Sentiment analysis on Covid-19 tweets

Objective:

Classify 45k tweets on Covid-19 as positive or negative based on the following machine learning and deeplearning models:

- Multinomial Naive Bayes Model
- Random Forests
- ADABoost
- XGBoost
- Simple RNN
- LSTM
- GRU
- Bidirectional LSTM
- BERT

For machine learning models, the tweets are preprocessed using the following NLP methods:

- Bag-of-words model
- Bag-of-POS model
- Pre-trained Spacy word embeddings

For neural networks, we use the following preprocessing methods:

- Pre-trained Spacy word embeddings
- Keras embedding layers

Results:

- Among machine learning models, XGBoost trained on a bag-of-words model has the best performance in terms of accuracy (82%) and AUC ROC (90%)
- Among all models, BERT has the best performance (accuracy = 94%)

```
pip install xgboost==1.4.2
```

```
Collecting xgboost==1.4.2
  Downloading xgboost-1.4.2-py3-none-win_amd64.whl (97.8 MB)
     ---------------------------------------- 97.8/97.8 MB 4.4 MB/s eta 0:00:00
Requirement already satisfied: numpy in c:\users\bhara\appdata\local\programs\python\python39\lib\site-packages (from xgboost==1.4.2) (1
Requirement already satisfied: scipy in c:\users\bhara\appdata\local\programs\python\python39\lib\site-packages (from xgboost==1.4.2) (1
Installing collected packages: xgboost
Successfully installed xgboost-1.4.2
Note: you may need to restart the kernel to use updated packages.

[notice] A new release of pip is available: 23.1.2 -> 23.2.1
[notice] To update, run: python.exe -m pip install --upgrade pip
```

```
from xgboost import XGBClassifier
```

```
pip install spacy
```

```
Note: you may need to restart the kernel to use updated packages.
Collecting spacy
  Using cached spacy-3.6.1-cp39-cp39-win_amd64.whl (12.1 MB)
Collecting spacy-legacy<3.1.0,>=3.0.11 (from spacy)
  Using cached spacy_legacy-3.0.12-py2.py3-none-any.whl (29 kB)
Collecting spacy-loggers<2.0.0,>=1.0.0 (from spacy)
  Using cached spacy_loggers-1.0.5-py3-none-any.whl (22 kB)
Collecting murmurhash<1.1.0,>=0.28.0 (from spacy)
  Using cached murmurhash-1.0.10-cp39-cp39-win_amd64.whl (25 kB)
Collecting cymem<2.1.0,>=2.0.2 (from spacy)
  Using cached cymem-2.0.8-cp39-cp39-win_amd64.whl (39 kB)
Collecting preshed<3.1.0,>=3.0.2 (from spacy)
  Using cached preshed-3.0.9-cp39-cp39-win_amd64.whl (122 kB)
Collecting thinc<8.2.0,>=8.1.8 (from spacy)
  Using cached thinc-8.1.12-cp39-cp39-win_amd64.whl (1.5 MB)
Collecting wasabi<1.2.0,>=0.9.1 (from spacy)
  Using cached wasabi-1.1.2-py3-none-any.whl (27 kB)
Collecting srsly<3.0.0,>=2.4.3 (from spacy)
```

```
  Using cached srsly-2.4.7-cp39-cp39-win_amd64.whl (483 kB)
Collecting catalogue<2.1.0,>=2.0.6 (from spacy)
  Using cached catalogue-2.0.9-py3-none-any.whl (17 kB)
Collecting typer<0.10.0,>=0.3.0 (from spacy)
  Using cached typer-0.9.0-py3-none-any.whl (45 kB)
Collecting pathy>=0.10.0 (from spacy)
  Using cached pathy-0.10.2-py3-none-any.whl (48 kB)
Collecting smart-open<7.0.0,>=5.2.1 (from spacy)
  Using cached smart_open-6.4.0-py3-none-any.whl (57 kB)
Requirement already satisfied: tqdm<5.0.0,>=4.38.0 in c:\users\bhara\appdata\local\programs\python\python39\lib\site-packages (from sp
Requirement already satisfied: numpy>=1.15.0 in c:\users\bhara\appdata\local\programs\python\python39\lib\site-packages (from spacy)
Requirement already satisfied: requests<3.0.0,>=2.13.0 in c:\users\bhara\appdata\local\programs\python\python39\lib\site-packages (fro
Collecting pydantic!=1.8,!=1.8.1,<3.0.0,>=1.7.4 (from spacy)
  Using cached pydantic-2.3.0-py3-none-any.whl (374 kB)
Requirement already satisfied: jinja2 in c:\users\bhara\appdata\local\programs\python\python39\lib\site-packages (from spacy) (3.1.2)
Requirement already satisfied: setuptools in c:\users\bhara\appdata\local\programs\python\python39\lib\site-packages (from spacy) (49
Requirement already satisfied: packaging>=20.0 in c:\users\bhara\appdata\local\programs\python\python39\lib\site-packages (from spacy
Collecting langcodes<4.0.0,>=3.2.0 (from spacy)
  Using cached langcodes-3.3.0-py3-none-any.whl (181 kB)
Collecting annotated-types>=0.4.0 (from pydantic!=1.8,!=1.8.1,<3.0.0,>=1.7.4->spacy)
  Using cached annotated_types-0.5.0-py3-none-any.whl (11 kB)
Collecting pydantic-core==2.6.3 (from pydantic!=1.8,!=1.8.1,<3.0.0,>=1.7.4->spacy)
  Using cached pydantic_core-2.6.3-cp39-none-win_amd64.whl (1.7 MB)
Collecting typing-extensions>=4.6.1 (from pydantic!=1.8,!=1.8.1,<3.0.0,>=1.7.4->spacy)
  Using cached typing_extensions-4.8.0-py3-none-any.whl (31 kB)
Requirement already satisfied: charset-normalizer<4,>=2 in c:\users\bhara\appdata\local\programs\python\python39\lib\site-packages (f
Requirement already satisfied: idna<4,>=2.5 in c:\users\bhara\appdata\local\programs\python\python39\lib\site-packages (from requests<
Requirement already satisfied: urllib3<3,>=1.21.1 in c:\users\bhara\appdata\local\programs\python\python39\lib\site-packages (from req
Requirement already satisfied: certifi>=2017.4.17 in c:\users\bhara\appdata\local\programs\python\python39\lib\site-packages (from req
Collecting blis<0.8.0,>=0.7.8 (from thinc<8.2.0,>=8.1.8->spacy)
  Using cached blis-0.7.10-cp39-cp39-win_amd64.whl (7.4 MB)
Collecting confection<1.0.0,>=0.0.1 (from thinc<8.2.0,>=8.1.8->spacy)
  Using cached confection-0.1.3-py3-none-any.whl (34 kB)
Requirement already satisfied: colorama in c:\users\bhara\appdata\local\programs\python\python39\lib\site-packages (from tqdm<5.0.0,>=
Requirement already satisfied: click<9.0.0,>=7.1.1 in c:\users\bhara\appdata\local\programs\python\python39\lib\site-packages (from ty
Requirement already satisfied: MarkupSafe>=2.0 in c:\users\bhara\appdata\local\programs\python\python39\lib\site-packages (from jinja2
Installing collected packages: cymem, wasabi, typing-extensions, spacy-loggers, spacy-legacy, smart-open, murmurhash, langcodes, catal
  Attempting uninstall: typing-extensions
    Found existing installation: typing_extensions 4.5.0
```

```
pip install transformers
```

```
Collecting transformers
  Downloading transformers-4.33.2-py3-none-any.whl (7.6 MB)
     ---------------------------------------- 7.6/7.6 MB 19.8 MB/s eta 0:00:00
Requirement already satisfied: filelock in /usr/local/lib/python3.10/dist-packages (from transformers) (3.12.2)
Collecting huggingface-hub<1.0,>=0.15.1 (from transformers)
  Downloading huggingface_hub-0.17.2-py3-none-any.whl (294 kB)
     ---------------------------------------- 294.9/294.9 kB 23.1 MB/s eta 0:00:00
Requirement already satisfied: numpy>=1.17 in /usr/local/lib/python3.10/dist-packages (from transformers) (1.23.5)
Requirement already satisfied: packaging>=20.0 in /usr/local/lib/python3.10/dist-packages (from transformers) (23.1)
Requirement already satisfied: pyyaml>=5.1 in /usr/local/lib/python3.10/dist-packages (from transformers) (6.0.1)
Requirement already satisfied: regex!=2019.12.17 in /usr/local/lib/python3.10/dist-packages (from transformers) (2023.6.3)
Requirement already satisfied: requests in /usr/local/lib/python3.10/dist-packages (from transformers) (2.31.0)
Collecting tokenizers!=0.11.3,<0.14,>=0.11.1 (from transformers)
  Downloading tokenizers-0.13.3-cp310-cp310-manylinux_2_17_x86_64.manylinux2014_x86_64.whl (7.8 MB)
     ---------------------------------------- 7.8/7.8 MB 54.1 MB/s eta 0:00:00
Collecting safetensors>=0.3.1 (from transformers)
  Downloading safetensors-0.3.3-cp310-cp310-manylinux_2_17_x86_64.manylinux2014_x86_64.whl (1.3 MB)
     ---------------------------------------- 1.3/1.3 MB 37.5 MB/s eta 0:00:00
Requirement already satisfied: tqdm>=4.27 in /usr/local/lib/python3.10/dist-packages (from transformers) (4.66.1)
Requirement already satisfied: fsspec in /usr/local/lib/python3.10/dist-packages (from huggingface-hub<1.0,>=0.15.1->transformers) (2023
Requirement already satisfied: typing-extensions>=3.7.4.3 in /usr/local/lib/python3.10/dist-packages (from huggingface-hub<1.0,>=0.15.1-
Requirement already satisfied: charset-normalizer<4,>=2 in /usr/local/lib/python3.10/dist-packages (from requests->transformers) (3.2.0)
Requirement already satisfied: idna<4,>=2.5 in /usr/local/lib/python3.10/dist-packages (from requests->transformers) (3.4)
Requirement already satisfied: urllib3<3,>=1.21.1 in /usr/local/lib/python3.10/dist-packages (from requests->transformers) (2.0.4)
Requirement already satisfied: certifi>=2017.4.17 in /usr/local/lib/python3.10/dist-packages (from requests->transformers) (2023.7.22)
Installing collected packages: tokenizers, safetensors, huggingface-hub, transformers
Successfully installed huggingface-hub-0.17.2 safetensors-0.3.3 tokenizers-0.13.3 transformers-4.33.2
```

```
from __future__ import absolute_import, division, print_function, unicode_literals

import pandas as pd
import numpy as np
import re
from matplotlib import pyplot as plt
import string
import joblib

from warnings import filterwarnings
```

```
filterwarnings("ignore")

import nltk
from nltk import pos_tag, word_tokenize
from nltk.stem import WordNetLemmatizer
from nltk.stem.porter import PorterStemmer
from nltk.corpus import stopwords
from sklearn.feature_extraction.text import CountVectorizer

import spacy
from spacy.lang.en.stop_words import STOP_WORDS
import collections
from collections import Counter
from wordcloud import WordCloud, STOPWORDS, ImageColorGenerator

import sklearn
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.utils import shuffle
from sklearn.model_selection import train_test_split
from sklearn.model_selection import StratifiedKFold
from sklearn.model_selection import GridSearchCV
from sklearn.metrics import roc_curve
from sklearn.metrics import auc
from sklearn.metrics import roc_auc_score
from sklearn.metrics import accuracy_score
from sklearn.pipeline import Pipeline
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import AdaBoostClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.preprocessing import LabelEncoder
from sklearn.naive_bayes import MultinomialNB

import tensorflow as tf
from tensorflow.keras import layers
from tensorflow.keras.preprocessing.text import Tokenizer
from tensorflow.keras.preprocessing.sequence import pad_sequences
from tensorflow.keras.optimizers import Adam

from numpy import asarray
from numpy import savetxt
from numpy import loadtxt

import transformers
from transformers import BertTokenizer, TFBertModel
```

```
def preprocessor(text):
    # convert to lowercase, susbstitute non alphanumerical characters with whitespaces
    text = re.sub('<[^>]*>', '', text)
    emoticons = re.findall('(?::|;|=)(?:-)?(?:\)|\(|D|P)', text)
    text = (re.sub('[\W]+', ' ', text.lower()) +
            ' '.join(emoticons).replace('-', ''))
    return text
```

```
def test_performance(model, X_test):
    ''' Given a fitted model and a test set, returns optimal hyperparameters, AUC and accuracy '''
    clf_b = model.best_estimator_
    y_pred_proba = clf_b.predict_proba(X_test)
    y_pred = clf_b.predict(X_test)

    # performance metrics
    auc_res=roc_auc_score(y_test, y_pred_proba[:, 1])
    acc = accuracy_score(y_test, y_pred)

    return (model.best_params_, auc_res, acc)
```

```
stopwords = list(set(stopwords.words("english")))
```

Analytics + NLP preprocessing

- There are no duplicated values in the dataset
- Missing values only in the location feature

- Most tweets have positive sentiment
- On average, each tweet has about 30 words

```
train_data=pd.read_csv("Corona_NLP_test.csv",encoding="latin1")
test_data=pd.read_csv("Corona_NLP_test.csv",encoding="latin1")
```

```
# merge train and test datasets
df = pd.concat([train_data, test_data], axis = 0)
df.shape
```

```
(7596, 6)
```

```
# duplicated tweets
duplicates = df[df.duplicated()]
len(duplicates)
```

```
3798
```

```
df.isnull().sum()
```

```
UserName            0
ScreenName          0
Location         1668
TweetAt             0
OriginalTweet       0
Sentiment           0
dtype: int64
```

```
df.head()
```

	UserName	ScreenName	Location	TweetAt	OriginalTweet	Sentiment
0	1	44953	NYC	02-03-2020	TRENDING: New Yorkers encounter empty supermar...	Extremely Negative
1	2	44954	Seattle, WA	02-03-2020	When I couldn't find hand sanitizer at Fred Me...	Positive
2	3	44955	NaN	02-03-2020	Find out how you can protect yourself and love...	Extremely Positive
					#Panic buying hits	

```
counts = df.groupby("Sentiment")["Sentiment"].agg("count")

fig, ax = plt.subplots(figsize=(10,5))
counts.plot.bar()
```

```
<Axes: xlabel='Sentiment'>
```


```
# remove neutral tweets
df.drop(df[df.Sentiment == "Neutral"].index, inplace = True)

df.Sentiment.unique()
```

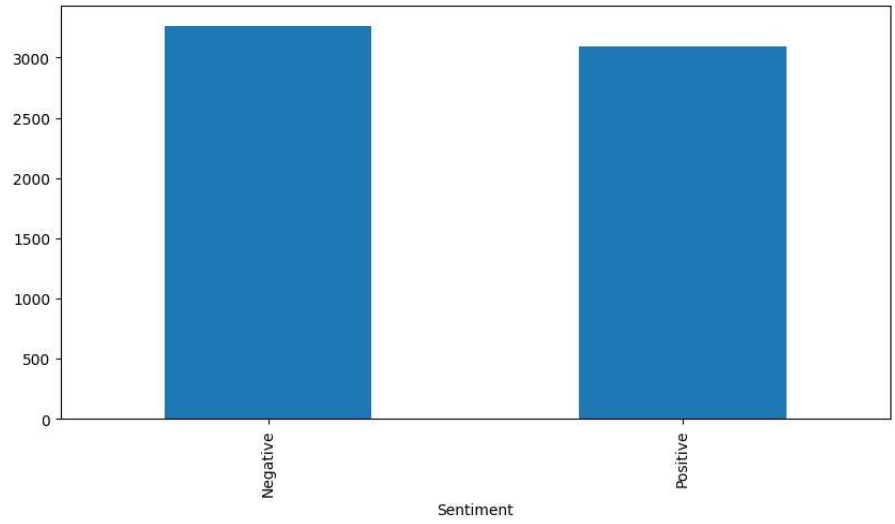
```
array(['Extremely Negative', 'Positive', 'Extremely Positive', 'Negative'],
      dtype=object)
```


```
# Aggregate extremely positive/negative values
df = df.replace("Extremely Positive", "Positive")
df = df.replace("Extremely Negative", "Negative")

df.Sentiment.unique()
```

```
array(['Negative', 'Positive'], dtype=object)
```

```
counts = df.groupby("Sentiment")["Sentiment"].agg("count")
fig, ax = plt.subplots(figsize=(10,5))

counts.plot.bar()
```

```
<Axes: xlabel='Sentiment'>
```




```
df.head()
```

