918 Report

u2151813

I. The Dataset

We had shown in the jypyter notebook the total number of tweets for each set and their distribution. One thing to be aware of is that there is a data imbalance problem: there are far more examples for the neutral and positive classes compared to the negative class.

This class imbalance is also reflected in the results of the classifiers. For all the models we developed, we observed that the scores for classes other than negative are generally better than those for the negative class. This indicates that the model is not learning the negative class well and is biased towards the other classes.

With the classical models, we tried several methods to address this data imbalance, including using balanced class weights, employing the Synthetic Minority Over-sampling Technique (SMOTE), and standard up-sampling. However, these methods provided only limited improvements. We found that the best performing approach was the use of balanced class weights, so we applied it to the SVM model, as this method is already implemented in the scikit-learn library.

II. Data Preprocessing

The data consists of TSV files with the following format:

```
tweet-id <tab> sentiment <tab> tweet-text
```

For preprocessing, the following steps were performed:

- **Remove URLs:** A regular expression is applied to remove URLs from tweets.
- **Remove User Mentions:** Tweets are cleansed of tokens starting with '@'.
- **Remove Hashtags:** All hashtags (tokens beginning with '#') are removed.
- **Remove Non-Alphanumeric Characters:** Except for spaces, non-alphanumeric characters are removed.
- **Remove Full-Digit Numbers and Short Words:** Tokens that consist only of digits or have only one character are removed.
- **Remove Stopwords and Surrounding Spaces:** Standard stopwords are removed and extraneous spaces are stripped.
- **Lemmatization:** Part-of-speech tagging is applied using NLTK, and tokens are lemmatized with their corresponding WordNet POS tags.
- **Tokenization:** Finally, tweets are tokenized using the BERT tokenizer (or NLTK) as needed.

These funtions are written in the same way as in CourseWork 1. The ordering of the functions is also inspired by CourseWork 1. We further added the code to remove stopwords and tokenizations using the nltk tokenizer.

We found that after cleanning, there can be while spaces at the beginning and the end of the tweet, we used the strip() function to remove these spaces.

Aslo, we apply lower case to all the tweets when we load the data into dataframes.

This data preprocessing pipeline is applied to all the datasets, including the training, development, and test sets. And it is applyed to all the models we developed, except the large language models.

III. Classical Machine Learning

A. Bag of Words (BoW)

1) Model Building: With bag-of-words, we used a variant of the feature representation, which is TF-IDF. The difference between TF-IDF and the traditional bag-of-words model is that TF-IDF is a weighted version. For the SVM, we used stochastic gradient descent (SGD) with hinge loss, and for Naive Bayes, we used Multinomial Naive Bayes. When SGD is used with hinge loss, it is equivalent to a linear SVM, as can be seen from their equations.

When using SGD, it is much faster than the traditional SVM because it updates the weights using a single sample, whereas SVM updates weights using the entire dataset.

For Naive Bayes, we used Multinomial Naive Bayes, which is more suitable for discrete data like the TF-IDF features.

These implementations are available from the scikit-learn library. We used TfidfVectorizer to extract the features, and SGDClassifier and MultinomialNB for the classifiers.

We then employed RandomizedSearchCV to tune the hyper-parameters for both the classifiers and the vectorizer.

The best model parameters are presented in the notebook.

2) Evaluation: We report the performance of the models in Table VII. We also show the confusion matrix for each model in Tables I, II, and III for the bag-of-words and SVM combination, and in Tables IV, V, and VI for the bag-of-words and Naive Bayes combination.

	positive	**negative**	**neutral**
positive	0.751	0.049	0.200
negative	0.170	0.623	0.207
neutral	0.255	0.122	0.623

TABLE I
CONFUSION MATRIX FOR BOW-SVM ON TWITTER-TEST1.

In Table VIII, we show the top five important words per class for the SVM model. This is also illustrated in Figure 1.

	positive	**negative**	**neutral**
positive	0.773	0.037	0.189
negative	0.228	0.574	0.198
neutral	0.344	0.071	0.585

TABLE II
CONFUSION MATRIX FOR BOW-SVM ON TWITTER-TEST2.

	positive	**negative**	**neutral**
positive	0.738	0.065	0.197
negative	0.222	0.536	0.243
neutral	0.306	0.105	0.589

TABLE III
CONFUSION MATRIX FOR BOW-SVM ON TWITTER-TEST3.

	positive	**negative**	**neutral**
positive	0.634	0.070	0.295
negative	0.196	0.580	0.224
neutral	0.260	0.149	0.591

TABLE IV
CONFUSION MATRIX FOR NB WITH BOW ON TWITTER-TEST1.

	positive	**negative**	**neutral**
positive	0.700	0.053	0.247
negative	0.180	0.571	0.248
neutral	0.340	0.104	0.556

TABLE V
CONFUSION MATRIX FOR NB WITH BOW ON TWITTER-TEST2.

