Phase 4 Project - NLP

Julia Müller

Data Science Flex

Summary abstract

For my Natural Language Processing project, I am using a dataset from CrowdFlower including more than 9000 Tweets about Apple and Google products.

For my business problem, it is well suited because I can advise Google and Apple about which features or activities create positive sentiment to that they can leverage it better. The data set is unbalanced with around 35% of the Tweets being positive, 7% negative and more than 50% neutral or undefined.

- Data cleaning: After basic data cleaning activities, I removed basic stop words as well as product specific words and used Tokenization from TweetTokenizer and WordNetLemmatizer from NLTK which are suitable for handling tweets taking care of # and mentions.
- Modeling: For modeling, I mainly used sklearn libraries and libraries from imblearn to address the class imbalance. Also, I recategorized to have a binary classifier (positive sentiment vs. non-positive including neutrals and negatives).
- Baseline model: I started with a baseline logistic regression model, tuned it by removing stopwords, lemmatization and tokenization.
- Iteration: Furthermore, I applied a random forest model and tuned it with the same steps. For both models, I applied hyperparameter tuning using sklearn’s GridSearchCv. My final model is a tuned random forest model with CountVectorizer, random oversampling and specified parameters such as minimum sample leafs, minimum sample split and the number of estimators.
- Evaluation: My final model has a weighted average precision of .73. When my model predicts positive cases, it is correct 76% of the time which is acceptable. A limitation is that the model is not as great in predicting non-positive sentiments. We can improve the model by collecting more data.

1. Introduction & Business Problem

In today's highly competitive digital landscape, understanding and managing brand perception is paramount for sustained success. Our client, a leading technology company, faces the challenge of effectively gauging public sentiment towards their products in real time. With a massive volume of customer-generated content on social media platforms, particularly Twitter, the client is seeking to harness the power of data science to gain actionable insights from this unstructured data. The business problem at hand revolves around the need to develop an accurate sentiment analysis model capable of classifying tweets related to their products—specifically Apple and Google offerings—as either positive or negative. By doing so, our client aims to proactively identify areas of concern, measure the impact of product launches, marketing campaigns, and other business initiatives, and ultimately refine their strategies to enhance customer satisfaction and loyalty. This project serves as a strategic tool to transform raw social media data into valuable insights, enabling our client to stay ahead in a dynamic and ever-evolving market. For my model evaluation, I will prioritize to correctly identify positive tweets and therefore minimizing false positives. I want to ensure that tweets with positive sentiment are correctly classified as such even if it means potentially having more false negatives.

2. Data Collection & Understanding

First, I will load the required packages and then load and familiarize myself with the data. I will look at the first 100 columns to get a feeling about the content of the tweets, the structure of the data (columns and missing values etc).

In [1]:

```
#loading required packages
import re
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
%matplotlib inline
import seaborn as sns
#train test split and undersampling
from sklearn.model_selection import train_test_split
from imblearn.under_sampling import RandomUnderSampler
from imblearn.over_sampling import RandomOverSampler
from imblearn.over_sampling import SMOTE
#packages for preprocessing
import nltk
from nltk.corpus import stopwords
from nltk.stem import WordNetLemmatizer
from nltk.tokenize import TweetTokenizer
import string
from nltk import FreqDist
#packages for modeling and feature selection
from sklearn.feature_extraction.text import TfidfVectorizer
from imblearn.pipeline import Pipeline
from sklearn.ensemble import RandomForestClassifier
from sklearn.naive_bayes import MultinomialNB
from sklearn.linear_model import LogisticRegression
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.svm import SVC
from sklearn.model_selection import GridSearchCV
#model evaluation
from sklearn.metrics import classification_report, confusion_matrix, ConfusionMatrixDispl
from sklearn.metrics import precision_score, accuracy_score, recall_score, f1_score
import warnings
# Suppress all warnings
warnings.filterwarnings("ignore")
#nltk.download('punkt')
#nltk.download('stopwords')
#nltk.download('wordnet')
```

In [2]:

```
#read data
df = pd.read_csv("data/tweets.csv", encoding="latin-1")
```

In [3]:

```
# Set display options to show all rows and increase the column width
pd.set_option('display.max_rows', None)
pd.set_option('display.max_colwidth', None)
#simplify column names
df.columns = ['Tweet','Brand/Product','Emotion']
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 9093 entries, 0 to 9092
Data columns (total 3 columns):
 #   Column         Non-Null Count  Dtype 
---  ------         --------------  ----- 
 0   Tweet          9092 non-null   object
 1   Brand/Product  3291 non-null   object
 2   Emotion        9093 non-null   object
dtypes: object(3)
memory usage: 213.2+ KB
```

In [4]:

```
df.head(100)
```

Out[4]:

	Tweet	Brand/Product	Emotion
0	.@wesley83 I have a 3G iPhone. After 3 hrs tweeting at #RISE_Austin, it was dead! I need to upgrade. Plugin stations at #SXSW.	iPhone	Negative emotion
1	@jessedee Know about @fludapp ? Awesome iPad/iPhone app that you'll likely appreciate for its design. Also, they're giving free Ts at #SXSW	iPad or iPhone App	Positive emotion
2	@swonderlin Can not wait for #iPad 2 also. They should sale them down at #SXSW.	iPad	Positive emotion
3	@sxsw I hope this year's festival isn't as crashy as this year's iPhone app. #sxsw	iPad or iPhone App	Negative emotion
4	@sxtxstate great stuff on Fri #SXSW: Marissa Mayer (Google), Tim O'Reilly (tech books/conferences) &amp; Matt Mullenweg (Wordpress)	Google	Positive emotion
	@teachntech00 New iPad Apps For #SpeechTherapy And		No emotion

My dataset has more than 9000 tweets and is split in 3 columns. The first column is the tweet, the second one is the information if the tweet is directed at a specific product (Apple or Google) and the third one is the sentiment towards the product. The second column only contains 3200 data points so we don't know about every of the 9000 tweets at which product they are directed at. Also the 3rd column shows for the majority of tweets no emotion. My next steps are to summarize the different products into the two brands Apple or Google and to check if the missing values in the product column really don't include any information about a product.

3. Data Cleaning

3.1 Cleaning of the Brand/Product column

First, I will rename the different products and map them to the brand Apple or Google.

In [5]:

```
#identify distribution of column
df["Brand/Product"].value_counts()
```

Out[5]:

```
iPad                               946
Apple                              661
iPad or iPhone App                 470
Google                             430
iPhone                             297
Other Google product or service    293
Android App                         81
Android                             78
Other Apple product or service      35
Name: Brand/Product, dtype: int64
```

In [6]:

```
#map product category to brand
product_mapping = {
    "iPad": "Apple",
    "iPad or iPhone App": "Apple",
    "iPhone": "Apple",
    "Other Apple product or service": "Apple",
    "Other Google product or service": "Google",
    "Android App": "Google",
    "Android": "Google"
}


df["Brand"] = df["Brand/Product"].replace(product_mapping)
print(df["Brand"].value_counts())
print(df.info())
```

```
Apple     2409
Google     882
Name: Brand, dtype: int64
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 9093 entries, 0 to 9092
Data columns (total 4 columns):
 #   Column         Non-Null Count  Dtype 
---  ------         --------------  ----- 
 0   Tweet          9092 non-null   object
 1   Brand/Product  3291 non-null   object
 2   Emotion        9093 non-null   object
 3   Brand          3291 non-null   object
dtypes: object(4)
memory usage: 284.3+ KB
None
```

3.1.1 Populate missing brand column

Now, where this is cleaned up, I will look at the na values to see if there are no information connected to Apple or Google.

In [7]:

```
filtered_df = df[df['Brand'].isna()]

# Select the first 100 lines of column A from the filtered DataFrame
column_a_subset = filtered_df['Tweet']
column_a_subset[:100]
```

Out[7]:

```
5           @teachntech00 New iPad Apps For #SpeechTherapy And Communica
tion Are Showcased At The #SXSW Conference http://ht.ly/49n4M (http://h
t.ly/49n4M) #iear #edchat #asd
6
NaN
16                                                          Holler Gram
for iPad on the iTunes App Store -  http://t.co/kfN3f5Q (http://t.co/kfN
3f5Q) (via @marc_is_ken) #sxsw
32                                                   Attn: All  #SXSW fr
ineds, @mention Register for #GDGTLive  and see Cobra iRadar for Androi
d. {link}
33
Anyone at  #sxsw want to sell their old iPad?
34
Anyone at  #SXSW who bought the new iPad want to sell their older iPad t
o me?
35                                           At #sxsw.  Oooh. RT @mention
Google to Launch Major New Social Network Called Circles, Possibly Today
```

It looks like there are indeed words in the tweets that will let us identify the brand from the comment. I will create a list of keywords and map them to the different brands

In [8]:

```
# Assign rows to Brand/Product for the unknown one

keywords = ['google', 'apple', 'ipad', 'android', 'iphone']

for index, row in df.iterrows():
    text = row['Tweet']
    if pd.isna(row['Brand/Product']) and isinstance(text, str):
        for keyword in keywords:
            if keyword in text.lower():
                df.at[index, 'Brand/Product'] = keyword
                break
# fill the rest with Unknown
df['Brand/Product'] = df['Brand/Product'].fillna('Unknown')
```

In [9]:

```
brand_keywords = {
    "Apple": ["ipad", "iphone", "itunes", "apple"],
    "Google": ["android", "google"]
}

# Iterate over the DataFrame and update the 'Brand' column for tweets with missing brand
for index, row in df.iterrows():
    if pd.isna(row['Brand']):  # Check if the brand is NaN
        tweet = row['Tweet']
        if isinstance(tweet, str):  # Check if the tweet is a string
            tweet = tweet.lower()  # Transform the tweet to lowercase
            for brand, keywords in brand_keywords.items():
                for keyword in keywords:
                    if keyword in tweet:
                        df.at[index, 'Brand'] = brand
                        break  # Break the loop if a matching keyword is found
        else:
            df.at[index, 'Brand'] = 'unknown'  # Assign 'unknown' for NaN values in 'Brar

# Assign 'unknown' for any remaining NaN values in 'Brand' column
df['Brand'].fillna('unknown', inplace=True)
```

In [10]:

```
df["Brand"].value_counts()
```

Out[10]:

```
Apple      5401
Google     2985
unknown     707
Name: Brand, dtype: int64
```

Since I want to advise Google and Apple about their products, I will drop the rows where we don't have the brand or product information.

