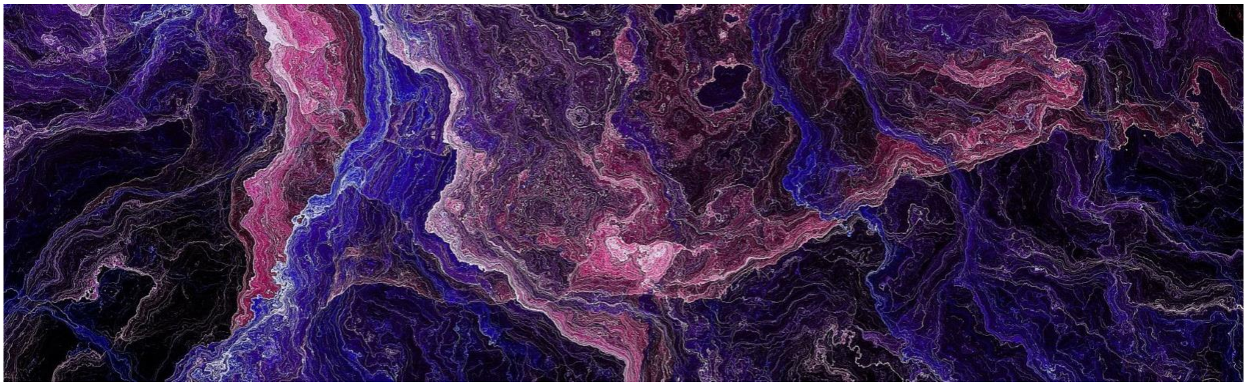

# Milestone 5

# Sentiment Analysis

## Team:


Ahmed Elsayed 21100818

Hana Mohamed 21100863

Ahella Mohamed 21100824

Aysha Mohamed 21100798

Ayten Hossam 21100785


## Summarization:


Our aim was doing sentiment analysis for tweets, we did preprocessing on the data we collected, and we worked on different types of pre-trained models, we created LSTM model, and we used it to classify tweets from our data set and we created GUI tab to classify new random texts

## 1-Methodology:

We collected our data from a Kaggle competition that had dataset that contains Text column that are tweets from twitter, language column that indicates every tweet language and Label column that indicates the sentiment analysis of the text as if it was (Positive, Negative, Uncertainty, Litigious)

Kaggle competition: [Sentiment Dataset with 1 Million Tweets]

Preprocessing of the data:

First we uploaded our data and we read it by pandas library

```
data = pd.read_csv("dataset.csv")
print(data.shape)
```

```
(937854, 3)
```

```
data.dropna(inplace=True)
data['Text'] = data['Text'].fillna('')
data.isna().sum()
```

```
Text        0
Language    0
Label       0
dtype: int64
```

Our first preprocessing step is to make sure there isn't any null values in any of the rows

```
data = data[data['Language'] == 'en']

string_to_remove = 'litigious'
data = data[~data['Label'].str.contains(string_to_remove)]

print(data)
```

```
                                                       Text Language  \
3       Rwanda is set to host the headquarters of Unit...       en
5       It sucks for me since I'm focused on the natur...       en
7       @ShawnTarloff @itsmieu you can also relate thi...       en
8       Social Security. Constant political crises dis...       en
9       @FilmThePoliceLA A broken rib can puncture a l...       en
...                                                   ...      ...
937849            @Juice_Lemons in the dark. it’s so good       en
937850  8.SSR &amp; Disha Salian case should be solved...       en
937851  *ACCIDENT:  Damage Only* - Raleigh Fire Depart...       en
937852  @reblavoie So happy for her! She’s been incred...       en
937853                       I'm lost and I'm found but       en

              Label
3          positive
5          negative
7       uncertainty
8          negative
9          negative
...             ...
937849     positive
937850     negative
937851     negative
937852     positive
937853     negative

[691248 rows x 3 columns]
```

We specified our project on only classifying on the English texts, so we only took the ‘en’ from the column (Language)

