nightly) (0.2.0)
```

```
Requirement already satisfied: setuptools in
/usr/local/lib/python3.7/dist-packages (from tf-nightly) (57.4.0)
Requirement already satisfied: astunparse>=1.6.0 in
/usr/local/lib/python3.7/dist-packages (from tf-nightly) (1.6.3)
Requirement already satisfied: tensorflow-io-gcs-filesystem>=0.23.1 in
/usr/local/lib/python3.7/dist-packages (from tf-nightly) (0.24.0)
Requirement already satisfied: termcolor>=1.1.0 in
/usr/local/lib/python3.7/dist-packages (from tf-nightly) (1.1.0)
Requirement already satisfied: absl-py>=1.0.0 in
/usr/local/lib/python3.7/dist-packages (from tf-nightly) (1.0.0)
Requirement already satisfied: tb-nightly~=2.9.0.a in
/usr/local/lib/python3.7/dist-packages (from tf-nightly)
(2.9.0a20220227)
Requirement already satisfied: wheel<1.0,>=0.23.0 in
/usr/local/lib/python3.7/dist-packages (from astunparse>=1.6.0->tf-
nightly) (0.37.1)
Requirement already satisfied: cached-property in
/usr/local/lib/python3.7/dist-packages (from h5py>=2.9.0->tf-nightly)
(1.5.2)
Requirement already satisfied: markdown>=2.6.8 in
/usr/local/lib/python3.7/dist-packages (from tb-nightly~=2.9.0.a->tf-
nightly) (3.3.6)
Requirement already satisfied: google-auth<3,>=1.6.3 in
/usr/local/lib/python3.7/dist-packages (from tb-nightly~=2.9.0.a->tf-
nightly) (1.35.0)
Requirement already satisfied: tensorboard-data-server<0.7.0,>=0.6.0
in /usr/local/lib/python3.7/dist-packages (from tb-nightly~=2.9.0.a-
>tf-nightly) (0.6.1)
Requirement already satisfied: tensorboard-plugin-wit>=1.6.0 in
/usr/local/lib/python3.7/dist-packages (from tb-nightly~=2.9.0.a->tf-
nightly) (1.8.1)
Requirement already satisfied: werkzeug>=0.11.15 in
/usr/local/lib/python3.7/dist-packages (from tb-nightly~=2.9.0.a->tf-
nightly) (1.0.1)
Requirement already satisfied: google-auth-oauthlib<0.5,>=0.4.1 in
/usr/local/lib/python3.7/dist-packages (from tb-nightly~=2.9.0.a->tf-
nightly) (0.4.6)
Requirement already satisfied: requests<3,>=2.21.0 in
/usr/local/lib/python3.7/dist-packages (from tb-nightly~=2.9.0.a->tf-
nightly) (2.23.0)
Requirement already satisfied: pyasn1-modules>=0.2.1 in
/usr/local/lib/python3.7/dist-packages (from google-auth<3,>=1.6.3-
>tb-nightly~=2.9.0.a->tf-nightly) (0.2.8)
Requirement already satisfied: cachetools<5.0,>=2.0.0 in
/usr/local/lib/python3.7/dist-packages (from google-auth<3,>=1.6.3-
>tb-nightly~=2.9.0.a->tf-nightly) (4.2.4)
Requirement already satisfied: rsa<5,>=3.1.4 in
/usr/local/lib/python3.7/dist-packages (from google-auth<3,>=1.6.3-
>tb-nightly~=2.9.0.a->tf-nightly) (4.8)
```

```
Requirement already satisfied: requests-oauthlib>=0.7.0 in
/usr/local/lib/python3.7/dist-packages (from google-auth-
oauthlib<0.5,>=0.4.1->tb-nightly~=2.9.0.a->tf-nightly) (1.3.1)
Requirement already satisfied: importlib-metadata>=4.4 in
/usr/local/lib/python3.7/dist-packages (from markdown>=2.6.8->tb-
nightly~=2.9.0.a->tf-nightly) (4.11.1)
Requirement already satisfied: zipp>=0.5 in
/usr/local/lib/python3.7/dist-packages (from importlib-metadata>=4.4-
>markdown>=2.6.8->tb-nightly~=2.9.0.a->tf-nightly) (3.7.0)
Requirement already satisfied: pyasn1<0.5.0,>=0.4.6 in
/usr/local/lib/python3.7/dist-packages (from pyasn1-modules>=0.2.1-
>google-auth<3,>=1.6.3->tb-nightly~=2.9.0.a->tf-nightly) (0.4.8)
Requirement already satisfied: certifi>=2017.4.17 in
/usr/local/lib/python3.7/dist-packages (from requests<3,>=2.21.0->tb-
nightly~=2.9.0.a->tf-nightly) (2021.10.8)
Requirement already satisfied: urllib3!=1.25.0,!=1.25.1,<1.26,>=1.21.1
in /usr/local/lib/python3.7/dist-packages (from requests<3,>=2.21.0-
>tb-nightly~=2.9.0.a->tf-nightly) (1.24.3)
Requirement already satisfied: chardet<4,>=3.0.2 in
/usr/local/lib/python3.7/dist-packages (from requests<3,>=2.21.0->tb-
nightly~=2.9.0.a->tf-nightly) (3.0.4)
Requirement already satisfied: idna<3,>=2.5 in
/usr/local/lib/python3.7/dist-packages (from requests<3,>=2.21.0->tb-
nightly~=2.9.0.a->tf-nightly) (2.10)
Requirement already satisfied: oauthlib>=3.0.0 in
/usr/local/lib/python3.7/dist-packages (from requests-oauthlib>=0.7.0-
>google-auth-oauthlib<0.5,>=0.4.1->tb-nightly~=2.9.0.a->tf-nightly)
(3.2.0)
Requirement already satisfied: pyparsing!=3.0.5,>=2.0.2 in
/usr/local/lib/python3.7/dist-packages (from packaging->tf-nightly)
(3.0.7)
Requirement already satisfied: tensorflow-datasets in
/usr/local/lib/python3.7/dist-packages (4.0.1)
Requirement already satisfied: numpy in /usr/local/lib/python3.7/dist-
packages (from tensorflow-datasets) (1.21.5)
Requirement already satisfied: promise in
/usr/local/lib/python3.7/dist-packages (from tensorflow-datasets)
(2.3)
Requirement already satisfied: six in /usr/local/lib/python3.7/dist-
packages (from tensorflow-datasets) (1.15.0)
Requirement already satisfied: dill in /usr/local/lib/python3.7/dist-
packages (from tensorflow-datasets) (0.3.4)
Requirement already satisfied: protobuf>=3.6.1 in
/usr/local/lib/python3.7/dist-packages (from tensorflow-datasets)
(3.17.3)
Requirement already satisfied: requests>=2.19.0 in
/usr/local/lib/python3.7/dist-packages (from tensorflow-datasets)
(2.23.0)
Requirement already satisfied: termcolor in
```