For the **negative** class, the words are commonly curse words, such as *fuck*, *bad*, and *shit*. For the **positive** class, the words align with common sense; they mainly include words like *good*, *love*, *best*, and *happy*.

For the **neutral** class, the pattern is generally unclear. We observe brand names like *gucci* and verbs like *say*. These words may be used broadly in general topics and typically do not carry strong emotional sentiment.

From Figure 1, we observe that the SVM is somewhat uncertain about words such as *saturday*, and slightly unsure about words like *cant* and *know*. We attempted to replace these words with more general terms; however, we did not observe a clear improvement in the results. The model still appeared heavily influenced by the replaced words, which may be due to the limited vocabulary.

In Table IX, we present the top five important words per class for the Naive Bayes model. Since we used features from both unigrams and bigrams, some of the features consist of two words. Compared with the SVM, we see that for the **negative** class, all the features clearly express negative sentiment. For the **positive** class, the top feature *lay sun* seems a bit unusual, but the remaining ones mostly make sense. Interestingly, we observe that *fuck love* is classified as positive. This suggests that the model can recognize that the word *fuck* is used more as an intensifier (e.g., ”big love”) rather than solely as a curse word.

For the **neutral** class, the identified words generally do not express any emotional sentiment, which is consistent with the behavior observed in the SVM model.

B. GloVe Embeddings

In this work, we develop a classical sentiment analysis pipeline that leverages pre-trained GloVe embeddings in combination with traditional machine learning classifiers. The goal is to classify tweets into three sentiment categories: *negative*, *neutral*, and *positive*. Our approach consists of several key steps, as described below.

1) GloVe Embedding Extraction: We use GloVe embeddings for the tokens in the tweets, specifically the 100-dimensional version of the GloVe embeddings.

We explored two methods for forming feature representations. The first approach is to compute the mean of the embeddings of the tokens in each tweet. The second approach involves stacking the embeddings of the tokens in a tweet to form a 1D vector.

We found that 95% of the processed tweets have a length of 16 tokens. Therefore, we set 16 as the maximum length and applied padding to tweets shorter than 16 tokens and truncation to those longer than 16.

2) Modeling and Hyperparameter Tuning: We used the same models from scikit-learn as in the bag-of-words approach for both SVM and Naive Bayes. The same hyperparameter tuning method, RandomizedSearchCV, was also applied.

However, in this pipeline, we did not use the TF-IDF vectorizer, since we are using GloVe embeddings for feature representation.

The best model parameters are presented in the notebook.

3) Evaluation: We present the performance of the models using GloVe embeddings in Table X.

Additionally, we show the confusion matrices of the models in Tables XI, XII, and XIII for the SVM model, and in Tables XIV, XV, and XVI for the Naive Bayes model.

IV. LSTM-BASED SENTIMENT ANALYSIS WITH GLOVE EMBEDDINGS

In this work, we develop a deep learning approach for Twitter sentiment classification using a bidirectional LSTM network initialized with pre-trained GloVe embeddings.

Our methodology is organized into several stages: data preprocessing, embedding matrix creation, vocabulary and token conversion, model definition, training, and evaluation.

	positive	**negative**	**neutral**
positive	0.649	0.074	0.277
negative	0.242	0.474	0.284
neutral	0.302	0.137	0.561

TABLE VI
CONFUSION MATRIX FOR NB WITH BOW ON TWITTER-TEST3.

Classifier	**Test1**	**Test2**	**Test3**
BOW-SVM	0.612	0.643	0.572
BOW-NB	0.548	0.586	0.522

TABLE VII
OVERALL F1 SCORES FOR BOW-SVM AND BOW-NB CLASSIFIERS ACROSS THREE TEST DATASETS.

A. Data Preprocessing

The same data preprocessing steps are applied as in the SVM and Naive Bayes approaches.

We then extract the 100-dimensional GloVe embeddings for the tokens in the tweets using the same process. One key difference here is that we do not use the mean of the embeddings; instead, we pass the original 2D embeddings directly into the LSTM model.

Since we are limited to a vocabulary size of 5000, we selected the top 5000 most frequent words from the training data, and then created the embedding matrix by assigning each word its corresponding GloVe vector.

It also shown that 95% of the processed tweets have a length of 16 tokens. Therefore, we set 16 as the maximum length and applied padding to tweets shorter than 16 tokens and truncation to those longer than 16.

B. Model Architecture and Training

For the LSTM model, we experimented with different hyperparameters, including the hidden dimension size (128, 256, 512), the number of LSTM layers (2 or 3), and the use of bidirectional LSTMs.

The best-performing configuration was found to be a three-layer bidirectional LSTM with a hidden dimension of 128. The model was trained using cross-entropy loss and the Adam optimizer over 15 epochs.

