# Untitled

March 4, 2024

[175]:
```
import pandas as pd
import numpy as np
import seaborn as sns
import re
import time
import matplotlib.pyplot as plt
from wordcloud import WordCloud
import nltk
from nltk.stem import PorterStemmer

from sklearn.model_selection import train_test_split, GridSearchCV
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.metrics import accuracy_score
from sklearn.feature_selection import chi2

from sklearn.linear_model import LogisticRegression
from sklearn.naive_bayes import GaussianNB, BernoulliNB, MultinomialNB
from sklearn.ensemble import RandomForestClassifier
from sklearn.tree import DecisionTreeClassifier

import tensorflow as tf
import tensorflow_hub as hub
from tensorflow.keras import layers
import keras_tuner as kt
```

## 1 Import Data

[163]:
```
df = pd.read_csv('./train.csv')
df.head()
```

[163]:
```
              TweetId     Label  \
0  304271250237304833  Politics
1  304834304222064640  Politics
2  303568995880144898    Sports
3  304366580664528896    Sports
4  296770931098009601    Sports
```

```
                                           TweetText
0  '#SecKerry: The value of the @StateDept and @U…
1                            '@rraina1481 I fear so'
2  'Watch video highlights of the #wwc13 final be…
3  'RT @chelscanlan: At Nitro Circus at #AlbertPa…
4  '@cricketfox Always a good thing. Thanks for t…
```

[164]:
```python
df.shape
```

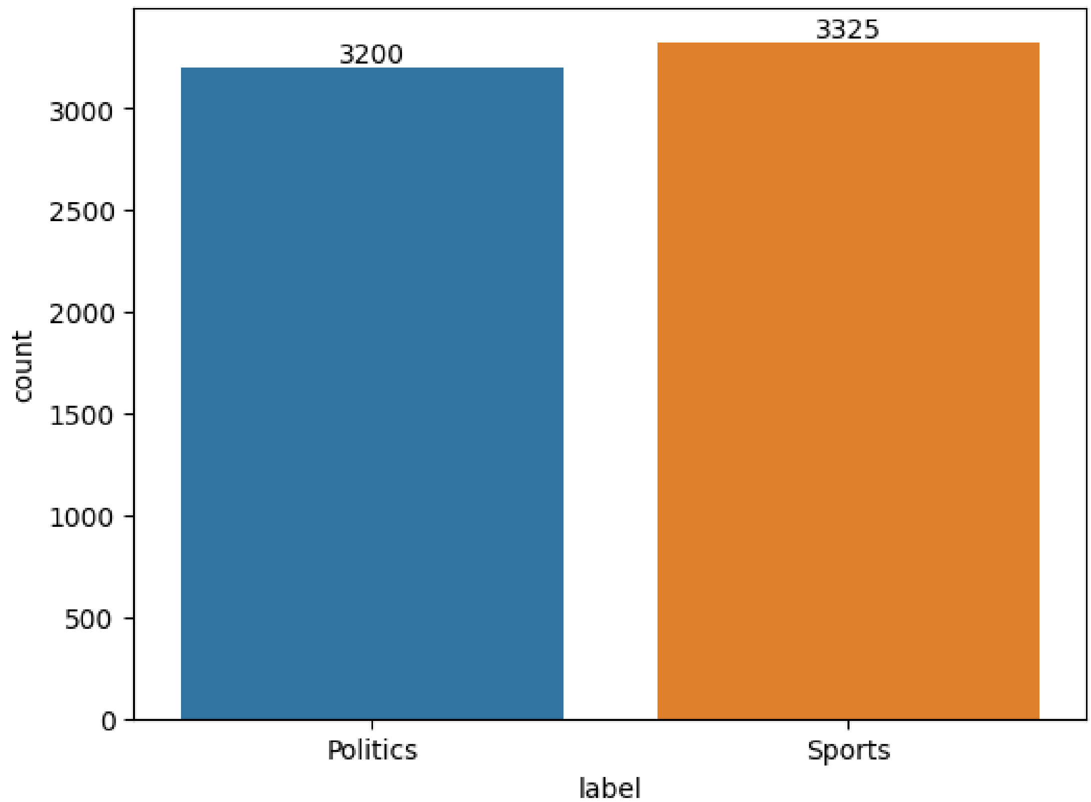
[164]:
```
(6525, 3)
```

[165]:
```python
df.rename({'Label': 'label', 'TweetText':'tweet'}, axis=1, inplace=True)
df.drop(columns=['TweetId'], axis=1, inplace=True)
```