```
/usr/local/lib/python3.7/dist-packages (from tensorflow-datasets)
(1.1.0)
Requirement already satisfied: tqdm in /usr/local/lib/python3.7/dist-
packages (from tensorflow-datasets) (4.62.3)
Requirement already satisfied: future in
/usr/local/lib/python3.7/dist-packages (from tensorflow-datasets)
(0.16.0)
Requirement already satisfied: attrs>=18.1.0 in
/usr/local/lib/python3.7/dist-packages (from tensorflow-datasets)
(21.4.0)
Requirement already satisfied: dm-tree in
/usr/local/lib/python3.7/dist-packages (from tensorflow-datasets)
(0.1.6)
Requirement already satisfied: absl-py in
/usr/local/lib/python3.7/dist-packages (from tensorflow-datasets)
(1.0.0)
Requirement already satisfied: tensorflow-metadata in
/usr/local/lib/python3.7/dist-packages (from tensorflow-datasets)
(1.6.0)
Requirement already satisfied: importlib-resources in
/usr/local/lib/python3.7/dist-packages (from tensorflow-datasets)
(5.4.0)
Requirement already satisfied: idna<3,>=2.5 in
/usr/local/lib/python3.7/dist-packages (from requests>=2.19.0-
>tensorflow-datasets) (2.10)
Requirement already satisfied: chardet<4,>=3.0.2 in
/usr/local/lib/python3.7/dist-packages (from requests>=2.19.0-
>tensorflow-datasets) (3.0.4)
Requirement already satisfied: urllib3!=1.25.0,!=1.25.1,<1.26,>=1.21.1
in /usr/local/lib/python3.7/dist-packages (from requests>=2.19.0-
>tensorflow-datasets) (1.24.3)
Requirement already satisfied: certifi>=2017.4.17 in
/usr/local/lib/python3.7/dist-packages (from requests>=2.19.0-
>tensorflow-datasets) (2021.10.8)
Requirement already satisfied: zipp>=3.1.0 in
/usr/local/lib/python3.7/dist-packages (from importlib-resources-
>tensorflow-datasets) (3.7.0)
Requirement already satisfied: googleapis-common-protos<2,>=1.52.0
in /usr/local/lib/python3.7/dist-packages (from tensorflow-metadata-
>tensorflow-datasets) (1.54.0)
[nltk_data] Downloading package punkt to /root/nltk_data...
[nltk_data]   Unzipping tokenizers/punkt.zip.
[nltk_data] Downloading package stopwords to /root/nltk_data...
[nltk_data]   Unzipping corpora/stopwords.zip.
2.9.0-dev20220301

# IMPORT DATA
# %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

"""
```

```
This cell imports the dataset used by the project:
     -> We are using the wget command to do this
     -> These files are from freeCodeCamp
     -> We are downloading the train and test data directly into the
notebook
          -> The train data is being stored in the variable called
train_file_path
          -> The file paths are stored in the variable called
test_file_path
     -> These are SMS formats (text messages), with the TSV file
extension
"""

!wget https://cdn.freecodecamp.org/project-data/sms/train-data.tsv
!wget https://cdn.freecodecamp.org/project-data/sms/valid-data.tsv

train_file_path = "train-data.tsv"
test_file_path = "valid-data.tsv"
```

```
--2022-03-01 11:03:14--
https://cdn.freecodecamp.org/project-data/sms/train-data.tsv
Resolving cdn.freecodecamp.org (cdn.freecodecamp.org)...
172.67.70.149, 104.26.2.33, 104.26.3.33, ...
Connecting to cdn.freecodecamp.org (cdn.freecodecamp.org)|
172.67.70.149|:443... connected.
HTTP request sent, awaiting response... 200 OK
Length: 358233 (350K) [text/tab-separated-values]
Saving to: ‘train-data.tsv’

train-data.tsv      100%[===================>] 349.84K  --.-KB/s    in
0.09s

2022-03-01 11:03:15 (3.63 MB/s) - ‘train-data.tsv’ saved
[358233/358233]

--2022-03-01 11:03:15--
https://cdn.freecodecamp.org/project-data/sms/valid-data.tsv
Resolving cdn.freecodecamp.org (cdn.freecodecamp.org)...
172.67.70.149, 104.26.2.33, 104.26.3.33, ...
Connecting to cdn.freecodecamp.org (cdn.freecodecamp.org)|
172.67.70.149|:443... connected.
HTTP request sent, awaiting response... 200 OK
Length: 118774 (116K) [text/tab-separated-values]
Saving to: ‘valid-data.tsv’

valid-data.tsv      100%[===================>] 115.99K  --.-KB/s    in
0.08s

2022-03-01 11:03:15 (1.34 MB/s) - ‘valid-data.tsv’ saved
```

```
[118774/118774]


# Read data into pandas' dataframes

"""
     Converting from text messages into pandas data frames:
          -> The previous cell imports the text message data (TSV
files) into the variables called train_file_path and test_file_path
          -> This cell takes this data and converts it into pandas
CSV data frames
          -> The first variable in this cell does this for the
training dataset and the second does this for the test dataset
          -> The final line in this cell prints out the head of one
of these data frames, so we can see its syntax
          -> We are doing this using the pandas module
          -> We are reading the TSV files into the data frames
               -> There is no header in these data frames
               -> We are telling it the names of the columns under
the array `name`

