CS6320: Final Project Report

`https://github.com/COftedahl/cs6320_group_project`

Brendan Martel BXM240013
Cole Oftedahl CXO220001
Brad Stover BES170230
Zhaotong Zhang ZXZ220016

1 Introduction

Sentiment analysis aspires to determine the emotional polarity expressed in natural language and is typically categorized into three classes: positive, negative, and neutral. Although this task has continuously been widely explored, social media platforms–especially Twitter–bring up unique challenges. The language in social media is very informal, with creative spelling and punctuation, misspellings, slang, new words, URLs, and genre-specific terminology and abbreviations, e.g., RT for re-tweet and #hashtags1[1]. In order to train and evaluate our project, we combine several publicly available tweet datasets, including Sentiment140 and two additional tweet corpora spanning multiple years and topics. Gathering these datasets did yield an extensive and heterogeneous collection suitable for this experiment.

With the growing availability of large-scale Tweet datasets spanning several years and different topics, it has become increasingly critical to develop sentiment classification systems. These systems not only perform well on small, homogeneous datasets; they also scale to enormous, heterogeneous corpora. Modern transformer-based language models, such as BERT and RoBERTa, have demonstrated powerful performance in many NLP tasks due to their contextual understanding. Classical lexicon-based tools, such as VADER, remain widely used for real-time or resource-constrained sentiment inference. In recent years, large-language models (LLMs), such as LLaMA, have made it feasible to fine-tune powerful generative architectures on domain-specific tasks, enabling improvements when working with massive datasets.

Using the combined Tweet datasets from several sources, we train and evaluate several models on consistent training, validation, and testing splits. Our results show that transformer models outperform the classical baseline, with Sentiment-BERT exhibiting the strongest performance. The transformer models demonstrate an advantage in capturing subtle sentiment features in large and diverse Tweet corpora. These findings emphasize the importance of both dataset scale and model architecture/capacity in Twitter sentiment classification.

2 Data

In this project, we formed a sentiment classification system using a massive combined Twitter corpus derived from multiple public datasets. We parsed and normalized data from Sentiment140, Twitter Tweets Sentiment Dataset, Sentiment Analysis of Tweets dataset, and additional labeled Tweet collections. After doing this, we gathered and unified all entries into a consistent three-class sentiment format consisting of positive, neutral, and negative sentiments. Then we applied our preprocessing pipeline, which cleaned noise, removed URLs, and standardized tokens. In the meantime, we split the resulting corpus into three equal partitions for training, validation, and testing.

Dataset Split	**Number of Examples**
Training Set	1,329,985
Validation Set	168,748
Test Set	168,748

Table 1: Number of examples in the training, validation, and test sets.

3 Methodology

3.1 Data Sources and Integration

To create a general-purpose sentiment classification corpus, we aggregated three publicly available tweet-based datasets:

- **Sentiment_Analysis.csv**: A collection of labeled English posts annotated for positive, negative, or neutral sentiment.
- **Sentiment140.csv**: A large-scale Twitter dataset containing polarity-annotated tweets collected across multiple years.
- **Tweets.csv**: A mixed-source Twitter dataset containing short and informal posts covering diverse subjects.

Even though these datasets originate from different time periods and subject domains, their sentiment labels were standardized to a unified three-class scheme:

- **Positive** – 2
- **Neutral** – 1
- **Negative** – 0

This merging procedure allows us to form a wide, diverse, and domain-agnostic sentiment corpus suitable for evaluating the robustness of modern language models.

3.2 Data Preprocessing and Normalization

Before building the final training, validation, and testing splits, all raw data files underwent a unified preprocessing stage:

- **URL and username removal:** Links such as `http://...` and user handles like `@name` were stripped using the `cleanLine` function; @'s were replaced by a token.
- **Consistent text formatting:** All text was converted to lowercase and cleaned from excessive whitespace or punctuation.
- **Character filtering:** Non-English characters and redundant punctuation marks were removed or collapsed.
- **Label normalization:** Dataset-specific labels (e.g., -1, 0, 1) were normalized to the range 0–2 through a constant shift.