[166]:
```python
df.head()
```

[166]:
```
      label                                            tweet
0  Politics  '#SecKerry: The value of the @StateDept and @U…
1  Politics                          '@rraina1481 I fear so'
2    Sports  'Watch video highlights of the #wwc13 final be…
3    Sports  'RT @chelscanlan: At Nitro Circus at #AlbertPa…
4    Sports  '@cricketfox Always a good thing. Thanks for t…
```

[167]:
```python
plot1 = sns.countplot(df, x='label')
for container in plot1.containers:
    plot1.bar_label(container)
```




the dataset is pretty balance so we can use accuracy as an evaluation metric

[168]:
```
df.isna().sum()
```

[168]:
```
label    0
tweet    0
dtype: int64
```

## 2 Data Preprocessing

[169]:
```
stemmer = PorterStemmer()
stop_words = nltk.corpus.stopwords.words('english')
def preprocess_text(text,
                    remove_single_char=True,
                    tokenize=True,
                    stemming=True,
                    remove_stop_words=True):

    # remove unicode and quotes on both ends
    text = text.encode().decode('unicode-escape').strip('\'"')
```

```
    # remove urls
    text = re.sub(r'https?://\S+|www\.\S+|http?://\S+', ' ', text)

    # remove non-alphanumeric values
    text = re.sub(r'\W', ' ', text)

    # remove single characters
    if remove_single_char:
        text = re.sub(r'\s+[a-zA-Z]\s+', ' ', text)

    # remove number
    text = re.sub(r'\d+', '', text)

    # remove extra spaces
    text = re.sub(r'\s+', ' ', text)

    # lower case
    text = text.lower().strip()

    # tokenize text
    if tokenize:
        text = nltk.word_tokenize(text)

    # remove stop words
    if remove_stop_words:
        if isinstance(text, list):
            text = [word for word in text if word not in stop_words]
        else:
            text = " ".join([word for word in text.split() if word not in␣
 ↪stop_words])

    # stemming
    if stemming:
        if isinstance(text, list):
            text = " ".join([stemmer.stem(token) for token in text])
        else:
            text = " ".join([stemmer.stem(word) for word in text.split()])

    return text
```