     We now have:
          -> The text message data imported into the project
          -> The text message data for testing the model converted
into CSV (spreadsheet) format and stored in the variable called
`df_train`
          -> The same thing with the data for testing the trained
model at the end of this notebook -> with this data stored in the
variable called `df_test`
          -> The test and training datasets are now stored in pandas
data frames
"""

df_train = pd.read_csv(train_file_path, sep='\t', header=None,
names=['label', 'message'])
df_test = pd.read_csv(test_file_path, sep='\t', header=None,
names=['label', 'message'])
df_test.head()
```

	label	message
0	ham	i am in hospital da. . i will return home in e...
1	ham	not much, just some textin'. how bout you?
2	ham	i probably won't eat at all today. i think i'm...
3	ham	don‘t give a flying monkeys wot they think and...
4	ham	who are you seeing?

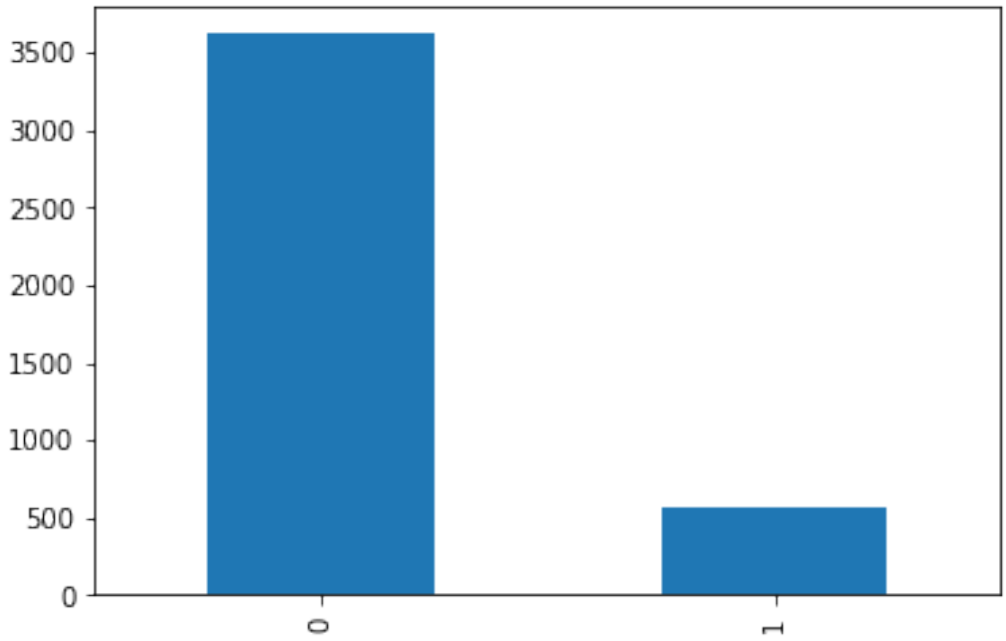
```
# Convert labels to categorical data and replace text with code values

     """
     We are taking the data from the previous cell and converting it
```

```
into numerical form:
          -> We can't train the model on words -> we need them to be
in a numerical format first
          -> We first imported the modules and data for the project
          -> Now we are getting this data ready to train the model
          -> We need to get rid of all of the categorical data in the
labels  column in these data frames

     To do this:
          -> We have two data frames -> the first is for the training
data and the second is for the test data
          -> We are targeting the labels column in both of these data
frames -> since it contains this categorical data
          -> We want to convert this data into numbers
          -> Replacing the text with code values -> each of the
categorical data values is being converted into a code form
          -> We are telling it to target the categorical data with
the .astype method, and then to convert this into code with
the .cat.codes method
          -> The first line in this cell is doing this for the
training data and the second line is doing this for the test data
          -> We are directly targeting the `labels` collumns in both
of those datasets
"""

df_train['label'] = df_train['label'].astype('category').cat.codes
df_test['label'] = df_test['label'].astype('category').cat.codes

# Split data into labels and features

"""
     -> From the previous cell, we now have the training and test
datasets stored in two different pandas frames
     -> The previous cell took the categorical data in the `labels`
column for these frames and converted it into numerical data
     -> This cell is taking the data from those datasets and splitting
the data we want to input into the model (the features) from the data
which we want the model to predict
     -> This is done via the .pop method -> this returns the same
dataset but without the column with the label which is taken as its
argument
     -> When you use this method, it takes the dataset which you give
it and sets it equal to itself but without the data which you tell it
to pop off
     -> Since data is lost when using this method, the first two lines
of code in this cell create copies of these two data frames (for the
train and test datasets)
     -> And then the final two lines are popping off the columns which
we want the model to predict
     -> This ensures we aren't training the model on data that we want
```

```
to predict
      -> This process is separating the input data from the output data
for our model -> training and test features
           -> The labels are the output data and the features are the
data which we are predicting
"""

train_features = df_train.copy()
test_features = df_test.copy()
train_labels = train_features.pop('label')
test_labels = test_features.pop('label')
```

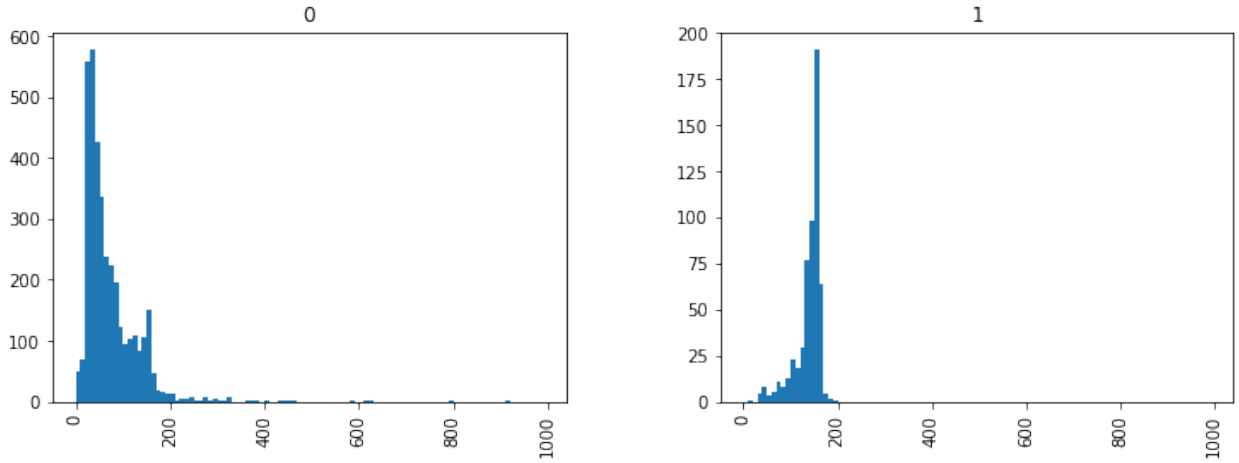
4. Analysing the Dataset

