MMA 865, Individual Assignment 1

Last Updated December 11, 2023.

- Anthony Ramelo
- 20499391
- Jan 21, 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 [1]:
```
import pandas as pd
import re
import nltk
from nltk.corpus import stopwords
from nltk.stem import WordNetLemmatizer
from spellchecker import SpellChecker
from imblearn.over_sampling import SMOTE
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.ensemble import StackingClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import train_test_split
from sklearn.metrics import classification_report, accuracy_score, roc_auc_score, f1_score
from xgboost import XGBClassifier
from lightgbm import LGBMClassifier

# Download necessary NLTK data
nltk.download('stopwords')
nltk.download('wordnet')

# Initialize resources
default_stop_words = set(stopwords.words('english'))
custom_stop_words = default_stop_words - {"not", "no", "never"}
lemmatizer = WordNetLemmatizer()
spell = SpellChecker()

# Load datasets
df_train = pd.read_csv("sentiment_train.csv")
df_test = pd.read_csv("sentiment_test.csv")

# Ensure 'Sentence' column exists and remove null/empty rows
df_train.dropna(subset=['Sentence'], inplace=True)
df_test.dropna(subset=['Sentence'], inplace=True)

# Updated text cleaning function to retain double negatives
def clean_text_v4(text):
    # Remove placeholders, URLs, special characters, and numbers
    text = re.sub(r'#NAME\?|http\S+|www\S+|https\S+|\@\w+|\#|\d+', '', text, flags=re.MULTILINE)
    text = re.sub(r'[^\w\s]', '', text)  # Remove punctuations
    text = text.lower()  # Convert to lowercase
    # Tokenize the text and remove stop words while retaining 'not', 'no', and 'never'
    words = text.split()
    filtered_words = [word for word in words if word not in custom_stop_words]
    # Perform spell check
    corrected_words = [spell.correction(word) if spell.correction(word) else word for word in filtered_words]
    return ' '.join(corrected_words)

# Apply updated cleaning function to datasets
df_train['Cleaned_Sentence'] = df_train['Sentence'].apply(clean_text_v4)
df_test['Cleaned_Sentence'] = df_test['Sentence'].apply(clean_text_v4)
df_train.dropna(subset=['Cleaned_Sentence'], inplace=True)
df_test.dropna(subset=['Cleaned_Sentence'], inplace=True)

# Features and target
X_train = df_train['Cleaned_Sentence']
y_train = df_train['Polarity']
```

```
X_test = df_test['Cleaned_Sentence']
y_test = df_test['Polarity']
```

```
[nltk_data] Downloading package stopwords to
[nltk_data]     /Users/anthonyramelo/nltk_data...
[nltk_data]   Package stopwords is already up-to-date!
[nltk_data] Downloading package wordnet to
[nltk_data]     /Users/anthonyramelo/nltk_data...
[nltk_data]   Package wordnet is already up-to-date!
```

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.