In [11]:

```
df = df[df["Brand"] != "unknown"]
```

3.1.2 Clean up Brand/Product column

In [12]:

```
df["Brand/Product"].value_counts()
```

Out[12]:

```
google                             1740
apple                              1195
ipad                               1069
iPad                                946
iphone                              710
Apple                               661
iPad or iPhone App                  470
Google                              430
android                             326
iPhone                              297
Other Google product or service     293
Android App                          81
Android                              78
Unknown                              55
Other Apple product or service       35
Name: Brand/Product, dtype: int64
```

We need to do a bit of clean up because of lower case and upper case values. I will map the different categories.

In [13]:

```
product_mapping = {
    "google": "Google",
    "apple": "Apple",
    "ipad": "iPad",
    "iphone": "iPhone",
    "android": "Android"
}
df["Brand/Product"] = df["Brand/Product"].replace(product_mapping)
print(df["Brand/Product"].value_counts())
```

```
Google                             2170
iPad                               2015
Apple                              1856
iPhone                             1007
iPad or iPhone App                  470
Android                             404
Other Google product or service     293
Android App                          81
Unknown                              55
Other Apple product or service       35
Name: Brand/Product, dtype: int64
```

3.1.3 Drop duplicates and missing values

Also, I want to check for duplicates or missing values and remove them

In [14]:

```
print("Before removal: ", df.duplicated().value_counts())
df.drop_duplicates(inplace=True)
print("After removal: ",df.duplicated().value_counts())
```

```
Before removal:  False    8366
True       20
dtype: int64
After removal:  False    8366
dtype: int64
```

In [15]:

```
#check for missing values
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 8366 entries, 0 to 9092
Data columns (total 4 columns):
 #   Column         Non-Null Count  Dtype 
---  ------         --------------  ----- 
 0   Tweet          8366 non-null   object
 1   Brand/Product  8366 non-null   object
 2   Emotion        8366 non-null   object
 3   Brand          8366 non-null   object
dtypes: object(4)
memory usage: 326.8+ KB
```

3.1.4 Clean up Emotion column

Next, I want to clean up the Emotion column. There are 4 different options: Positive emotion, negative emotion, no emotion and can't tell. The "no emotion" option is the most common one and since I want to create a binary classifier, I will leave positive as positive and combine the neutral and negative ones as non-positive. Also, I will drop the can't tell rows as they are only a very small fraction of the dataset.

In [16]:

```
df["Emotion"].value_counts()
```

Out[16]:

```
No emotion toward brand or product    4688
Positive emotion                      2960
Negative emotion                       568
I can't tell                           150
Name: Emotion, dtype: int64
```

In [17]:

```
#emotions dictionary for mapping
emotions = {
    "No emotion toward brand or product": "Non-positive",
    "Positive emotion": "Positive",
    "Negative emotion": "Non-positive"
}
#mapping old labels to new ones
df["Emotion"] = df["Emotion"].map(emotions)
#check for nas and drop them (can't tell)
print(df['Emotion'].isnull().sum())
# Drop NaN in the emotion column
df.dropna(subset = ["Emotion"], inplace = True)
#check for distribution
df["Emotion"].value_counts(normalize=True)
```

```
150
```

Out[17]:

```
Non-positive    0.639727
Positive        0.360273
Name: Emotion, dtype: float64
```

In [18]:

```
#changing to numerical
df["Emotion"] = df["Emotion"].map({'Non-positive': 1, 'Positive': 0})
```

For my future target variable, I can note that I have a class imbalance. 64% of the cases are not positive, so I will perform different oversampling or undersampling techniques after the train test split to avoid data leakage.

4. Exploratory Data Analysis

Before I start with splitting my dataset and doing the preprocessing, I want to get more familiar with the most frequent words. In this section, I will see how my tokens look like so that I can remove certain stop words in my model.

4.1 Basic preprocessing

Now that I have my dataframe cleaned up, I will start with preparing the tweet texts. Here are the decisions, I have taken:

Stop word removal: I will remove some basic stop words

Tokenization: I will use a specific Tweet Tokenizer that handles hashtags and mentions

4.1.1: Tokenization using TweetTokenizer

In [19]:

```
#initialising Tokenizer
tknzr = TweetTokenizer(strip_handles=True, preserve_case=False)
df['Tokens'] = df['Tweet'].apply(tknzr.tokenize)

#writing a function to get the 20 most common words
def get_most_common_words(df, column_name, N=20):
    # Flatten the list of tokens into a single list
    all_tokens = [token for tokens in df[column_name] for token in tokens]

    # Calculate the frequency distribution
    freq_dist = FreqDist(all_tokens)

    # Get the top N common words
    most_common_words = freq_dist.most_common(N)

    return most_common_words

# applying the function
top_words = get_most_common_words(df, 'Tokens', N=20)
print(top_words)
```

```
[('#sxsw', 8219), ('.', 5525), ('the', 4043), ('link', 3613), ('}', 3599),
('{', 3596), (',', 3261), ('to', 3244), ('at', 2816), ('rt', 2654), ('ipa
d', 2367), ('for', 2309), ('a', 2164), ('!', 2125), ('google', 2082), ('i
n', 1780), ('apple', 1778), ('is', 1593), ('of', 1560), ('"', 1553)]
```

In the list of most common words, there are a lot of common words / stopwords included that I will get rid of. Also, I will include "sxsw" which is an acronym for the Southwest Bank and "rt" which probably stands for retweet to my list of stopwords.I will first remove the stopwords and then see what else I can remove.