```
# Analyse dataset
# Draw histogram showing number of 'ham' and 'spam' messages

"""
     From the previous cell:
          -> We now have the datasets imported
          -> These were split into training and test datasets and
cleaned so that they only contained categorical (and not numerical
data)
          -> We then popped off the features from these datasets
which we want to predict
          -> This cell is taking the column in the training dataset
with the data we want to predict (the data in the `label` column) and
printing it out into a bar chart

     What this cell does:
          -> The data which is contained in the `label` column of
this dataset is whether the text message in that row is a "spam" or
"ham"
          -> 1 is spam and 0 is ham
          -> We are taking the data in the labels column and using
the .value_counts method to return the number of "spams" and "hams" in
that data
          -> We are then taking this data and returning it in a bar
chart
          -> This allows us to see the number of "ham" and "spam"
text messages in the datasets relative to each other and prints them
out in a distribution
          -> From this, we can see that there are significantly more
"hams" than "spams" in the dataset which we later train the model on
          -> We are doing this using the Pandas library
"""

df_train['label'].value_counts().plot.bar();
```



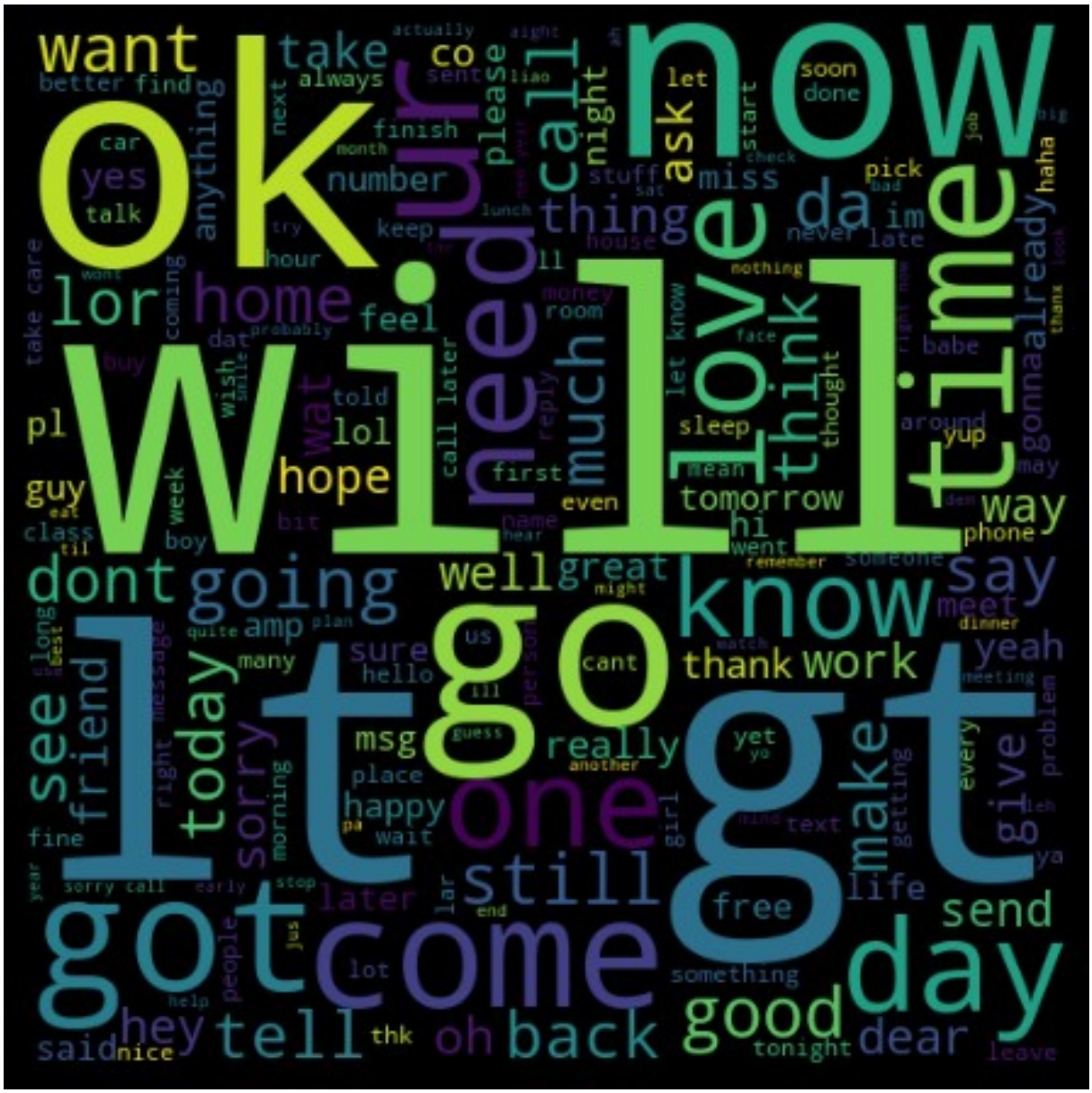

```
# Draw histogram showing length of 'ham' and 'spam' messages

"""
      What this block of code does:
            -> We are taking the wanted ("ham") and unwanted ("spam")
messages in the training dataset and plotting them according to how
long they are
            -> The dataset we are plotting doesn't contain categorical
data -> we have converted the entire thing into numerical data
            -> Since we are dealing with "ham" and "spam" messages, we
are creating two histograms
            -> We are doing this to inspect the dataset before training
it with the aim of distinguishing between these two datatypes

      How this is achieved:
            The first line of code in this cell creates a copy of the
test data frame:
                  -> This cell plots the lengths of the "ham" and "spam"
messages in this dataset -> so we are first creating a copy of it in a
variable called `df_len`
                  -> These histograms are for the data which the model
is being trained on
                  -> We are creating a copy of this dataset because we
are going to add an extra column which contains the data which we are
plotting here

            The second line adds a column in which contains the data
that we are doing to plot:
                  -> df_len is the name of the dataset which is the copy
```