In [2]:
```
# TF-IDF Vectorization
vectorizer = TfidfVectorizer(max_features=5000, ngram_range=(1, 2))
X_train_tfidf = vectorizer.fit_transform(X_train)
X_test_tfidf = vectorizer.transform(X_test)

# SMOTE for balancing
smote = SMOTE(random_state=42)
X_train_resampled, y_train_resampled = smote.fit_resample(X_train_tfidf, y_train)

# ------------------------------------------------------------------------------
# Model 1: XGBoost
xgb_model = XGBClassifier(random_state=42)
xgb_model.fit(X_train_resampled, y_train_resampled)
y_pred_xgb = xgb_model.predict(X_test_tfidf)
y_proba_xgb = xgb_model.predict_proba(X_test_tfidf)[:, 1]

print("XGBoost Metrics:")
print(f"Accuracy: {accuracy_score(y_test, y_pred_xgb):.4f}")
print(f"F1 Score: {f1_score(y_test, y_pred_xgb):.4f}")
print(f"ROC-AUC: {roc_auc_score(y_test, y_proba_xgb):.4f}\n")

# ------------------------------------------------------------------------------
# Model 2: LightGBM
lgbm_model = LGBMClassifier(random_state=42, class_weight='balanced')
lgbm_model.fit(X_train_resampled, y_train_resampled)
y_pred_lgbm = lgbm_model.predict(X_test_tfidf)
y_proba_lgbm = lgbm_model.predict_proba(X_test_tfidf)[:, 1]

print("LightGBM Metrics:")
print(f"Accuracy: {accuracy_score(y_test, y_pred_lgbm):.4f}")
print(f"F1 Score: {f1_score(y_test, y_pred_lgbm):.4f}")
print(f"ROC-AUC: {roc_auc_score(y_test, y_proba_lgbm):.4f}\n")

# ------------------------------------------------------------------------------
# Model 3: Stacked (XGBoost + LightGBM)
stacked_model = StackingClassifier(
    estimators=[('xgb', xgb_model), ('lgbm', lgbm_model)],
    final_estimator=LogisticRegression(),
    n_jobs=-1
)
stacked_model.fit(X_train_resampled, y_train_resampled)
y_pred_stacked = stacked_model.predict(X_test_tfidf)
y_proba_stacked = stacked_model.predict_proba(X_test_tfidf)[:, 1]

print("Stacked Model Metrics:")
print(f"Accuracy: {accuracy_score(y_test, y_pred_stacked):.4f}")
print(f"F1 Score: {f1_score(y_test, y_pred_stacked):.4f}")
print(f"ROC-AUC: {roc_auc_score(y_test, y_proba_stacked):.4f}\n")

# ------------------------------------------------------------------------------
# Model 4: Logistic Regression
lr_model = LogisticRegression(solver='liblinear', random_state=42)
lr_model.fit(X_train_resampled, y_train_resampled)
y_pred_lr = lr_model.predict(X_test_tfidf)
y_proba_lr = lr_model.predict_proba(X_test_tfidf)[:, 1]

print("Logistic Regression Metrics:")
print(f"Accuracy: {accuracy_score(y_test, y_pred_lr):.4f}")
print(f"F1 Score: {f1_score(y_test, y_pred_lr):.4f}")
print(f"ROC-AUC: {roc_auc_score(y_test, y_proba_lr):.4f}\n")
```

```
XGBoost Metrics:
Accuracy: 0.7233
F1 Score: 0.6914
ROC-AUC: 0.8127

[LightGBM] [Info] Number of positive: 1213, number of negative: 1213
[LightGBM] [Info] Auto-choosing row-wise multi-threading, the overhead of testing was 0.003917 seconds.
You can set `force_row_wise=true` to remove the overhead.
And if memory is not enough, you can set `force_col_wise=true`.
[LightGBM] [Info] Total Bins 1691
[LightGBM] [Info] Number of data points in the train set: 2426, number of used features: 103
[LightGBM] [Info] [binary:BoostFromScore]: pavg=0.500000 -> initscore=0.000000
LightGBM Metrics:
Accuracy: 0.6383
F1 Score: 0.6004
ROC-AUC: 0.7314

[LightGBM] [Info] Number of positive: 1213, number of negative: 1213
[LightGBM] [Info] Auto-choosing row-wise multi-threading, the overhead of testing was 0.004819 seconds.
You can set `force_row_wise=true` to remove the overhead.
And if memory is not enough, you can set `force_col_wise=true`.
[LightGBM] [Info] Total Bins 1691
[LightGBM] [Info] Number of data points in the train set: 2426, number of used features: 103
[LightGBM] [Info] [binary:BoostFromScore]: pavg=0.500000 -> initscore=0.000000
[LightGBM] [Info] Number of positive: 970, number of negative: 971
[LightGBM] [Info] Number of positive: 970, number of negative: 970
[LightGBM] [Info] Number of positive: 971, number of negative: 970
[LightGBM] [Info] Number of positive: 971, number of negative: 970
[LightGBM] [Info] Number of positive: 970, number of negative: 971
[LightGBM] [Info] Auto-choosing row-wise multi-threading, the overhead of testing was 0.007236 seconds.
You can set `force_row_wise=true` to remove the overhead.
And if memory is not enough, you can set `force_col_wise=true`.
[LightGBM] [Info] Total Bins 1180
[LightGBM] [Info] Number of data points in the train set: 1941, number of used features: 75
[LightGBM] [Info] [binary:BoostFromScore]: pavg=0.500000 -> initscore=-0.000000
[LightGBM] [Info] Start training from score -0.000000
[LightGBM] [Info] Auto-choosing col-wise multi-threading, the overhead of testing was 0.013729 seconds.
You can set `force_col_wise=true` to remove the overhead.
[LightGBM] [Info] Total Bins 1219
[LightGBM] [Info] Auto-choosing col-wise multi-threading, the overhead of testing was 0.014237 seconds.
You can set `force_col_wise=true` to remove the overhead.
[LightGBM] [Info] Number of data points in the train set: 1941, number of used features: 75
[LightGBM] [Info] Total Bins 1208
[LightGBM] [Info] Number of data points in the train set: 1940, number of used features: 78
[LightGBM] [Info] [binary:BoostFromScore]: pavg=0.500000 -> initscore=0.000000
[LightGBM] [Info] [binary:BoostFromScore]: pavg=0.500000 -> initscore=0.000000
[LightGBM] [Info] Start training from score 0.000000
[LightGBM] [Info] Auto-choosing col-wise multi-threading, the overhead of testing was 0.013480 seconds.
You can set `force_col_wise=true` to remove the overhead.
[LightGBM] [Info] Total Bins 1211
[LightGBM] [Info] Number of data points in the train set: 1941, number of used features: 77
[LightGBM] [Info] Auto-choosing col-wise multi-threading, the overhead of testing was 0.013346 seconds.
You can set `force_col_wise=true` to remove the overhead.
[LightGBM] [Info] Total Bins 1195
[LightGBM] [Info] Number of data points in the train set: 1941, number of used features: 76
[LightGBM] [Info] [binary:BoostFromScore]: pavg=0.500000 -> initscore=0.000000
[LightGBM] [Info] Start training from score 0.000000
[LightGBM] [Info] [binary:BoostFromScore]: pavg=0.500000 -> initscore=-0.000000
[LightGBM] [Info] Start training from score -0.000000
Stacked Model Metrics:
Accuracy: 0.7200
F1 Score: 0.6900
ROC-AUC: 0.8096

Logistic Regression Metrics:
Accuracy: 0.7733
F1 Score: 0.7631
ROC-AUC: 0.8573
```

