NANYANG TECHNOLOGICAL UNIVERSITY

SCHOOL OF COMPUTER SCIENCE AND ENGINEERING

Project report for SC4001 / CE4042 / CZ4042
AY2023-2024

Sentiment Analysis

Chen Yiting

Dong Luojie

Zhao Yu

November 2023

Contents

Abstract

We performed a sentiment analysis on the Stanford Sentiment Treebank (SST)[1] and IMDB movie review[2] data sets to identify the opinions towards movies expressed in reviews by deploying Bayes classification, Long-Short Term Memory (LSTM) classification, multi-channel Convolutional Neural Network (CNN), and CNN_LSTM classification models. Different transformer architectures were also deployed on a subset of SST dataset and evaluated with quantitative score metrics. In this project, we worked on deep learning techniques to achieve the aim of adapting the domain, comparing the performance of transformer architectures, and tackling datasets with insufficient training samples.

1 Introduction

Sentiment analysis is a field in Natural Language Processing (NLP), focusing on identifying and categorizing opinions or attitudes expressed in text (1). In most scenarios, it is used to determine whether the writer's attitude is positive, neutral, or negative towards a certain topic, usually in the field of commerce, elections, and finance (2).

Various methods have been applied in an ample amount of research to carry out sentiment analysis. Some of the methods that have produced the most renowned works are machine learning, rule-based, and lexical-based approaches (3). A more recent review in 2021 on secondary studies in sentiment analysis suggested that LSTM and CNN algorithms are the most popular machine learning algorithms used in studies (4). Transformer-based methods have become a hot topic in sentiment analysis and have seen many successful applications and publications(5; 6; 7; 8; 9).

Sentiment analysis on movie review datasets would offer information on movie viewers' opinions, which is essential for producers, marketers, and filmmakers to comprehend how their work is received. The trend of viewers' likings can be analyzed to provide a deeper understanding for the practitioner in the movie-making business. Such analysis can be also performed on the reviews made by an individual, towards more accurate suggestions that are tailored to the end-user.

Specific aims The study aims of this project are as follows:

- **Aim 1.** To analyze the attitude in reviews by performing the sentiment analysis task with different techniques;
- **Aim 2.** Based on **Aim 1,** to benchmark the sentiment analysis techniques deployed and to determine the optimal methods based on requirements in a real-world scenario;
- **Aim 3.** Based on **Aim 1** and **Aim 2**, to discuss the reason behind the model performance based on our implementation, the input data and the model architecture, thus gain profound insights in sentiment analysis.

Project highlights Some salient points that have been touched on in this project are:

- **Domain adaptation**: Getting a model that was trained on one domain to function well in another is one of the fundamental problems in sentiment analysis. We use a fine-tuning method on pre-trained models to solve this. In particular, we adjust movie review TSA data using the 'bert-yelp-review' model, which was trained on customer reviews and are able to maximize the model's performance in the new domain thanks to this modification. To further improve the adaptation process, we also experiment with hyperparameter adjustment, paying special attention to learning rates.
- **Comparison between transformer architectures**: We carry out extensive benchmarking analysis in order to comprehend the performance variations across different transformer designs. The efficacy of several models, such as BERT, DistilBERT, and DistilRoBERTa is assessed and contrasted. Our goal is to determine which architecture best fits the TSA task while highlighting the advantages and disadvantages of each.

[1] https://www.kaggle.com/atulanandjha/stanford-sentimenttreebank-v2-sst2
[2] https://www.kaggle.com/lakshmi25npathi/imdb-dataset-of-50kmovie-reviews

- **Handling of smaller datasets**: We tackle the third criteria, which is handling tiny datasets—a problem that TSA frequently faces. In order to get around this restriction, we use many techniques:

  - **Transfer Learning** : We use big text corpora-trained models such as BERT, leveraging the capabilities of pre-trained models. We next use our limited dataset to fine-tune these models, making use of the knowledge that was already ingrained in the pre-trained model to improve performance.

  - **Regularisation methods** : We use crucial regularisation methods like dropout and early halting to avoid overfitting on sparse data. These techniques contribute to the successful generalization of our models.