```
of the training dataset -> this is the one which we are operating on

                -> We are setting its length column equal to the
length of the data contained in its message column

          The third line plots this length data in a histogram:
                -> df_len <- this is the name of the dataset which
contains what we want to plot
                -> The arguments in this line format the appearance of
these plots
                      -> We are plotting the length column in that
data frame
                      -> We don't have to use matplotlib to generate a
1x2 matrix of subplots -> the 'label' argument in this line is telling
it to render the histograms according to the labels which they store
("spam" or "ham")
                      -> Since there are two labels for this, it
automatically generates a 1x2 matrix of plots
                      -> The columns in the histogram serve as bins
for this data -> we are telling it the number of bins per histogram to
create (1,000), combined with the width that we want each one to be
                      -> The final argument is for the width and
height of the plot with all of the histograms in it

     Interpreting these histograms:
           -> The first histogram is for the length of the "ham"
messages and the second is for the length of the "spam" messages
           -> These are all for the test datasets
           -> From this, we can see that the "spam" messages are
shorter in length, that their lengths follow a negative skew and that
there are about three times less than them in the training data
           -> This is the data which we later train the model on -> to
make predictions about the status of the input message
           -> x in these figures is the length of the messages which
they contain, and y is the amount of messages of that length in the
training dataset which are 0 "ham" and 1 "spam"
"""

df_len = pd.DataFrame(df_train)
df_len['length'] = df_len['message'].apply(len)
df_len.hist(column='length', by='label', bins=range(0, 1000, 10),
figsize=(12,4));
```




```
# Draw WordCloud for 'ham' words

"""
     The aim of this block of code:
          -> The "ham" messages are the ones which aren't from
companies and which don't contain adverts
          -> This cell is taking the training dataset and plotting
the content of these text messages in a WordCloud
          -> We are only doing this for the friendly messages in the
training data (not for the "spams")
          -> When we converted the categorical data into numerical
data in that dataset, we converted the "hams" into 0's and the "spams"
into 1's
               -> The data which we want to generate the WordCloud of
has a label of 0
               -> The text message data stores entire strings of
words -> we want to separate these out into individual ones for the
WordCloud

     Acquiring the data which we want to use to create the WordCloud:
          -> The data which we want to create this cloud for is the
"hams" in the training dataset
          -> The first line of code in this cell acquires this data
          -> We are telling it to only include rows where the label
is 0 <- "ham" messages
          -> We are first selecting the "ham" messages
          -> Then we join all of them into a string, via the ' '.join
method
          -> We are setting this entire string equal to the
`ham_words` variable
               -> This is a string which contains all of the words in
the ham text messages for the training dataset
          -> So we are telling it to take the messages in the
df_train dataset, so long as the value which is stored in the `label`
column of that dataset is 0 (the message is a "ham")
```

```
          -> That is just the "ham" messages in that dataset -> and
then for the entire thing to be converted into a string
          -> All of those messages are then put into their own string
-> which is stored in the `ham_words` variable

     Using this data to generate the WordCloud:
          We create a WordCloud object:
               -> We have the list of words we want to generate the
WordCloud from -> in `ham_words`
               -> We are then creating a WordCloud object based on
this and storing it in the `ham_wc` variable
               -> We are telling it the width and height we want for
this -> and which string of words it should be based on

          We then plot this WordCloud object:
               -> We are plotting this using matplotlib
               -> When doing this, we are setting the dimensions of
the plot we want it to create, background colour, whether we want axes
displayed, the shape we want the layout in and telling it to display
the plot
               -> It shows the words which are most frequently used
in the "ham" text messages in the training dataset
"""

ham_words = ' '.join(list(df_train[df_train['label'] == 0]
['message']))
ham_wc = WordCloud(width=512, height=512).generate(ham_words)
plt.figure(figsize=(10,6), facecolor='k')
plt.imshow(ham_wc)
plt.axis('off')
plt.tight_layout(pad=0)
plt.show()
```



5. Preprocessing Data for Training the Machine Learning Model

```
# Preprocess data

"""
     Context:
          -> This cell defines a function which takes a message and
pre-processes it
          -> We need the data to be in a specific form before we can
train the model on it
```

```
            -> This function is called `process_message`
            -> This is to process the text data in those messages
            -> We previously took the categorical labels and converted
them into numerical data -> this did not target or process the text
content of those text messages themselves

        How this function works:
            -> The function takes a text message and returns a
processed list of those words
                -> The processed list of words is the input list of
words after lowercasing, punctuation removal, tokenisation and
stopword removal have been performed
                -> This function can be used on each of the text
messages in the datasets so that they can be used to train the model
on
            -> There are two other lines inside the definition of the
function which it is using to process these messages
            -> The first makes all of the text in the input string
lowercase and removes punctuation, by using a generator expression
                -> `msg` is the name of the input string (the argument
to the function)
                -> We are converting that entire string into
lowercase, using the .lower method
                -> Then we are getting rid of the punctuation by
iterating through each character of the lowercased method and asking
the cell if it is in a list of punctuation or not
                -> This is a generator expression
                -> The list of punctuation characters we are filtering
it against are standard
                -> We are setting the argument to the function equal
to itself, but without the punctuation in its strings
            -> The second removes stopwords and performs tokenisation
                -> `msg` is now the input message, in lowercase and
without punctuation
                -> This line of code is taking that message and
tokenising the words -> putting them into an array where each element
is a ["single", "string"]
                -> We are iterating through that array <- each element
in it is `word` (like for i in) and including it in the message which
is returned if it's not in the library of English stopwords
                    -> Stopwords are common words (like "and," which
otherwise void our message of meaning
                    -> We are using the NLTK library here to void
our message of these words, and to put those messages in array ->
where each element is one of the letters which we input into it
"""

# Define function for message processing
def process_message(msg):
```

```
  # Make everything lowercase and skip punctuation characters
  msg = ''.join(char for char in msg.lower() if char not in
set(string.punctuation))
  # Tokenize message but skip stopwords like 'the', 'a', etc.
  word_list = [word for word in word_tokenize(msg) if word not in
stopwords.words('english')]
  return word_list