Then we removed litigious analysis because we wanted to make sure our data will only have (positive , negative , neutral)

```
stop_words = set(stopwords.words('english'))
lemmatizer = WordNetLemmatizer()
def preprocess_text(Text):
    Text = Text.lower()
    Text = re.sub(r'[^a-zA-Z\s]', '', Text)
    tokens = nltk.word_tokenize(Text)
    tokens = [lemmatizer.lemmatize(word) for word in tokens if word not in stop_words]
    return ' '.join(tokens)
data['Text'] = data['Text'].apply(preprocess_text)
```

```
x= data['Text']
y= data['Label']
```

```
train_texts, test_texts, train_labels, test_labels = train_test_split(x, y, test_size=0.2, random_state=42)
```

Next step we converted all capital letters to small letters by using .lower() function

Then we removed any special characters from the texts

Then we used from the NTLK library word Lemmatization it is a technique used to reduce inflected words to their root word by using stop words : it is a set of commonly used words in a language. Examples of stop words in English are “a,” “the,” “is,” “are,” etc

Then we used word tokenization library: Tokenization breaks text into smaller parts for easier machine analysis, helping machines understand human language

## 2-Results

In this Sentiment analysis project, several different methodologies and pretrained models were employed to analyze text data. Initially, the project utilized a pretrained model like VADER, which yielded moderate validation and test accuracies of approximately 0.53 and 0.54, respectively.

```
import nltk
nltk.download('vader_lexicon')
train_texts, test_texts, train_labels, test_labels = train_test_split(x, y, test_size=0.2, random_state=42)
train_texts, val_texts, train_labels, val_labels = train_test_split(train_texts, train_labels, test_size=0.2, random_state=42)

# Initialize VADER
vader = SentimentIntensityAnalyzer()

# Predict labels for validation set
predicted_labels_val = val_texts.apply(lambda text: 'positive' if vader.polarity_scores(text)['compound'] >= 0.05
                                       else 'negative' if vader.polarity_scores(text)['compound'] <= -0.05
                                       else 'neutral')

# Predict labels for test set
predicted_labels_test = test_texts.apply(lambda text: 'positive' if vader.polarity_scores(text)['compound'] >= 0.05
                                       else 'negative' if vader.polarity_scores(text)['compound'] <= -0.05
                                       else 'neutral')

# Calculate validation accuracy
val_1 = accuracy_score(val_labels, predicted_labels_val)

# Calculate test accuracy
test_1 = accuracy_score(test_labels, predicted_labels_test)

print("Validation Accuracy:", val_1)
print("Test Accuracy:", test_1)
```

```
[nltk_data] Downloading package vader_lexicon to
[nltk_data]     /Users/aytenhossamahmed/nltk_data...
Validation Accuracy: 0.5344060978191827
Test Accuracy: 0.5440379403794038
```

Subsequently, TextBlob was implemented, resulting in a slightly higher accuracy of around 0.56.

```
label_mapping = {'positive': 1, 'negative': 0, 'uncertainty': 2}
data['Label'] = data['Label'].map(label_mapping)

# Split data into training and validation sets
x = data['Text']
y = data['Label']
x_train, x_val, y_train, y_val = train_test_split(x, y, test_size=0.3, random_state=42)

# Analyze sentiment using TextBlob pretrained model
def analyze_sentiment(sentence):
    blob = TextBlob(sentence)
    polarity = blob.sentiment.polarity

    # Classify the sentiment based on the polarity score
    if polarity > 0:
        return 1  # Positive
    elif polarity < 0:
        return 0  # Negative
    else:
        return 2  # Neutral

# Apply sentiment analysis to validation set
y_val_predicted = x_val.apply(analyze_sentiment)

# Calculate accuracy
accuracy = accuracy_score(y_val, y_val_predicted)
print(f"Accuracy: {accuracy}")
```

```
Accuracy: 0.5616531165311653
```