Due to the limited amount of data and the relatively simple model architecture, validation accuracy for many configurations began to improve after around 10 epochs. Therefore, we trained the final model for 15 epochs to allow additional learning without overfitting.

C. Evaluation and Results

Due to shuffling of the training set during training, we obtained the best model by chance; however, we unfortunately lost the model because we forgot to save it. As a result of the shuffling and time constraints, we were unable to reproduce the exact same model.

Nevertheless, we did save the evaluation results, which we refer to as the *theoretical best performance* of the model. Below, we list the theoretical best F1 scores:

- Test set `test1`: 0.5807
- Test set `test2`: 0.5993
- Test set `test3`: 0.5618

We present this best performance in Table X, and the corresponding confusion matrices for each test set are shown in Tables XVII, XVIII, and XIX.

negative	**neutral**	**positive**
dont	say	love
fuck	saturday	good
cant	via	best
bad	gucci	happy
shit	know	great

TABLE VIII
FEATURES WITH TOP EFFECT WITH SVM

negative	**neutral**	**positive**
shit tomorrow	distich	lay sun
barbaric	sox tie	nice see
spread lie	muhammadu	fun night
make sick	muhammadu buhari	april live
rhetoric	mull	fuck love

TABLE IX
FEATURES WITH TOP EFFECT WITH NAIVE BAYES

V. SENTIMENT ANALYSIS USING A PRE-TRAINED TRANSFORMER

A. roBERTa

We use the pre-trained model `cardiffnlp/twitter-roberta-base-sentiment-latest` from Hugging Face for this work.

The model, based on a transformer architecture, has been pre-trained on over 124 million tweets and outputs probabilities over three sentiment classes: *positive*, *neutral*, and *negative*.

1) Methodology: We downloaded and performed inference using the model with Hugging Face's `transformers` library.

We input raw tweets directly into the model, as this approach was shown to yield the best performance. When we attempted to apply preprocessing techniques similar to those used in our other models, the performance actually degraded. This is understandable, as the model was trained on raw tweets, where usernames were anonymized by replacing them with `@USER` to protect user privacy.

Interestingly, our experiments showed that some preprocessing steps worsened the class imbalance problem. For example, after preprocessing, the positive class showed much higher accuracy, while the other two classes—especially the neutral class—experienced a noticeable drop in performance. This suggests that the model captures information that traditional models tend to treat as noise, such as hashtags, URLs, mentions, stopwords, non-alphanumeric characters, and non-lemmatized

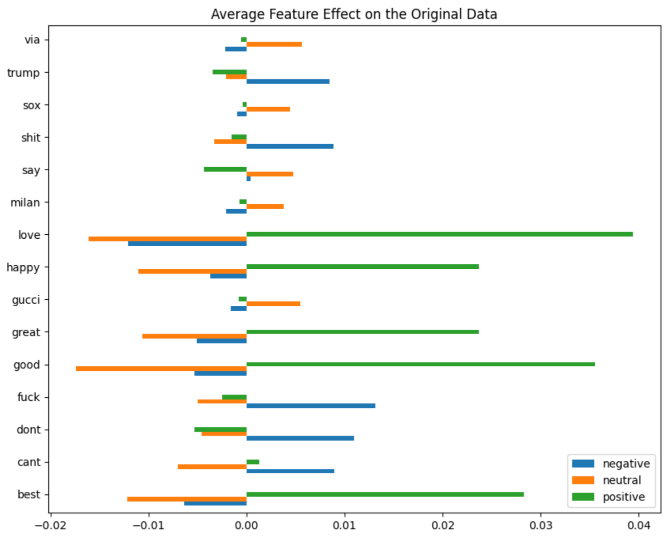

Fig. 1. Keywords for Different Classes With SVM

Classifier	Test1	Test2	Test3
GloVe-SVM	0.541	0.535	0.530
GloVe-NB	0.484	0.463	0.466
GloVe-LSTM	0.541	0.606	0.547

TABLE X
OVERALL F1 SCORES FOR GLOVE-SVM AND GLOVE-NB CLASSIFIERS ACROSS THREE TEST DATASETS.

	positive	negative	neutral
positive	0.498	0.107	0.395
negative	0.196	0.445	0.359
neutral	0.175	0.079	0.746

TABLE XI
CONFUSION MATRIX FOR GLOVE-SVM ON TWITTER-TEST1.

	positive	negative	neutral
positive	0.578	0.081	0.341
negative	0.274	0.374	0.353
neutral	0.304	0.052	0.643

TABLE XII
CONFUSION MATRIX FOR GLOVE-SVM ON TWITTER-TEST2.

words. This makes sense, as heavily preprocessed tweets often become difficult to interpret, even for humans. While classical models are effective at capturing simple patterns, more advanced models like transformers are capable of identifying complex linguistic cues, which are essential for accurate predictions.