# Create bag-of-words transformer

"""
     Creating a bag of words transformer for the dataset:
          -> We are creating a bag of words transformer -> taking the
words and converting them into a numerical format
          -> We are doing this using the CountVectorizer from scikit-
learn, so that the words can be used to train the machine learning
model with NLP
          -> The first two lines of code in this cell are making a
data frame out of the training and test labels
               -> The labels are the data which we are predicting and
the labels are the other data
               -> We have the entire dataset which was split into
train and test sets
               -> We are taking those two datasets and combining them
back into one -> and only extracting the features (top row) and labels
(second row) from them
               -> So we have two variables, `data_features` and
`data_labels`, the first which stores the data which we aren't
predicting, and the second which stores the data which we are
          -> The last two lines create a bag of words transformer
from these variables, and store this in a variable called
`bow_transformer` whose length is printed by the last line in this
cell
               -> CountVectorizer is used to do this
                    -> We are converting those words into vectors in
a higher dimensional space, whose coordinates relate to their
sentiment
                    -> This is from scikit-learn
                    -> One of the arguments to this is `analyzer` <-
this is the function defined in the previous cell, which processes a
message passed into it
               -> This has created a vocabulary from the words passed
into it in the dataset and forms a bag-of-words model of it
               -> We are printing the number of unique values and
their indices, by targeting bow_transformer.vocabulary_
          -> This bow transformer can be used to convert the words in
a text message into numerical form
"""

# Concatenate train and test data
```

```
data_features = pd.concat([train_features, test_features])
data_labels = pd.concat([train_labels, test_labels])
# Create bag-of-words transformer that keeps vocabulary with all words
bow_transformer =
CountVectorizer(analyzer=process_message).fit(data_features['message']
)
# Print number of words in vocabulary
print(len(bow_transformer.vocabulary_))
```

```
9496
```

```
# Vectorize a message using a bag-of-words transformer

"""
     This cell uses the BOW transformer defined in the previous
question:
          -> We are taking a text message and vectorising it
          -> The first line of code in this cell sets the value of
the variable `sample_msg` equal to the test message that we want to
vectorise:
               -> This is the first message in the message column of
the training_features dataset
          -> The second line is applying the bow transformer to this:
               -> In the previous cell, this transformer was stored
in the variable called `bow_transformer` -> the .transform method is
used on this to apply it to the text message
               -> This converts the message into a numerical version
based on the vocabulary which the transformer previously learned

     The outputs of this are then printed:
          -> The first line this cell prints out is the original
processed message
          -> The second line this cell prints out is the message with
the bow transformer applied
          -> This is this same message in a numerical format
               -> This is a sparse matrix representation of the
message in the bag-of-words format
               -> Each row is one message and each column is one word
in its vocabulary -> the values represent the count of each word in
the corresponding document
          -> We can use this transformer to convert messages into a
form to train the machine-learning model with
"""

sample_msg = train_features['message'][0]
bow_sample_msg = bow_transformer.transform([sample_msg])
print(sample_msg)
print(bow_sample_msg)
```

```
ahhhh...just woken up!had a bad dream about u tho,so i dont like u
right now :) i didnt know anything about comedy night but i guess im
up for it.
  (0, 1066)     1
  (0, 1213)     1
  (0, 1444)     1
  (0, 2292)     1
  (0, 2775)     1
  (0, 2898)     1
  (0, 2940)     1
  (0, 3916)     1
  (0, 4356)     1
  (0, 4799)     1
  (0, 4989)     1
  (0, 5782)     1
  (0, 7010)     1
  (0, 8299)     1
  (0, 8600)     2
  (0, 8693)     1
  (0, 9141)     1
```

```
# Apply bag-of-words transformer to all messages

"""
     -> The previous cell tested the bag of words transformer on a
single message
     -> This cell applies this transformer to all of the messages in
the set
          -> These messages are stored in `data_features`
          -> We are taking the transformer in the same syntax which
the previous cell uses, and instead applying it to an entire column
(the `message` column) in the dataset
          -> Then we are printing out the shape of the sparse matrix
which this creates
     -> We are storing this data (the messages in numerical format) in
`bow_data`, since it is now in this bag-of-word format
"""

bow_data = bow_transformer.transform(data_features['message'])
print(bow_data.shape)
```

```
(5571, 9496)
```

```
# Percentage of non-zero entries to the matrix size

"""
     -> We are calculating the percentage of entries which aren't zero

     -> Most of the elements in the sparse matrix which the bag of
words transformer generates aren't numbers
```

```
     -> Since the words which the bag of words transformer has
transformed to get there are a tiny subset of the total vocabulary of
words in the English language, the percentage of elements in this
matrix which aren't zero is very small
     -> To calculate the percentage of non-zero elements in this
matrix, we are
          -> Using a print function (to print the result)
          -> Taking the number of values in that matrix which aren't
zero -> we do this by using the .nnz method
          -> Calculating this as a percentage of the total number of
entries in the matrix -> this is the number of rows times the number
of columns in that matrix
     -> This number reflects the density of this sparse matrix
"""

print(bow_data.nnz / (bow_data.shape[0] * bow_data.shape[1]) * 100)
```

```
0.09482211482407467
```

```
# Use TF-IDF (Term Frequency times Inverse Document Frequency)
transformer to account for the total number of words in each document
and to downscale weights for words that occur in many documents

"""
     -> In this cell, we are writing a Term Frequency times Inverse
Document Frequency) transformer
     -> This accounts for the total number of words in each document
and ensures that when training the model, that the weights per word
are reduced for documents with larger numbers of words
     -> We are fitting this transformer to the BOW data
     -> This is done using the .fit method on the bow_data
          -> The bow_data being the messages in the dataset which we
imported, cleaned and then transformed into numerical values in vector
space
          -> The module we are using to do this is scikit-learn
          -> We are altering the weights of the words in the model,
if for example the total number of words in the document we are
processing is different
     -> We are transforming the bag of words matrix
          -> This is also considering the importance of terms based
off of their rarity in the set -> for example, more frequent words
being considered as less important
"""

tfidf_transformer = TfidfTransformer().fit(bow_data)