However, the most notable performance improvements were achieved through more sophisticated techniques like logistic regression and SVM. Logistic regression demonstrated remarkable accuracy during training, achieving around 0.98, and maintained a high validation accuracy of approximately 0.96.

```
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score, classification_report
logistic_regression_model = LogisticRegression(max_iter=500)
logistic_regression_model.fit(X_train_tfidf, y_train)

train_preds = logistic_regression_model.predict(X_train_tfidf)
val_preds = logistic_regression_model.predict(X_val_tfidf)

train_accuracy = accuracy_score(y_train, train_preds)
val_accuracy = accuracy_score(y_val, val_preds)

print("Train Accuracy:", train_accuracy)
print("Validation Accuracy:", val_accuracy)
print("\nTrain Classification Report:")
print(classification_report(y_train, train_preds))
print("\nValidation Classification Report:")
print(classification_report(y_val, val_preds))
```

```
Train Accuracy: 0.9759957411798867
Validation Accuracy: 0.962059620596206

Train Classification Report:
              precision    recall  f1-score   support

           0       0.98      0.98      0.98      7390
           1       0.98      0.98      0.98      7411
           2       0.97      0.97      0.97      5862

    accuracy                           0.98     20663
   macro avg       0.98      0.98      0.98     20663
weighted avg       0.98      0.98      0.98     20663
```

Similarly, SVM also showcased impressive results, with a training accuracy exceeding 0.98 and a validation accuracy of around 0.96.

```
from sklearn.svm import SVC
from sklearn.feature_extraction.text import CountVectorizer

count_vectorizer = CountVectorizer(max_features=1000)
X_train_counts = count_vectorizer.fit_transform(x_train)
X_val_counts = count_vectorizer.transform(x_val)

svm_model = SVC(kernel='linear', C=1.0)
svm_model.fit(X_train_counts, y_train)
train_accuracy = svm_model.score(X_train_counts, y_train)
val_accuracy = svm_model.score(X_val_counts, y_val)

print("Training Accuracy:", train_accuracy)
print("Validation Accuracy:", val_accuracy)
y_pred = svm_model.predict(X_val_counts)
print(classification_report(y_val, y_pred))
```

```
Training Accuracy: 0.9868847698785268
Validation Accuracy: 0.9601400180668473
```

Finally, we tried LSTM and gave us an accuracy of 0.97

```
Epoch 1/40
265/265 ━━━━━━━━━━━━━━━━━━━━ 271s 997ms/step - accuracy: 0.8144 - loss: 0.4430 - precision: 0.8573 - recall: 0.7546 - val_accuracy: 0.9411 - val_loss: 0.1569 - val_precision: 0.9469 - val_recall: 0.9354
Epoch 2/40
265/265 ━━━━━━━━━━━━━━━━━━━━ 304s 1s/step - accuracy: 0.9583 - loss: 0.1064 - precision: 0.9643 - recall: 0.9544 - val_accuracy: 0.9641 - val_loss: 0.0856 - val_precision: 0.9677 - val_recall: 0.9611
Epoch 3/40
265/265 ━━━━━━━━━━━━━━━━━━━━ 343s 1s/step - accuracy: 0.9643 - loss: 0.0850 - precision: 0.9711 - recall: 0.9596 - val_accuracy: 0.9659 - val_loss: 0.0856 - val_precision: 0.9724 - val_recall: 0.9589
Epoch 4/40
265/265 ━━━━━━━━━━━━━━━━━━━━ 311s 1s/step - accuracy: 0.9675 - loss: 0.0755 - precision: 0.9732 - recall: 0.9623 - val_accuracy: 0.9663 - val_loss: 0.0856 - val_precision: 0.9710 - val_recall: 0.9602
Epoch 5/40
265/265 ━━━━━━━━━━━━━━━━━━━━ 331s 1s/step - accuracy: 0.9676 - loss: 0.0749 - precision: 0.9730 - recall: 0.9631 - val_accuracy: 0.9656 - val_loss: 0.0807 - val_precision: 0.9699 - val_recall: 0.9624
Epoch 6/40
265/265 ━━━━━━━━━━━━━━━━━━━━ 337s 1s/step - accuracy: 0.9690 - loss: 0.0697 - precision: 0.9742 - recall: 0.9643 - val_accuracy: 0.9661 - val_loss: 0.0848 - val_precision: 0.9680 - val_recall: 0.9643
Epoch 7/40
265/265 ━━━━━━━━━━━━━━━━━━━━ 399s 1s/step - accuracy: 0.9685 - loss: 0.0718 - precision: 0.9736 - recall: 0.9632 - val_accuracy: 0.9650 - val_loss: 0.0871 - val_precision: 0.9703 - val_recall: 0.9613
Epoch 8/40
265/265 ━━━━━━━━━━━━━━━━━━━━ 375s 1s/step - accuracy: 0.9699 - loss: 0.0651 - precision: 0.9745 - recall: 0.9657 - val_accuracy: 0.9664 - val_loss: 0.0857 - val_precision: 0.9682 - val_recall: 0.9628
Epoch 9/40
265/265 ━━━━━━━━━━━━━━━━━━━━ 398s 2s/step - accuracy: 0.9739 - loss: 0.0611 - precision: 0.9771 - recall: 0.9703 - val_accuracy: 0.9638 - val_loss: 0.1020 - val_precision: 0.9653 - val_recall: 0.9624
Epoch 10/40
265/265 ━━━━━━━━━━━━━━━━━━━━ 351s 1s/step - accuracy: 0.9744 - loss: 0.0581 - precision: 0.9763 - recall: 0.9722 - val_accuracy: 0.9622 - val_loss: 0.1056 - val_precision: 0.9647 - val_recall: 0.9606
```