  - We also propose to use ensemble methods to reduce instability with small datasets. It will be covered in detail in Section 6.

Scope and limitations In this project, based on the scope of the course, we decided to use naive Bayes classification, LSTM, CNN, CNN_LSTM, and some of the transformer-based architectures only.

It should be noted that there are more advanced approaches that might achieve better accuracy, especially transformer-based ones. However, we chose not to include them based on the scope of the course as well as the project. The limitations of computing power and RAM/VRAM were also a consideration. For example, we did not have sufficient resources for inference of the GPT model, not to mention training. Similarly, due to the limitation of computing power, for most transformer-based models, only a fraction of the full dataset was used for training and testing. This was done to cut down time on training, as well as to accomplish the goal of dealing with small datasets.

2 Methodology

In this project, we employed a diverse range of methodologies and models to tackle Text Sentiment Analysis. We begin with traditional neural network models and followed by the fast.ai pretrained models. This section briefly explains how the dataset was processed and transformed, and how the deep neural network models were designed, trained, and evaluated.

2.1 Dataset Discovery and Basic Preprocessing

Given the IMDB movie review dataset, which comprises 5000 entries with two columns, 'review' and 'sentiment,' both containing English text and other symbols such as special characters, HTML entities, etc. A text cleaning was carried out first to remove non-English content. Afterward, considering the different requirements that different models might need, stemming as a commonly used text normalization technique was applied.

Index Stemming The Stemming efficiently helped remove suffixes like "ing," "ed," "es," and others so that the core meaning could be better captured and the noise of the text corpus decreased largely. The cleaned data preview in Figure 1:

	review	sentiment	language	cleaned_review	cleaned_stemmed_review
0	One of the other reviewers has mentioned that ...	positive	en	one reviewers mentioned watching 1 oz episode ...	one review mention watch 1 oz episod hook righ...
1	A wonderful little production. The...	positive	en	wonderful little production filming technique ...	wonder littl product film techniqu unassum old...
2	I thought this was a wonderful way to spend ti...	positive	en	thought wonderful way spend time hot summer we...	thought wonder way spend time hot summer weeke...
3	Basically there's a family where a little boy ...	negative	en	basically family little boy jake thinks zombie...	basic famili littl boy jake think zombi closet...
4	Petter Mattei's "Love in the Time of Money" is...	positive	en	petter mattei love time money visually stunnin...	petter mattei love time money visual stun film...

Figure 1: Cleaned movie review comparison

The processed text became more concise and valid for further training and prediction. We did an analysis of word frequency. Figure 2 below displays the most frequently occurring words in processed text data.

(a) Word cloud

	common_words	count
0	movie	87937
1	film	79674
2	one	53585
3	like	40160
4	good	29737
5	time	25100
6	even	24855
7	would	24599
8	story	23110
9	really	23089
10	see	23021

(b) Most frequent words

Figure 2: Word frequency insights

Data Splitting The transformed data was split into training, validation, and test sets with the ratio of 6:2:2, with the training dataset having 30000 entries and the rest having 10000. Figure 12 in the appendix section displays the program output of shapes of the training, validation, and test data after splitting.

Text Vectorization Text Vectorization is to transform the text data into numerical vector representation that can be used as input for the models to process. In this project, three techniques were applied.

- **Index Vectorization** It uses a unique index for every word or token in the text.
- **BoW (Bag of Words)** It concentrates on word frequency within a document, ignoring word order. Every document is represented as a vector, with each dimension denoting a distinct word across the corpus.
- **TF-IDF (Term Frequency-Inverse Document Frequency)** It weighs words according to their significance within a given document and takes into account their frequency throughout the entire corpus.

After vectorization, the shapes of the `X_train` data are all `(30000,500)`. The output showing the shape of the vectorized data is shown in Figure13 in the appendix section. The vectorized data was then serialized into a .pkl file for later training and prediction.