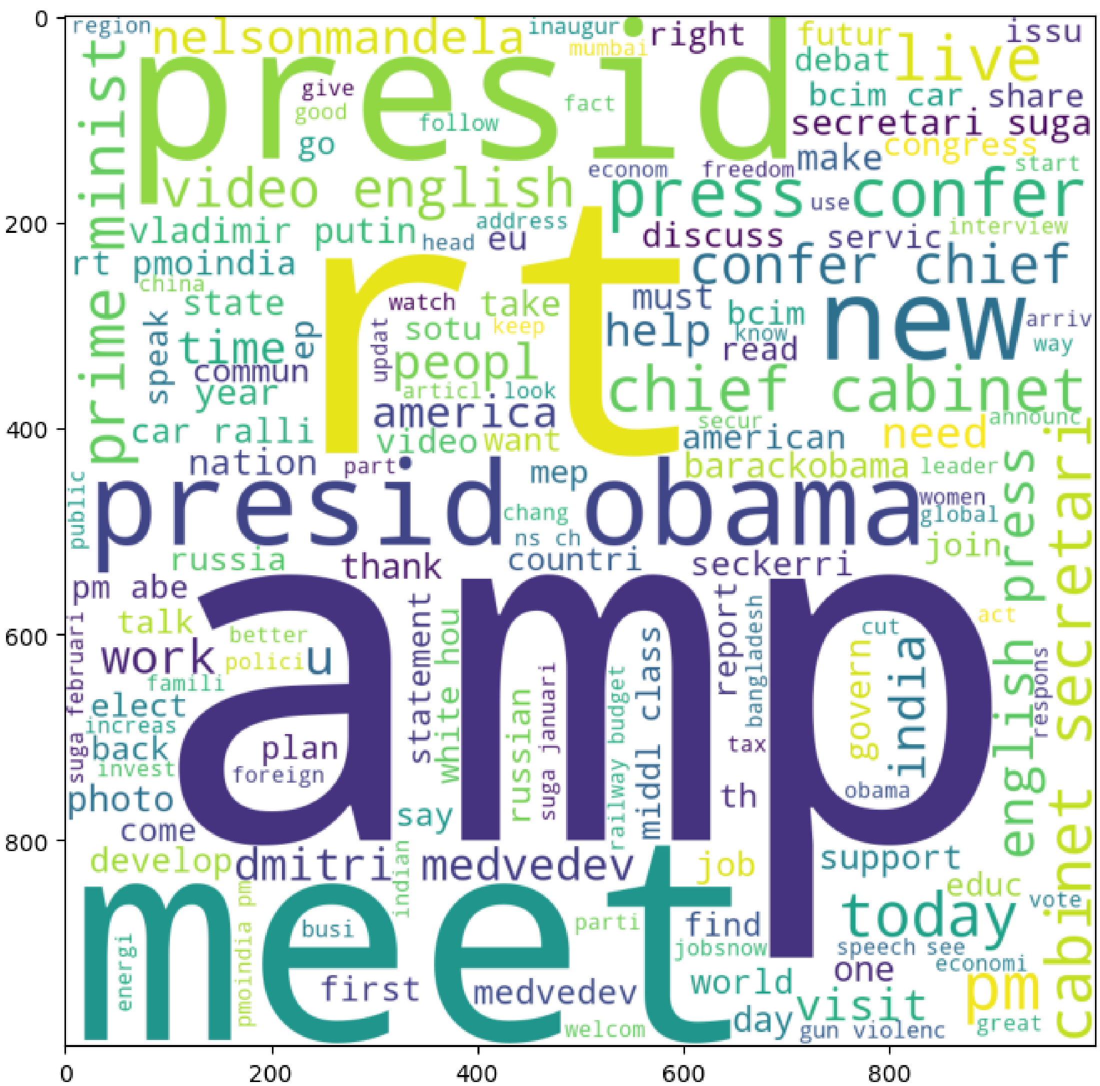
[170]:
```
df['cleaned_text'] = df['tweet'].apply(preprocess_text)

X = pd.DataFrame(df['cleaned_text'])
y = df['label']
```

## 2.1 WordCloud

[102]:
```
wc = WordCloud(width=1000, height=1000, background_color='white',
    min_font_size=15)
wc_politics = wc.generate(df[df.label == "Politics"]['cleaned_text'].str.
    cat(sep=' '))
plt.figure(figsize=(8,8))
plt.imshow(wc_politics)
```

[102]: <matplotlib.image.AxesImage at 0x200d1669300>




[103]:
```
wc = WordCloud(width=1000, height=1000, background_color='white',
    min_font_size=15)
```

```
wc_sports = wc.generate(df[df.label == "Sports"]['cleaned_text'].str.cat(sep='␣
↪'))
plt.figure(figsize=(8,8))
plt.imshow(wc_sports)
```

[103]: <matplotlib.image.AxesImage at 0x200d0dd7fd0>


# 3 Feature Extraction

[104]:
```
def generate_data(vectorizer, X, y):
    X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,␣
↪random_state=42)
```

```
    X_train = vectorizer.fit_transform(X_train['cleaned_text']).toarray()
    X_test = vectorizer.transform(X_test['cleaned_text']).toarray()
    return X_train, X_test, y_train, y_test
```

[105]:
```
vectorizer1 = TfidfVectorizer(max_features=10000)
X_train, X_test, y_train, y_test = generate_data(vectorizer1, X, y)
```

# 4 Model Selection

[108]:
```
def test_models(X_train, X_test, y_train, y_test):

    models = {
        "Logistic Regression": LogisticRegression(),
        "GNB": GaussianNB(),
        "BNB": BernoulliNB(),
        "MNB": MultinomialNB(),
#        "KNN": KNeighborsClassifier(),
#        "lightgbm": LGBMClassifier(),
#        "XGBoost": XGBClassifier(),
      "Random Forest": RandomForestClassifier(),
      'Decision Tree': DecisionTreeClassifier(),
    }
    models_performance = {"model":[], "training_accuracy":[],␣
 ↪"testing_accuracy":[], "training_time":[]}
    for key, model in models.items():
        print(f"model: {key}")
        models_performance['model'].append(key)

        start = time.time()
        model.fit(X_train, y_train)
        stop = time.time()
        training_time = stop - start
        models_performance['training_time'].append(training_time)

        # Training Accuracy
        y_train_pred = model.predict(X_train)
        training_accuracy = accuracy_score(y_train, y_train_pred)
        models_performance['training_accuracy'].append(training_accuracy)

        # Testing Accuracy
        y_test_pred = model.predict(X_test)
        testing_accuracy = accuracy_score(y_test, y_test_pred)
        models_performance['testing_accuracy'].append(testing_accuracy)

    return pd.DataFrame(models_performance)
```