# Show a sample message with weights calculated using TF-IDF
transformer

"""
```

```
      -> This cell uses the TF-IDF transformer which was defined in the
previous cell, on an example message
      -> The name of our TF-IDF transformer is `tfidf_transformer`
            -> To apply this, we are using the .transform method
            -> The argument which we are passing into this method is
`bow_sample_msg`
            -> This is the sample message that we are vectorising
            -> Using our TF-IDF transformer like this allows us to do
this in a way which allows all of the words to have frequency-
dependent weights
      -> We then output this cell
            -> This is the input message in vectorised form
      -> This shows how we can use the TF-IDF transformer to vectorise
a text message
      -> We are turning the words in the message into a sparse matrix
      -> The outputs of this cell show the TIDF weights for each of
those words for the test message
"""

tfidf_sample = tfidf_transformer.transform(bow_sample_msg)
print(tfidf_sample)
```

```
  (0, 9141)     0.3193536147571175
  (0, 8693)     0.33453929686752293
  (0, 8600)     0.21963536951370996
  (0, 8299)     0.33453929686752293
  (0, 7010)     0.19323996644726754
  (0, 5782)     0.18549026826246273
  (0, 4989)     0.157156822738756
  (0, 4799)     0.15477050617704852
  (0, 4356)     0.13383352007920915
  (0, 3916)     0.22328984800432256
  (0, 2940)     0.2742618228607118
  (0, 2898)     0.15267261779364014
  (0, 2775)     0.19685264460210067
  (0, 2292)     0.3085792018329111
  (0, 1444)     0.23069891672907558
  (0, 1213)     0.20032755250826478
  (0, 1066)     0.33453929686752293
```

```
# Transform all data using TF-IDF transformer

"""
      -> This cell uses the same code which the previous cell uses,
except that we are applying this to all of the `bow_data`, rather than
to one sample message
      -> The final line of this cell then prints out the shape of the
matrix which this creates
      -> This essentially vectorises all of the messages in the
training dataset
```

```
     -> We have taken the data in `bow_data` and transformed it into a
TF-IDF matrix using our bow transformer
     -> That matrix represents all of the words in the entire training
dataset
          -> Each row in this is a message
          -> Each column in this is a unique word in its vocabulary
          -> The values which are stored there are the TF-IDF weights
for the words in the document which are stored at that point in the
matrix
     -> We now have the entire dataset in vectorised / matrix form,
which will allow us to train a model using it
     -> The aim being to classify the next messages which we have
processed into "ham" and "spam" types
"""

data_tfidf = tfidf_transformer.transform(bow_data)
np.shape(data_tfidf)
```

```
(5571, 9496)
```

```
# Split data into training and testing part

"""
     -> This cell takes the bow transformed data and splits it back
into training and test datasets
     -> We are doing this using the scikit-learn train_test_split
method

     -> The arguments this method takes:
          -> `data_tfidf` is the matrix with the transformed data ->
each row in this matrix represents a document and each column
represents a unique word in the vocabulary
               -> This is the feature matrix
               -> The features are the data which we aren't
predicting (we are predicting the labels, whether the messages are a
"ham" or "spam")
               -> It contains TF-IDF representations of those
messages
          -> `data_labels` <- this contains the labels for the
documents in the dataset
               -> We are using this as the target (y) variable
          -> test_size <- this sets the amount of data we want to
train the model with
               -> We will use the rest to test it
          -> random_state=0 <- when we split the data into training
and test sets, we are doing this randomly
               -> Each time this is done, our set will again be based
on a different random split (hence epochs)
     -> The variables which we are storing these data in are:
          -> For the training data:
```

```
                -> `X_train` <- this contains the TF-IDF
representation of the training data
                -> `y_train` <- this contains the TF-IDF
representation of the training labels -> the output data we want
           -> This is similarly to the test data
                -> This is stored in `X_test` and `y_test`
     -> This gives us four sets of data for training and testing the
models
"""

X_train, X_test, y_train, y_test = train_test_split(data_tfidf,
data_labels, test_size=0.25, random_state=0)
```

6. Obtaining the Optimal Machine Learning Model for the SMS Text-Classifier

```
# First attempt: use random forest classifier

"""
     -> We now have the training and test data for the model
     -> Three different attempts were made at training the model
          -> This cell contains the first of these attempts, which
was a random forest classifier
          -> The other two attempts were to use a XGBoost regressor
and GridSearchCV classifier method (the last having the most accurate
results)
     -> The random forest classifier
          -> The first line in this cell creates a random forest
classifier called `rf_clf`
                -> This is with 250 random trees and a seed for random
number generation of 0
                -> This is created using the scikit-learn library
          -> The second line in this cell trains the model on this
classifier
                -> We are using the .fit method on the random forest
classifier defined in the first line of this cell
                -> This trains this model
                -> The data we are using to train this is the TF-IDF
transformed training data and training labels, defined from the
previous cell
           -> The final line in this is the trained model, to make
predictions
                -> We are using the .predict method on the trained
classifier (`rf_clf`)
                -> We are giving it the features on the test dataset
for this (stored in `X_test`) and storing the predictions this makes
in `y_pred`
```

```
     -> We are using the model to make predictions on the `X_test`
data and storing these in `y_pred`
     -> We are doing this for three different model types, the random
forest classifier (in this cell), the XGBoost regressor in the next
and the GridSearchCV classifier in the one after this
     -> We are looking for the model which produces the most accurate
predictions -> so we are first building out these predictions
     -> We first imported and pre-processed the data -> now we are
trying to determine the best model to use to make predictions for it

"""

#1 Create the classifier
rf_clf = RandomForestClassifier(n_estimators=250, random_state=0)

#2 Train the classifier on the TF-IDF transformed training data
rf_clf.fit(X_train, y_train)

#3 Use the classifier to make predictions
y_pred = rf_clf.predict(X_test)

# Second attempt: use XGBoost regressor

"""
     -> In this cell, we are training the TF-IDF transformed training
data using an XGBoost (Extreme Gradient Boosting) regressor
     -> The previous cell used a random forest approach, this is using
another approach to see if the predictions of this model are more
accurate
     -> The code in this cell is the same as in the previous cell,
except that instead of a random forest classifier, an XGBoost
regressor method is used
          -> The first line of this cell is initialising the model
          -> The second line of the cell is training it
          -> The final line of the cell is using this model to make
predictions
     -> The data which this uses is the same as in the previous cell
     -> When setting up this model
          -> We are giving this 120 boosting rounds
          -> A learning rate of 0.125 <- this controls how quickly
the model learns during the optimisation process
"""

xgb_clf = XGBRegressor(n_estimators=120, learning_rate=0.125)
xgb_clf.fit(X_train, y_train)
y_pred = xgb_clf.predict(X_test)
```

```
[11:17:33] WARNING: /workspace/src/objective/regression_obj.cu:152:
reg:linear is now deprecated in favor of reg:squarederror.
```

```
# Third attempt: use GridSearchCV classifier. This method is the best
according to the classification results compared to the other methods
that were tested.