2.2 Training of Traditional Neural Network Models

We experiment with traditional neural network models such as the Naive Bayes Classifier, LSTM classification, Multi-channel CNN, and CNN-LSTM. The models are trained and evaluated with different performance metrics such as F1 score. Naive Bayes would serve as a baseline for benchmarking due to its simplicity. In the training of the traditional neural network models, a few techniques were implemented including Early Stopping for overfitting prevention, Model Checkpoint for saving the best weights, and learning rate adjustment with ReduceLROnPlateau. More details in Appendix A.2.1

2.3 Using fast.ai for Transformer-Based Methods

The data preprocessing for Transformers is different from the traditional models.

A maximum 512 or the 1/20 training data of Stanford Sentiment Treebank v2 was used for training and a total of six transformers architectures were experimented with the fast.ai framework.

We carried out tokenization, input formatting, and data preparation to ensure that our text data met the requirements of the pre-trained Transformers.

TokBatchTransform: We used `TokBatchTransform`, which wraps a pre-trained HuggingFace tokenizer, to tokenize the inputs. To expedite the process, the text is processed in batches. Whenever possible, we want to avoid using explicit Python loops. Relevant code can be found in Listing 1.

TransCallback: TransCallback creates a tuple from the input dict that the dataloader yielded and saves the valid model argument. By default, the model produces an object that resembles a dictionary that contains logits and potentially additional outputs specified by the model configuration (such as intermediate hidden representations). Preds is typically expected to be a tensor containing model predictions (logits) in the fastai training loop. The callback correctly formats the press.

Transformer models calculate the loss and return it along with the output logits if labels are detected in the input. The callback that follows is made to use the loss that the model returns rather than recalculating it with learn.loss_func. Although it isn't used in this example, this could be useful in other situations. A code snippet of the implementation is in Listing 2 in the appendix.

Customized Learner: A series of positional arguments are fed into the model by fastai Learner. We can create a callback to handle unrolling the input dict into the correct xb tuple in order to ensure that everything goes smoothly. Relevant code can be found in Listing 3.

3 Experiments and Results

3.1 Traditional TSA Models

3.1.1 Naive Bayes Classification

Naive Bayes classifiers are based on applying Bayes' theorem with strong (naive) independence assumptions between the features (10; 11). The model calculates the probability of each class and the conditional probability of each class given each input value. These probabilities are estimated for new data and multiplied together, assuming they are all independent, and the class with the highest probability is output. This is used as the baseline model.

The Naive bayes was trained and evaluated using the different vectorized dataset, there are four evaluation measures, F1 score, precision score, recall score, and average precision-recall score and the results are shown in the table below:

	F1 Score	Precision Score	Recall Score	Average Precision-Recall Score
index_cleaned	0.635	0.503	0.862	0.503
index_cleaned_stemmed	0.634	0.503	0.857	0.502
bow_cleaned	0.814	0.829	0.799	0.763
bow_cleaned_stemmed	0.816	0.828	0.804	0.764
tfidf_cleaned	0.819	0.836	0.803	0.769
tfidf_cleaned_stemmed	0.819	0.835	0.804	0.769

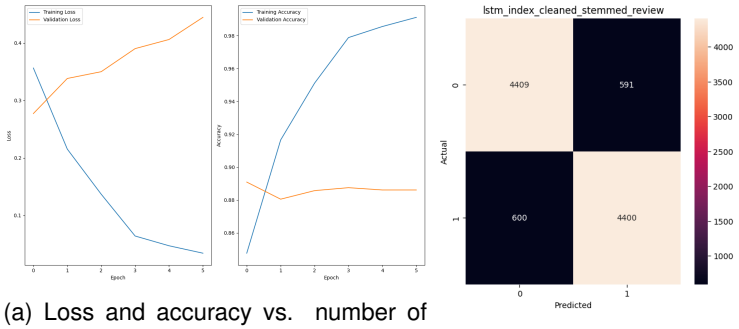
From the above evaluation, NaiveBayes performs optimally when dealing with stemmed text using tfidf vectorization, this further proves the hypothesis that the stemming text will reduce the noise in the data hence improve the model performing. Worth mentioning is that, when using index vectorization, the precision score is close to 0.5, which means not much better than taking random guess.