Part 1.c. Assessing

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

In [3]:
```
print("Logistic Regression Metrics:")
print(f"Accuracy: {accuracy_score(y_test, y_pred_lr):.4f}")
print(f"F1 Score: {f1_score(y_test, y_pred_lr):.4f}")
print(f"ROC-AUC: {roc_auc_score(y_test, y_proba_lr):.4f}\n")
```

```
Logistic Regression Metrics:
Accuracy: 0.7733
F1 Score: 0.7631
ROC-AUC: 0.8573
```

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 point of view, the model's performance is not satisfactory, particularly for scenarios where there is a high cost if the decision of the output is wrong. Although the model achieves an accuracy of 77.33%, this means that around 23% of predictions are incorrect, which is a significant error rate. This inaccuracy could lead to poor decision-making, especially in situations where the sentiment analysis is critical to business success.

The F1-score of 76.31% shows that the model balances precision and recall but has trouble with harder cases and less common sentence structures. This is a concern if accurately identifying nuanced or ambiguous sentiments is important, as the model's performance may not consistently meet business needs.

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 [4]:

```python
import pandas as pd

# Identify incorrect predictions
incorrect_indices = (y_test != y_pred_lr)

# Extract incorrect predictions
incorrect_examples = df_test.loc[incorrect_indices].copy()
incorrect_examples['Predicted_Polarity'] = y_pred_lr[incorrect_indices]
incorrect_examples['Probability'] = y_proba_lr[incorrect_indices]

# Select 5 random incorrect examples for detailed analysis
sample_incorrect = incorrect_examples.sample(5, random_state=42)

# Add true labels to the sample
sample_incorrect['True_Polarity'] = y_test.loc[sample_incorrect.index].values

# Add cleaned sentences for context
sample_incorrect['Cleaned_Sentence'] = sample_incorrect['Sentence'].apply(clean_text_v4)
sample_incorrect
```

Out[4]:

	Sentence	Polarity	Cleaned_Sentence	Predicted_Polarity	Probability	True_Polarity
351	But I thought his acting was skilled.	1	thought acting skilled	0	0.473471	1
240	But it is entertaining, nonetheless.	1	entertaining nonetheless	0	0.471712	1
294	You wont regret it!	1	wont regret	0	0.463258	1
214	Omit watching this.	0	omit watching	1	0.504836	0
572	If you act in such a film, you should be glad ...	0	act film glad your gonna drift away earth far ...	1	0.584955	0

The model made mistakes because it struggles with certain types of sentences. For example, in "But I thought his acting was skilled," the word "but" made the model think the sentence was negative, even though it was positive. In "But it is entertaining, nonetheless," the word "nonetheless" may have confused it. It also had trouble with negatives, like in "You won't regret it!", where it focused on "regret" but didn't recognize the "won't." The model also misunderstood commands, like in "Omit watching this," where it saw "watching" as positive and ignored "omit." Finally, it didn't pick up on sarcasm in "If you act in such a film, you should be glad...", treating it as a positive statement. These mistakes show that the model needs to get better at understanding negatives, context, and sarcasm to make more accurate predictions.

Analysis on each of the sentences:

1. "But I thought his acting was skilled."

   • Words like "but" might dilute the sentiment. Additionally, "skilled" is a positive word, but the phrase "I thought" might weaken its overall weight.

2. "But it is entertaining, nonetheless."

• The word “entertaining” is positive, but “nonetheless” makes the sentence complex. The "But" may have also have some weight to the sentence.

3. “You won’t regret it!”

   • The model did not handle the double negative of "won't" and "regret" well.

4. “Omit watching this.” • The word "watching" being positive and "Omit" being negative. Did have an affect on the classification of the sentence.

5. “If you act in such a film, you should be glad your gonna drift away earth far ...” • The phrase “you should be glad” might have confused the model since this sentence is sarcasm.