	UserName	ScreenName	Location	TweetAt	OriginalTweet	Sentiment
0	1	44953	NYC	02-03-2020	TRENDING: New Yorkers encounter empty supermar...	Negative
1	2	44954	Seattle, WA	02-03-2020	When I couldn't find hand sanitizer at Fred Me...	Positive
2	3	44955	NaN	02-03-2020	Find out how you can protect yourself and love...	Positive
				02-03-	#Panic buying hits #NewYork	

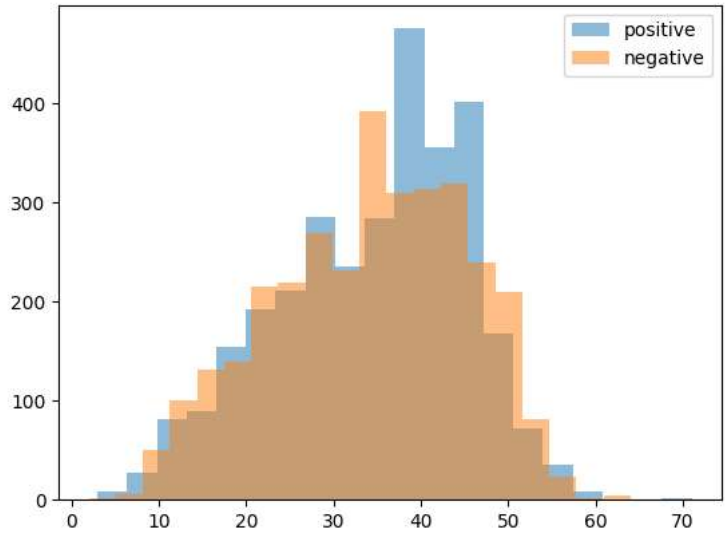
```
# count number of words per headline. strip whitespaces at the beginning/end of the sentence
# and tokenize by whitespace
df_select = df[["OriginalTweet", "Sentiment"]]
df_select["word_count"] = df_select["OriginalTweet"].apply(lambda x: len(x.strip().split(" ")))
df_select.head()
```

	OriginalTweet	Sentiment	word_count
0	TRENDING: New Yorkers encounter empty supermar...	Negative	23
1	When I couldn't find hand sanitizer at Fred Me...	Positive	31
2	Find out how you can protect yourself and love...	Positive	13
3	#Panic buying hits #NewYork City as anxious sh...	Negative	35

```
# summary statistics of word counts
df_select["word_count"].describe()
```

```
count    6358.000000
mean       34.329663
std        10.979474
min         2.000000
25%        26.000000
50%        36.000000
75%        43.000000
max        71.000000
Name: word_count, dtype: float64
```

```
# distributions of word counts by sentiment
pos = df_select[df_select.Sentiment == "Positive"].word_count
neg =  df_select[df_select.Sentiment == "Negative"].word_count

plt.hist(pos, bins = 20, alpha = 0.5, label='positive')
plt.hist(neg, bins = 20, alpha = 0.5, label='negative')
plt.legend(loc='upper right')
plt.show()
```




```
df_select
```

	OriginalTweet	Sentiment	word count
1	When I couldn't find hand sanitizer at Fred Me...	Positive	31

```
# save dataset
df_select.to_csv("./binary_data.csv",index = False, encoding = "utf-8")
```

NLP basic preprocessing

- Convert tweets to lowercase
- Remove punctuation and non alphanumerical characters

```
df = pd.read_csv("./binary_data.csv")
```

3793	Meanwhile In A Supermarket in Israel -- People...	Positive	18

```
# select random tweet
text = df.OriginalTweet.iloc[400]
text
```

```
'Have been talking to lots of Trump supporters across the country these last couple of
days, and the disdain for and distrust of the media right now is worse than I've seen a
t any point throughout this presidency.'
```

```
# preprocess entire corpus
df['tweet_clean'] = df['OriginalTweet'].apply(preprocessor)
df.head()
```

	OriginalTweet	Sentiment	word_count	tweet_clean
0	TRENDING: New Yorkers encounter empty supermar...	Negative	23	trending new yorkers encounter empty supermark...
1	When I couldn't find hand sanitizer at Fred Me...	Positive	31	when i couldn t find hand sanitizer at fred me...
2	Find out how you can protect yourself and love...	Positive	13	find out how you can protect yourself and love...
3	#Panic buying hits #NewYork City			panic buying hits newyork city as

Bag-of- models

Bag-of- models are used to produce a real-valued representation of a text document that can used to train a ML model.

- Bag-of-words: count (with normalization) the occurences of each token in a text
- Bag-of-POS: count the occurence of each POS tag in a text. A POS is a set of words with similar sytactic behaviour (nouns, adjective, verbs, pronouns etc). POS tags are assigned to every word in a document and used in bag-of machine learning models.

Bag-of-word models suffer from high dimensionality and sparseness of the real-valued representation. Bag-of-POS is usually more parsimonious than bag-of-words, since there usually are less POS-tags than words in a document, and can therefore be used to reduce dimensionality.

```
# tokenize the dataset with nltk tokenizer
import nltk
nltk.download('punkt')
text_processed = df.tweet_clean.apply( word_tokenize )
text_processed
```

```
[nltk_data] Downloading package punkt to /root/nltk_data...
[nltk_data]   Package punkt is already up-to-date!
0       [trending, new, yorkers, encounter, empty, sup...
1       [when, i, couldn, t, find, hand, sanitizer, at...
2       [find, out, how, you, can, protect, yourself, ...
3       [panic, buying, hits, newyork, city, as, anxio...
4       [voting, in, the, age, of, coronavirus, hand, ...
                              ...
6353    [ricepolitics, mdcounties, craig, will, you, c...
6354    [meanwhile, in, a, supermarket, in, israel, pe...
6355    [did, you, panic, buy, a, lot, of, non, perish...
6356    [gov, need, to, do, somethings, instead, of, b...
6357    [i, and, forestandpaper, members, are, committ...
Name: tweet_clean, Length: 6358, dtype: object
```

```
# POS-tags for the whole dataset
nltk.download('all')
```

```
text_tagged = text_processed.apply(pos_tag)
text_tagged
```

```
[nltk_data] Downloading collection 'all'
[nltk_data]    |
[nltk_data]    | Downloading package abc to /root/nltk_data...
[nltk_data]    |   Unzipping corpora/abc.zip.
[nltk_data]    | Downloading package alpino to /root/nltk_data...
[nltk_data]    |   Unzipping corpora/alpino.zip.
[nltk_data]    | Downloading package averaged_perceptron_tagger to
[nltk_data]    |     /root/nltk_data...
[nltk_data]    |   Unzipping taggers/averaged_perceptron_tagger.zip.
[nltk_data]    | Downloading package averaged_perceptron_tagger_ru to
[nltk_data]    |     /root/nltk_data...
[nltk_data]    |   Unzipping
[nltk_data]    |       taggers/averaged_perceptron_tagger_ru.zip.
[nltk_data]    | Downloading package basque_grammars to
[nltk_data]    |     /root/nltk_data...
[nltk_data]    |   Unzipping grammars/basque_grammars.zip.
[nltk_data]    | Downloading package bcp47 to /root/nltk_data...
[nltk_data]    | Downloading package biocreative_ppi to
[nltk_data]    |     /root/nltk_data...
[nltk_data]    |   Unzipping corpora/biocreative_ppi.zip.
[nltk_data]    | Downloading package bllip_wsj_no_aux to
[nltk_data]    |     /root/nltk_data...
[nltk_data]    |   Unzipping models/bllip_wsj_no_aux.zip.
[nltk_data]    | Downloading package book_grammars to
[nltk_data]    |     /root/nltk_data...
[nltk_data]    |   Unzipping grammars/book_grammars.zip.
[nltk_data]    | Downloading package brown to /root/nltk_data...
[nltk_data]    |   Unzipping corpora/brown.zip.
[nltk_data]    | Downloading package brown_tei to /root/nltk_data...
[nltk_data]    |   Unzipping corpora/brown_tei.zip.
[nltk_data]    | Downloading package cess_cat to /root/nltk_data...
[nltk_data]    |   Unzipping corpora/cess_cat.zip.
[nltk_data]    | Downloading package cess_esp to /root/nltk_data...
[nltk_data]    |   Unzipping corpora/cess_esp.zip.
[nltk_data]    | Downloading package chat80 to /root/nltk_data...
[nltk_data]    |   Unzipping corpora/chat80.zip.
[nltk_data]    | Downloading package city_database to
[nltk_data]    |     /root/nltk_data...
[nltk_data]    |   Unzipping corpora/city_database.zip.
[nltk_data]    | Downloading package cmudict to /root/nltk_data...
[nltk_data]    |   Unzipping corpora/cmudict.zip.
[nltk_data]    | Downloading package comparative_sentences to
[nltk_data]    |     /root/nltk_data...
[nltk_data]    |   Unzipping corpora/comparative_sentences.zip.
[nltk_data]    | Downloading package comtrans to /root/nltk_data...
[nltk_data]    | Downloading package conll2000 to /root/nltk_data...
[nltk_data]    |   Unzipping corpora/conll2000.zip.
[nltk_data]    | Downloading package conll2002 to /root/nltk_data...
[nltk_data]    |   Unzipping corpora/conll2002.zip.
[nltk_data]    | Downloading package conll2007 to /root/nltk_data...
[nltk_data]    | Downloading package crubadan to /root/nltk_data...
[nltk_data]    |   Unzipping corpora/crubadan.zip.
[nltk_data]    | Downloading package dependency_treebank to
[nltk_data]    |     /root/nltk_data...
[nltk_data]    |   Unzipping corpora/dependency_treebank.zip.
[nltk_data]    | Downloading package dolch to /root/nltk_data...
[nltk_data]    |   Unzipping corpora/dolch.zip.
[nltk_data]    | Downloading package europarl_raw to
```

```
def retrieve_tags(text):
    # form a string of tags from POS-tag text
    return "-".join(tag for (word, tag) in text)

df["text_pos"] = text_tagged.apply(retrieve_tags)
df.text_pos
```

```
0       VBG-JJ-NNS-VBP-JJ-NN-NNS-VBD-NNS-IN-NN-VBN-RP-...
1       WRB-NN-VBP-NN-VBP-NN-NN-IN-JJ-NN-NN-VBD-TO-VB-...
2                 VB-RP-WRB-PRP-MD-VB-PRP-CC-VBD-NNS-IN-NN
3       JJ-NN-NNS-VBP-NN-IN-JJ-NNS-NN-RB-IN-NN-NN-JJ-N...
4                  NN-IN-DT-NN-IN-NN-NN-NN-NN-VBD-JJ-NN-NN
                              ...
6353    NNS-NNS-VBP-MD-PRP-VB-IN-DT-JJ-NN-TO-VB-DT-JJ-...
6354    RB-IN-DT-NN-IN-JJ-NNS-NN-CC-VBG-RB-TO-VB-JJ-NN...
6355    VBD-PRP-VB-VB-DT-NN-IN-NN-JJ-NNS-VBP-NNS-NN-NN...
6356    NNS-VBP-TO-VB-NNS-RB-IN-JJ-NN-NN-VBP-JJ-NN-CC-...
6357    NN-CC-JJ-NNS-VBP-VBN-TO-DT-NN-IN-PRP$-NNS-CC-P...
Name: text_pos, Length: 6358, dtype: object
```

Word embeddings

Associate a real-valued vector to each word in a document, so that words that occur in similar context (i.e. are semantically close) have a similar vector representation.

- word2vec: compute word embeddings using the concept of context, i.e. the words around a certain word in the text. The embeddings are calculated based on the probability of each word to appear in the context of every other word (word-word co-occurrences).
- Glove: unsupervised learning model trained on word-word co-occurence counts, computed with respect to the entire document as opposed to one context window at a time. Embeddings are computed based on the probability of a word to appear in the context of another word, given the entire vocabulary.

Spacy offers pre-trained word embedding models (such as "en_core_web_md")

```
# load pre-trained word embedding model from spacy
!python -m spacy download en_core_web_md

nlp = spacy.load('en_core_web_md')
```

```
2023-09-19 21:16:39.714180: W tensorflow/compiler/tf2tensorrt/utils/py_utils.cc:38] TF-TRT Warning: Could not find TensorRT
Collecting en-core-web-md==3.6.0
  Downloading https://github.com/explosion/spacy-models/releases/download/en_core_web_md-3.6.0/en_core_web_md-3.6.0-py3-none-any.whl (42
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 42.8/42.8 MB 12.2 MB/s eta 0:00:00
Requirement already satisfied: spacy<3.7.0,>=3.6.0 in /usr/local/lib/python3.10/dist-packages (from en-core-web-md==3.6.0) (3.6.1)
Requirement already satisfied: spacy-legacy<3.1.0,>=3.0.11 in /usr/local/lib/python3.10/dist-packages (from spacy<3.7.0,>=3.6.0->en-core
Requirement already satisfied: spacy-loggers<2.0.0,>=1.0.0 in /usr/local/lib/python3.10/dist-packages (from spacy<3.7.0,>=3.6.0->en-core
Requirement already satisfied: murmurhash<1.1.0,>=0.28.0 in /usr/local/lib/python3.10/dist-packages (from spacy<3.7.0,>=3.6.0->en-core-w
Requirement already satisfied: cymem<2.1.0,>=2.0.2 in /usr/local/lib/python3.10/dist-packages (from spacy<3.7.0,>=3.6.0->en-core-web-md=
Requirement already satisfied: preshed<3.1.0,>=3.0.2 in /usr/local/lib/python3.10/dist-packages (from spacy<3.7.0,>=3.6.0->en-core-web-m
Requirement already satisfied: thinc<8.2.0,>=8.1.8 in /usr/local/lib/python3.10/dist-packages (from spacy<3.7.0,>=3.6.0->en-core-web-md=
Requirement already satisfied: wasabi<1.2.0,>=0.9.1 in /usr/local/lib/python3.10/dist-packages (from spacy<3.7.0,>=3.6.0->en-core-web-mo
Requirement already satisfied: srsly<3.0.0,>=2.4.3 in /usr/local/lib/python3.10/dist-packages (from spacy<3.7.0,>=3.6.0->en-core-web-md=
Requirement already satisfied: catalogue<2.1.0,>=2.0.6 in /usr/local/lib/python3.10/dist-packages (from spacy<3.7.0,>=3.6.0->en-core-web
Requirement already satisfied: typer<0.10.0,>=0.3.0 in /usr/local/lib/python3.10/dist-packages (from spacy<3.7.0,>=3.6.0->en-core-web-mo
Requirement already satisfied: pathy>=0.10.0 in /usr/local/lib/python3.10/dist-packages (from spacy<3.7.0,>=3.6.0->en-core-web-md==3.6.0
Requirement already satisfied: smart-open<7.0.0,>=5.2.1 in /usr/local/lib/python3.10/dist-packages (from spacy<3.7.0,>=3.6.0->en-core-we
Requirement already satisfied: tqdm<5.0.0,>=4.38.0 in /usr/local/lib/python3.10/dist-packages (from spacy<3.7.0,>=3.6.0->en-core-web-md=
Requirement already satisfied: numpy>=1.15.0 in /usr/local/lib/python3.10/dist-packages (from spacy<3.7.0,>=3.6.0->en-core-web-md==3.6.0
Requirement already satisfied: requests<3.0.0,>=2.13.0 in /usr/local/lib/python3.10/dist-packages (from spacy<3.7.0,>=3.6.0->en-core-web
Requirement already satisfied: pydantic!=1.8,!=1.8.1,<3.0.0,>=1.7.4 in /usr/local/lib/python3.10/dist-packages (from spacy<3.7.0,>=3.6.0
Requirement already satisfied: jinja2 in /usr/local/lib/python3.10/dist-packages (from spacy<3.7.0,>=3.6.0->en-core-web-md==3.6.0) (3.1.
Requirement already satisfied: setuptools in /usr/local/lib/python3.10/dist-packages (from spacy<3.7.0,>=3.6.0->en-core-web-md==3.6.0) (
Requirement already satisfied: packaging>=20.0 in /usr/local/lib/python3.10/dist-packages (from spacy<3.7.0,>=3.6.0->en-core-web-md==3.6
Requirement already satisfied: langcodes<4.0.0,>=3.2.0 in /usr/local/lib/python3.10/dist-packages (from spacy<3.7.0,>=3.6.0->en-core-web
Requirement already satisfied: typing-extensions>=4.2.0 in /usr/local/lib/python3.10/dist-packages (from pydantic!=1.8,!=1.8.1,<3.0.0,>=
Requirement already satisfied: charset-normalizer<4,>=2 in /usr/local/lib/python3.10/dist-packages (from requests<3.0.0,>=2.13.0->spacy<
Requirement already satisfied: idna<4,>=2.5 in /usr/local/lib/python3.10/dist-packages (from requests<3.0.0,>=2.13.0->spacy<3.7.0,>=3.6.
Requirement already satisfied: urllib3<3,>=1.21.1 in /usr/local/lib/python3.10/dist-packages (from requests<3.0.0,>=2.13.0->spacy<3.7.0,
Requirement already satisfied: certifi>=2017.4.17 in /usr/local/lib/python3.10/dist-packages (from requests<3.0.0,>=2.13.0->spacy<3.7.0,
Requirement already satisfied: blis<0.8.0,>=0.7.8 in /usr/local/lib/python3.10/dist-packages (from thinc<8.2.0,>=8.1.8->spacy<3.7.0,>=3.
Requirement already satisfied: confection<1.0.0,>=0.0.1 in /usr/local/lib/python3.10/dist-packages (from thinc<8.2.0,>=8.1.8->spacy<3.7.
Requirement already satisfied: click<9.0.0,>=7.1.1 in /usr/local/lib/python3.10/dist-packages (from typer<0.10.0,>=0.3.0->spacy<3.7.0,>=
Requirement already satisfied: MarkupSafe>=2.0 in /usr/local/lib/python3.10/dist-packages (from jinja2->spacy<3.7.0,>=3.6.0->en-core-web
Installing collected packages: en-core-web-md
Successfully installed en-core-web-md-3.6.0
✓ Download and installation successful
You can now load the package via spacy.load('en_core_web_md')
```

```
# example on random tweet
print( nlp(text) )
print( )
print( nlp(text).vector[:20])
```

```
Have been talking to lots of Trump supporters across the country these last couple of days, and the disdain for and distrust of the medi

[-2.2430875   0.31259587 -2.511582    1.1530114   5.0358505   1.0344031
  0.76018673  5.1734204   0.21933225  0.474815    6.314838    2.230242
 -3.6703875   0.5104935   0.81468946  2.1584044   1.1790328  -1.8672558
 -1.9416398  -1.9658836 ]
```

```
# stack embeddings of tweets
# each tweet has a 1x300 embedding
emb = np.vstack(df.tweet_clean.apply(lambda x: nlp(x).vector))
len(emb[0])
```

```
300
```

```
# convert embeddings to dataframe
emb_df = pd.DataFrame(emb, columns = np.array([str(x) for x in range(0, len(emb[0]))]) )
print(emb_df.shape)
emb_df.head()
```

```
(6358, 300)
```