4.1.2: Removing stopwords

In [20]:

```
# Get the set of English stopwords
stop_words = set(stopwords.words('english'))
additional_stopwords = ["#sxsw", "sxsw", "sxswi", "#sxswi", "rt"]
stop_words.update(additional_stopwords)

# Function to remove stopwords from a list of tokens
def remove_stopwords(tokens):
    return [token for token in tokens if token not in stop_words]

# Apply the remove_stopwords function to the 'tokens' column
df['Tokens'] = df['Tokens'].apply(remove_stopwords)
#get the top 20 words
top_words = get_most_common_words(df, 'Tokens', N=20)
print(top_words)
```

```
[('.', 5525), ('link', 3613), ('}', 3599), ('{', 3596), (',', 3261), ('ipa
d', 2367), ('!', 2125), ('google', 2082), ('apple', 1778), ('"', 1553),
(':', 1485), ('store', 1440), ('?', 1436), ('2', 1289), ('iphone', 1278),
('-', 1072), ('new', 1035), ('austin', 776), ('app', 753), ('&', 738)]
```

I will also remove the product specific words and treat them as stopwords. I have this information already in my brand and product column, so I know which Apple or Google product the tweet is about. Also, I will remove punctuation

In [21]:

```
additional_stopwords = [
    "ipad", "google", "apple", "iphone", "amp",
    "android", "sxswi", "link", "#apple",
    "#google", "...", "\x89", "#ipad2",
    "0","1","2","3","4","5","6","7","8","9",
    "#iphone", "#android", "store", "austin", "#ipad"]
stop_words.update(additional_stopwords)

# Apply the remove_stopwords function to the 'tokens' column
df['Tokens'] = df['Tokens'].apply(remove_stopwords)
# Remove punctuation from the tokens
df['Tokens'] = df['Tokens'].apply(lambda tokens: [token for token in tokens if token not
#get most common words
top_words = get_most_common_words(df, 'Tokens', N=20)
print(top_words)
```

```
[('new', 1035), ('app', 753), ('launch', 608), ('social', 597), ('circle
s', 539), ('today', 516), ('network', 440), ('pop-up', 410), ('via', 397),
('line', 392), ('get', 365), ('called', 337), ('party', 303), ('mobile', 2
98), ('major', 290), ('free', 274), ('like', 269), ('temporary', 261), ('o
ne', 260), ('time', 257)]
```

4.1.3: Check most common words for Apple and Google products

To understand the most common words a bit better in order to decide if I need to exclude anything else in the stopwords, I will check for both Google and Apple products the most common words.

In [22]:

```
#splitting up data in brands and emotions
apple = df[df["Brand"]=="Apple"]
apple_pos = apple[apple["Emotion"]==1]
apple_nonpos = apple[apple["Emotion"]==0]
google = df[df["Brand"]=="Google"]
google_pos = google[google["Emotion"]==1]
google_nonpos = google[google["Emotion"]==0]
```

In [23]:

```
top_words = get_most_common_words(apple_pos, "Tokens", N=20)
print(top_words)
```

```
[('app', 286), ('pop-up', 259), ('new', 257), ('line', 250), ('temporary',
172), ('opening', 166), ('get', 147), ('via', 137), ('free', 131), ('downt
own', 130), ('popup', 129), ('open', 129), ('one', 121), ('launch', 118),
('pop', 102), ('like', 99), ('need', 96), ("i'm", 95), ('win', 93), ('peop
le', 92)]
```

In [24]:

```
top_words = get_most_common_words(apple_nonpos, "Tokens", N=20)
print(top_words)
```

```
[('app', 309), ('new', 219), ('pop-up', 151), ('line', 123), ('get', 120),
('via', 103), ('one', 98), ("i'm", 96), ('cool', 96), ('temporary', 89),
('free', 88), ('opening', 87), ('downtown', 86), ('like', 81), ('go', 79),
('launch', 78), ('time', 78), ('great', 77), ('popup', 76), ('day', 73)]
```

In [25]:

```
top_words = get_most_common_words(google_pos, "Tokens", N=20)
print(top_words)
```

```
[('social', 439), ('circles', 433), ('new', 420), ('network', 345), ('laun
ch', 331), ('today', 316), ('called', 262), ('major', 228), ('possibly', 1
87), ('mobile', 166), ('party', 140), ('via', 128), ('mayer', 124), ("goog
le's", 121), ('marissa', 117), ('maps', 106), ('app', 86), ('#circles', 7
9), ('search', 77), ('bing', 69)]
```

