Sentiment Analysis

In this notebook, we will use a dataset of movie reviews to train a model to predict whether a review is positive or negative. We will use the ultc-movies.csv to train the model (this is not o the Github repo due to it's size).

To train the model, we will use the `sklearn.feature_extraction.text.CountVectorizer` to create a bag of words representation of the reviews with a `nltk.stem.SnowballStemmer` to stem the words, and `sklearn.naive_bayes.MultinomialNB` as our machine learning model.

We will use the `sklearn.metrics.classification_report` to evaluate the model.

Importing the libraries

Here we import the libraries we will use.

In [ ]:
```
import os
import re
import warnings
import unicodedata
import pandas as pd
import numpy as np
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.naive_bayes import MultinomialNB
from sklearn.metrics import classification_report
from nltk import word_tokenize
from nltk.corpus import stopwords
from nltk.stem.snowball import SnowballStemmer
```

Creating a custom tokenizer

We'll need a custom tokenizer that stems the words in our reviews. We will use the `nltk.stem.SnowballStemmer` to stem the words.

This tokenizer will be used to create a bag of words representation of the reviews.

In [ ]:
```
class StemmerTokenizer:
    def __init__(self):
        self.stemmer = SnowballStemmer("portuguese")
    def __call__(self, doc):
        return [self.stemmer.stem(t) for t in word_tokenize(doc)]
```

Functions

Now we will define the functions we will use. We will use the following functions:

- **create_dataframe**: to create a dataframe from the csv file
- **load_directory**: to load the data from the directory
- **get_training_data**: to get the training data
- **drop_useless_columns**: to drop the columns that we don't need
- **get_csv**: to get the csv file
- **filter_string**: to filter the string
- **integer**: to convert the string to an integer
- **get_count_vectorizer_with_stopwords**: to get the count vectorizer with stopwords
- **normalize**: to normalize the data
- **emotion_from_int**: to get the emotion from the integer
- **predict**: to predict the emotion

Function: create_dataframe

This function will create a dataframe from the csv file.

In [ ]:
```
def create_dataframe(filename):
    nan_value = float("NaN")
    df = pd.read_csv(filename)
    df.replace("", nan_value, inplace=True)
    df.dropna(how="any", inplace=True)
    return df
```

Function: load_directory

This function will load the data from the directory.

In [ ]:
```
def load_directory(directory):
    files = []
    for filename in os.listdir(directory):
        if filename.endswith(".csv") and not filename.startswith("list"):
            files.append(filename)
    return files
```

Function: get_training_data

This function will get the training data.

In [ ]:
```
def get_training_data():
    df = create_dataframe(
        "models/utlc_movies.csv"
    )  # original_index,review_text,review_text_processed,review_text_tokenized,polarity,rating,kfold_polarity,k
    df = drop_useless_columns(df)
    df = df.rename(columns={"polarity": "Class", "review_text_processed": "Data"})
    df = df.reindex(columns=["Class", "Data"])
    return df.sample(frac=1).reset_index(drop=True)
```

Function: drop_useless_columns

This function will drop the columns that we don't need.

In [ ]:
```
def drop_useless_columns(df):
    df.drop(
        [
            "original_index",
            "review_text",
            "review_text_tokenized",
            "rating",
            "kfold_polarity",
            "kfold_rating",
        ],
        axis=1,
        inplace=True,
    )
    return df
```

Function: get_csv

This function will get the csv file.

In [ ]:
```
def get_csv(path):
    df = create_dataframe(path)
    return df
```

Function: filter_string

This function will filter the string.

In [ ]:
```
def filter_string(df, column):
    ret = (
        df[column]
        .apply(lambda x: re.sub("[^a-zA-Z]", " ", x))
        .apply(lambda x: re.sub(" +", " ", x))
        .apply(lambda x: x.strip())
        .apply(lambda x: x.lower())
        .apply(lambda x: normalize(x))
        .values
    )
    return ret
```

Function: integer

This function will convert the string to an integer.

In [ ]:
```
def integer(x):
    return int(x)
```

Function: get_count_vectorizer_with_stopwords

This function will get the count vectorizer with stopwords.

In [ ]:
```
def get_count_vectorizer_with_stopwords():
    return CountVectorizer(
        tokenizer=StemmerTokenizer(),
        ngram_range=(1, 2),
        stop_words=stopwords.words("portuguese"),
    )
```

Function: normalize

This function will normalize the data.

In [ ]:
```
def normalize(text):
    text = (
        unicodedata.normalize("NFKD", text)
        .encode("ascii", "ignore")
        .decode("utf-8", "ignore")
    )
    return text
```

Function: emotion_from_int

This function will get the emotion from the integer.

In [ ]:
```
def emotion_from_int(x):
    if x == 0:
        return "Negative"
    elif x == 1:
        return "Positive"
    else:
        return "Unknown"
```

Function: predict

This function will predict the emotion.

In [ ]:
```
def predict(model, vec, directory):
    files = load_directory(directory)
    for filename in files:
        df_predict = get_csv(directory + "/" + filename)
        predict_data = filter_string(df_predict, "Avaliacoes")
        # print("Loaded data to predict")
        reviews = []
        sentiments = []
        for review in predict_data:
            sentiment = emotion_from_int(
                model.predict(vec.transform([review]).toarray())[0]
            )
            # print("Model predicts that: " + str(review) + " is " + str(sentiment))
            reviews.append(str(review))
            sentiments.append(str(sentiment))
        with open(
            directory.replace("scrapes", "sentimentanalysis") + "/" + filename,
            "w",
            encoding="utf-8",
        ) as f:
            f.write("Sentiment, Review\n")
            for i in range(len(reviews)):
                f.write('"' + sentiments[i] + '", "' + reviews[i] + '"\n')
```

Execution

Now we will execute the code.

In [ ]:
```
warnings.filterwarnings("ignore")

df = get_training_data()
```

```
df_test = df.sample(frac=0.1, random_state=42).reset_index(drop=True)

df = df[:15000]
df_test = df_test[:5000]

train_data, train_class = (
    filter_string(df, "Data"),
    df["Class"].apply(lambda x: integer(x)).values,
)
print("Loaded training data")

test_data, test_class = (
    filter_string(df_test, "Data"),
    df_test["Class"].apply(lambda x: integer(x)).values,
)
print("Loaded test data")

vec = get_count_vectorizer_with_stopwords()
print("Created vectorizer")

train_data = vec.fit_transform(train_data).toarray()
print("Transformed training data")

test_data = vec.transform(test_data).toarray()
print("Transformed test data")

model = MultinomialNB()
print("Created model")

model.fit(train_data, train_class)
print("Trained model")

print(
    "Tested model: " + str(model.score(test_data, test_class) * 100) + "%" + " accuracy"
)
```

Prediction of the emotion of the reviews of various establishments of various types and platforms and export the results to a csv file.

In [ ]:

```
predict(model, vec, "../scrapes/booking/hotels")
predict(model, vec, "../scrapes/zomato/restaurantes")
predict(model, vec, "../scrapes/tripadvisor/hotels")
predict(model, vec, "../scrapes/tripadvisor/restaurants")
predict(model, vec, "../scrapes/tripadvisor/activities")
```