	0	1	2	3	4	5	6	7	
0	-0.891958	0.325154	-2.062366	1.340930	2.533503	0.112686	0.823051	2.036370	-1.687
1	-1.256169	0.411010	-2.785302	-0.033508	2.118214	0.287657	0.180948	3.885340	-2.748
2	0.879349	0.786262	-5.566750	-1.598292	1.157152	1.936206	-0.577861	2.262260	-3.088
3	-0.733919	0.446307	-2.309876	0.746547	3.198810	-0.813633	0.090123	2.263667	-0.182
4	-2.309330	0.901132	-0.473967	1.618246	4.785238	0.299894	1.590768	4.593878	-0.224

5 rows × 300 columns

```
# add embeddings to dataset
df_embed = pd.concat([df, emb_df], axis = 1)
df = df_embed.drop(columns=["OriginalTweet",'word_count'])
```

Convert target label to real-valued

```
# convert sentiment label to real-valued target
le = LabelEncoder()
le.fit(df.Sentiment)

df['target_encoded'] = le.transform(df.Sentiment)
df.target_encoded
```

```
0       0
1       1
2       1
3       0
4       1
       ..
6353    0
6354    1
6355    0
6356    0
6357    1
Name: target_encoded, Length: 6358, dtype: int64
```

```
# save dataset
df.to_csv("./preprocessed_data.csv",index = False, encoding = "utf-8")
```

Sentiment analysis with machine learning models

We consider the following classifiers:

- Multinomial Naive Bayes Model (benchmark)
- Random forests
- Adaptive boosting (ADABoost)
- Extreme Gradient Boosting (XGBoost)

The machine learning models are trained on a real-valued representation of the training data, produced with the following nlp preprocesssing methods:

- Bag-of-words model
- Bag-of-POS model
- Pre-trained word embeddings

To reduce computational time, we train the machine learning models on a random sample with dimensions equal to 30% of the original data

```
df_imp = pd.read_csv("./preprocessed_data.csv")
df_imp.shape
```

```
(6358, 304)
```

```
# train the models on about 30% of the original data
df = df_imp.sample(frac = 0.3 )
df.shape
```

```
(1907, 304)
```

```
# reshuffle data
np.random.seed(0)
df = df.reindex(np.random.permutation(df.index))
df = df.reset_index(drop=True) # reset the index after the permutation
```

```
# define train and test size
train_size = int( len(df)*0.8 )
test_size = int( len(df) - train_size )
print(train_size, test_size)
```

```
1525 382
```

```
# split dataset
df_train = df.head(train_size)
y_train = df.head(train_size).target_encoded

df_test = df.tail(test_size)
y_test = df.tail(test_size).target_encoded
```

```
# set tfidf and cross validation parameters for all models
tfidf = TfidfVectorizer(strip_accents=None,
                        lowercase=False,
                        preprocessor=None)

cv = StratifiedKFold(n_splits=5,
                     shuffle=False)
```

Multinomial naive bayes model

Probabilistic classifier with independence assumption between fetures

The model is trained on the real-valued representation of the tweets produces by nltk CountVectorizer(), which transforms text into a vector of token counts. In the resulting sparse matrix, each column represents a unique word in the vocabulary, and each row represents a tweet in the dataset. The values of the matrix are the word counts. Words that do not appear in a certain tweet are given the value zero.

```
X_train = df_train.tweet_clean
X_test = df_test.tweet_clean
```

```
# convert tweets into a real-valued matrix of token counts (sparse matrix with dim nr tweets, nr words)
vect = CountVectorizer(stop_words='english')
vect.fit(X_train)

x_train_dtm = vect.transform(X_train)
x_test_dtm = vect.transform(X_test)
```

```
# nrows = nr tweets, ncols = nr words
print("Shape training data: {}".format(x_train_dtm.shape) + "\nShape test data:{}".format(x_test_dtm.shape))
```

```
Shape training data: (1525, 7066)
Shape test data:(382, 7066)
```

```
# Multinomial Naive Bayes model
model = MultinomialNB()
model.fit(x_train_dtm, y_train)
# save fitted model
joblib.dump(model, "./NaiveBayes.joblib")
```

```
['./NaiveBayes.joblib']
```

```
# predict target label (0 or 1)
y_pred = model.predict(x_test_dtm)

# predict probability of class == 1
y_pred_prob = model.predict_proba(x_test_dtm)

# performance metrics
auc_res = list()
acc = list()

auc_res.append( roc_auc_score(y_test, y_pred_prob[:, 1]) )
acc.append( accuracy_score(y_test, y_pred) )
```

Random Forests

Random forests are an ensemble learning method for classification and regression that operates by constructing a multitude of decision trees at training time. Random decision forests correct for decision trees' habit of overfitting to their training set.

The training algorithm for random forests applies the general technique of bootstrap aggregating, or bagging, to tree learners. Given a training set $X = x_1, \ldots, x_n$ with responses $Y = y_1, \ldots, y_n$, bagging repeatedly (B times) selects a random sample with replacement of the training set and fits trees to these samples:

For $b = 1, \ldots, B$:

- Sample, with replacement, n training examples from X, Y; call these X_b, Y_b.
- Train a classification or regression tree f_b on X_b, Y_b.

After training, predictions for unseen samples can be made by averaging the predictions from all the individual regression trees on or by taking the majority vote in the case of classification trees.

This bootstrapping procedure leads to better model performance because it decreases the variance of the model, without increasing the bias. This means that while the predictions of a single tree are highly sensitive to noise in its training set, the average of many trees is not, as long as the trees are not correlated. Simply training many trees on a single training set would give strongly correlated trees; bootstrap sampling is a way of de-correlating the trees by showing them different training sets.

Random forests also include another type of bagging scheme: they use a modified tree learning algorithm that selects, at each candidate split in the learning process, a random subset of the features. This process is sometimes called "feature bagging". The reason for doing this is the correlation of the trees in an ordinary bootstrap sample: if one or a few features are very strong predictors for the response variable, these features will be selected in many of the $B trees, causing them to become correlated.

For bag-of models, NLP preprocessing is performed by sklearn TfidfVectorizer(), which implements lowercasing, tokenization and stopword removal and produces a "document-term matrix" with tf-idf normalization via the .fit transform() method.

Input of TfidfVectorizer():

- Bag-of-words: clean tweets
- Bag-of-POS: strings of POS tags

For pretrained word embeddings, NLP preprocessing is not necessary and the model is trained directly on the word embeddings.

```
tfidf = TfidfVectorizer(strip_accents=None,
                        lowercase=False,
                        preprocessor=None)
```

```
# cross validation parameters
cv = StratifiedKFold(n_splits=5,
                     shuffle=False)
```

Bag of words

```
X_train = df_train.tweet_clean
X_test = df_test.tweet_clean
model_name = "randomforest_BOW"
```

```
pipeline = Pipeline([('vect', tfidf),
                     ('clf', RandomForestClassifier())]) # use default base learner (decision trees)
```

```
param_grid = {'vect__ngram_range': [(1, 1)],
              'vect__stop_words': [stopwords, None],
              'vect__max_df': [1.0, 0.1, 0.3, 0.5],
              'vect__max_features': [None, 1000],
              'clf__n_estimators': [100, 200, 300, 400], # number of trees trained in the ensamble
              'clf__max_depth': [1, 5, 10] # tree depth
              }

model = GridSearchCV(pipeline,
                     param_grid,
                     scoring='roc_auc',
                     cv=cv,
                     n_jobs = -1,
                     verbose = 1)
```

```
model.fit(X_train, y_train)

# save fitted model
joblib.dump(model, model_name + ".joblib")
```

```
# load fitted model
model = joblib.load( "" + model_name + ".joblib")
```

```
params, auc, accuracy = test_performance(model, X_test)

auc_res.append( auc )
acc.append( accuracy )

print('Best parameter set: %s ' % params)
```

```
Best parameter set: {'clf__max_depth': 10, 'clf__n_estimators': 400, 'vect__max_df': 0.1, 'vect__max_features': None, 'vect__ngram_range
```

Bag of POS

```
X_train = df_train.text_pos
X_test = df_test.text_pos
model_name = "randomforest_POS"
```

```
param_grid = {'vect__ngram_range': [(1, 1)],
              'clf__n_estimators': [100, 200, 300, 400],
              'clf__max_depth': [1, 5, 10]}

model = GridSearchCV(pipeline,
                     param_grid,
                     scoring='roc_auc',
                     cv=cv,
                     n_jobs = -1,
                     verbose = 1)
```

```
model.fit(X_train, y_train)

# save fitted model
joblib.dump(model,  model_name + ".joblib")
```

```
# load fitted model
model = joblib.load( "" + model_name + ".joblib")
```

```
params, auc, accuracy = test_performance(model, X_test)

auc_res.append( auc )
acc.append( accuracy )

print('Best parameter set: %s ' % params)
```

```
Best parameter set: {'clf__max_depth': 10, 'clf__n_estimators': 300, 'vect__ngram_range': (1, 1)}
```

Embeddings

```
# select columns corresponding to embeddings
select = [i for i in list(df.columns) if i not in ["Sentiment", "tweet_clean", "text_pos", "target_encoded"]]

X_train = df_train[select]
X_test = df_test[select]
model_name = "randomforest_EMB"
```

```
pipe = Pipeline([('clf', RandomForestClassifier())])

param_grid = {'clf__n_estimators': [100, 200, 300, 400],
              'clf__max_depth': [1, 5, 10]}

model = GridSearchCV(pipe, param_grid,
                     scoring='roc_auc',
                     cv=cv,
                     n_jobs=-1,
                     verbose = 1)
```

```
model.fit(X_train, y_train)

# save fitted model
joblib.dump(model, model_name + ".joblib")
```

```
# load fitted model
model = joblib.load("" + model_name + ".joblib")
```

```
params, auc, accuracy = test_performance(model, X_test)

auc_res.append( auc )
acc.append( accuracy )

print('Best parameter set: %s ' % params)
```

```
Best parameter set: {'clf__max_depth': 10, 'clf__n_estimators': 400}
```

Boosting Models

Boosting algorithms consist of iteratively training weak classifiers with respect to a distribution and adding them to a final strong classifier. When they are added, they are weighted in a way that is related to the weak learners' accuracy. After a weak learner is added, the data weights are readjusted, known as "re-weighting". Misclassified input data gain a higher weight and examples that are classified correctly lose weight. Thus, future weak learners focus more on the examples that previous weak learners misclassified.

The main variation between many boosting algorithms is their method of weighting training data points and hypotheses.

ADABoost

Bag of words

```
X_train = df_train.tweet_clean
X_test = df_test.tweet_clean
model_name = "ADA_BOW"
```

```
tree = DecisionTreeClassifier(max_depth=5) # use decision tree with max depth 5 as base learners

pipeline = Pipeline([('vect', tfidf),
                     ('clf', AdaBoostClassifier(base_estimator = tree))]) # use default base learner (decision trees)

param_grid = {'vect__ngram_range': [(1, 1)],
              'vect__stop_words': [stopwords, None],
              'vect__max_df': [1.0, 0.1, 0.3, 0.5],
              'vect__max_features': [None, 1000],
              'clf__n_estimators': [100, 200, 300, 400], # number of trees trained in the ensamble
              'clf__learning_rate': [0.001, 0.01, 0.1, 1.0]
              }
```

```
model = GridSearchCV(pipeline,
                     param_grid,
                     scoring='roc_auc',
                     cv=cv,
                     n_jobs = -1,
                     verbose = 1)
```

```
model.fit(X_train, y_train)

# save fitted model
joblib.dump(model,  model_name + ".joblib")
```

```
# load fitted model
model = joblib.load( "" + model_name + ".joblib")
```

```
params, auc, accuracy = test_performance(model, X_test)

auc_res.append( auc )
acc.append( accuracy )

print('Best parameter set: %s ' % params)
```

```
Best parameter set: {'clf__learning_rate': 0.1, 'clf__n_estimators': 200, 'vect__max_df': 0.1, 'vect__max_features': None, 'vect__ngram_
```

Bag of POS

```
X_train = df_train.text_pos
X_test = df_test.text_pos
model_name = "ADA_POS"
```

```
param_grid = {'vect__ngram_range': [(1, 1)],
              'clf__n_estimators': [100, 200, 300, 400],
              'clf__learning_rate': [0.001, 0.01, 0.1, 1.0]}

model = GridSearchCV(pipeline,
                     param_grid,
                     scoring='roc_auc',
                     cv=cv,
                     n_jobs = -1,
                     verbose = 1)
```

```
model.fit(X_train, y_train)

# save fitted model
joblib.dump(model, model_name + ".joblib")
```

```
# load fitted model
model = joblib.load( "" + model_name + ".joblib")
```

```
params, auc, accuracy = test_performance(model, X_test)

auc_res.append( auc )
acc.append( accuracy )

print('Best parameter set: %s ' % params)
```

```
Best parameter set: {'clf__learning_rate': 0.001, 'clf__n_estimators': 400, 'vect__ngram_range': (1, 1)}
```