[109]:
```
models_performance = test_models(X_train, X_test, y_train, y_test)
models_performance
```

```
model: Logistic Regression
model: GNB
model: BNB
model: MNB
model: Random Forest
model: Decision Tree
```

[109]:

| | model | training_accuracy | testing_accuracy | training_time |
|---|---|---|---|---|
| 0 | Logistic Regression | 0.986207 | 0.957854 | 1.988721 |
| 1 | GNB | 0.998467 | 0.936398 | 0.789900 |
| 2 | BNB | 0.982184 | 0.963985 | 0.377522 |
| 3 | MNB | 0.985441 | 0.963218 | 0.117687 |
| 4 | Random Forest | 1.000000 | 0.926437 | 28.553573 |
| 5 | Decision Tree | 1.000000 | 0.891188 | 20.684208 |

## 4.1 Feature Importance

since not every feature generated by Tfidf vectorizer relevant to our classification we'll try to filter only the important features using chi-squared test and see if there'll be any improvement in the performance

[110]:
```
feature_names = vectorizer1.get_feature_names_out()
confidence_level = 0.95

_, p_value = chi2(X_train, y_train)
feature_importance = pd.DataFrame({"feature":feature_names, "score":1-p_value})
feature_importance =␣
 ↪feature_importance[feature_importance["score"]>=confidence_level]

important_features = feature_importance['feature'].unique()
```

[111]:
```
vectorizer2 = TfidfVectorizer(max_features=10000, vocabulary=important_features)
X_train, X_test, y_train, y_test = generate_data(vectorizer2, X, y)
```

[112]:
```
models_performance = test_models(X_train, X_test, y_train, y_test)
models_performance
```

```
model: Logistic Regression
model: GNB
model: BNB
model: MNB
model: Random Forest
model: Decision Tree
```

[112]:

| | model | training_accuracy | testing_accuracy | training_time |
|---|---|---|---|---|
| 0 | Logistic Regression | 0.927586 | 0.926437 | 0.131649 |
| 1 | GNB | 0.918966 | 0.914943 | 0.059841 |
| 2 | BNB | 0.924713 | 0.924904 | 0.026959 |
| 3 | MNB | 0.925862 | 0.928736 | 0.012938 |
| 4 | Random Forest | 0.943103 | 0.900383 | 2.788981 |
| 5 | Decision Tree | 0.943103 | 0.885057 | 0.522591 |

[113]:
```
X_train.shape
```

[113]: (5220, 402)

as you can see we went from 10k feature to only 397 feature but had a performance downgrade we'll stick to 10k feature

### 4.2 Hyperparameter tuning

[114]:
```
tfidf = TfidfVectorizer(max_features=10000)
X_train, X_test, y_train, y_test = generate_data(tfidf, X, y)
```

[115]:
```
models = {
    "Logistic Regression": LogisticRegression(),
    "BNB": BernoulliNB(),
    "MNB": MultinomialNB()
}

param_grids = {
    "Logistic Regression": {'C': [0.001, 0.01, 0.1]},
    "BNB": {'alpha': [0.1, 0.5, 1, 1.5, 2]},
    "MNB": {'alpha': [0.1, 0.5, 1, 1.5, 2]}
}
```

[116]:
```
scores = pd.DataFrame({'Model': models.keys(), 'Accuracy': np.
 ↪zeros(len(models))})
model_best_params = models.copy()
i = 0
for key, model in models.items():
    grid_search = GridSearchCV(estimator=model, param_grid=param_grids[key],␣
 ↪cv=4, n_jobs=-1)

    grid_search.fit(X_train, y_train)
    scores.iloc[i, 1] = grid_search.score(X_test, y_test)

    model_best_params[key] = grid_search.best_params_
    i += 1
```