"""
      -> This is the final cell which we are using to train models
      -> The three models we are using are the random forest
classifier, the XGBoost regressor and the GridSearchCV classifier
      -> We are doing this to see which of these models produces the
best predictions
      -> This cell trains the third model, which uses the GridSearchCV
classifier method
            -> This is the Grid Search Cross-Validation approach
      -> The first two lines of this cell initialise this model
            -> We set the parameter grid for hyperparameter tuning and
create the classifier using Support Vector Machines as the base model
            -> Defining the grid which will be searched during the
tuning process
            -> We are allowing probability estimates
      -> The next line trains this model
            -> It searches through the hyperparameter grid and gives us
the best ones which give the model the best predictions
      -> The final line in this cell uses this model to make
predictions
      -> We now have three models which are being trained on the same
data and have been used to make predictions
      -> In the next cells we choose the best model and run unit tests
on them, to pass the project
"""

param_grid = {'C': [0.1, 1, 10, 100],
  'gamma': [1, 0.1, 0.01, 0.001, 0.0001],
  'gamma':['scale', 'auto'],
  'kernel': ['linear']}

gsc_clf = GridSearchCV(SVC(probability=True), param_grid, refit=True,
verbose=0, n_jobs=-1)
gsc_clf.fit(X_train, y_train)
y_pred = gsc_clf.predict(X_test)
```

7. Testing the Predictions Produced by the Model

```
# Show classification results. Assume that probability of 0.5 is a
threshold between 'spam' and 'ham' message.

"""
```

```
     -> This prints the predictions of the last (most accurate)
classification model on the testing data
     -> This includes the confusion matrix, classification report and
accuracy score of this model on the test data

     The confusion matrix:
          -> The confusion matrix shows the true positives, false
positives and vice versa with the negatives <- this is the first data
which this cell returns
          -> This is showing us how many (for example) "spams" those
models predicted which were actually "spams", how many "spams" which
it predicted that were actually "hams" and vice versa
          -> We are using this to return the accuracy of the
predictions that these models make on the training data
          -> 0 and 1 correspond to "ham" and "spam" text messages

     The classification report:
          -> The second line of code in this cell prints the
classification report for these predictions
          -> This also returns statistical results (the weighted
average, accuracy)

     -> The final line in this cell returns the accuracy score of this
model
          -> This is the percentage of the predictions which the
model makes which are actually correct
          -> The last of the three model types which we try produces
a largest accuracy score -> this is the one which we use
"""

print(confusion_matrix(y_test,(y_pred>0.5)))
print(classification_report(y_test,(y_pred>0.5)))
print(accuracy_score(y_test,(y_pred>0.5)))
```

```
[[1211    0]
 [  26  156]]
              precision    recall  f1-score   support

           0       0.98      1.00      0.99      1211
           1       1.00      0.86      0.92       182

    accuracy                           0.98      1393
   macro avg       0.99      0.93      0.96      1393
weighted avg       0.98      0.98      0.98      1393

0.9813352476669059
```

```
# function to predict messages based on model
# (should return list containing prediction and label, ex.
[0.008318834938108921, 'ham'])
```

```
"""
      -> This cell defines a prediction function called
`predict_message`
      -> This takes an input text message and returns the probability
that it is "spam"
            -> This result is between 0 and 1
      -> We have the trained model -> we tried three different methods
to find the most accurate one, and found the last gave the most
accurate results for this
      -> The output is the probability that the result is a "spam,"
combined with the prediction about what the message is
      -> To do this, we:
            -> Transform the input message using the bow transformer ->
this encodes it into a numerical tensor form
            -> This is the first two lines of code in this function
            -> Then we are using the trained model to predict the
probability that the input message is "spam"
            -> Then we create a prediction list with the predicted
probability label -> if the message we are passing into the model was
"ham" or "spam" and return this prediction
            -> We are taking the prediction that the model is either
"ham" or "spam" in array form and returning just its first element
            -> Depending on the probability of the message being "ham"
or not (if this value is above or below 0.5), we choose the output
      -> Once this function has been defined, we place a test text
message into it
"""

def predict_message(pred_text):
  bow_pred_text = bow_transformer.transform([pred_text])
  tfidf_pred_text = tfidf_transformer.transform(bow_pred_text)
  pred = gsc_clf.predict_proba(tfidf_pred_text)[0]
  prediction = [pred[1], 'ham' if pred[0] > 0.5 else 'spam']
  return (prediction)

pred_text = "how are you doing today?"

prediction = predict_message(pred_text)
print(prediction)
```

```
[0.011547273098205363, 'ham']
```

8. Running Unit Tests for the Model in Python

```
# Run this cell to test your function and model. Do not modify
contents.
```

```
"""
     -> This cell contains unit tests for the prediction function
     -> test_predictions is the function that does this
     -> This contains test text messages in an array, and the answers
which our prediction function should return
     -> If the result from our prediction function matches the result
from their list of what those predictions should be then this function
prints out a given message
     -> We are iterating through the messages in that array and
printing out whether our predictions match the expected ones
"""

def test_predictions():
  test_messages = ["how are you doing today",
                   "sale today! to stop texts call 98912460324",
                   "i dont want to go. can we try it a different day?
available sat",
                   "our new mobile video service is live. just install
on your phone to start watching.",
                   "you have won £1000 cash! call to claim your
prize.",
                   "i'll bring it tomorrow. don't forget the milk.",
                   "wow, is your arm alright. that happened to me one
time too"
                  ]

  test_answers = ["ham", "spam", "ham", "spam", "spam", "ham", "ham"]
  passed = True

  for msg, ans in zip(test_messages, test_answers):
    prediction = predict_message(msg)
    if prediction[1] != ans:
      passed = False

  if passed:
    print("You passed the challenge. Great job!")
  else:
    print("You haven't passed yet. Keep trying.")

test_predictions()
```

```
You passed the challenge. Great job!
```