Embeddings

```
# select columns corresponding to embeddings
select = [i for i in list(df.columns) if i not in ["Sentiment", "tweet_clean", "text_pos", "target_encoded"]]

X_train = df_train[select]
X_test = df_test[select]
model_name = "ADA_EMB"
```

```
pipe = Pipeline([('clf', AdaBoostClassifier(base_estimator=tree))])

param_grid = {'clf__n_estimators': [100, 200, 300, 400],
              'clf__learning_rate': [0.001, 0.01, 0.1, 1.0]}

model = GridSearchCV(pipe, param_grid,
                     scoring='roc_auc',
                     cv=cv,
                     n_jobs=-1,
                     verbose = 1)
```

```
model.fit(X_train, y_train)

# save fitted model
joblib.dump(model, model_name + ".joblib")
```

```
Fitting 5 folds for each of 16 candidates, totalling 80 fits
---------------------------------------------------------------------------
KeyboardInterrupt                         Traceback (most recent call last)
<ipython-input-95-a5853140fb22> in <cell line: 1>()
----> 1 model.fit(X_train, y_train)
      2
      3 # save fitted model
      4 joblib.dump(model, model_name + ".joblib")

 6 frames
/usr/local/lib/python3.10/dist-packages/joblib/parallel.py in _retrieve(self)
   1705                 (self._jobs[0].get_status(
   1706                     timeout=self.timeout) == TASK_PENDING)):
-> 1707                 time.sleep(0.01)
   1708                 continue
   1709

KeyboardInterrupt:

SEARCH STACK OVERFLOW
```

```
# load fitted model
model = joblib.load( "" + model_name + ".joblib")
```

```
params, auc, accuracy = test_performance(model, X_test)

auc_res.append( auc )
acc.append( accuracy )

print('Best parameter set: %s ' % params)
```

```
Best parameter set: {'clf__learning_rate': 0.01, 'clf__n_estimators': 400}
```

XGBoost

Bag of words

```
X_train = df_train.tweet_clean
X_test = df_test.tweet_clean
model_name = "XGB_BOW"
```

```
from sklearn.pipeline import Pipeline
from sklearn.feature_extraction.text import TfidfVectorizer
from xgboost import XGBClassifier  # Import XGBClassifier from xgboost

# Create a TfidfVectorizer and XGBClassifier pipeline
```

```
pipeline = Pipeline([
    ('vect', TfidfVectorizer()),
    ('clf', XGBClassifier())  # Use XGBClassifier as the classifier
])

# Rest of your code...
```

```
pipeline = Pipeline([('vect', tfidf),
                     ('clf', XGBClassifier())]) # use default base learner (decision trees)

param_grid = {'vect__ngram_range': [(1, 1)],
              'vect__stop_words': [stopwords, None],
              'vect__max_df': [1.0, 0.1, 0.3, 0.5],
              'vect__max_features': [None, 1000],
              'clf__n_estimators': [100, 200, 300, 400], # number of trees trained in the ensamble
              'clf__learning_rate': [0.001, 0.01, 0.1, 1.0],
              'clf__max_depth': [1, 5, 10] # tree depth
              }

model = GridSearchCV(pipeline,
                     param_grid,
                     scoring='roc_auc',
                     cv=cv,
                     n_jobs = -1,
                     verbose = 1)
```

```
model.fit(X_train, y_train)

# save fitted model
joblib.dump(model, model_name + ".joblib")
```

```
# load fitted model
model = joblib.load( "" + model_name + ".joblib")
```

```
[21:34:45] WARNING: ../src/learner.cc:1203:
  If you are loading a serialized model (like pickle in Python, RDS in R) generated by
  older XGBoost, please export the model by calling `Booster.save_model` from that version
  first, then load it back in current version. See:

    https://xgboost.readthedocs.io/en/latest/tutorials/saving_model.html

  for more details about differences between saving model and serializing.

[21:34:45] WARNING: ../src/learner.cc:888: Found JSON model saved before XGBoost 1.6, please save the model using current version again.
[21:34:45] WARNING: ../src/learner.cc:553:
  If you are loading a serialized model (like pickle in Python, RDS in R) generated by
  older XGBoost, please export the model by calling `Booster.save_model` from that version
  first, then load it back in current version. See:

    https://xgboost.readthedocs.io/en/latest/tutorials/saving_model.html

  for more details about differences between saving model and serializing.
```

```
# Create an XGBoost classifier
from xgboost import XGBClassifier

model = XGBClassifier()

# Access the parameters of the XGBoost classifier
params = model.get_params()

# Now you can access specific parameters like this:
learning_rate = params['learning_rate']
n_estimators = params['n_estimators']

# You can also set parameters if needed:
model.set_params(n_estimators=100)  # Set the number of estimators to 100
```

```
XGBClassifier(base_score=None, booster=None, callbacks=None,
              colsample_bylevel=None, colsample_bynode=None,
              colsample_bytree=None, early_stopping_rounds=None,
              enable_categorical=False, eval_metric=None, feature_types=None,
              gamma=None, gpu_id=None, grow_policy=None, importance_type=None,
              interaction_constraints=None, learning_rate=None, max_bin=None,
              max_cat_threshold=None, max_cat_to_onehot=None,
              max_delta_step=None, max_depth=None, max_leaves=None,
              min_child_weight=None, missing=nan, monotone_constraints=None,
              n_estimators=100, n_jobs=None, num_parallel_tree=None,
              predictor=None, random_state=None, ...)
```

```
from xgboost import XGBRegressor
```

```
auc_res.append( auc )
acc.append( accuracy )

print('Best parameter set: %s ' % params)
```

```
Best parameter set: {'objective': 'binary:logistic', 'use_label_encoder': None, 'base_score': None, 'booster': None, 'callbacks': None,
```

Bag of POS

```
X_train = df_train.text_pos
X_test = df_test.text_pos
model_name = "XGB_POS"
```

```
param_grid = {'vect__ngram_range': [(1, 1)],
              'clf__n_estimators': [100, 200, 300, 400],
              'clf__learning_rate': [0.001, 0.01, 0.1, 1.0],
              'clf__max_depth': [1, 5, 10]}

model = GridSearchCV(pipeline,
                     param_grid,
                     scoring='roc_auc',
                     cv=cv,
                     n_jobs = -1,
                     verbose = 1)
```

```
model.fit(X_train, y_train)

# save fitted model
joblib.dump(model, model_name + ".joblib")
```

```
# load fitted model
model = joblib.load( "" + model_name + ".joblib")
```

```
[21:45:30] WARNING: ../src/learner.cc:1203:
  If you are loading a serialized model (like pickle in Python, RDS in R) generated by
  older XGBoost, please export the model by calling `Booster.save_model` from that version
  first, then load it back in current version. See:

    https://xgboost.readthedocs.io/en/latest/tutorials/saving_model.html

  for more details about differences between saving model and serializing.

[21:45:30] WARNING: ../src/learner.cc:888: Found JSON model saved before XGBoost 1.6, please save the model using current version again.
[21:45:30] WARNING: ../src/learner.cc:553:
  If you are loading a serialized model (like pickle in Python, RDS in R) generated by
  older XGBoost, please export the model by calling `Booster.save_model` from that version
  first, then load it back in current version. See:

    https://xgboost.readthedocs.io/en/latest/tutorials/saving_model.html

  for more details about differences between saving model and serializing.
```

```
auc_res.append( auc )
acc.append( accuracy )
```

```
print('Best parameter set: %s ' % params)
```

```
Best parameter set: {'objective': 'binary:logistic', 'use_label_encoder': None, 'base_score': None, 'booster': None, 'callbacks': None,
```

Embeddings

```
# select columns corresponding to embeddings
select = [i for i in list(df.columns) if i not in ["Sentiment", "tweet_clean", "text_pos", "target_encoded"]]

X_train = df_train[select]
X_test = df_test[select]
model_name = "XGB_EMB"
```

```
pipe = Pipeline([('clf', XGBClassifier())])

param_grid = {'clf__n_estimators': [100, 200, 300, 400],
              'clf__learning_rate': [0.001, 0.01, 0.1, 1.0],
              'clf__max_depth': [1, 5, 10]}

model = GridSearchCV(pipe, param_grid,
                     scoring='roc_auc',
                     cv=cv,
                     n_jobs=-1,
                     verbose = 1)
```

```
model.fit(X_train, y_train)

# save fitted model
joblib.dump(model, model_name + ".joblib")
```

```
# load fitted model
model = joblib.load(  "" + model_name + ".joblib")
```

```
[21:52:33] WARNING: ../src/learner.cc:1203:
  If you are loading a serialized model (like pickle in Python, RDS in R) generated by
  older XGBoost, please export the model by calling `Booster.save_model` from that version
  first, then load it back in current version. See:

    https://xgboost.readthedocs.io/en/latest/tutorials/saving_model.html

  for more details about differences between saving model and serializing.

[21:52:33] WARNING: ../src/learner.cc:888: Found JSON model saved before XGBoost 1.6, please save the model using current version again.
[21:52:33] WARNING: ../src/learner.cc:553:
  If you are loading a serialized model (like pickle in Python, RDS in R) generated by
  older XGBoost, please export the model by calling `Booster.save_model` from that version
  first, then load it back in current version. See:

    https://xgboost.readthedocs.io/en/latest/tutorials/saving_model.html

  for more details about differences between saving model and serializing.
```

```
auc_res.append( auc )
acc.append( accuracy )

print('Best parameter set: %s ' % params)
```

```
Best parameter set: {'objective': 'binary:logistic', 'use_label_encoder': None, 'base_score': None, 'booster': None, 'callbacks': None,
```

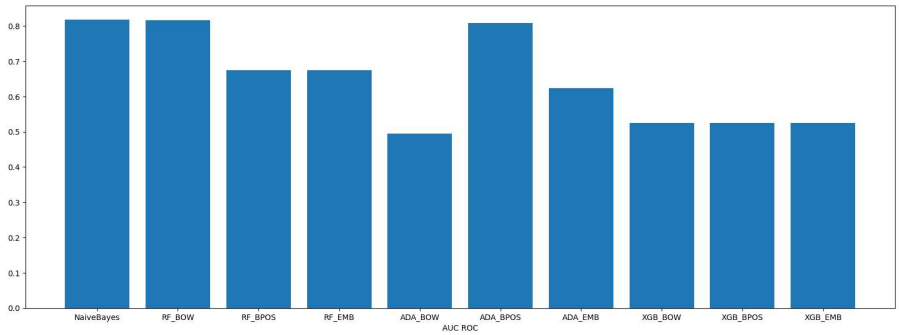
Compare machine learning models

- $Accuracy = \frac{TP+TN}{TP+TN+FP+FN}$
- AUC ROC (Area Under the Receiver Operating Characteristics): measure of how much the model is capable of distinguishing between classes. The higher the AUC, the better the model is at predicting 0 classes as 0 and 1 classes as 1.

```
model_names = ["NaiveBayes", "RF_BOW", "RF_BPOS", "RF_EMB", "ADA_BOW", "ADA_BPOS", "ADA_EMB",
               "XGB_BOW", "XGB_BPOS", "XGB_EMB"]

accuracy = dict()
for n in range(len(model_names)):
    accuracy[model_names[n]] = acc[n]

auc_roc = dict()
for n in range(len(model_names)):
    auc_roc[model_names[n]] = auc_res[n]
```

```
fig, ax = plt.subplots(figsize=(20,7))
plt.xlabel('AUC ROC')
keys = auc_roc.keys()
values = auc_roc.values()

plt.bar(keys, values)
auc_roc
```

```
{'NaiveBayes': 0.8171403392817802,
 'RF_BOW': 0.816424322538004,
 'RF_BPOS': 0.6749283983256223,
 'RF_EMB': 0.6749283983256223,
 'ADA_BOW': 0.4946849526327385,
 'ADA_BPOS': 0.8090025335977087,
 'ADA_EMB': 0.624242674597929,
 'XGB_BOW': 0.5243996474994492,
 'XGB_BPOS': 0.5243996474994492,
 'XGB_EMB': 0.5243996474994492}
```



Sentiment analysis with deeplearning and transformers models

We classify Covid-19 tweets using the following deeplearning models:

- Simple RNN
- LSTM
- GRU
- Bidirectional LSTM
- BERT

The tweets are preprocessed using the following NLP methods before being fed as inputs to the neural network models:

- Keras embedding layers
- Pre-trained Spacy word embeddings

NLP preprocessing steps

- Randomly reshuffle data
- Estimate embedding matrix with Keras

- Initialize Keras embedding layer with pre-trained Spacy word embedding

```
df = pd.read_csv("./binary_data.csv")
df.shape
```

```
(6358, 3)
```

```
# reshuffle data
np.random.seed(0)
df = df.reindex(np.random.permutation(df.index))
df = df.reset_index(drop=True) # reset the index after the permutation
```

```
# remove punctuation, non alphanumerical characters and lowercase
df["tweet_clean"] = df['OriginalTweet'].apply(preprocessor)
```

```
# load pre-trained word embedding model from spacy
nlp = spacy.load('en_core_web_md')

# stack embeddings of tweets
emb = np.vstack(df.tweet_clean.apply(lambda x: nlp(x).vector))

savetxt('./embedded_data.csv', emb, delimiter=',')
```

Basic preprocessing:

- Convert to lowercase and split tokens on whitespaces and puncutuation
- Map every word to an integer value
- Set each tweet to an equal length T by padding with 0 or slicing

Hyperparameters for Keras preprocessor

- num_words: size of the vocabulary based on most frequent words in tweets
- maxlen: sequence length T to ensure that all tweets have the same length

```
# find size of vocabulary = number of words that appear more than 1 time
counts = Counter()
for i, news in enumerate(df['OriginalTweet']):
    text = news.lower() # convert to lowercase
    df.loc[i, "OriginalTweet"] = text
    counts.update(text.split()) # splitting on whitespace
```

```
print('Number of unique words:', len(counts))
print('Number of words that appear more than 1 time:', len( [k for k, v in counts.items() if v > 1] ))
print('Number of words that appear more than 10 times:', len( [k for k, v in counts.items() if v > 10] ))
```

```
Number of unique words: 18506
Number of words that appear more than 1 time: 18506
Number of words that appear more than 10 times: 1899
```

```
# our vocabulary consists of words that appear more than 10 times
vocab_size = len( [k for k, v in counts.items() if v > 10] )
```

```
# most frequent words that appear more than 5 times
word_counts = sorted(counts, key=counts.get, reverse=True)
word_counts = word_counts[1:vocab_size]
print( word_counts[:20] ) # show first 20 words
```

```
['to', 'and', 'of', 'a', 'in', 'for', '#covid_19', 'is', 'i', 'are', '#coronavirus', 'food', 'you', 'on', 'at', 'this', 'grocery', 'be',
```

```
# assign an index to all words in word_counts
word_to_int = {}
for i, word in enumerate(word_counts):
    word_to_int[word] = i
```

```
# create a list with all tweets, where each tweet contains only the indexes of most frequent words
mapped_tweet = []
for tweet in df['OriginalTweet']:
```