After preprocessing, each dataset was written in the format:

$$\{\text{sentence}, \text{label}\}.$$

The `splitData()` routine was then executed to construct the final splits:

- **80% Training set:** `train.csv`
- **10% Validation set:** `val.csv`
- **10% Test set:** `test.csv`

This split guaranteed that all models were trained and evaluated on the same unified distribution.

3.3 Model Architecture and Training Procedure

The project primarily experimented with transformer-based architectures due to their state-of-the-art performance in text classification. The implemented and planned models include:

- **RoBERTa Twitter:** `cardiffnlp/twitter-roberta-base-sentiment-latest`
- **BERT-based sentiment models** from HuggingFace
- **VADER:** a lexicon-based baseline

Training Pipeline

All transformer models are fine-tuned using HuggingFace's `Trainer` with the following defaults:

- **Loss:** Cross-entropy
- **Optimizer:** AdamW
- **Batch size:** 8
- **Epochs:** 2–3
- **Evaluation metric:** Confusion matrix using `evaluate.load("confusion_matrix")`
- **Tokenization:** `AutoTokenizer` with truncation to 256 tokens

The same training/validation/testing splits were used for all models in order to ensure a fair comparison across different architectures.

VADER, as a non-transformer, was used as a baseline against which to evaluate the transformer models. In this study, VADER was not trained, but it was used to provide baseline classification scores by being run on a concatenation of all the datasets and evaluating its performance.

4 Implementations

The transformer models were implemented using the HuggingFace transformers library in Python. Since multiple different models were evaluated, abstractions were used to allow simple reuse by passing a different parameter to indicate which model to use.

To this end, the LanguageModel class was developed, incorporating a constructor method as well as training and testing methods. Development of this class was largely based on the documentation available from HuggingFace, and thus the class closely follows example code from the provided tutorials.

```
  def test(self, data):
    return self.pipe(data)
```

Listing 1: LanguageModel test() method

The constructor method simply accepts the model name as a parameter and then initializes variables to store the necessary model details used for training and testing. The test method is very simple, composed of only one line, listed above. The data parameter is the prompt being evaluated, and the simple interface provided by the transformers library allows calling pipe() with the data to compute the result using the loaded language model.

```
def train(self, trainData,
  trainingArgs
):
  tokenizedTrainData = trainData.map(
    tokenizeDataset, batched=True)
  metric = evaluate.load(
    "confusion_matrix")
  trainer = Trainer(
    model=self.model,
    args=trainingArgs,
    train_dataset=tokenizedTrainData,
    eval_dataset=tokenizedTrainData,
    compute_metrics=compute_metrics,
  )
  trainer.train()
```

Listing 2: LanguageModel train() method

The train method is more complex, requiring setup to utilize the provided Trainer interface from the transformers library. First, the method accepts the training dataset ”trainData”, and then accepts ”trainingArgs” to define parameters for the training. To implement the training, the Trainer class is used, which requires the training data to be tokenized and also requires providing compute metrics, which allow the model to accurately compute the loss for each pass.

```
  def tokenizeDataset(dataset):
    return self.tokenizer(
      dataset[TEXT_COLUMN_NAME],
      padding="max_length",
      truncation=True
    )
  tokenizedTrainData = trainData.map(
    tokenizeDataset, batched=True
  )
```

Listing 3: Tokenizing data in train() method

The above listing details how the datasets are tokenized. This method is called to map ”trainData” to ”tokenizedTrainData”, which ensures each input sent to the language model is of the proper size by using padding and truncation.

```
  metric = evaluate.load(
    "confusion_matrix")

  def compute_metrics(eval_pred):
    logits, labels = eval_pred
    # convert the logits to
    # their predicted class
    predictions = np.argmax(logits,
      axis=-1)
    return metric.compute(
      predictions=predictions,
      references=labels
    )
```

Listing 4: Defining training metrics in train() method

Finally, the metrics must be provided to the Trainer to allow it to evaluate its performance during training. Using the confusion matrix approach, the predictions are checked against the reference labels evaluate the performance of the model in the training cycle.

