MMA 865, Individual Assignment 1

- [Jose, Chua]
- [20069208]
- [January 26, 2025]

Part 1: Sentiment Analysis via the ML-based approach

Download the “Product Sentiment” dataset from the course portal: sentiment_train.csv and sentiment_test.csv.

Part 1.a. Loading and Prep

Load, clean, and preprocess the data as you find necessary.

In [23]:
```
#libraries
import pandas as pd
import string
import nltk
from nltk.tokenize import word_tokenize
from nltk.corpus import stopwords
from nltk.stem import PorterStemmer, WordNetLemmatizer
import matplotlib.pyplot as plt
from unidecode import unidecode
import re
from textblob import TextBlob
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.svm import SVC
from sklearn.model_selection import train_test_split
from sklearn.metrics import classification_report, accuracy_score
from sklearn.model_selection import GridSearchCV
from sklearn.model_selection import RandomizedSearchCV
```

In [24]:
```
#importing test and train data

df_train = pd.read_csv("sentiment_train.csv")

print(df_train.info())
print(df_train.head())

df_test = pd.read_csv("sentiment_test.csv")

print(df_train.info())
print(df_train.head())
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 2400 entries, 0 to 2399
Data columns (total 2 columns):
 #   Column    Non-Null Count  Dtype
---  ------    --------------  -----
 0   Sentence  2400 non-null   object
 1   Polarity  2400 non-null   int64
dtypes: int64(1), object(1)
memory usage: 37.6+ KB
None
                                            Sentence  Polarity
0                           Wow... Loved this place.         1
1                                 Crust is not good.         0
2          Not tasty and the texture was just nasty.         0
3  Stopped by during the late May bank holiday of...         1
4  The selection on the menu was great and so wer...         1
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 2400 entries, 0 to 2399
Data columns (total 2 columns):
 #   Column    Non-Null Count  Dtype
---  ------    --------------  -----
 0   Sentence  2400 non-null   object
 1   Polarity  2400 non-null   int64
dtypes: int64(1), object(1)
memory usage: 37.6+ KB
None
                                            Sentence  Polarity
0                           Wow... Loved this place.         1
1                                 Crust is not good.         0
2          Not tasty and the texture was just nasty.         0
3  Stopped by during the late May bank holiday of...         1
4  The selection on the menu was great and so wer...         1
```

Cleaning

1. Remove Duplicates
2. Case normalization
3. Removing punctuation
4. Removing special characters
5. Removing extra whitespace
6. removing numbers
7. Spell Checking
8. Tokenization
9. Stop words

In [25]:
```
#checking for duplicates (train)

duplicates = df_train[df_train['Sentence'].duplicated()]

print("Duplicate Rows:")
print(duplicates)
```

```
Duplicate Rows:
                                               Sentence  Polarity
71                                              #NAME?         1
219                                             #NAME?         1
814                                  I love this place.         1
816                              The food was terrible.         0
843                                    I won't be back.         0
846                   I would not recommend this place.         0
904                                             #NAME?         0
1285                                      Great phone!.         1
1407                                       Works great.         1
1524                                      Works great!.         1
1543                           Don't buy this product.         0
1744  If you like a loud buzzing to override all you...         0
1748                                      Does not fit.         0
1778                              This is a great deal.         1
1792                                       Great Phone.         1
1892                   Excellent product for the price.         1
1896                                       Great phone.         1
2363                   Definitely worth checking out.          1
```

In [26]:
```
#deleting duplicates

df_train=df_train[df_train['Sentence'] != '#NAME?']
```

In [27]:
```
#checking for duplicates (test)

duplicates = df_test[df_test['Sentence'].duplicated()]

print("Duplicate Rows:")
print(duplicates)
```

```
Duplicate Rows:
               Sentence  Polarity
185  Not recommended.           0
```

In [28]:
```
#my cleaning function

def preprocess_text(sentence):

    # Case normalization
    sentence = sentence.lower()

    # Remove punctuation
    sentence = re.sub(f"[{string.punctuation}]", "", sentence)

    # Replace special characters
    sentence = unidecode(sentence)

    # Remove extra whitespace
    sentence = re.sub(r"\s+", " ", sentence).strip()

    # Remove numbers
    sentence = re.sub(r"\d+", "", sentence)

    # Spell checking
    sentence = str(TextBlob(sentence).correct())

    # Tokenization
    tokens = word_tokenize(sentence)

    # Stopword removal
    stop_words = set(stopwords.words('english'))
    critical_stopwords = {
        "not", "no", "nor", "never",
        "but", "however", "yet", "though", "although",
        "very", "too", "much", "more", "most", "few", "less", "least",
        "all", "some", "any", "only", "every", "each", "none",
        "if", "then", "when", "while"
    }
    stop_words -= critical_stopwords
    tokens = [word for word in tokens if word.lower() not in stop_words]

    return tokens
```