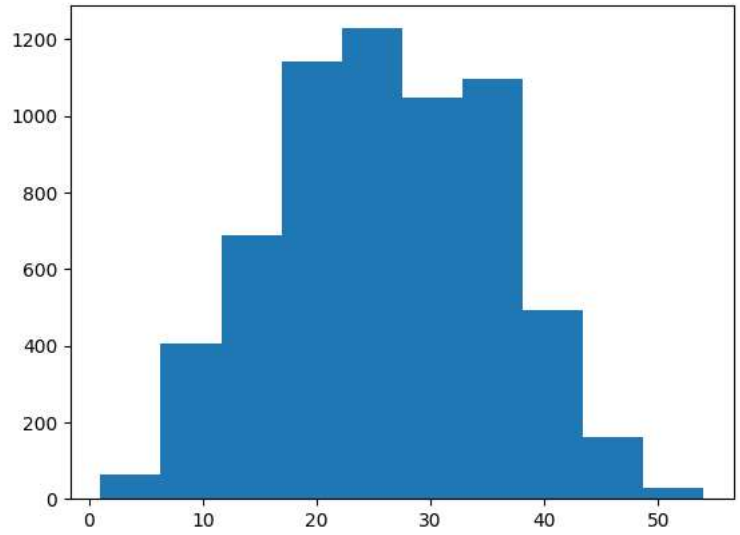
```
    mapped_tweet.append([word_to_int[word]
                         for word in tweet.split()
                         if word in word_to_int.keys()])

print( mapped_tweet[:10] )
```

```
[[486, 341, 0, 26, 355, 487, 2, 1, 2, 26, 655, 88, 18, 72, 2, 2], [918, 90, 14, 16, 19, 8, 96, 14, 7, 1655, 3, 396, 72, 2, 26, 381, 2, 1
```

```
# set length of each tweet equal to median length of mapped_tweet
sequence_length = np.median([len(tweet) for tweet in mapped_tweet])
sequence_length
```

```
26.0
```

```
rev_lengths = np.array([len(tweet) for tweet in mapped_tweet])
plt.hist(rev_lengths)
plt.show()
```




```
# map words to integers with preprocessing using keras tokenizer
tokenizer = Tokenizer(num_words=vocab_size,
                      filters='!"#$%&()*+,-./:;<=>?@[\\]^_`{|}~\t\n', # filters out all punctuation
                      lower=True, # convert to lowercase
                      split=' ') # split on whitespaces

tokenizer.fit_on_texts(df['OriginalTweet'])
list_tokenized = tokenizer.texts_to_sequences(df['OriginalTweet'])

print( list_tokenized[:10] )
```

```
[[995, 483, 377, 2, 242, 35, 357, 468, 319, 484, 7, 535, 4, 1, 7, 35, 696, 97, 24, 79, 7, 1, 7, 72, 12, 10, 11], [162, 93, 22, 1, 25, 21
```

```
# dictionary of words with indices
word_index = tokenizer.word_index
```

```
# convert all tweets to same length by padding with 0 or slicing
sequence_length = 35 # select 35 as max length based on distribution of mapped tweets
sequences = pad_sequences(list_tokenized, maxlen= sequence_length)
print(sequences)
```

```
[[   0    0    0 ...   12   10   11]
 [ 162   93   22 ...   14    5    6]
 [   1  190    4 ...  149   14 1508]
 ...
 [   0    0    0 ...   36    5    6]
 [   1   81    3 ...   12   10   11]
 [   0    0    0 ...   12   10   11]]
```

```
# convert sentiment to real valued
le = LabelEncoder()
```

```
le.fit(df['Sentiment'])
df['target_encoded'] = le.transform(df['Sentiment'])
```

```
df_processed = pd.concat([df[['Sentiment', "target_encoded", "tweet_clean"]], pd.DataFrame(sequences)], axis=1)
df_processed.head()
```

	Sentiment	target_encoded	tweet_clean	0	1	2	3	4	5	6	...	25	26
0	Negative	0	breaking ontario s chief medical officer dr d...	0	0	0	0	0	0	0	...	97	24
1	Negative	0	well everyone at the grocery store i work at i...	162	93	22	1	25	21	17	...	18	197
2	Negative	0	i am more and more concerned about the	1	190	4	385	9	57	698	...	44	42

```
# save data
df_processed.to_csv("./data_deeplearning.csv", index = False, encoding = "utf-8")
```

Recurrent neural networks (RNN)

- The machine learning approach to text analysis relies heavily on text preprocessing
- RNNs are used to implement most of the NLP preprocessing as well as classification tasks

A RNN is a neural network architecture suited to deal with sequential data. In contrast to feedforward neural networks, they allow for cycles in their architecture in order to take into account the activations of the previous step. Hence, they have some form of "memory" of the structure of the training data.

For input data $x = (x_1, \ldots, x_T)$, the RNN architecture computes a sequence of outputs $(z_1^1, \ldots, z_T^1)$, where each output isa function of the feature information at the same timestep and the output information from the previous timestep, i.e.
$z_{j+1}^1 = z^1(x_{j+1}, z_j^1), \forall j = 1, \ldots, T-1$

The most commonly used gradient-based learning algorithm to update RNN parameters is Backpropagation Through Time (BPTT), which is applied to the unfolded network in order to compute the gradients at each timestep. However, the recurisve computation of these gradients often lead to exploding or vanishing gradients. Gated RNN architectures, such as Long Short-Term Memory (LSTM) or Gated Recurrent Units (GRU) solve this problem.

The models are trained on the following NLP preprocessed data:

- **Word embeddings**: word embeddings are trained using Keras embedding layer, which converts each word into fixed length, dense, real-valued vector of defined size. Parameters:
  - input_dim : Size of the vocabulary (equal to the size of the embedding matrix)
  - output_dim : Length of the vector for each word
  - input_length : Maximum length of a sequence
- **Pre-trained Spacy word embeddings**: used as weights in Keras embedding layer (transfer learning). Additional parameters:
  - weights: Embedding matrix computed from pretrained Spacy embeddings
  - trainable = False: do not retrain word embeddings

```
df = pd.read_csv("./data_deeplearning.csv")
```

```
# define train and test size
train_size = int( len(df)*0.8 )
test_size = int( len(df) - train_size )
print(train_size, test_size)
```

```
5086 1272
```

```
remove = ['Sentiment', "target_encoded", "tweet_clean"]
```

```
# split dataset
X_train = np.array(df.head(train_size).drop(remove, axis=1)) # replace with n_train
y_train = df.head(train_size).target_encoded.values

X_test = np.array(df.tail(test_size).drop(remove, axis=1)) # replace with n_test
y_test = df.tail(test_size).target_encoded.values

print('Training data X, y:', X_train.shape, y_train.shape)
print('Testing data X, y:', X_test.shape, y_test.shape)
```

```
Training data X, y: (5086, 35) (5086,)
Testing data X, y: (1272, 35) (1272,)
```

```
# load embedding matrix
emb = loadtxt('embedded_data.csv', delimiter=',')
```

```
emb
```

```
array([[-2.13883114,  0.0549665 , -1.46543276, ..., -1.89626944,
        -2.30746913,  0.97282922],
       [-0.76660615, -1.21993959, -2.24405622, ..., -1.29237962,
        -1.69344974,  0.97751206],
       [-1.43679869,  0.76140064, -3.09800816, ..., -1.44790125,
        -4.52036142,  1.80674946],
       ...,
       [ 0.44486162,  0.43326563, -1.66728425, ...,  0.69858426,
        -2.41147494,  1.95321858],
       [-1.0397501 ,  0.94362527, -1.61163914, ..., -0.33368155,
        -2.35109544,  2.11889815],
       [ 0.78431338,  0.2940124 , -0.54890811, ..., -0.44269753,
        -2.071172  ,  1.07922041]])
```

Simple RNN

```
# paramerers, based on results from preprocessing
emb_length = 300
max_length = 58 # length of longest sequence
sequence_length = 35
```

Keras embedding layer

```
# define model
model = tf.keras.Sequential()
model.add(layers.Embedding(input_dim = len(emb),
                 output_dim = emb_length,
                 input_length = sequence_length))

model.add(layers.SimpleRNN(sequence_length, activation='sigmoid'))
model.add(layers.Dropout(0.5))
model.add(layers.Dense(1, activation='sigmoid'))

model.compile(loss='binary_crossentropy', optimizer='adam', metrics=['accuracy'])
```

```
# define callbacks for early stopping during training
# stop training when the validation loss `val_accuracy` is no longer improving
callbacks = [tf.keras.callbacks.EarlyStopping(
    monitor='val_accuracy',
    min_delta=1e-2,
    patience=3,
    verbose=1,
    restore_best_weights=True)]
```

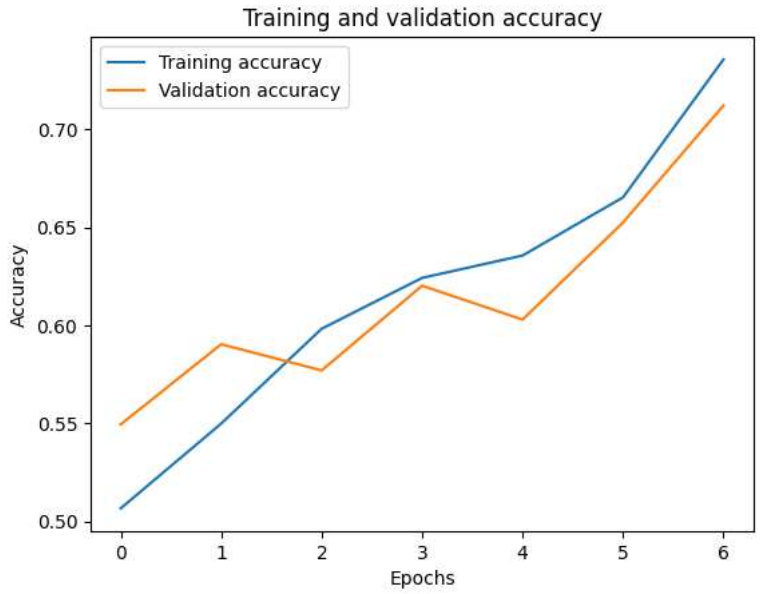
```
# train the model
history = model.fit(X_train,
         y_train,
         validation_data = (X_test, y_test),
         epochs=7,
         batch_size=32,
         callbacks=callbacks,
         verbose=1)
```

```
model.save("SimpleRNN_1")
```

```
Epoch 1/7
159/159 [==============================] - 16s 77ms/step - loss: 0.7323 - accuracy: 0.5067 - val_loss: 0.6841 - val_accuracy: 0.5495
Epoch 2/7
159/159 [==============================] - 9s 54ms/step - loss: 0.6869 - accuracy: 0.5499 - val_loss: 0.6639 - val_accuracy: 0.5904
Epoch 3/7
159/159 [==============================] - 11s 71ms/step - loss: 0.6361 - accuracy: 0.5983 - val_loss: 0.6467 - val_accuracy: 0.5770
Epoch 4/7
159/159 [==============================] - 12s 74ms/step - loss: 0.5976 - accuracy: 0.6243 - val_loss: 0.6318 - val_accuracy: 0.6203
Epoch 5/7
159/159 [==============================] - 10s 60ms/step - loss: 0.5763 - accuracy: 0.6357 - val_loss: 0.6172 - val_accuracy: 0.6030
Epoch 6/7
159/159 [==============================] - 7s 43ms/step - loss: 0.5510 - accuracy: 0.6654 - val_loss: 0.5873 - val_accuracy: 0.6525
Epoch 7/7
159/159 [==============================] - 8s 48ms/step - loss: 0.4947 - accuracy: 0.7357 - val_loss: 0.5431 - val_accuracy: 0.7123
```

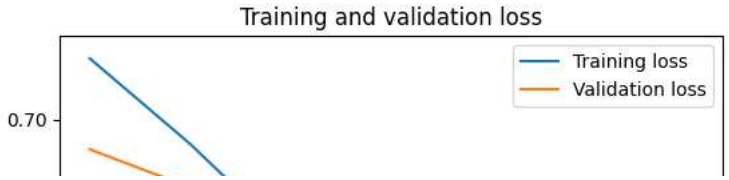
```
# plot the development of the accuracy over epochs to see the training progress
plt.plot(history.history['accuracy'], label='Training accuracy')
plt.plot(history.history['val_accuracy'], label='Validation accuracy')
plt.title('Training and validation accuracy')
plt.xlabel('Epochs')
plt.ylabel('Accuracy')
plt.legend()
```

```
<matplotlib.legend.Legend at 0x7b143023a290>
```




```
# plot the development of the loss over epochs to see the training progress
plt.plot(history.history['loss'], label='Training loss')
plt.plot(history.history['val_loss'], label='Validation loss')
plt.title('Training and validation loss')
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.legend()
```

```
<matplotlib.legend.Legend at 0x7b146d1c7e80>
```



Pre-trained Spacy embedding

```
# define model
model = tf.keras.Sequential()
model.add(layers.Embedding(input_dim = len(emb),
                 output_dim = emb_length,
                 weights=[emb],
                 input_length = sequence_length,
                 trainable = False))

model.add(layers.SimpleRNN(sequence_length, activation='sigmoid'))
model.add(layers.Dropout(0.5))
model.add(layers.Dense(1, activation='sigmoid'))

model.compile(loss='binary_crossentropy', optimizer='adam',metrics=['accuracy'])
```

```
# define callbacks for early stopping during training
# stop training when the validation loss `val_accuracy` is no longer improving
callbacks = [tf.keras.callbacks.EarlyStopping(
    monitor='val_accuracy',
    min_delta=1e-3,
    patience=5,
    verbose=1,
    restore_best_weights=True)]
```

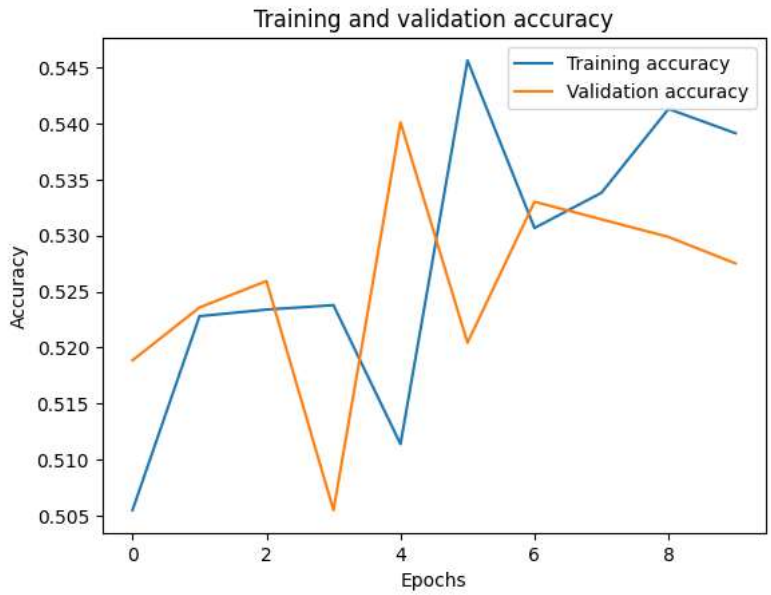
```
# train the model
history = model.fit(X_train,
         y_train,
         validation_data = (X_test, y_test),
         epochs=20,
         batch_size=32,
         callbacks=callbacks,
         verbose=1)

model.save("SimpleRNN_2")
```

```
Epoch 1/20
159/159 [==============================] - 4s 17ms/step - loss: 0.7181 - accuracy: 0.5055 - val_loss: 0.6899 - val_accuracy: 0.5189
Epoch 2/20
159/159 [==============================] - 4s 23ms/step - loss: 0.6927 - accuracy: 0.5228 - val_loss: 0.6895 - val_accuracy: 0.5236
Epoch 3/20
159/159 [==============================] - 2s 14ms/step - loss: 0.6903 - accuracy: 0.5234 - val_loss: 0.6888 - val_accuracy: 0.5259
Epoch 4/20
159/159 [==============================] - 2s 12ms/step - loss: 0.6905 - accuracy: 0.5238 - val_loss: 0.6897 - val_accuracy: 0.5055
Epoch 5/20
159/159 [==============================] - 2s 11ms/step - loss: 0.6881 - accuracy: 0.5114 - val_loss: 0.6868 - val_accuracy: 0.5401
Epoch 6/20
159/159 [==============================] - 2s 13ms/step - loss: 0.6844 - accuracy: 0.5456 - val_loss: 0.6868 - val_accuracy: 0.5204
Epoch 7/20
159/159 [==============================] - 2s 12ms/step - loss: 0.6844 - accuracy: 0.5307 - val_loss: 0.6872 - val_accuracy: 0.5330
Epoch 8/20
159/159 [==============================] - 3s 20ms/step - loss: 0.6838 - accuracy: 0.5338 - val_loss: 0.6916 - val_accuracy: 0.5314
Epoch 9/20
159/159 [==============================] - 3s 19ms/step - loss: 0.6819 - accuracy: 0.5413 - val_loss: 0.6858 - val_accuracy: 0.5299
Epoch 10/20
155/159 [============================>.] - ETA: 0s - loss: 0.6804 - accuracy: 0.5389Restoring model weights from the end of the best epo
159/159 [==============================] - 2s 13ms/step - loss: 0.6800 - accuracy: 0.5391 - val_loss: 0.6846 - val_accuracy: 0.5275
Epoch 10: early stopping
```

```
# plot the development of the accuracy over epochs to see the training progress
plt.plot(history.history['accuracy'], label='Training accuracy')
plt.plot(history.history['val_accuracy'], label='Validation accuracy')
plt.title('Training and validation accuracy')
plt.xlabel('Epochs')
```

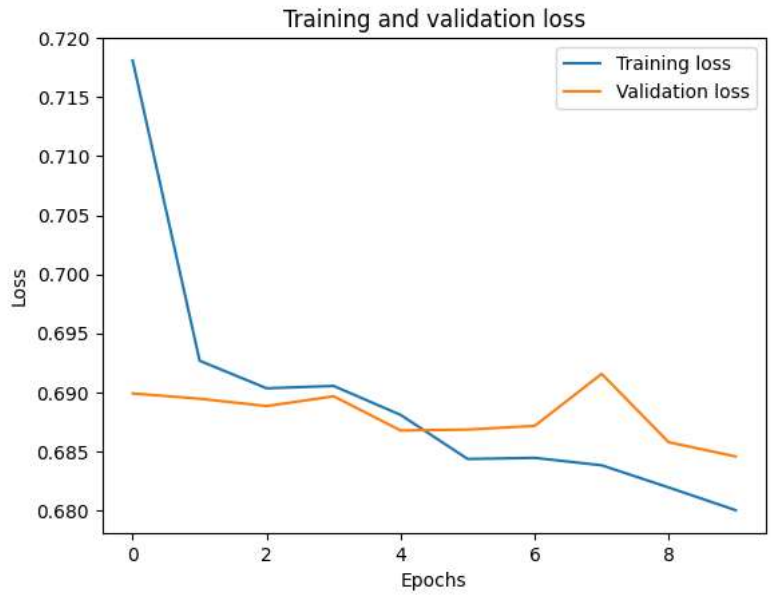
```
plt.ylabel('Accuracy')
plt.legend()
```

```
<matplotlib.legend.Legend at 0x7b146175f7c0>
```




```
# plot the development of the loss over epochs to see the training progress
plt.plot(history.history['loss'], label='Training loss')
plt.plot(history.history['val_loss'], label='Validation loss')
plt.title('Training and validation loss')
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.legend()
```

```
<matplotlib.legend.Legend at 0x7b14656ec730>
```



LSTM

LSTM units replace hidden neurons in RNN architectures by more complex memory cells. A memory cell takes three inputs: the input features x_t, the neural activations z_{t-1}^1 and the cell state c_{t-1}^1 from the previous step, and outputs a new cell state c_t^1 and neural activations z_t^1. Each memory cell consists of three gates that control the flow of information through the cell:

- **Forget Gate**: this gate decides what information should be thrown away or kept. Information from the previous hidden state and information from the current input is passed through the sigmoid function. Values come out between 0 and 1. The closer to 0 means to forget, and the closer to 1 means to keep.