[117]:
```
scores
```

[117]:
```
                 Model  Accuracy
0  Logistic Regression  0.941762
1                  BNB  0.970881
2                  MNB  0.968582
```

[118]:
```
model_best_params
```

[118]:
```
{'Logistic Regression': {'C': 0.1},
 'BNB': {'alpha': 0.1},
 'MNB': {'alpha': 0.1}}
```

## 4.3 Deep Learning Models

### 4.3.1 Word Embedding + Basic NN network

using google nnlm-en-dim50 word embedding

[119]:
```
tf_ds = pd.concat([
                    df.tweet.apply(lambda x: preprocess_text(x,␣
 ↪tokenize=False,stemming=False, remove_stop_words=True)),
                    pd.DataFrame(np.where(df.label=="Politics", 1, 0),␣
 ↪columns=['label'])
                  ],
                  axis=1)
```

[120]:
```
train, val, test = np.split(tf_ds.sample(frac=1), [int(0.8*len(tf_ds)), int(0.
 ↪9*len(tf_ds))])
```

change pandas dataframe to tensorflow dataset as it's much faster to process

[121]:
```
def df_to_dataset(dataframe, shuffle=True, batch_size=256,x_col='tweet',␣
 ↪y_col='label'):
    df = dataframe.copy()
    labels = df.pop(y_col)
    df = df[x_col]
    ds = tf.data.Dataset.from_tensor_slices((df, labels))
    if shuffle:
        ds = ds.shuffle(buffer_size=len(dataframe))
    ds = ds.batch(batch_size)
    ds = ds.prefetch(tf.data.AUTOTUNE)
    return ds
```

[122]:
```
train = df_to_dataset(train)
test = df_to_dataset(test)
val = df_to_dataset(val)
```

[123]:
```
hub_url = 'https://tfhub.dev/google/nnlm-en-dim50/2'
embedding_hub_layer = hub.KerasLayer(hub_url, dtype=tf.string, trainable=True)
```

[124]:
```
model = tf.keras.Sequential()
model.add(embedding_hub_layer)
model.add(layers.Dense(16, activation='relu', kernel_regularizer=tf.keras.
 ↪regularizers.l2(0.001)))
model.add(layers.Dropout(0.5))
model.add(layers.Dense(8, activation='relu', kernel_regularizer=tf.keras.
 ↪regularizers.l2(0.001)))
model.add(layers.Dropout(0.4))
model.add(layers.Dense(1, activation='sigmoid'))
```

[125]:
```
early_stopping = tf.keras.callbacks.EarlyStopping(monitor='val_loss',␣
 ↪patience=3, restore_best_weights=True)
```

[126]:
```
model.compile(
    optimizer=tf.keras.optimizers.Adam(learning_rate=0.001),
    loss=tf.keras.losses.BinaryCrossentropy(),
    metrics=['accuracy']
)
```

[127]:
```
history = model.fit(train, epochs=10, validation_data=val,␣
 ↪callbacks=[early_stopping])
```

```
Epoch 1/10
21/21 [==============================] - 2s 75ms/step - loss: 0.7206 - accuracy:
0.5536 - val_loss: 0.6788 - val_accuracy: 0.7132
Epoch 2/10
21/21 [==============================] - 1s 57ms/step - loss: 0.6620 - accuracy:
0.6584 - val_loss: 0.6172 - val_accuracy: 0.8252
Epoch 3/10
21/21 [==============================] - 1s 58ms/step - loss: 0.5970 - accuracy:
0.7335 - val_loss: 0.5289 - val_accuracy: 0.8911
Epoch 4/10
21/21 [==============================] - 1s 57ms/step - loss: 0.5101 - accuracy:
0.8077 - val_loss: 0.4222 - val_accuracy: 0.9141
Epoch 5/10
21/21 [==============================] - 1s 57ms/step - loss: 0.4351 - accuracy:
0.8420 - val_loss: 0.3336 - val_accuracy: 0.9294
Epoch 6/10
21/21 [==============================] - 1s 57ms/step - loss: 0.3545 - accuracy:
0.8852 - val_loss: 0.2660 - val_accuracy: 0.9371
Epoch 7/10
21/21 [==============================] - 1s 57ms/step - loss: 0.2964 - accuracy:
0.9109 - val_loss: 0.2193 - val_accuracy: 0.9387
Epoch 8/10
21/21 [==============================] - 1s 57ms/step - loss: 0.2402 - accuracy:
0.9374 - val_loss: 0.1933 - val_accuracy: 0.9433
Epoch 9/10
21/21 [==============================] - 1s 57ms/step - loss: 0.2032 - accuracy:
```