3.1.2 Long Short Term Memory (LSTM) Classification

LSTM is a type of recurrent neural network (RNN) that is capable of learning long-term dependencies. It has a chain-like structure, but the repeating module has a different structure: instead of having a single neural network layer, there are four interacting layers(12). LSTMs are significant for their ability to retain information over long sequences.

TPU provided on Kaggle was used to reduce the training time. In this experiment, the stemmed text with index vectorization was selected as training data as the LSTM model requires a numerical data input format.

Each epoch took approximately 465 seconds to run in the training of the LSTM model.



(a) Loss and accuracy vs. number of Epochs

(b) Confusion matrix

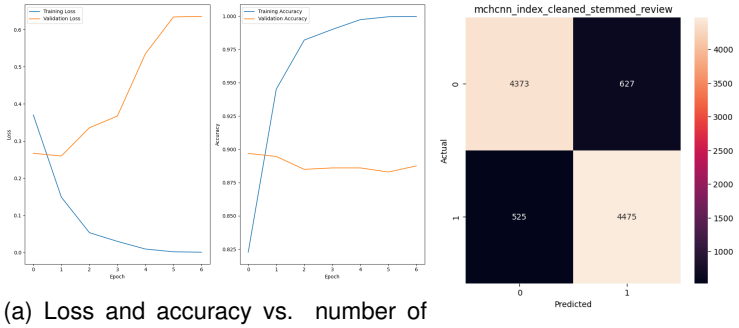
Figure 3: Summary of Performance Metrics of LSTM

Figure 3 shows a summary of various performance metrics of the LSTM model. Figure 3a shows the accuracy and the loss in the training and validation datasets; figure 3b shows the confusion matrix of the model output. The final training accuracy reached 99.11%, while the validation accuracy peaked at 88.09% but then dropped to 88.61%, suggesting possible overfitting. The same trend can be observed in loss in validation. There are comparable amounts of Type I errors and Type II errors (591 vs 600).

3.1.3 Multi Channel Convolutional Neural Network(CNN)

A multi-channel CNN uses multiple convolutions to process the input data independently and in parallel, each extracting different features and then combines them further in the network. This architecture is akin to having multiple senses and integrating their inputs to understand a scene better, and has performed better than the traditional CNN methods (13), and has an advantage in complex tasks where different kinds of features are relevant.

A multi-channel CNN was also trained using index cleaned stemmed data, from the training accuracy and loss plots in Figure 4, it appears that each epoch takes approximately 230 seconds about 3.83 minutes to run.



(a) Loss and accuracy vs. number of Epochs

(b) Confusion matrix

Figure 4: Summary of Performance Metrics of Multi-Channel CNN

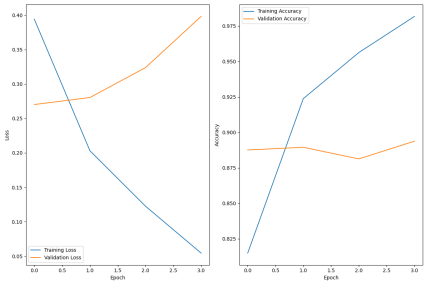
Following the same procedure, Figure 4 demonstrates a summary of performance metrics of the multi-channel CNN model. The final training accuracy reached 99.98%, while the validation accuracy peaked at 89.69% but then dropped to 88.61%. The loss of validation also suggests overfitting. There are similar numbers of Type I errors and Type II errors (627 vs 527).

3.1.4 CNN LSTM

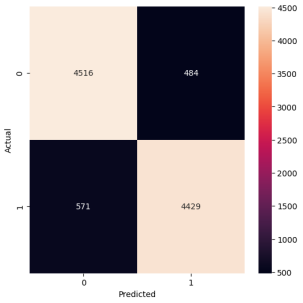
The CNN LSTM architecture involves using Convolutional Neural Network (CNN) layers for feature extraction on input data combined with LSTMs to support sequence prediction (14). CNN layers extract

spatial features from the input data, and these features are then passed to LSTM layers to capture temporal dynamics.