- **Input Gate**: to update the cell state, we have the input gate. First, we pass the previous hidden state and current input into a sigmoid function. That decides which values will be updated by transforming the values to be between 0 and 1. 0 means not important, and 1 means important.
- **Output gate**: the output gate decides what the next hidden state, which contains information on previous inputs, should be. The hidden state is also used for predictions. First, we pass the previous hidden state and the current input into a sigmoid function. Then we pass the newly modified cell state to the tanh function. We multiply the tanh output with the sigmoid output to decide what information the hidden state should carry. The output is the hidden state. The new cell state and the new hidden is then carried over to the next time step.

Keras embedding layer

```
# define model
model = tf.keras.Sequential()
model.add(layers.Embedding(input_dim = len(emb),
                 output_dim = emb_length,
                 input_length = sequence_length))

model.add(layers.LSTM(sequence_length, dropout=0.3, recurrent_dropout=0.3))
model.add(layers.Dropout(0.5))
model.add(layers.Dense(1, activation='sigmoid'))

model.compile(loss='binary_crossentropy', optimizer='adam',metrics=['accuracy'])
```

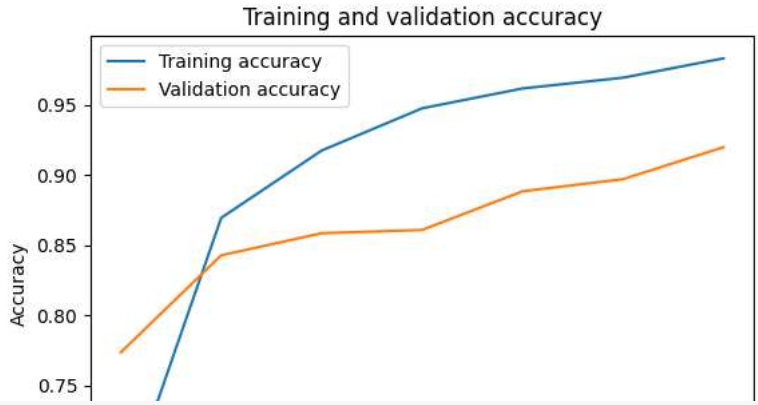
```
# define callbacks for early stopping during training
# stop training when the validation loss `val_accuracy` is no longer improving
callbacks = [tf.keras.callbacks.EarlyStopping(
    monitor='val_accuracy',
    min_delta=1e-2,
    patience=3,
    verbose=1,
    restore_best_weights=True)]
```

```
# train the model
history = model.fit(X_train,
                    y_train,
                    validation_data = (X_test, y_test),
                    epochs=7,
                    batch_size=32,
                    callbacks=callbacks,
                    verbose=1)

model.save("LSTM_1")
```

```
Epoch 1/7
159/159 [==============================] - 23s 112ms/step - loss: 0.5975 - accuracy: 0.6632 - val_loss: 0.4866 - val_accuracy: 0.7736
Epoch 2/7
159/159 [==============================] - 18s 111ms/step - loss: 0.3303 - accuracy: 0.8694 - val_loss: 0.3887 - val_accuracy: 0.8428
Epoch 3/7
159/159 [==============================] - 17s 108ms/step - loss: 0.2270 - accuracy: 0.9174 - val_loss: 0.3703 - val_accuracy: 0.8585
Epoch 4/7
159/159 [==============================] - 17s 109ms/step - loss: 0.1583 - accuracy: 0.9475 - val_loss: 0.3882 - val_accuracy: 0.8608
Epoch 5/7
159/159 [==============================] - 19s 119ms/step - loss: 0.1222 - accuracy: 0.9617 - val_loss: 0.3283 - val_accuracy: 0.8884
Epoch 6/7
159/159 [==============================] - 18s 114ms/step - loss: 0.0972 - accuracy: 0.9693 - val_loss: 0.3536 - val_accuracy: 0.8970
Epoch 7/7
159/159 [==============================] - 18s 114ms/step - loss: 0.0653 - accuracy: 0.9831 - val_loss: 0.3297 - val_accuracy: 0.9198
```

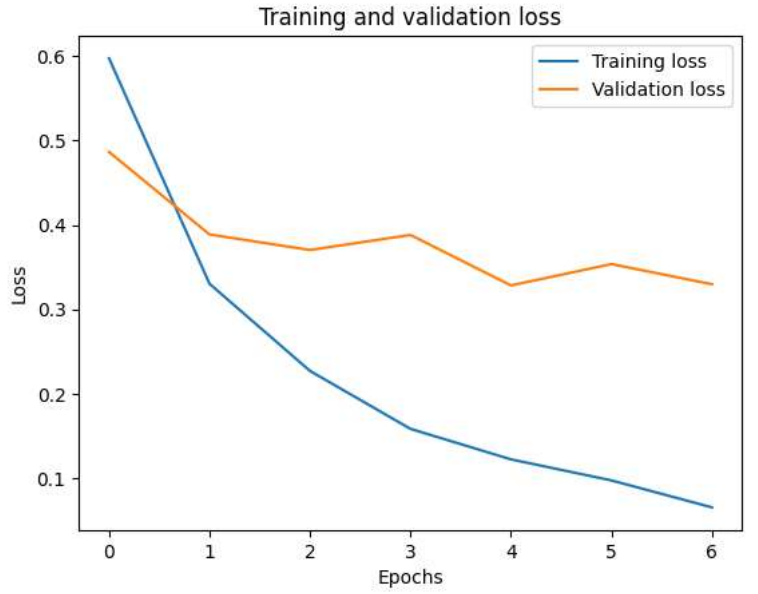
```
# plot the development of the accuracy over epochs to see the training progress
plt.plot(history.history['accuracy'], label='Training accuracy')
plt.plot(history.history['val_accuracy'], label='Validation accuracy')
plt.title('Training and validation accuracy')
plt.xlabel('Epochs')
plt.ylabel('Accuracy')
plt.legend()
```

```
<matplotlib.legend.Legend at 0x7b14646ed240>
```




```
# plot the development of the loss over epochs to see the training progress
plt.plot(history.history['loss'], label='Training loss')
plt.plot(history.history['val_loss'], label='Validation loss')
plt.title('Training and validation loss')
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.legend()
```

```
<matplotlib.legend.Legend at 0x7b14643e14b0>
```



Pre-trained word embeddings

```
# define model
model = tf.keras.Sequential()
model.add(layers.Embedding(input_dim = len(emb),
                 output_dim = emb_length,
                 weights=[emb],
                 input_length = sequence_length,
                 trainable = False))

model.add(layers.LSTM(sequence_length, dropout=0.3, recurrent_dropout=0.3))
model.add(layers.Dropout(0.5))
model.add(layers.Dense(1, activation='sigmoid'))

model.compile(loss='binary_crossentropy', optimizer='adam',metrics=['accuracy'])
```

```
# define callbacks for early stopping during training
# stop training when the validation loss `val_accuracy` is no longer improving
callbacks = [tf.keras.callbacks.EarlyStopping(
    monitor='val_accuracy',
    min_delta=1e-3,
    patience=5,
```

```
    verbose=1,
    restore_best_weights=True)]
```

```
# train the model
history = model.fit(X_train,
                    y_train,
                    validation_data = (X_test, y_test),
                    epochs=30,
                    batch_size=32,
                    callbacks=callbacks,
                    verbose=1)

model.save("LSTM_2")
```

```
Epoch 1/30
159/159 [==============================] - 16s 80ms/step - loss: 0.6992 - accuracy: 0.5171 - val_loss: 0.6913 - val_accuracy: 0.5330
Epoch 2/30
159/159 [==============================] - 12s 76ms/step - loss: 0.6929 - accuracy: 0.5171 - val_loss: 0.6909 - val_accuracy: 0.5197
Epoch 3/30
159/159 [==============================] - 13s 79ms/step - loss: 0.6918 - accuracy: 0.5102 - val_loss: 0.6908 - val_accuracy: 0.5244
Epoch 4/30
159/159 [==============================] - 10s 62ms/step - loss: 0.6908 - accuracy: 0.5212 - val_loss: 0.6911 - val_accuracy: 0.5307
Epoch 5/30
159/159 [==============================] - 12s 75ms/step - loss: 0.6901 - accuracy: 0.5326 - val_loss: 0.6896 - val_accuracy: 0.5354
Epoch 6/30
159/159 [==============================] - 12s 75ms/step - loss: 0.6891 - accuracy: 0.5289 - val_loss: 0.6927 - val_accuracy: 0.5197
Epoch 7/30
159/159 [==============================] - 12s 75ms/step - loss: 0.6895 - accuracy: 0.5269 - val_loss: 0.6898 - val_accuracy: 0.5212
Epoch 8/30
159/159 [==============================] - 10s 61ms/step - loss: 0.6888 - accuracy: 0.5263 - val_loss: 0.6888 - val_accuracy: 0.5197
Epoch 9/30
159/159 [==============================] - 12s 76ms/step - loss: 0.6889 - accuracy: 0.5326 - val_loss: 0.6885 - val_accuracy: 0.5165
Epoch 10/30
159/159 [==============================] - ETA: 0s - loss: 0.6859 - accuracy: 0.5354Restoring model weights from the end of the best epo
159/159 [==============================] - 12s 73ms/step - loss: 0.6859 - accuracy: 0.5354 - val_loss: 0.6885 - val_accuracy: 0.5236
Epoch 10: early stopping
```

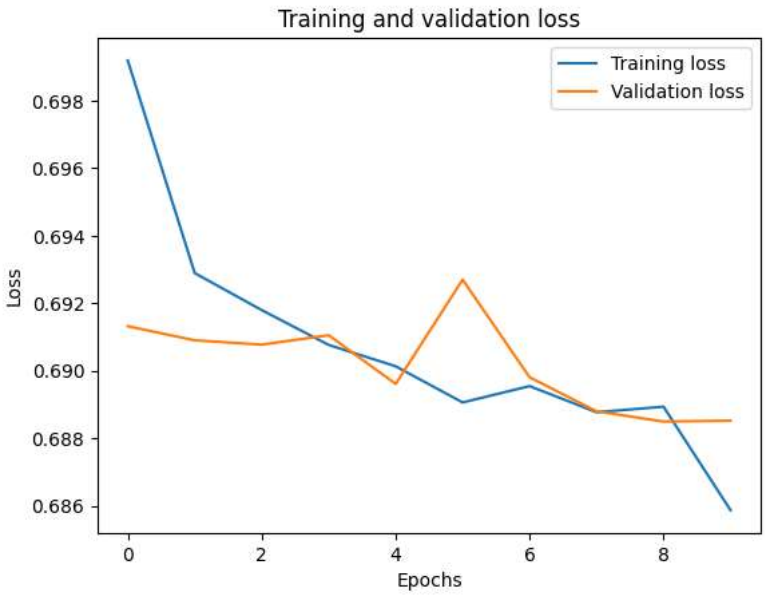
```
# plot the development of the accuracy over epochs to see the training progress
plt.plot(history.history['accuracy'], label='Training accuracy')
plt.plot(history.history['val_accuracy'], label='Validation accuracy')
plt.title('Training and validation accuracy')
plt.xlabel('Epochs')
plt.ylabel('Accuracy')
plt.legend()
```

```
<matplotlib.legend.Legend at 0x7b14305c3190>
```


```
# plot the development of the loss over epochs to see the training progress
plt.plot(history.history['loss'], label='Training loss')
plt.plot(history.history['val_loss'], label='Validation loss')
plt.title('Training and validation loss')
plt.xlabel('Epochs')
```

```
plt.ylabel('Loss')
plt.legend()
```

```
<matplotlib.legend.Legend at 0x7b1430625690>
```



GRU

Uses a similar but less complex memory cell architecture as LSTM, composed by:

- **Update gate**: It determines how much of the past knowledge needs to be passed along into the future. It is analogous to the output gate in an LSTM recurrent unit.
- **Reset gate**: It determines how much of the past knowledge to forget. It is analogous to the combination of the input Gate and the forget Gate in an LSTM recurrent unit.
- **Current Memory Gate**: It is incorporated into the Reset Gate and is used to introduce some non-linearity into the input and to also make the input zero-mean. It also reduces the effect that previous information has on the current information that is being passed into the future.