In [26]:

```
top_words = get_most_common_words(google_nonpos, "Tokens", N=20)
print(top_words)
```

```
[('new', 139), ('party', 105), ('circles', 105), ('social', 103), ('maps',
101), ('network', 84), ('launch', 81), ('mobile', 73), ('app', 72), ('maye
r', 64), ('today', 63), ('called', 60), ('great', 59), ('marissa', 59),
("google's", 56), ('major', 54), ('time', 49), ('w', 41), ('possibly', 4
1), ('get', 38)]
```

5. Model building

My steps for modeling are the following:

1. Removing stop-words
2. Train-Test-Split
3. Address class imbalance
4. Build and train baseline model with basic preprocessing and a vectorizer
5. Evaluate the baseline model
6. Finetune the preprocessing
7. Potentially include other features
8. Iterate through different models

5.1: Removing stopwords

Based on my EDA, I will remove a specific list of stopwords that has to do with the sxsw festival and product related words that won't have a lot of value.

In [27]:

```
#write function to remove stopwords
def remove_stopwords(tweet):
    stop_words = set(stopwords.words('english')) #basic stopwords
    additional_stopwords = [
        "#sxsw", "sxsw", "sxswi", "#sxswi", "rt","ipad",
        "google", "apple", "iphone", "amp",
        "android", "sxswi", "link", "#apple",
        "#google", "...", "\x89", "#ipad2",
        "0","1","2","3","4","5","6","7","8","9",
        "#iphone", "#android", "store", "austin", "#ipad"
    ] + list(string.punctuation)
    stop_words.update(additional_stopwords)

    filtered_tweet = ' '.join([word for word in tweet.split() if word.lower() not in stop
    return filtered_tweet
# add column with filtered tweets
df["Tweets_filtered"] = df["Tweet"].apply(remove_stopwords)
```

5.2: Train-Test-Split

To avoid data leakage, I will now split my cleaned dataset into train and test data

In [28]:

```
# Split the dataset into training and testing sets
X = df[['Tweets_filtered']]  # Feature
y = df['Emotion']  # Target variable

# Split the data into 75% training and 25% testing
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.25, random_state=42

# Print the shapes of the resulting datasets
print("Training data shape:", X_train.shape, y_train.shape)
print("Testing data shape:", X_test.shape, y_test.shape)
# Reshape X_train and X_test
X_train = X_train.values.ravel()
X_test = X_test.squeeze()
# Print the updated shapes
print("Training data shape:", X_train.shape, y_train.shape)
print("Testing data shape:", X_test.shape, y_test.shape)
```

```
Training data shape: (6162, 1) (6162,)
Testing data shape: (2054, 1) (2054,)
Training data shape: (6162,) (6162,)
Testing data shape: (2054,) (2054,)
```

5.3: Building a baseline model - logistic regression, oversampling , count vectorization

I will now build a logistic regression model with random oversampling and Count vectorization but without applying a specific tokenizer.

In [29]:

```
# Define pipeline
pipe_lr = Pipeline([
    ('vectorizer', CountVectorizer()),
    ('oversample', RandomOverSampler(random_state=42)),
    ('classifier', LogisticRegression())
])

# Fit the pipeline on the resampled training data
pipe_lr.fit(X_train, y_train)
```

Out[29]:

```
Pipeline(steps=[('vectorizer', CountVectorizer()),
                ('oversample', RandomOverSampler(random_state=42)),
                ('classifier', LogisticRegression())])
```

In [30]:

```
# Predict on the test data
y_pred = pipe_lr.predict(X_test)

# Evaluate the model
accuracy = accuracy_score(y_test, y_pred)
print(f"Accuracy: {accuracy:.2f}")

report = classification_report(y_test, y_pred)
print("Classification Report:\n", report)
```

```
Accuracy: 0.70
Classification Report:
               precision    recall  f1-score   support

           0       0.58      0.58      0.58       728
           1       0.77      0.77      0.77      1326

    accuracy                           0.70      2054
   macro avg       0.67      0.67      0.67      2054
weighted avg       0.70      0.70      0.70      2054
```

The classification report of the linear regression model indicates moderate performance. The weighted average for precision, recall and f1-score is 0.7. The overall accuracy of the model was 0.7.

5.4 Model iteration - logistic regression, oversample, count vectorizer, TweetTokenizer

Next, I will try the same model but with TweetTokenizer. Based on my EDA, I will do tokenization by using TweetTokenizer that handles hashtags and mentions and apply lemmatization.

In [31]:

```
# Instantiate lemmatizer and tokenizer
lemmatizer = WordNetLemmatizer()
tokenizer = TweetTokenizer()
# Define a custom tokenizer function that applies lemmatization
def custom_tokenizer(text):
    tokens = tokenizer.tokenize(text)
    lemmatized_tokens = [lemmatizer.lemmatize(token) for token in tokens]
    return lemmatized_tokens

# Instantiate tweet tokenizer to later include in the pipeline
lr_pipe_tknzr = Pipeline([
    ('vectorizer', CountVectorizer(tokenizer=custom_tokenizer)),
    ('oversample', RandomOverSampler(random_state=42)),
    ('lr', LogisticRegression(random_state=42, max_iter=1000))
])

lr_pipe_tknzr.fit(X_train, y_train)
```