The other portion of code written specifically for language processing tasks was the use of VADER. For this, the Python package vaderSentiment was used, specifically utilizing the SentimentIntensityAnalyzer for computing a baseline sentiment prediction for the datasets. Much of this code is also based on the documentation of the vaderSentiment package.

```
analyzer = SentimentIntensityAnalyzer()
predictions = []
for sentence in tqdm(trainingData[
    TEXT_COLUMN_NAME]):
  vs = analyzer.polarity_scores(
      sentence)
  label = getOverallSentiment(vs)
  predictions.append(label)
metrics = computeMetrics(
    predictions, trainingData)
```

Listing 5: Main VADER Evaluation Loop

VADER provides most functionality already, with each call of the "polarity_scores" method returning the probability of the given input belonging to each of the three possible classifications, along with an aggregration of those probabilities in a field called "compound". Thus, the program simply calls this method on each input sentence to get the classification, storing the results in an array to later be passed to a helper method for computing metrics of the model. However, since the "polarity_scores" method returns an object with multiple classification scores, the method "getOverallSentiment" was developed to classify the sentiment using the returned compound score.

```
THRESHOLD_VALUE = 0.5
POSITIVE_THRESHOLD = THRESHOLD_VALUE
NEGATIVE_THRESHOLD = -THRESHOLD_VALUE
def getOverallSentiment(sentiment):
  if sentiment['compound'] >=
      POSITIVE_THRESHOLD:
    return SENTIMENT_RESULT_ENUM.
        POSITIVE
  elif sentiment['compound'] <=
      NEGATIVE_THRESHOLD:
    return SENTIMENT_RESULT_ENUM.
        NEGATIVE
  else:
    return SENTIMENT_RESULT_ENUM.
        NEUTRAL
```

Listing 6: VADER Sentiment Classification

The above listing details the thresholds used for classifications, which are based on the thresholds used in the documentation. Based on the compound score, a negative score below the threshold is classified as negative sentiment, a positive score above the threshold as positive sentiment, and any other score as neutral.

5 Experiments and Results

5.1 Overall Model Performance

Table 2 summarizes the accuracy, macro-averaged precision, recall, and F1 scores for all systems. The numerical values are imported from `table.tex`.

VADER performed the worst of all the models, which was expected, as it does not follow the transformer architecture and also because VADER is not trained on the Twitter data before testing (like the transformer models are). Sentiment-BERT achieved the best overall results with 77.12% accuracy and 0.7737 macro F1, outperforming both BERT Multilingual and Twitter-RoBERTa. This aligns with expectations: Sentiment-BERT is explicitly trained for sentiment classification, whereas BERT Multilingual supports broader multilingual tasks, and Twitter-RoBERTa is optimized for tweet semantics but not solely sentiment.

5.2 Class-Level Performance

Across every model, the neutral class consistently obtained the highest recall but the lowest precision. It reflects the dataset's inherent ambiguity and the linguistic overlap between mild positive, mild negative, and neutral tweets.

BERT Multilingual

- Positive precision: 0.8599 (strongest)
- Negative recall: 0.6703 (weakest)
- Frequently mislabels mild sentiment as Neutral
- Confusion matrix reveals strong spillover from Positive $\rightarrow$ Neutral with 60 instances

Sentiment-BERT

- Highest precision, recall, and F1 across all classes
- Best detection of Neutral sentiment with Recall is 0.8432
- More robust separation between Positive and Negative
- Lower misclassification rate overall as shown in its confusion matrix

Table 2: Model Performance on Test Set

Model	Acc.	Prec.	Rec.	F1
BERT Multilingual	0.7232	0.7529	0.7166	0.7252
Sentiment-BERT	0.7712	0.7974	0.7648	0.7737
Twitter-RoBERTa	0.7568	0.7728	0.7540	0.7604
VADER	0.6746	0.0244	0.3483	0.0455

Twitter-RoBERTa

- Strong Positive Precision - 0.84
- Strong Negative Precision - 0.80
- More balanced performance than BERT Multilingual
- Slightly inferior to Sentiment-BERT on Neutral recall
- Tends to confuse borderline Negative/Neutral cases

These patterns imply that pretraining domain specificity–including Sentiment datasets vs. general-purpose vs. Twitter corpora–has a measurable impact on classification behaviour.