Keras embedding layer

```
# define model
model = tf.keras.Sequential()
model.add(layers.Embedding(input_dim = len(emb),
                  output_dim = emb_length,
                  input_length = sequence_length))


model.add(layers.SpatialDropout1D(0.3))
model.add(layers.GRU(sequence_length))
model.add(layers.Dropout(0.5))
model.add(layers.Dense(1, activation='sigmoid'))

model.compile(loss='binary_crossentropy', optimizer='adam',metrics=['accuracy'])
```

```
# define callbacks for early stopping during training
# stop training when the validation loss `val_accuracy` is no longer improving
callbacks = [tf.keras.callbacks.EarlyStopping(
    monitor='val_accuracy',
    min_delta=1e-3,
    patience=5,
    verbose=1,
    restore_best_weights=True)]
```

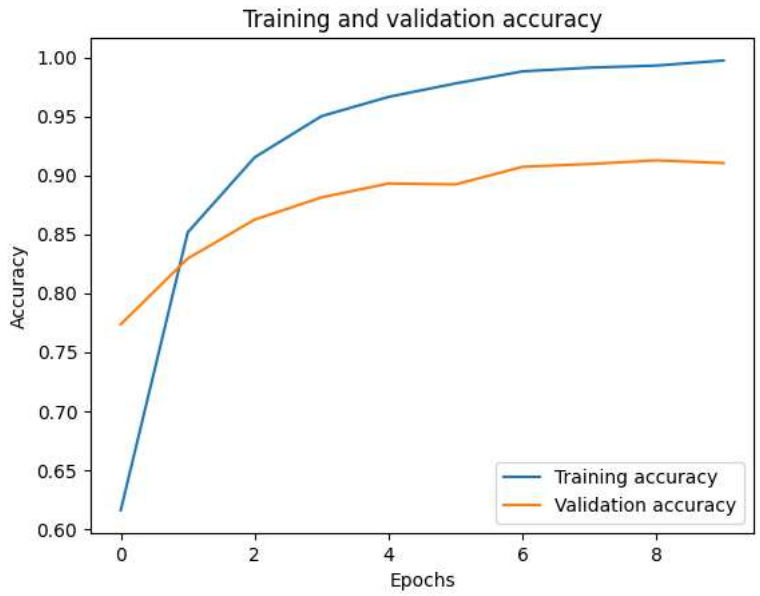
```
# train the model
history = model.fit(X_train,
                    y_train,
```

```
                    validation_data = (X_test, y_test),
                    epochs=10,
                    batch_size=32,
                    callbacks=callbacks,
                    verbose=1)

model.save("GRU_1")
```

```
Epoch 1/10
159/159 [==============================] - 15s 74ms/step - loss: 0.6352 - accuracy: 0.6158 - val_loss: 0.4720 - val_accuracy: 0.7736
Epoch 2/10
159/159 [==============================] - 11s 70ms/step - loss: 0.3592 - accuracy: 0.8516 - val_loss: 0.4043 - val_accuracy: 0.8294
Epoch 3/10
159/159 [==============================] - 10s 60ms/step - loss: 0.2328 - accuracy: 0.9153 - val_loss: 0.3623 - val_accuracy: 0.8624
Epoch 4/10
159/159 [==============================] - 10s 63ms/step - loss: 0.1510 - accuracy: 0.9503 - val_loss: 0.3681 - val_accuracy: 0.8813
Epoch 5/10
159/159 [==============================] - 11s 67ms/step - loss: 0.1074 - accuracy: 0.9666 - val_loss: 0.3645 - val_accuracy: 0.8931
Epoch 6/10
159/159 [==============================] - 9s 57ms/step - loss: 0.0749 - accuracy: 0.9780 - val_loss: 0.3614 - val_accuracy: 0.8923
Epoch 7/10
159/159 [==============================] - 10s 65ms/step - loss: 0.0463 - accuracy: 0.9882 - val_loss: 0.3828 - val_accuracy: 0.9072
Epoch 8/10
159/159 [==============================] - 10s 65ms/step - loss: 0.0381 - accuracy: 0.9913 - val_loss: 0.4085 - val_accuracy: 0.9096
Epoch 9/10
159/159 [==============================] - 10s 62ms/step - loss: 0.0275 - accuracy: 0.9931 - val_loss: 0.4091 - val_accuracy: 0.9127
Epoch 10/10
159/159 [==============================] - 11s 69ms/step - loss: 0.0150 - accuracy: 0.9974 - val_loss: 0.4971 - val_accuracy: 0.9104
```

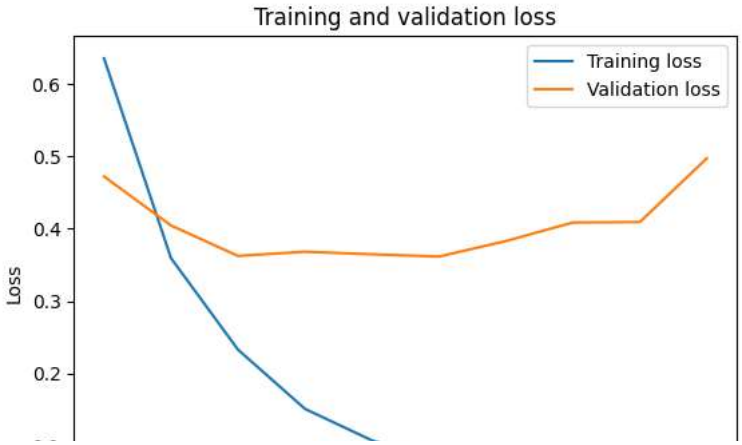
```
# plot the development of the accuracy over epochs to see the training progress
plt.plot(history.history['accuracy'], label='Training accuracy')
plt.plot(history.history['val_accuracy'], label='Validation accuracy')
plt.title('Training and validation accuracy')
plt.xlabel('Epochs')
plt.ylabel('Accuracy')
plt.legend()
```

```
<matplotlib.legend.Legend at 0x7b145b742320>
```




```
# plot the development of the loss over epochs to see the training progress
plt.plot(history.history['loss'], label='Training loss')
plt.plot(history.history['val_loss'], label='Validation loss')
plt.title('Training and validation loss')
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.legend()
```

```
<matplotlib.legend.Legend at 0x7b1464ddf970>
```



Pre-trained word embeddings

```
# define model
model = tf.keras.Sequential()
model.add(layers.Embedding(input_dim = len(emb),
                 output_dim = emb_length,
                 weights=[emb],
                 input_length = sequence_length,
                 trainable = False))


model.add(layers.SpatialDropout1D(0.3))
model.add(layers.GRU(sequence_length))
model.add(layers.Dropout(0.5))
model.add(layers.Dense(1, activation='sigmoid'))

model.compile(loss='binary_crossentropy', optimizer='adam',metrics=['accuracy'])
```

```
# define callbacks for early stopping during training
# stop training when the validation loss `val_accuracy` is no longer improving
callbacks = [tf.keras.callbacks.EarlyStopping(
    monitor='val_accuracy',
    min_delta=1e-3,
    patience=5,
    verbose=1,
    restore_best_weights=True)]
```

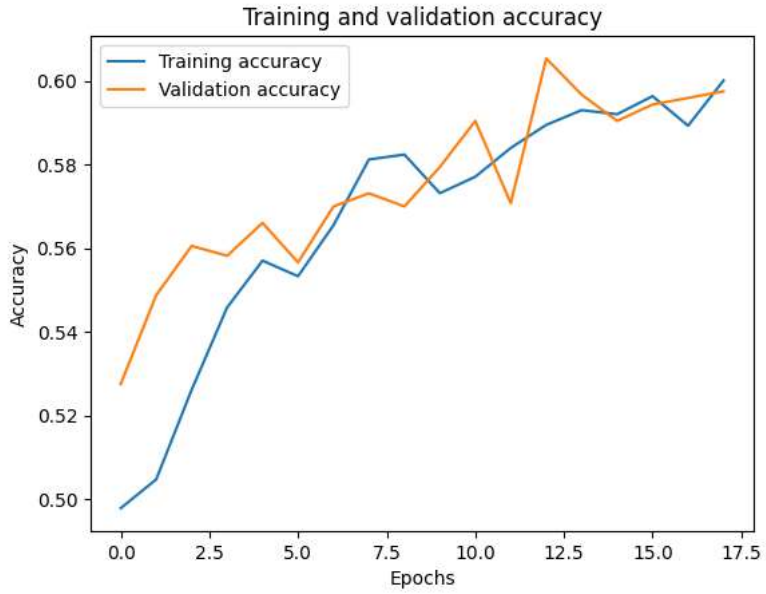
```
# train the model
history = model.fit(X_train,
                    y_train,
                    validation_data = (X_test, y_test),
                    epochs=30,
                    batch_size=32,
                    callbacks=callbacks,
                    verbose=1)

model.save("GRU_2")
```

```
Epoch 1/30
159/159 [==============================] - 10s 37ms/step - loss: 0.8035 - accuracy: 0.4978 - val_loss: 0.6900 - val_accuracy: 0.5275
Epoch 2/30
159/159 [==============================] - 4s 27ms/step - loss: 0.7262 - accuracy: 0.5047 - val_loss: 0.6870 - val_accuracy: 0.5487
Epoch 3/30
159/159 [==============================] - 5s 33ms/step - loss: 0.6928 - accuracy: 0.5262 - val_loss: 0.6842 - val_accuracy: 0.5605
Epoch 4/30
159/159 [==============================] - 5s 30ms/step - loss: 0.6854 - accuracy: 0.5458 - val_loss: 0.6800 - val_accuracy: 0.5582
Epoch 5/30
159/159 [==============================] - 4s 26ms/step - loss: 0.6827 - accuracy: 0.5570 - val_loss: 0.6775 - val_accuracy: 0.5660
Epoch 6/30
159/159 [==============================] - 5s 30ms/step - loss: 0.6801 - accuracy: 0.5533 - val_loss: 0.6756 - val_accuracy: 0.5566
Epoch 7/30
159/159 [==============================] - 6s 37ms/step - loss: 0.6801 - accuracy: 0.5655 - val_loss: 0.6747 - val_accuracy: 0.5700
Epoch 8/30
159/159 [==============================] - 4s 25ms/step - loss: 0.6748 - accuracy: 0.5812 - val_loss: 0.6723 - val_accuracy: 0.5731
Epoch 9/30
159/159 [==============================] - 4s 27ms/step - loss: 0.6722 - accuracy: 0.5824 - val_loss: 0.6703 - val_accuracy: 0.5700
```

```
Epoch 10/30
159/159 [==============================] - 6s 37ms/step - loss: 0.6741 - accuracy: 0.5731 - val_loss: 0.6683 - val_accuracy: 0.5794
Epoch 11/30
159/159 [==============================] - 4s 26ms/step - loss: 0.6717 - accuracy: 0.5771 - val_loss: 0.6663 - val_accuracy: 0.5904
Epoch 12/30
159/159 [==============================] - 4s 25ms/step - loss: 0.6684 - accuracy: 0.5840 - val_loss: 0.6683 - val_accuracy: 0.5708
Epoch 13/30
159/159 [==============================] - 6s 40ms/step - loss: 0.6694 - accuracy: 0.5895 - val_loss: 0.6624 - val_accuracy: 0.6053
Epoch 14/30
159/159 [==============================] - 4s 27ms/step - loss: 0.6664 - accuracy: 0.5930 - val_loss: 0.6629 - val_accuracy: 0.5967
Epoch 15/30
159/159 [==============================] - 4s 25ms/step - loss: 0.6650 - accuracy: 0.5920 - val_loss: 0.6611 - val_accuracy: 0.5904
Epoch 16/30
159/159 [==============================] - 6s 35ms/step - loss: 0.6606 - accuracy: 0.5963 - val_loss: 0.6575 - val_accuracy: 0.5943
Epoch 17/30
159/159 [==============================] - 5s 30ms/step - loss: 0.6630 - accuracy: 0.5893 - val_loss: 0.6568 - val_accuracy: 0.5959
Epoch 18/30
158/159 [============================>.] - ETA: 0s - loss: 0.6601 - accuracy: 0.5991Restoring model weights from the end of the best epc
159/159 [==============================] - 4s 27ms/step - loss: 0.6596 - accuracy: 0.6001 - val_loss: 0.6531 - val_accuracy: 0.5975
Epoch 18: early stopping
```

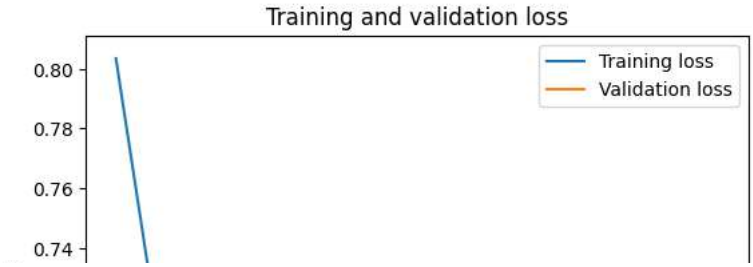
```
# plot the development of the accuracy over epochs to see the training progress
plt.plot(history.history['accuracy'], label='Training accuracy')
plt.plot(history.history['val_accuracy'], label='Validation accuracy')
plt.title('Training and validation accuracy')
plt.xlabel('Epochs')
plt.ylabel('Accuracy')
plt.legend()
```

```
<matplotlib.legend.Legend at 0x7b145affcb50>
```




```
# plot the development of the loss over epochs to see the training progress
plt.plot(history.history['loss'], label='Training loss')
plt.plot(history.history['val_loss'], label='Validation loss')
plt.title('Training and validation loss')
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.legend()
```

```
<matplotlib.legend.Legend at 0x7b1464f2bfd0>
```



Bidirectional RNN

Bidirectional RNN consists of putting two independent RNNs together: one whose input sequence is processed in normal time order, and another which is processed in reverse time order. This way, the network has both backward and forward information about the sequence at every time step. The outputs of the two RNNs are concatenated at each time step.

Keras embedding layer

```
# define model
model = tf.keras.Sequential()
model.add(layers.Embedding(input_dim = len(emb),
                output_dim = emb_length,
                input_length = sequence_length))

model.add(layers.Bidirectional(layers.LSTM(sequence_length, dropout = 0.3, recurrent_dropout = 0.3)))
model.add(layers.Dropout(0.5))
model.add(layers.Dense(1, activation='sigmoid'))

model.compile(loss='binary_crossentropy', optimizer='adam',metrics=['accuracy'])
```

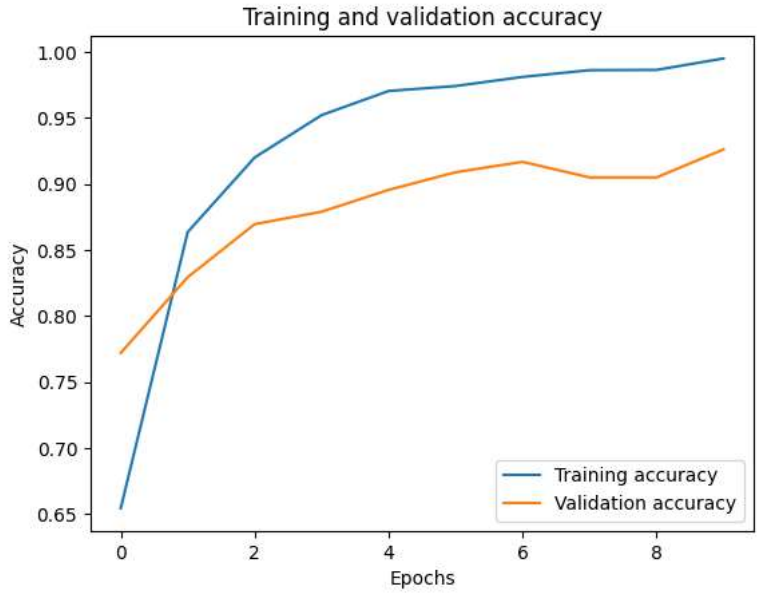
```
# define callbacks for early stopping during training
# stop training when the validation loss `val_accuracy` is no longer improving
callbacks = [tf.keras.callbacks.EarlyStopping(
    monitor='val_accuracy',
    min_delta=1e-3,
    patience=5,
    verbose=1,
    restore_best_weights=True)]
```

```
# train the model
history = model.fit(X_train,
                    y_train,
                    validation_data = (X_test, y_test),
                    epochs=10,
                    batch_size=32,
                    callbacks=callbacks,
                    verbose=1)

model.save("BI_LSTM_1")
```

```
Epoch 1/10
159/159 [==============================] - 37s 192ms/step - loss: 0.6075 - accuracy: 0.6545 - val_loss: 0.4830 - val_accuracy: 0.7720
Epoch 2/10
159/159 [==============================] - 32s 200ms/step - loss: 0.3393 - accuracy: 0.8635 - val_loss: 0.3910 - val_accuracy: 0.8294
Epoch 3/10
159/159 [==============================] - 30s 187ms/step - loss: 0.2205 - accuracy: 0.9200 - val_loss: 0.3682 - val_accuracy: 0.8695
Epoch 4/10
159/159 [==============================] - 29s 184ms/step - loss: 0.1482 - accuracy: 0.9520 - val_loss: 0.3491 - val_accuracy: 0.8789
Epoch 5/10
159/159 [==============================] - 29s 183ms/step - loss: 0.0963 - accuracy: 0.9703 - val_loss: 0.3464 - val_accuracy: 0.8954
Epoch 6/10
159/159 [==============================] - 29s 181ms/step - loss: 0.0786 - accuracy: 0.9740 - val_loss: 0.2987 - val_accuracy: 0.9088
Epoch 7/10
159/159 [==============================] - 29s 180ms/step - loss: 0.0577 - accuracy: 0.9809 - val_loss: 0.3044 - val_accuracy: 0.9167
Epoch 8/10
159/159 [==============================] - 30s 191ms/step - loss: 0.0459 - accuracy: 0.9860 - val_loss: 0.3442 - val_accuracy: 0.9049
Epoch 9/10
159/159 [==============================] - 29s 182ms/step - loss: 0.0449 - accuracy: 0.9862 - val_loss: 0.3446 - val_accuracy: 0.9049
Epoch 10/10
159/159 [==============================] - 29s 182ms/step - loss: 0.0190 - accuracy: 0.9949 - val_loss: 0.4065 - val_accuracy: 0.9261
```