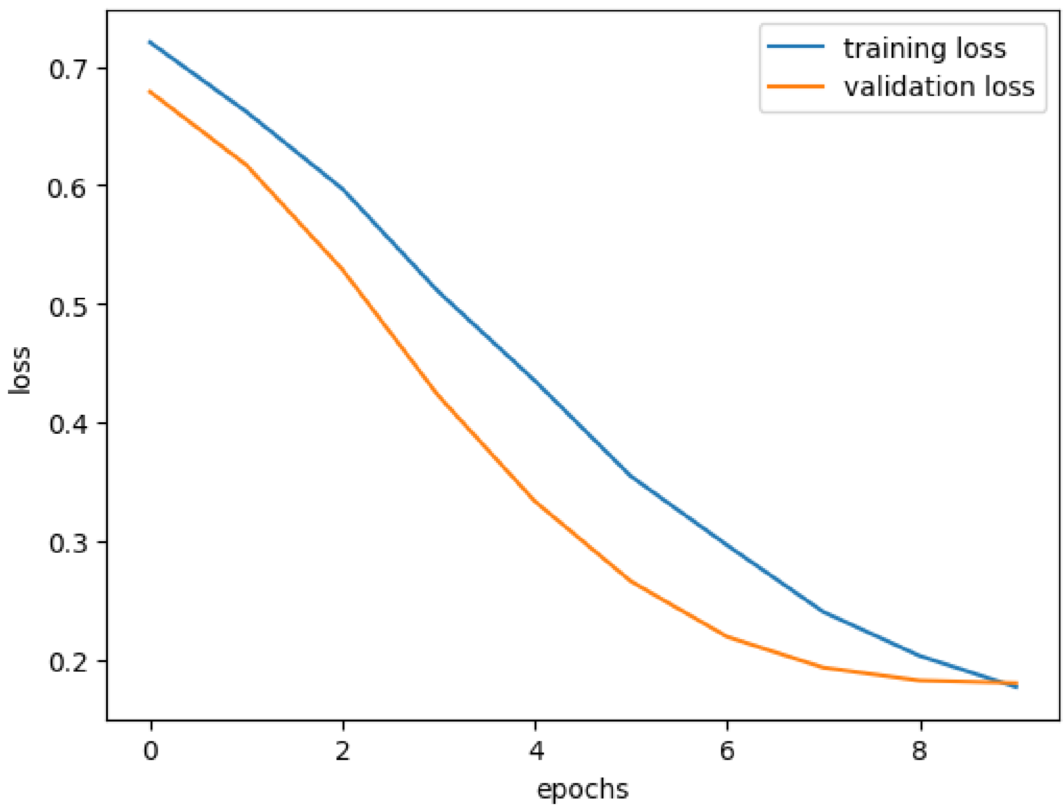
```
0.9494 - val_loss: 0.1825 - val_accuracy: 0.9387
Epoch 10/10
21/21 [==============================] - 1s 57ms/step - loss: 0.1770 - accuracy:
0.9573 - val_loss: 0.1803 - val_accuracy: 0.9402
```

[128]:
```
model.evaluate(test)
```

```
3/3 [==============================] - 0s 6ms/step - loss: 0.1563 - accuracy:
0.9510
```

[128]: [0.1563473641872406, 0.9509953856468201]

[129]:
```
plt.plot(history.history['loss'], label='training loss')
plt.plot(history.history['val_loss'], label='validation loss')
plt.ylabel('loss')
plt.xlabel('epochs')
plt.legend()
plt.show()
```




Basic NN don't seem to be a good model for sequence input data knowing we have overfitting in this case we'll try better architecture

### 4.4 LSTM

[130]:
```
tf_ds = pd.concat([
                    df.tweet.apply(lambda x: preprocess_text(x,␣
 ↪tokenize=False)),
                    pd.DataFrame(np.where(df.label=="Politics", 1, 0),␣
 ↪columns=['label'])
                  ],
                  axis=1)
```

[131]:
```
train, val, test = np.split(tf_ds.sample(frac=1), [int(0.8*len(tf_ds)), int(0.
 ↪9*len(tf_ds))])
```