In [ ]:
```
# Apply function to train
df_train['Sentence'] = df_train['Sentence'].apply(preprocess_text)

# Apply function to test
df_test['Sentence'] = df_test['Sentence'].apply(preprocess_text)

print(df_train)
print(df_test)
```

```
                                               Sentence  Polarity
0                                        [loved, place]         1
1                                   [crust, not, good]         0
2                         [not, taste, texture, nasty]         0
3     [stopped, late, may, bank, holiday, rich, stev...         1
4                        [selection, menu, great, prices]         1
...                                                 ...       ...
2395  [almost, all, songs, cover, girl, oldfashioned...         0
2396  [most, annoying, thing, cover, girl, way, rite...         0
2397  [unfortunately, cover, girl, example, hollywoo...         0
2398  [nonlinear, narration, thus, many, flashbacks,...         1
2399  [good, cinematography, also, makes, monica, be...         1

[2396 rows x 2 columns]
                                               Sentence  Polarity
0    [good, commentary, today, love, undoubtedly, f...         1
1    [people, first, times, film, making, think, ex...         1
2    [very, popular, when, cinema, good, house, ver...         1
3           [feelgood, film, felt, when, came, cinema]         1
4    [northern, humour, positive, community, repres...         1
..                                                 ...       ...
595  [got, bored, watching, justice, large, take, c...         0
596  [unfortunately, any, virtue, films, production...         0
597                              [word, embarrassing]         0
598                              [exceptionally, bad]         0
599  [all, all, insult, ones, intelligence, huge, w...         0

[600 rows x 2 columns]
```

Part 1.b. Modeling

Use your favorite ML algorithm to train a classification model. Don’t forget everything that we’ve learned in our ML course: hyperparameter tuning, cross validation, handling imbalanced data, etc. Make reasonable decisions and try to create the best-performing classifier that you can.

SVM

In [30]:
```
# Convert lists in the 'Sentence' column to strings
df_train['Sentence'] = df_train['Sentence'].apply(lambda x: ' '.join(x) if isinstance(x, list) else x)
df_test['Sentence'] = df_test['Sentence'].apply(lambda x: ' '.join(x) if isinstance(x, list) else x)
```

In [31]:
```
# Define features and target
X_train = df_train['Sentence']
y_train = df_train['Polarity']
X_test = df_test['Sentence']
y_test = df_test['Polarity']
```

In [32]:
```
# Modifying CountVectorizer to handle stopwords
vectorizer = CountVectorizer(stop_words=None)
X_train_vec = vectorizer.fit_transform(X_train)
X_test_vec = vectorizer.transform(X_test)
```

In [33]:
```
# Creating and training the SVM model
svm_model = SVC(kernel='linear', random_state=42)
svm_model.fit(X_train_vec, y_train)
```

Out[33]:
```
SVC(kernel='linear', random_state=42)
```

In [34]:
```
y_pred = svm_model.predict(X_test_vec)
```

In [35]:
```
# Evaluation
print("Accuracy:", accuracy_score(y_test, y_pred))
print("Classification Report:\n", classification_report(y_test, y_pred))
```

```
Accuracy: 0.7533333333333333
Classification Report:
               precision    recall  f1-score   support

           0       0.70      0.84      0.76       287
           1       0.82      0.68      0.74       313

    accuracy                           0.75       600
   macro avg       0.76      0.76      0.75       600
weighted avg       0.76      0.75      0.75       600
```

In [36]:
```
param_grid = {
    'C': [0.1, 1, 10, 100],  # Regularization parameter
    'kernel': ['linear', 'poly', 'rbf', 'sigmoid'],  # Kernel types
    'gamma': ['scale', 'auto'],  # Kernel coefficient
}

# Create an instance of the SVM model
svm_model = SVC(random_state=42)
```

In [37]:
```
# Set up GridSearchCV
grid_search = GridSearchCV(estimator=svm_model, param_grid=param_grid,
                           cv=5, scoring='accuracy', verbose=1, n_jobs=-1)
```

In [38]:
```
# Perform grid search
grid_search.fit(X_train_vec, y_train)
```

```
Fitting 5 folds for each of 32 candidates, totalling 160 fits
```

Out[38]:
```
GridSearchCV(cv=5, estimator=SVC(random_state=42), n_jobs=-1,
             param_grid={'C': [0.1, 1, 10, 100], 'gamma': ['scale', 'auto'],
                         'kernel': ['linear', 'poly', 'rbf', 'sigmoid']},
             scoring='accuracy', verbose=1)
```

In [39]:
```
# Best hyperparameters and corresponding score
print("Best Parameters:", grid_search.best_params_)
print("Best Cross-Validation Accuracy:", grid_search.best_score_)
```

```
Best Parameters: {'C': 10, 'gamma': 'scale', 'kernel': 'rbf'}
Best Cross-Validation Accuracy: 0.815937717466945
```

Part 1.c. Assessing

Use the testing data to measure the accuracy and F1-score of your model.