## 3-Conclusion:

In the process of sentiment analysis, we used different techniques like VADER, Text Blob, logistic regression, and (SVM) Support Vector Machines

By using these models, we found out

## Logistic Regression:

It is fast and with a high accuracy but dealing with a big dataset

```
[20]
    val_accuracy = accuracy_score(y_val, val_preds)

    print("Train Accuracy:", train_accuracy)
    print("Validation Accuracy:", val_accuracy)
    print("\nTrain Classification Report:")
    print(classification_report(y_train, train_preds))
    print("\nValidation Classification Report:")
    print(classification_report(y_val, val_preds))

    Train Accuracy: 0.9759957411798867
    Validation Accuracy: 0.962059620596206

    Train Classification Report:
                  precision    recall  f1-score   support

               0       0.98      0.98      0.98      7390
               1       0.98      0.98      0.98      7411
               2       0.97      0.97      0.97      5862

        accuracy                           0.98     20663
       macro avg       0.98      0.98      0.98     20663
    weighted avg       0.98      0.98      0.98     20663


    Validation Classification Report:
                  precision    recall  f1-score   support

               0       0.96      0.97      0.96      3113
               1       0.97      0.96      0.97      3279
               2       0.95      0.95      0.95      2464

        accuracy                           0.96      8856
       macro avg       0.96      0.96      0.96      8856
    weighted avg       0.96      0.96      0.96      8856
```

**(SVM) Support Vector Machines:**

It has the highest accuracy, but it takes too much time to run dealing with big dataset

```
from sklearn.svm import SVC
from sklearn.feature_extraction.text import CountVectorizer

count_vectorizer = CountVectorizer(max_features=1000)
X_train_counts = count_vectorizer.fit_transform(x_train)
X_val_counts = count_vectorizer.transform(x_val)

svm_model = SVC(kernel='linear', C=1.0)
svm_model.fit(X_train_counts, y_train)
train_accuracy = svm_model.score(X_train_counts, y_train)
val_accuracy = svm_model.score(X_val_counts, y_val)

print("Training Accuracy:", train_accuracy)
print("Validation Accuracy:", val_accuracy)
y_pred = svm_model.predict(X_val_counts)
print(classification_report(y_val, y_pred))

Training Accuracy: 0.9868847698785268
Validation Accuracy: 0.9601400180668473
              precision    recall  f1-score   support

           0       0.95      0.97      0.96      3113
           1       0.97      0.96      0.97      3279
           2       0.95      0.95      0.95      2464

    accuracy                           0.96      8856
   macro avg       0.96      0.96      0.96      8856
weighted avg       0.96      0.96      0.96      8856
```

## Vader:

It takes time but with right classification and with accuracy =0.53

```
print("Validation Accuracy:", val_1)
print("Test Accuracy:", test_1)
```

```
[nltk_data] Downloading package vader_lexicon to
[nltk_data]     /Users/aytenhossamahmed/nltk_data...
Validation Accuracy: 0.5344060978191827
Test Accuracy: 0.5440379403794038
```

## Text blob:

It is the same as Vader it takes time but with right classification and with accuracy=0.56

```
# Apply sentiment analysis to validation set
y_val_predicted = x_val.apply(analyze_sentiment)

# Calculate accuracy
accuracy = accuracy_score(y_val, y_val_predicted)
print(f"Accuracy: {accuracy}")
```

```
Accuracy: 0.5616531165311653
```