[132]:
```
train = df_to_dataset(train)
test = df_to_dataset(test)
val = df_to_dataset(val)
```

[133]:
```
encoder = layers.TextVectorization(max_tokens=5000)
encoder.adapt(train.map(lambda text, label: text))
```

[134]:
```
encoder.get_vocabulary()[:10]
```

[134]:
```
['', '[UNK]', 'rt', 'amp', 'test', 'presid', 'indvau', 'pm', 'obama', 'run']
```

#### 4.4.1 LSTM Hyperparameter tuning

[135]:
```
def model_builder(hp):
    model = tf.keras.Sequential()
    model.add(encoder)
    model.add(layers.Embedding(
                            input_dim=len(encoder.get_vocabulary()),
                            output_dim=32,
                            mask_zero=True
                              )
             )

    hp_activation = hp.Choice('activation', values=['relu', 'tanh'])
    hp_lstm_layer = hp.Int('lstm_layer', min_value=4, max_value=32, step=2)
    hp_dense_layer = hp.Int('dense_layer', min_value=4, max_value=32, step=2)
    hp_learning_rate = hp.Choice('learning_rate', values=[1e-2, 1e-3, 1e-4])
    hp_dropout_rate = hp.Choice('dropout_rate', values=[0.3, 0.4])

    model.add(layers.LSTM(units=hp_lstm_layer))
    model.add(layers.Dense(units=hp_dense_layer, activation=hp_activation))
    model.add(layers.Dropout(hp_dropout_rate))
    model.add(layers.Dense(1, activation='sigmoid'))

    model.compile(
```

```
        optimizer=tf.keras.optimizers.Adam(learning_rate=hp_learning_rate),
        loss=tf.keras.losses.BinaryCrossentropy(),
        metrics=['accuracy']
    )

    return model
```

[136]:
```
tuner = kt.Hyperband(model_builder,
                     objective='val_accuracy',
                     max_epochs=10,
                     factor=3,
                     directory='dir',
                     project_name='x')
```

```
Reloading Tuner from dir\x\tuner0.json
```

[137]:
```
early_stopping = tf.keras.callbacks.EarlyStopping(monitor='val_loss',␣
↪patience=3, restore_best_weights=True)
```

[138]:
```
tuner.search(train, epochs=20, validation_data=val, callbacks=[early_stopping])
```

[139]:
```
best_hyper_parameters = tuner.get_best_hyperparameters()[0]
```

[140]:
```
model = tuner.hypermodel.build(best_hyper_parameters)
history = model.fit(train, epochs=50, validation_data=val,␣
↪callbacks=[early_stopping])
```

```
Epoch 1/50
21/21 [==============================] - 5s 72ms/step - loss: 0.4263 - accuracy:
0.7929 - val_loss: 0.1598 - val_accuracy: 0.9356
Epoch 2/50
21/21 [==============================] - 0s 15ms/step - loss: 0.1027 - accuracy:
0.9738 - val_loss: 0.1601 - val_accuracy: 0.9463
Epoch 3/50
21/21 [==============================] - 0s 13ms/step - loss: 0.0401 - accuracy:
0.9893 - val_loss: 0.2093 - val_accuracy: 0.9463
Epoch 4/50
21/21 [==============================] - 0s 12ms/step - loss: 0.0216 - accuracy:
0.9931 - val_loss: 0.2813 - val_accuracy: 0.9387
```