Out[31]:

```
Pipeline(steps=[('vectorizer',
                 CountVectorizer(tokenizer=<function custom_tokenizer at 0
x000001BC1E4C9C10>)),
                ('oversample', RandomOverSampler(random_state=42)),
                ('lr', LogisticRegression(max_iter=1000, random_state=4
2))])
```

In [32]:

```
# Predict on the test data
y_pred = lr_pipe_tknzr.predict(X_test)

# Evaluate the model
accuracy = accuracy_score(y_test, y_pred)
print(f"Accuracy: {accuracy:.2f}")

report = classification_report(y_test, y_pred)
print("Classification Report:\n", report)
```

```
Accuracy: 0.71
Classification Report:
               precision    recall  f1-score   support

           0       0.59      0.59      0.59       728
           1       0.77      0.77      0.77      1326

    accuracy                           0.71      2054
   macro avg       0.68      0.68      0.68      2054
weighted avg       0.71      0.71      0.71      2054
```

This slightly improved the model. I will try a different vectorization (TFIDF) to see if that improves the model

5.5 Model iteration - Logistic regression, oversample, TFIDF vectorizer

In [33]:

```
lr_pipe_tknzr_oversample = Pipeline([
    ('vectorizer', TfidfVectorizer(tokenizer=custom_tokenizer)),
    ('oversample', RandomOverSampler(random_state=42)),
    ('lr', LogisticRegression(random_state=42))
])

lr_pipe_tknzr_oversample.fit(X_train, y_train)
```

Out[33]:

```
Pipeline(steps=[('vectorizer',
                 TfidfVectorizer(tokenizer=<function custom_tokenizer at 0
x000001BC1E4C9C10>)),
                ('oversample', RandomOverSampler(random_state=42)),
                ('lr', LogisticRegression(random_state=42))])
```

In [34]:

```
# Predict on the test data
y_pred = lr_pipe_tknzr_oversample.predict(X_test)

# Evaluate the model
accuracy = accuracy_score(y_test, y_pred)
print(f"Accuracy: {accuracy:.2f}")

report = classification_report(y_test, y_pred)
print("Classification Report:\n", report)
```

```
Accuracy: 0.71
Classification Report:
               precision    recall  f1-score   support

           0       0.58      0.62      0.60       728
           1       0.78      0.76      0.77      1326

    accuracy                           0.71      2054
   macro avg       0.68      0.69      0.69      2054
weighted avg       0.71      0.71      0.71      2054
```

This is similar than before. I will go back to CountVectorizer but try a different classifier (random forest).

5.6 Model Iteration - Random Forest, Oversample, Count Vectorizer, Tokenizer

In [35]:

```
# Pipeline with Random Forest
rfc_pipe = Pipeline([
    ('vectorizer', CountVectorizer(tokenizer=tokenizer.tokenize)),
    ('oversample', RandomOverSampler(random_state=42)),
    ('rfc', RandomForestClassifier(random_state=42))
])
rfc_pipe.fit(X_train, y_train)
```

Out[35]:

```
Pipeline(steps=[('vectorizer',
                 CountVectorizer(tokenizer=<bound method TweetTokenizer.to
kenize of <nltk.tokenize.casual.TweetTokenizer object at 0x000001BC1E50919
0>>)),
                ('oversample', RandomOverSampler(random_state=42)),
                ('rfc', RandomForestClassifier(random_state=42))])
```

In [36]:

```
# Predict on the test data
y_pred = rfc_pipe.predict(X_test)

# Evaluate the model
accuracy = accuracy_score(y_test, y_pred)
print(f"Accuracy: {accuracy:.2f}")

report = classification_report(y_test, y_pred)
print("Classification Report:\n", report)
```

```
Accuracy: 0.72
Classification Report:
               precision    recall  f1-score   support

           0       0.62      0.55      0.58       728
           1       0.77      0.81      0.79      1326

    accuracy                           0.72      2054
   macro avg       0.69      0.68      0.69      2054
weighted avg       0.72      0.72      0.72      2054
```

5.7 Model iteration - Grid Search on baseline model for hyperparameter tuning

Now, I will try a GridSearch to tune hyperparameters.

In [37]:

```
# Instantiate tweet tokenizer to later include in the pipeline
tokenizer = TweetTokenizer()

# Define the pipeline
lr_pipe_grid = Pipeline([
    ('vectorizer', CountVectorizer(tokenizer=tokenizer.tokenize, ngram_range=(1, 3))),
    ('oversample', RandomOverSampler(random_state=42)),
    ('lr', LogisticRegression(random_state=42))
])

# Define the hyperparameter grid
param_grid = {
    'vectorizer__ngram_range': [(1, 1), (1, 2), (1, 3)],
    'lr__C': [0.1, 1, 10],
    'lr__solver': ['liblinear', 'sag', 'saga'],
    'lr__max_iter': [100, 1000, 10000]
}

# Perform grid search
grid_search = GridSearchCV(lr_pipe_grid, param_grid, cv=5)
grid_search.fit(X_train, y_train)

# Predict on the test data using the best model
best_model = grid_search.best_estimator_
y_pred = best_model.predict(X_test)

# Print the best hyperparameters
print("Best Hyperparameters:", grid_search.best_params_)
```

```
Best Hyperparameters: {'lr__C': 0.1, 'lr__max_iter': 100, 'lr__solver': 'l
iblinear', 'vectorizer__ngram_range': (1, 3)}
```