## LSTM:

It has bad classification and bad accuracy dealing with small data set

```
The training loss is : 1.36

The training accuracy is : 32.40%

The training precision is : 0.46

The training recall is : 0.01

The F1 score of the training set is : 0.0108
```

While it is better dealing with big dataset, but the epochs take time

```
print(f'The F1 score of the validation set is : {lstm_val_f1_score:0.4f}\n')

The training loss is : 0.07

The training accuracy is : 97.07%

The training precision is : 0.97

The training recall is : 0.97

The F1 score of the training set is : 0.9708

The validation loss is : 0.08

The validation accuracy is : 96.61%

The validation precision is : 0.97

The validation recall is : 0.96

The F1 score of the validation set is : 0.9664
```

Finally ,

**SVM:** can be an effective approach for sentiment analysis, especially when combined with appropriate feature extraction techniques.

**VADER:** can provide a quick and simple solution for sentiment analysis tasks.

**Text Blob:** can be a useful tool for basic sentiment analysis tasks with its easy-to-use interface and pretrained models.

**Logistic regression**: is a popular choice for sentiment analysis due to its simplicity and interpretability.

## 4-Future work

**To enhance our pretrained model and LSTM here are some future work suggestions:**

- **Word Embeddings:** Utilize more advanced word embedding techniques to represent words in the input text. Pretrained word embeddings such as Word2Vec, GloVe, or BERT embeddings can capture semantic relationships and improve the model's understanding of word meanings. You can initialize the LSTM model with these embeddings or even fine-tune them during the training process.

- **Transfer Learning with Language Models:** Consider leveraging large-scale language models like BERT, GPT, or XLNet as a starting point for training your sentiment analysis model. These models are pretrained on vast amounts of text data and can provide valuable contextualized representations of words. Fine-tuning these language models on sentiment analysis tasks can help improve the model's performance by leveraging their language understanding capabilities.

- **Regularization Techniques**: Apply regularization techniques to prevent overfitting and improve the generalization ability of the LSTM model. Techniques like dropout, L1 or L2 regularization, or early stopping can help reduce overfitting and improve the model's ability to generalize well on unseen data.

- **Data Augmentation:** Augment your training data by generating synthetic samples with slight variations or perturbations. This can help increase the diversity of the training data and improve the model's ability to handle different sentence structures, word orders, or sentiment expressions.

- **Handling Imbalanced Data:** If your sentiment analysis dataset is imbalanced, where one sentiment class has significantly more samples than the others, apply techniques to address this issue. You can use oversampling, undersampling, or class weighting techniques to balance the data distribution and prevent the model from being biased towards the majority class.

- **Error Analysis:** Perform a detailed analysis of the model's errors to identify common misclassifications and areas for improvement. Analyze cases where the model fails to predict sentiment accurately and examine the patterns or specific linguistic structures that pose challenges. This analysis can guide you in refining the model, identifying data biases, or incorporating additional features to address the identified limitations

- **Multimodal Sentiment Analysis:** Explore the integration of other modalities, such as images, audio, or video, along with text for sentiment analysis. Multimodal sentiment analysis can provide a more comprehensive understanding of sentiment by considering visual or auditory cues in addition to textual content. This can be particularly useful in social media analysis or analyzing sentiment in multimedia content.

- **Model Stacking:** Implement model stacking, which involves training multiple models and using their predictions as input features for another model. In this approach, you can train multiple sentiment analysis models with different architectures or algorithms. Then, you can use their predictions as input features to train a meta-model that learns to combine the outputs of the base models. Model stacking can help capture diverse patterns and improve the overall predictive power of your sentiment analysis system.

- **Fine-tuning:** Fine-tune your pretrained models using domain-specific data. If your sentiment analysis task is focused on specific domains or industries, fine-tuning the models with data from those domains can help improve their performance. This process involves training the models on the target domain data while leveraging the pretrained weights to speed up convergence and benefit from the pretrained knowledge.

- **Continuous Model Monitoring and Updating:** Deploy your sentiment analysis model in a production environment and continuously monitor its performance. Collect feedback from users or domain experts and use it to update and fine-tune your model iteratively. Sentiment analysis trends and language usage can change over time, so it's important to keep your model up to date with the latest data and adapt it to evolving sentiments and language patterns.