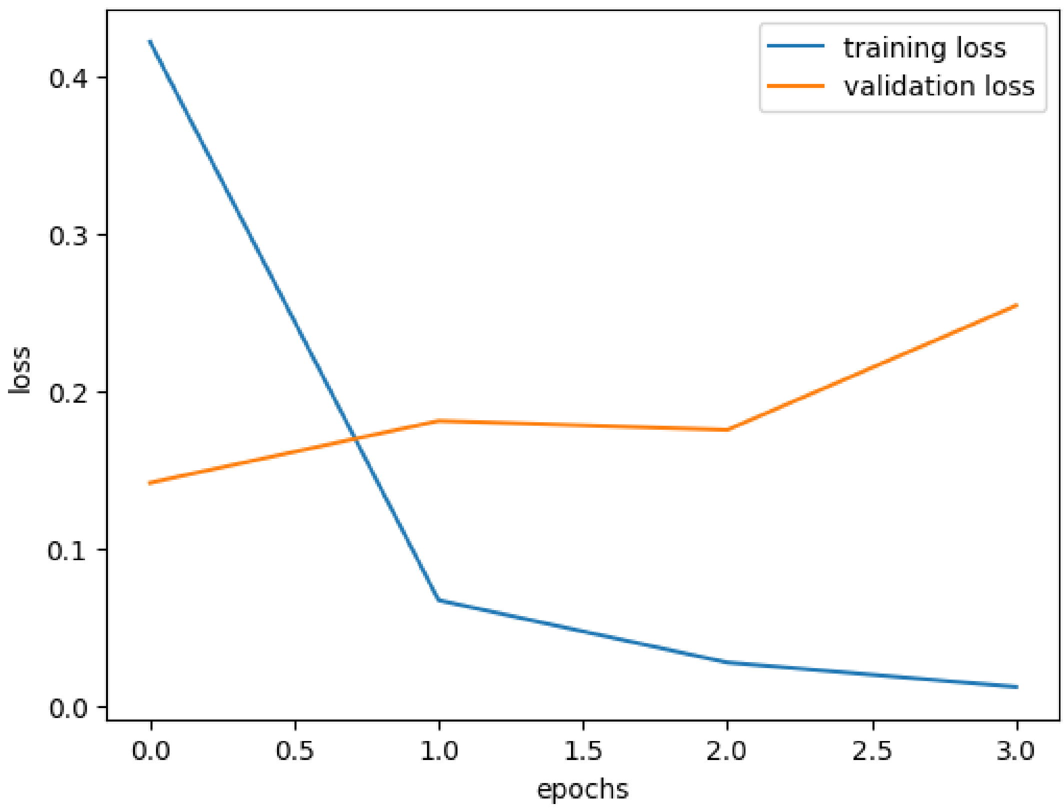
[141]:
```
model.evaluate(test)
```

```
3/3 [==============================] - 0s 6ms/step - loss: 0.1602 - accuracy:
0.9296
```

[141]: [0.16017460823059082, 0.9295558929443359]

[53]:
```
plt.plot(history.history['loss'], label='training loss')
plt.plot(history.history['val_loss'], label='validation loss')
```

```
plt.ylabel('loss')
plt.xlabel('epochs')
plt.legend()
plt.show()
```




Obviously we have an overfitting problem but because of tuner and early stopping we managed to get the best results, but after testing different models the BNB & MNB still perform better so we'll pick BNB for this classification task

### 4.5 BEST MODEL + Submition

[215]:
```
tfidf = TfidfVectorizer(max_features=10000)
# training data
X_train = df['cleaned_text']
y_train = df['label']
X_train = tfidf.fit_transform(X_train).toarray()

# test submission data
X_test = pd.read_csv('./test.csv')
submission = X_test[['TweetId']]
```

```
# preprocessing
X_test = X_test['TweetText'].apply(preprocess_text)
X_test = tfidf.transform(X_test).toarray()
```

[216]:
```
model = BernoulliNB(**model_best_params['BNB'])
model.fit(X_train, y_train)
```

[216]:
```
BernoulliNB(alpha=0.1)
```

[217]:
```
predictions = model.predict(X_test)
```

[218]:
```
submission['Label'] = predictions
```

[219]:
```
submission.head()
```

[219]:

| | TweetId | Label |
|---|---|---|
| 0 | 306486520121012224 | Sports |
| 1 | 286353402605228032 | Sports |
| 2 | 289531046037438464 | Politics |
| 3 | 306451661403062273 | Politics |
| 4 | 297941800658812928 | Sports |

[220]:
```
submission.to_csv('submissions.csv', index=False, index_label=False)
```

## 5 Other Approaches

there many things we could do that maybe will result in better performace such as:

**Use different word embedding**: there is plenty of word embeddings we can use for this task for example Word2Vec or bert word embedding

**Preprocessing**: we can't of course try all the possible pre processing techniques, we used previously stemming we could use lemmization instead or take advantage of ngrams in tfidf vectorizer

**Evaluation Metrics**: we can benefit from ROC-AUC metrics especially for the deep learning models that we used before to help us make a good decision for the threshold

**LLM**: in this era of llm models we could easily fine tune a large language model as they're trained on billions of data

[ ]:
```
```