In [38]:

```
# Instantiate lemmatizer and tokenizer
lemmatizer = WordNetLemmatizer()
tokenizer = TweetTokenizer()

# Instantiate tweet tokenizer to later include in the pipeline
lr_pipe_tknzr = Pipeline([
    ('vectorizer', CountVectorizer(tokenizer=custom_tokenizer, ngram_range=(1, 3))),
    ('oversample', RandomOverSampler(random_state=42)),
    ('lr', LogisticRegression(random_state=42, max_iter=100, C=0.1, solver="liblinear"))
])

lr_pipe_tknzr.fit(X_train, y_train)
```

Out[38]:

```
Pipeline(steps=[('vectorizer',
                 CountVectorizer(ngram_range=(1, 3),
                                 tokenizer=<function custom_tokenizer at 0
x0000022867CA8CA0>)),
                ('oversample', RandomOverSampler(random_state=42)),
                ('lr',
                 LogisticRegression(C=0.1, random_state=42,
                                    solver='liblinear'))])
```

In [39]:

```
# Predict on the test data
y_pred = lr_pipe_tknzr.predict(X_test)

# Evaluate the model
accuracy = accuracy_score(y_test, y_pred)
print(f"Accuracy: {accuracy:.2f}")

report = classification_report(y_test, y_pred)
print("Classification Report:\n", report)
```

```
Accuracy: 0.73
Classification Report:
               precision    recall  f1-score   support

           0       0.78      0.82      0.80      1326
           1       0.63      0.57      0.60       728

    accuracy                           0.73      2054
   macro avg       0.70      0.70      0.70      2054
weighted avg       0.73      0.73      0.73      2054
```

The hyperparameter tuning of my linear regression model resulted in better precision, recall and f1-scores of 0.73.

5.8 Model iteration - Hyperparameter tuning random forest

I will try another hyperparameter tuning of the random forest model.

In [40]:

```
# Instantiate tweet tokenizer to later include in the pipeline
tokenizer = TweetTokenizer()

# Define the pipeline
rf_pipe_grid = Pipeline([
    ('vectorizer', CountVectorizer(tokenizer=tokenizer.tokenize, ngram_range=(1, 3))),
    ('oversample', RandomOverSampler(random_state=42)),
    ('rf', RandomForestClassifier(random_state=42))
])

# Define the hyperparameter grid
param_grid_rf = {
    'vectorizer__ngram_range': [(1, 1), (1, 2), (1, 3)],
    'rf__n_estimators': [50, 100, 200],
    'rf__max_depth': [None, 10, 20],
    'rf__min_samples_split': [2, 5, 10],
    'rf__min_samples_leaf': [1, 2, 4]
}

# Perform grid search
grid_search = GridSearchCV(rf_pipe_grid, param_grid_rf, cv=5)
grid_search.fit(X_train, y_train)

# Predict on the test data using the best model
best_model = grid_search.best_estimator_
y_pred = best_model.predict(X_test)

# Print the best hyperparameters
print("Best Hyperparameters:", grid_search.best_params_)
```

```
Best Hyperparameters: {'rf__max_depth': None, 'rf__min_samples_leaf': 2,
'rf__min_samples_split': 5, 'rf__n_estimators': 200, 'vectorizer__ngram_ra
nge': (1, 2)}
```

In [37]:

```
# Pipeline with Random Forest
rfc_pipe_tuned = Pipeline([
    ('vectorizer', CountVectorizer(tokenizer=tokenizer.tokenize, ngram_range=(1,2))),
    ('oversample', RandomOverSampler(random_state=42)),
    ('rfc', RandomForestClassifier(random_state=42, max_depth=None, min_samples_leaf=2,
                                  min_samples_split=5, n_estimators=200))
])
rfc_pipe_tuned.fit(X_train, y_train)
```

Out[37]:

```
Pipeline(steps=[('vectorizer',
                 CountVectorizer(ngram_range=(1, 2),
                                 tokenizer=<bound method TweetTokenizer.to
kenize of <nltk.tokenize.casual.TweetTokenizer object at 0x000001BC1E50919
0>>)),
                ('oversample', RandomOverSampler(random_state=42)),
                ('rfc',
                 RandomForestClassifier(min_samples_leaf=2, min_samples_sp
lit=5,
                                        n_estimators=200, random_state=4
2))])
```

In [38]:

```
# Predict on the test data
y_pred = rfc_pipe_tuned.predict(X_test)

# Evaluate the model
accuracy = accuracy_score(y_test, y_pred)
print(f"Accuracy: {accuracy:.2f}")

report = classification_report(y_test, y_pred)
print("Classification Report:\n", report)
```