The training of CNN LSTM is much longer than the previous models due to its special architecture. Each epoch takes approximately 700 seconds to run. However, the model takes less epoch to converge which means the model learns the pattern in the data faster. The analysis of the performance is shown below.



(a) Loss and accuracy vs. number of Epochs



(b) Confusion matrix

Figure 5: Summary of Performance Metrics of CNN_LSTM

Figure 5a shows the steady improvement of accuracy on the training set. This indicates that the model performance converges faster comparing to Multi Channel Convolutional Neural Network(CNN), LSTM and NaiveBayes.

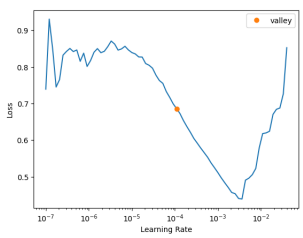
3.2 Transformers

In recent years, transformers have become a dominant architecture in NLP to handle sequences of data without the need for recurrence, using self-attention to weigh the influence of different parts of the input data (5; 15; 16; 17). There are six transformer techniques applied in this project and the pre-trained models provided by Hugging Face constitute a primary resource for transformer architectures.

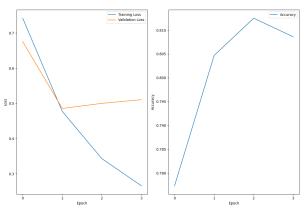
3.2.1 BERT

BERT (Bidirectional Encoder Representations from Transformers) uses the transformer's encoder mechanism in a bidirectional way to understand the context of a word based on all of its surroundings (18). Its bidirectional nature allows for a more nuanced context understanding, contributing to its state-of-the-art performance on many NLP tasks. It achieves high accuracy on various NLP tasks and has a deep understanding of context and nuance in language but requires significant computational resources to train and fine-tune (19; 20). In this application, a model specifically trained for polarity classification from huggingface was used.

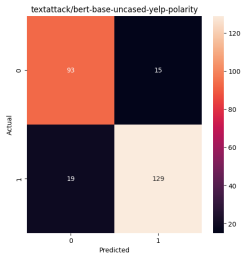
In our model run, each epoch took around 53 seconds.



(a) Optimal learning rate



(b) Loss and accuracy vs. number of Epochs



(c) Confusion matrix

Figure 6: Summary of Performance Metrics of BERT

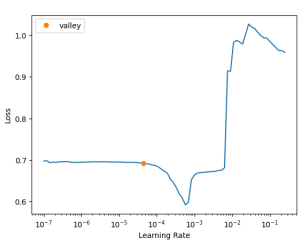
Figure 6 shows some performance metrics as well as the optimal learning rate. BERT seems to also

suffer from overfitting, which can be seen from the increasing validation loss since epoch 1. The best accuracy hit 81.25%. There are comparable amounts of Type I errors and Type II errors (15 vs 19).

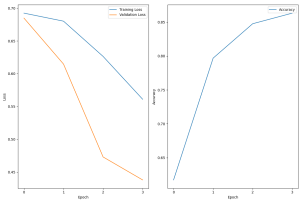
3.2.2 DistilBERT

DistilBERT is a smaller ('distilled') version of BERT designed to be faster and lighter while retaining most of BERT's performance (7). It's achieved by a method known as knowledge distillation, which provides a more efficient alternative for environments where computational resources are limited (21; 22; 23; 24). It is faster and requires less memory than BERT, making it more deployable on resource-constrained devices but some performance trade-offs compared to full BERT.

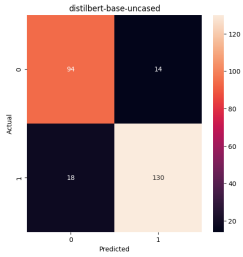
DistilBERT as a smaller model comparing to BERT, provided faster inference, averaging 29.75 seconds per epoch.