We also attempted to fine-tune the model using Hugging Face's `Trainer` API. However, we encountered an issue: in the original model, the label mapping is `0 = negative`, `1 = neutral`, and `2 = positive`, whereas we inadvertently used `0 = positive`, `1 = neutral`, and `2 = negative` during training. We were unable to correct this label mismatch in time.

Despite this, the model still achieved impressive results. With only a small amount of training data and 30 epochs, the fine-tuned model achieved macro F1 scores of 0.74, 0.75, and 0.77 on the `test1`, `test2`, and `test3` sets, respectively—surpassing the performance of all other classifiers used in this study.

This improved performance can be attributed to two main factors. First, the use of a transformer-based architecture, which has been shown to outperform simpler models such as LSTMs. Second, the model has been pre-trained on a massive corpus of over 124 million tweets—far more data than we have access to or the resources to process for training a model from scratch.

2) Evaluation: When performing direct inference with the pre-trained model on raw tweets, we obtained the following macro F1 scores:

- `Test1` Macro F1 Score: 0.8008
- `Test2` Macro F1 Score: 0.8030
- `Test3` Macro F1 Score: 0.7845

B. DeepSeek r1

We attempted inference using DeepSeek R1 via Ollama, as it is considered one of the best language models available today, and Ollama provides an easy interface for inference. We used the 7B version of the model, which has a size of 4.7GB.

One challenge we encountered was that we could not find a way to load the model from Ollama directly within a Jupyter Notebook. Instead, we had to download both the model and Ollama through the terminal. Fortunately, since the model is relatively small compared to other large language models, we were able to run it on our local machine.

Another limitation we faced was the model's inference speed. Due to time and resource constraints, setting up the model on a server took too long. As a result, we only ran inference on 21 samples from `test set 2`, saving the output in a JSON file.

We prompted the model to return responses in JSON format to ensure the output was structured and easy to handle. This approach allowed us to bypass any need for post-processing and use the results directly for evaluation.

1) Evaluation: The results of DeepSeek R1 are shown in Table XX. The model achieved a macro F1 score of 0.58, which is lower than the score obtained by the RoBERTa model, but still higher than the scores from the other models.

Given that DeepSeek is one of the most advanced language models available today, it is somewhat surprising that its performance is relatively low. This may be due to the limited

	positive	**negative**	**neutral**
positive	0.546	0.108	0.346
negative	0.239	0.338	0.423
neutral	0.188	0.052	0.760

TABLE XIII
CONFUSION MATRIX FOR GLOVE-SVM ON TWITTER-TEST3.

	positive	**negative**	**neutral**
positive	0.535	0.130	0.335
negative	0.279	0.463	0.259
neutral	0.265	0.116	0.619

TABLE XIV
CONFUSION MATRIX FOR NB WITH GLOVE ON TWITTER-TEST1.

number of samples used for evaluation, or the fact that we used a distilled version of the model, which may not possess the full capabilities of the original.

	positive	**negative**	**neutral**
positive	0.608	0.109	0.283
negative	0.333	0.364	0.303
neutral	0.365	0.056	0.579

TABLE XV
CONFUSION MATRIX FOR NB WITH GLOVE ON TWITTER-TEST2.

	positive	**negative**	**neutral**
positive	0.563	0.141	0.296
negative	0.315	0.352	0.332
neutral	0.286	0.098	0.616

TABLE XVI
CONFUSION MATRIX FOR NB WITH GLOVE ON TWITTER-TEST3.

	positive	**negative**	**neutral**
positive	0.725	0.049	0.226
negative	0.210	0.627	0.163
neutral	0.251	0.161	0.587

TABLE XVII
CONFUSION MATRIX FOR GLOVE-LSTM ON TWITTER-TEST1.

	positive	**negative**	**neutral**
positive	0.760	0.038	0.202
negative	0.198	0.698	0.103
neutral	0.356	0.099	0.544

TABLE XVIII
CONFUSION MATRIX FOR GLOVE-LSTM ON TWITTER-TEST2.

	positive	**negative**	**neutral**
positive	0.730	0.055	0.214
negative	0.201	0.569	0.230
neutral	0.295	0.137	0.568

TABLE XIX
CONFUSION MATRIX FOR GLOVE-LSTM ON TWITTER-TEST3.

Class	**Precision**	**Recall**	**F1-score**	**Support**
negative	0.40	1.00	0.57	2
neutral	0.67	0.29	0.40	7
positive	0.75	0.82	0.78	11
accuracy			0.65	20
macro avg	0.61	0.70	0.58	20
weighted avg	0.69	0.65	0.63	20

TABLE XX
DEEPSEEK R1 RESULTS