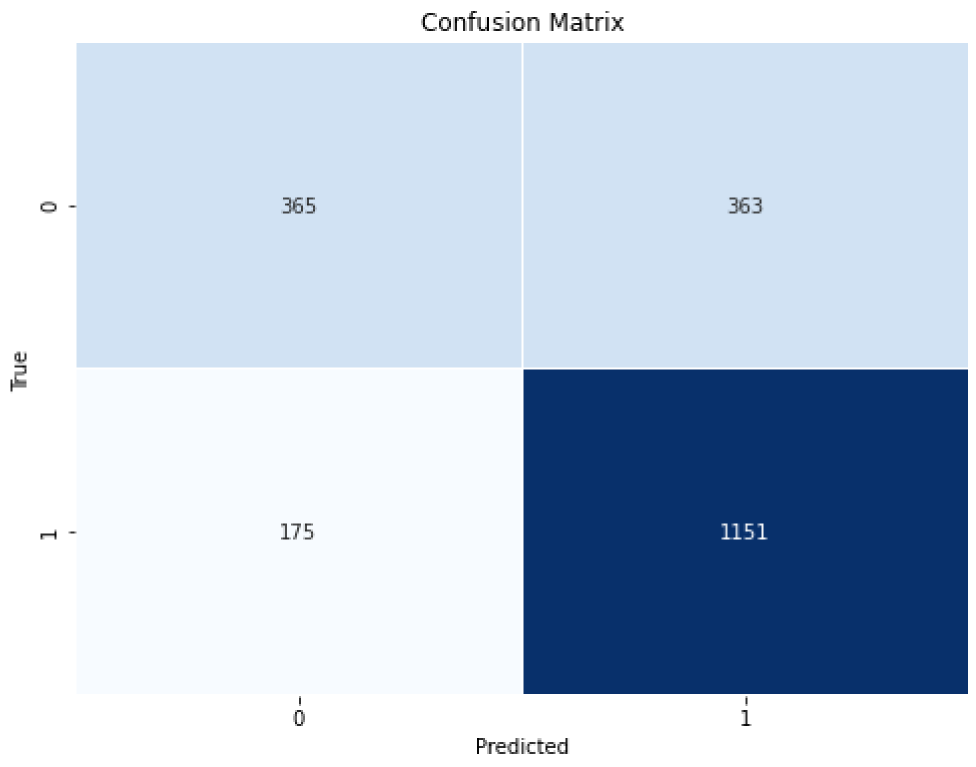
```
Accuracy: 0.74
Classification Report:
               precision    recall  f1-score   support

           0       0.68      0.50      0.58       728
           1       0.76      0.87      0.81      1326

    accuracy                           0.74      2054
   macro avg       0.72      0.68      0.69      2054
weighted avg       0.73      0.74      0.73      2054
```

In [39]:

```
# compute confusion matrix
conf_matrix = confusion_matrix(y_test, y_pred)
# Create a heatmap for visualization
plt.figure(figsize=(8, 6))
sns.heatmap(conf_matrix, annot=True, fmt='d', cmap='Blues', linewidths=0.5, cbar=False)
plt.xlabel('Predicted')
plt.ylabel('True')
plt.title('Confusion Matrix')
plt.show()
```



6 Model evaluation

The accuracy of 74% indicates that the model correctly predicts the positive sentiment for approximately 74% of the tweets in the dataset. However, since my primary interest is in positive sentiment, this general accuracy might not be as crucial as the model's performance in detecting positive sentiment.

Precision (Positive Class 0): Precision measures the proportion of true positive predictions among all positive predictions. In this case, for class 0 (positive sentiment), it's 0.76. This means that when the model predicts positive sentiment, it's correct 76% of the time. This high precision suggests that the model is reliable in identifying positive sentiment tweets.

Recall (Positive Class 0): Recall (Sensitivity) measures the proportion of true positive predictions among all actual positive instances. For class 0, it's 0.87, indicating that the model captures 87% of the actual positive sentiment tweets. This high recall means that the model is effective at identifying the majority of positive sentiment tweets.

F1-score (Positive Class 0): The F1-score, which is the harmonic mean of precision and recall, for class 0 is 0.81. This score confirms the overall effectiveness of the model in identifying positive sentiments.

Limitations:

Class Imbalance: While the model performs well in identifying positive sentiment, it's important to be aware of the class imbalance. There are more positive tweets (class 0) than negative tweets (class 1), which can affect the model's performance on the minority class (class 1). However, since my primary focus is on positive sentiments, this imbalance may not be a significant concern.

Performance on Negative Sentiment (Class 1): Given the emphasis on positive sentiments, it's acceptable that the model may not perform as well in detecting negative sentiment tweets (class 1). The lower recall (0.50) for class 1 indicates that the model may miss some negative tweets, but this might not be a critical issue for our business objective.

In conclusion, the primary goal is to focus on identifying positive sentiments and the model is performing well in terms of precision and recall for positive sentiment tweets.

Conclusion

In conclusion, we have successfully developed a sentiment analysis model with an average weighted precision of 76%, prioritizing the accurate identification of positive tweets to minimize false positives. However, it's worth noting that the model performs better in predicting positive tweets compared to negative ones. To further enhance its performance, we recommend collecting more data and addressing the class imbalance in the dataset. This model represents a critical step in harnessing the power of data science to gain actionable insights from social media data, enabling our client to refine their strategies and maintain a competitive edge in the dynamic digital landscape.

```
In [ ]:
```