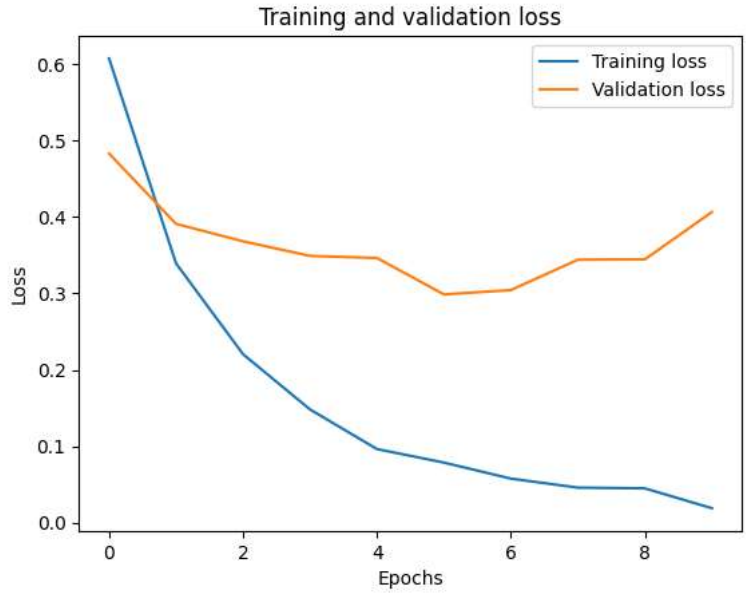
```
# plot the development of the accuracy over epochs to see the training progress
plt.plot(history.history['accuracy'], label='Training accuracy')
plt.plot(history.history['val_accuracy'], label='Validation accuracy')
plt.title('Training and validation accuracy')
plt.xlabel('Epochs')
plt.ylabel('Accuracy')
plt.legend()
```

```
<matplotlib.legend.Legend at 0x7b14615b6d70>
```




```
# plot the development of the loss over epochs to see the training progress
plt.plot(history.history['loss'], label='Training loss')
plt.plot(history.history['val_loss'], label='Validation loss')
plt.title('Training and validation loss')
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.legend()
```

```
<matplotlib.legend.Legend at 0x7b1465051c00>
```



Pre-trained word embeddings

```
# define model
model = tf.keras.Sequential()
model.add(layers.Embedding(input_dim = len(emb),
                 output_dim = emb_length,
```

```
                weights=[emb],
                input_length = sequence_length,
                trainable = False))

model.add(layers.Bidirectional(layers.LSTM(sequence_length, dropout = 0.3, recurrent_dropout = 0.3)))
model.add(layers.Dropout(0.5))
model.add(layers.Dense(1, activation='sigmoid'))

model.compile(loss='binary_crossentropy', optimizer='adam',metrics=['accuracy'])
```

```
# define callbacks for early stopping during training
# stop training when the validation loss `val_accuracy` is no longer improving
callbacks = [tf.keras.callbacks.EarlyStopping(
    monitor='val_accuracy',
    min_delta=1e-3,
    patience=5,
    verbose=1,
    restore_best_weights=True)]
```

```
# train the model
history = model.fit(X_train,
                    y_train,
                    validation_data = (X_test, y_test),
                    epochs=30,
                    batch_size=32,
                    callbacks=callbacks,
                    verbose=1)

model.save("BI_LSTM_2")
```

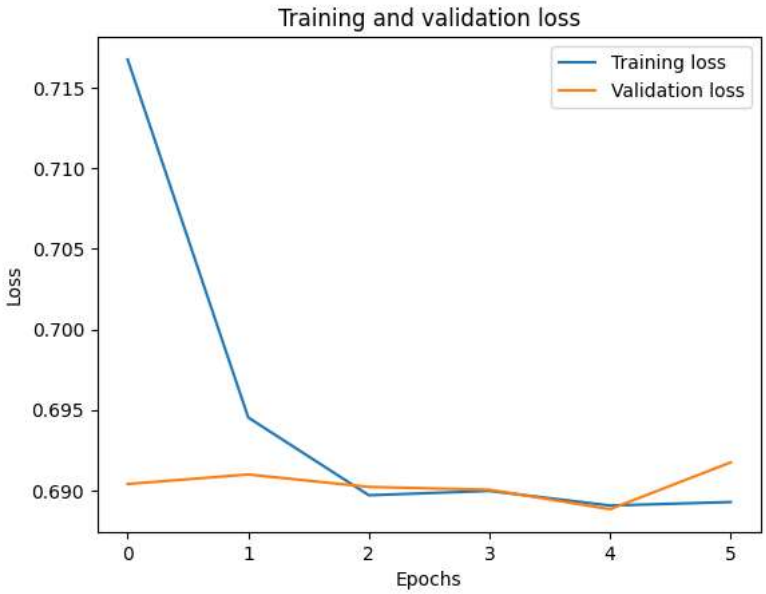
```
Epoch 1/30
159/159 [==============================] - 30s 146ms/step - loss: 0.7167 - accuracy: 0.5006 - val_loss: 0.6904 - val_accuracy: 0.5377
Epoch 2/30
159/159 [==============================] - 20s 129ms/step - loss: 0.6945 - accuracy: 0.5132 - val_loss: 0.6910 - val_accuracy: 0.5259
Epoch 3/30
159/159 [==============================] - 23s 143ms/step - loss: 0.6897 - accuracy: 0.5317 - val_loss: 0.6902 - val_accuracy: 0.5189
Epoch 4/30
159/159 [==============================] - 23s 142ms/step - loss: 0.6900 - accuracy: 0.5313 - val_loss: 0.6901 - val_accuracy: 0.5330
Epoch 5/30
159/159 [==============================] - 23s 145ms/step - loss: 0.6891 - accuracy: 0.5307 - val_loss: 0.6888 - val_accuracy: 0.5228
Epoch 6/30
159/159 [==============================] - ETA: 0s - loss: 0.6893 - accuracy: 0.5289Restoring model weights from the end of the best epo
159/159 [==============================] - 21s 130ms/step - loss: 0.6893 - accuracy: 0.5289 - val_loss: 0.6917 - val_accuracy: 0.5275
Epoch 6: early stopping
```

```
# plot the development of the accuracy over epochs to see the training progress
plt.plot(history.history['accuracy'], label='Training accuracy')
plt.plot(history.history['val_accuracy'], label='Validation accuracy')
plt.title('Training and validation accuracy')
plt.xlabel('Epochs')
plt.ylabel('Accuracy')
plt.legend()
```

```
<matplotlib.legend.Legend at 0x7b1461717910>
```


```
# plot the development of the loss over epochs to see the training progress
plt.plot(history.history['loss'], label='Training loss')
plt.plot(history.history['val_loss'], label='Validation loss')
plt.title('Training and validation loss')
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.legend()
```

```
<matplotlib.legend.Legend at 0x7b1467516bf0>
```




```
model_names = ["SimpleRNN_1", "SimpleRNN_2", "LSTM_1", "LSTM_2", "GRU_1", "GRU_2"]

acc = dict()
auc_res = dict()
for model in model_names:
    loaded_model = tf.keras.models.load_model(model)
    results = loaded_model.evaluate(X_test, y_test, batch_size=32, verbose=0)
    acc[model] = results[1]
```

```
fig, ax = plt.subplots(figsize=(20,7))
plt.xlabel('Accuracy')
keys = acc.keys()
values = acc.values()
plt.bar(keys, values)
acc
```

```
{'SimpleRNN_1': 0.7122641801834106,
 'SimpleRNN_2': 0.5400943160057068,
 'LSTM_1': 0.9198113083839417,
 'LSTM_2': 0.5353773832321167,
 'GRU_1': 0.9103773832321167,
 'GRU_2': 0.6053459048271179}
```

BERT

BERT (Bidirectional Encoder Representations from Transformers) is a pre-trained machine learning model for NLP tasks:

- Base model: 12-layers, 768-hidden, 12-heads, 110M parameter neural network
- Large model: 24-layer, 1024-hidden, 16-heads, 340M parameter neural network

A transformer is a deep learning model that adopts the mechanism of attention, differentially weighting the significance of each part of the input data. Like recurrent neural networks (RNNs), transformers are designed to handle sequential input data, such as natural language, although they do not necessarily process the data in order. Rather, the attention mechanism helps the transformer encoder identify the contextual relations between words in a text. This feature allows for more parallelization than RNNs and therefore reduces training times.

The transformer encoder takes as input a sequence of tokens, which are embedded into real-valued vectors and processed in the neural network.

Training strategies:

- Masked LM (MLM): before processing word sequences, 15% of the words in each sequence are masked, so that the model attempts to predict the masked tokens based on the context.
- Next sentence prediction (NSP): the model receives pairs of sentences as input and learns to predict if the second sentence in the pair is the subsequent sentence in the original document.

When training the BERT model, Masked LM and Next Sentence Prediction are trained together, with the goal of minimizing the combined loss function of the two strategies.

Before being used by BERT as inputs, the tweets are preprocessed using the transformers encoder:

- input ids: token indices, i.e. numerical representations of token sequences that will be used as input by the model
- attention mask: optional argument, indicates which tokens should be attended to and which should not. Sequences of different lengths are padded/sliced until they are of the same dimension; the attention mask indicates the position of the padded indices, so that the model knows which token ids to ignore.

We add a dense, a dropout and an output layer to the pre-trained BERT model and train only those layers for 5 epochs

```
df = pd.read_csv("./data_deeplearning.csv")
```

```
# define train and test size
train_size = int( len(df)*0.8 )
test_size = int( len(df) - train_size )
print(train_size, test_size)
```

```
5086 1272
```

```
# split dataset
X_train = df.head(train_size).tweet_clean
y_train = df.head(train_size).target_encoded

X_test = df.tail(test_size).tweet_clean
y_test = df.tail(test_size).target_encoded

print('Training data X, y:', X_train.shape, y_train.shape)
print('Testing data X, y:', X_test.shape, y_test.shape)
```

```
Training data X, y: (5086,) (5086,)
Testing data X, y: (1272,) (1272,)
```

```
# paramerers, based on results from preprocessing
emb_length = 300
max_length = 58 # length of longest sequence
sequence_length = 35
```

```
# load pre-trained bert tokenizer and model
tokenizer = BertTokenizer.from_pretrained('bert-large-uncased')
bert_model = TFBertModel.from_pretrained('bert-base-uncased')
```

```
Downloading (…)solve/main/vocab.txt: 100%  232k/232k [00:00<00:00, 4.28MB/s]
Downloading (…)okenizer_config.json: 100%  28.0/28.0 [00:00<00:00, 1.41kB/s]
Downloading (…)lve/main/config.json: 100%  571/571 [00:00<00:00, 29.0kB/s]
Downloading (…)lve/main/config.json: 100%  570/570 [00:00<00:00, 18.6kB/s]
Downloading model.safetensors: 100%  440M/440M [00:02<00:00, 147MB/s]
Some weights of the PyTorch model were not used when initializing the TF 2.0 model TFBertModel: ['cls.predictions.transform.LayerNorm.bi
- This IS expected if you are initializing TFBertModel from a PyTorch model trained on another task or with another architecture (e.g. i
- This IS NOT expected if you are initializing TFBertModel from a PyTorch model that you expect to be exactly identical (e.g. initializi
All the weights of TFBertModel were initialized from the PyTorch model.
If your task is similar to the task the model of the checkpoint was trained on, you can already use TFBertModel for predictions without
```

```
def inputs_bert(data, sequence_length):
    """ preprocess text for BERT model input """
    input_ids = []
    attention_masks = []

    for text in data:
        encoded = tokenizer.encode_plus(
        text,
        add_special_tokens=True,
        max_length=sequence_length, #set max length of tokens equal to median tweet length
        pad_to_max_length=True,
        return_attention_mask=True)

        input_ids.append(encoded['input_ids'])
        attention_masks.append(encoded['attention_mask'])

    return np.array(input_ids), np.array(attention_masks)
```

```
train_input_ids, train_attention_masks = inputs_bert(X_train, sequence_length)
test_input_ids, test_attention_masks = inputs_bert(X_test, sequence_length)
```

```
Truncation was not explicitly activated but `max_length` is provided a specific value, please use `truncation=True` to explicitly trunca
```

```
print(train_input_ids.shape, test_input_ids.shape)
```

```
(5086, 35) (1272, 35)
```

```
# add dense, dropout and output layer to pre-trained model and train only those
input_ids = tf.keras.Input(shape=(sequence_length,),dtype='int32')
attention_masks = tf.keras.Input(shape=(sequence_length,),dtype='int32')

output = bert_model([input_ids, attention_masks])
output = output[1]
output = tf.keras.layers.Dense(sequence_length, activation='relu')(output)
output = tf.keras.layers.Dropout(0.5)(output)
output = tf.keras.layers.Dense(1, activation='sigmoid')(output)

model = tf.keras.models.Model(inputs = [input_ids, attention_masks], outputs = output)
model.compile(optimizer=Adam(lr=1e-5), loss='binary_crossentropy', metrics=['accuracy'])
```

```
WARNING:absl:`lr` is deprecated in Keras optimizer, please use `learning_rate` or use the legacy optimizer, e.g.,tf.keras.optimizers.leg
```

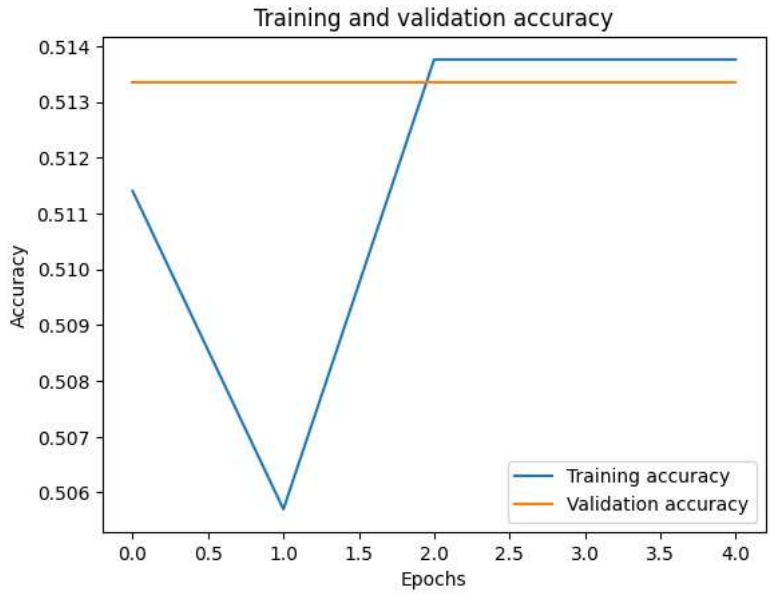
```
history = model.fit(
    [train_input_ids, train_attention_masks],
    y_train,
    validation_data = ([test_input_ids, test_attention_masks], y_test),
    epochs=5,
    batch_size=32)
```

```
Epoch 1/5
159/159 [==============================] - 2361s 15s/step - loss: 0.7265 - accuracy: 0.5114 - val_loss: 0.6931 - val_accuracy: 0.5134
Epoch 2/5
159/159 [==============================] - 2379s 15s/step - loss: 0.6932 - accuracy: 0.5057 - val_loss: 0.6929 - val_accuracy: 0.5134
Epoch 3/5
```

```
159/159 [==============================] - 2393s 15s/step - loss: 0.6931 - accuracy: 0.5138 - val_loss: 0.6929 - val_accuracy: 0.5134
Epoch 4/5
159/159 [==============================] - 2367s 15s/step - loss: 0.6929 - accuracy: 0.5138 - val_loss: 0.6928 - val_accuracy: 0.5134
Epoch 5/5
159/159 [==============================] - 2345s 15s/step - loss: 0.6928 - accuracy: 0.5138 - val_loss: 0.6928 - val_accuracy: 0.5134
```

```
# plot the development of the accuracy over epochs to see the training progress
plt.plot(history.history['accuracy'], label='Training accuracy')
plt.plot(history.history['val_accuracy'], label='Validation accuracy')
plt.title('Training and validation accuracy')
plt.xlabel('Epochs')
plt.ylabel('Accuracy')
plt.legend()
```

```
<matplotlib.legend.Legend at 0x7b1466905660>
```




```
# plot the development of the loss over epochs to see the training progress
plt.plot(history.history['loss'], label='Training loss')
plt.plot(history.history['val_loss'], label='Validation loss')
plt.title('Training and validation loss')
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.legend()
```

```
<matplotlib.legend.Legend at 0x7b1460b06b30>
```