In [ ]:
```
# Evaluation
best_model = grid_search.best_estimator_
y_pred = best_model.predict(X_test_vec)

print("Test Set Accuracy:", accuracy_score(y_test, y_pred))
print("Test F1 Score:", f1_score(y_test, y_pred))
print("Classification Report:\n", classification_report(y_test, y_pred))
```

```
Test Set Accuracy: 0.7566666666666667
Test F1 Score: 0.7345454545454545
Classification Report:
               precision    recall  f1-score   support

           0       0.69      0.88      0.78       287
           1       0.85      0.65      0.73       313

    accuracy                           0.76       600
   macro avg       0.77      0.76      0.75       600
weighted avg       0.78      0.76      0.75       600
```

Part 2. Given the accuracy and F1-score of your model, are you satisfied with the results, from a business point of view? Explain.

From a business POV my SVM model's accuracy of 75.67 and F1-score of 73.45 are decent and acceptable but not entirely satisfactory for a product sentiment analysis application. In a business application I would aim for 80-90% accuracy for more reliable insights. My F1-score tells me that my model performs better at identifying negative sentiment (class 0) with higher recall (88%) however my recall for positive sentiment (class 1) is only 65% meaning my model misclassifies a significant number of positive reviews as negative. This would not be suitable from a business point of view because it makes it seem like more customers are dissatisfied than they actually are. This could lead to unnecessary product changes, misplaced marketing efforts, or incorrect conclusions about customer satisfaction. In the future I could improve my model with feature engineering such as TF-IDF vectors or n-grams, data balancing techniques or try different parameters.

Part 3. Show five example instances in which your model’s predictions were incorrect. Describe why you think the model was wrong. Don’t just guess: dig deep to figure out the root cause.

In [41]:
```
#Compare predictions with actual values
df_test_with_predictions = pd.DataFrame({
    'Sentence': X_test,
    'Actual': y_test,
    'Predicted': y_pred
}).reset_index(drop=True)

# Display the new DataFrame
print(df_test_with_predictions.head())
```

```
                                            Sentence  Actual  Predicted
0  good commentary today love undoubtedly film wo...       1          1
1  people first times film making think excellent...       1          1
2  very popular when cinema good house very good ...       1          1
3                feelgood film felt when came cinema       1          0
4      northern humour positive community represents       1          1
```

In [42]:
```
selected_rows = df_test_with_predictions.loc[[307, 490, 242, 588, 433]]
selected_rows
```

Out[42]:

	Sentence	Actual	Predicted
307	fat computer week unbelievable bible thumped a...	0	1
490	only consistent thread holding series together...	0	1
242	barely boring moment film plenty humorous parts	1	0
588	riot see hugo weaving play sexobsessed gay rea...	1	0
433	feelings thoughtsgabriels discomfort danceall ...	1	0

1. Lack of Context Understanding

Example:

- "fat computer week unbelievable bible thumped adams girl actors"
- "riot see hugo weaving play sexobsessed gay real estate salesman uses clients houses trusts flaming warren tom holland"

The model may be relying too heavily on individual words rather than overall sentence meaning. Words like "unbelievable," "bible thumped," "fat," "sexobsessed," and "flaming" might have been learned as indicators of negative sentiment, even if the sentence isn’t actually negative.

2. Failure to Detect Subtle Positive Sentiment

Example:

- "only consistent thread holding series together amazing performances lent parker anita laselva two salons quiet ideological conflict"
- "barely boring moment film plenty humorous parts"

In the first sentence, the model might have focused on words like "quiet ideological conflict" rather than recognizing "amazing performances" as an indicator of positive sentiment. The second sentence contains "barely boring", which actually means not boring may have misclassified it as negative due to the presence of "boring."

3. Ambiguous or Abstract Language

Example:

- "feelings thoughtsgabriels discomfort danceall intangible leap life come within viewer grasp customs portray"

This review is philosophical and vague, making it difficult for my simple SVM model to classify it correctly. This sentence contains words like discomfort which could be taken as negative sentiment. The sentence overall might be neutral or positive depending on the context (I cant even tell, thats how ambiguous this sentence is).