5.3 Confusion Matrix Analysis

BERT Multilingual

- 54 cases: Neutral mispredictions for Negative
- 60 cases: Neutral mispredictions for Positive
- Strong preference to classify ambiguous emotions as Neutral
- Implies conservative sentiment boundaries and difficulty with informal tone

Sentiment-BERT

- Cleanest separation among classes
- Very low cross-polarity confusion, such as 6 cases: Positive $\rightarrow$ Negative
- Strong Neutral recall with 199 cases correct, indicating better understanding of ambiguous expressions

Twitter-RoBERTa

- Balanced overall but less stable than Sentiment-BERT
- Higher Negative $\leftrightarrow$ Neutral swaps: 49 cases and 26 cases
- Robust Positive sentiment identification, with 156 cases correct

5.4 Error Analysis

Common sources of error across all models include:

- **Sarcasm and irony**
  "yeah amazing job..." — actually negative, but predicted as neutral. Models lack explicit sarcasm-handling ability.
- **Implicit sentiment**
  "wish the app worked better" — expresses negative intent without explicit emotional keywords.
- **Mixed sentiment**
  "love the screen, hate the battery life"
- **Short or context-poor tweets**
  Single-token posts such as "great", "wow", "ok", where polarity depends heavily on external context.
- **Lexical ambiguity**
  Words like *fine*, *ok*, and *sick* vary in meaning by speaker and domain.

These patterns match prior findings in Twitter sentiment modeling literature and align with the analysis trends in Section 7 of the cited ACL 2022 paper.

5.5 Comparison Across Models

A consistent ranking emerges across all metrics:

- **Sentiment-BERT (best)**
- **Twitter-RoBERTa**
- **BERT Multilingual (worst)**

Interpretation:

- **Sentiment-BERT** benefits from pretraining directly on sentiment corpora, leading to superior generalization.
- **Twitter-RoBERTa** benefits from social media text modeling but lacks explicit polarity supervision.
- **BERT Multilingual** is trained on multilingual data, and its sentiment head is less aligned with informal, sarcastic, or domain-specific text.

This demonstrates that **pretraining domain and task alignment are essential** for effective sentiment classification.

5.6 Summary of Findings

- **Sentiment-BERT** provides the most accurate and stable performance, achieving **0.7712 accuracy** and **0.7737 macro F1**.
- **Twitter-RoBERTa** performs competitively but struggles with ambiguity around the Neutral boundary.
- **BERT Multilingual** displays the weakest polarity distinctions.
- The **Neutral class remains the primary source of error** across all models.
- **Dataset heterogeneity** increases robustness yet simultaneously introduces more ambiguity.
- Error patterns such as *sarcasm, implicit sentiment, and mixed polarity* align with challenges documented in prior research.

6 Conclusion

Sentiment analysis can be performed to a high degree of accuracy using modern technologies such as large-scale transformers trained to be attuned to the specific task, while older or non-trained styles of language models fail to achieve the same accuracy in their performance. However, even when using these modern models, difficulties present in the nuance of natural language–particularly in a realm of less grammatically-structured text (as in social media)–can cause inaccuracies in sentiment classification. Additionally, further work may need to be done to improve the ability of models to identify neutral sentiment, which posed the greatest challenge.

For most sentiment classification tasks, transformer models such as BERT and RoBERTa will achieve sufficient accuracy given they are trained on a sufficiently large and quality dataset. While other model architectures are usable, particularly in resource-constrained contexts, transformer models remain the standard in sentiment analysis performance.

7 Bibliography

[1] S. Rosenthal, P. Nakov, A. Ritter, and V. Stoyanov, “SemEval-2014 Task 9: Sentiment Analysis in Twitter,” 2014.

[2]A. Das et al., “Automatic Error Analysis for Document-level Information Extraction,” vol. 1, pp. 3960–3975, 2022, Accessed: Nov. 29, 2025. [Online]. Available: https://aclanthology.org/2022.acl-long.274.pdf