(a) Optimal learning rate



(b) Loss and accuracy vs. number of Epochs



(c) Confusion matrix

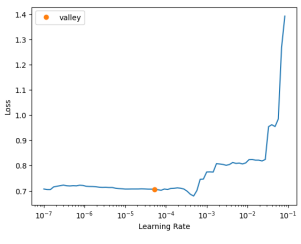
Figure 7: Summary of Performance Metrics of DistilBERT

Figure 7 shows DistilBERT's performance. Steady decreases of loss in both the training and validation datasets and an increase in accuracy suggest no signs of overfitting in this model.

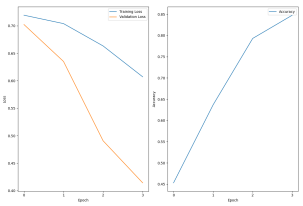
3.2.3 BART

The BART (Bidirectional and Auto-Regressive Transformers) model is a type of transformer-based machine learning model similar to BERT that was introduced before: it takes into account the full context of a sentence by processing data in both directions. Unlike BERT, BART also includes a decoding capability, much like the GPT (Generative Pretrained Transformer) models, which generate text in an auto-regressive manner. This means that it predicts the next word in a sequence while taking into account all the previous words.

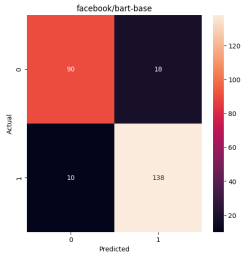
The model run of BART took longer compared to the previously mentioned transformers which usually took less than 1 minute for each epoch. The average time for each epoch was 68.75 seconds. The optimal learning rate and some performance metrics are shown below in Figure 8.



(a) Optimal learning rate



(b) Loss and accuracy vs. number of Epochs



(c) Confusion matrix

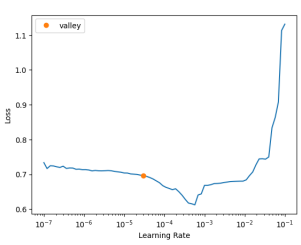
Figure 8: Summary of Performance Metrics of BART for Classification

No signs of overfitting are demonstrated in Figure 8b.Figure 8c shows that there are more Type I errors compared to Type II errors (18 vs 10), meaning the BART model is more likely to consider a negative review positive, than the other way round.

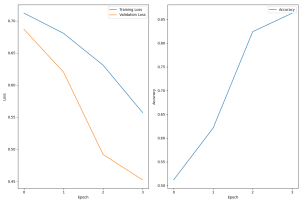
3.2.4 ALBERT

ALBERT (A Lite BERT) introduces two parameter-reduction techniques to lower memory consumption and increase the training speed of BERT (25). It factors the embedding matrix into two smaller matrices and shares parameters across the hidden layers. It's designed to scale up more efficiently, allowing for larger models without as much computational cost. It has reduced memory consumption and faster training without a significant drop in performance.

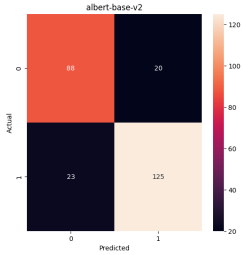
The mean run time for each epoch was 60 seconds in our experiment. Figure 9 shows the optimal learning rate obtained and some performance metrics of it.



(a) Optimal learning rate



(b) Loss and accuracy vs. number of Epochs



(c) Confusion matrix

Figure 9: Summary of Performance Metrics of ALBERT

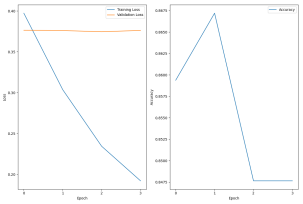
As shown in the above figure, the model is training effectively without signs of overfitting. The training and validation loss in Figure 9b are consistently decreasing, and the accuracy is improving for both sets. ALBERT has more false negatives than most other models, as seen in 9c.

3.2.5 ALBERT for Classification

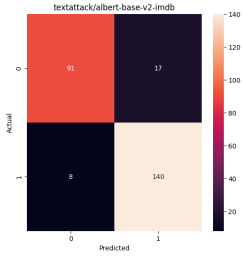
This model is a variant of the formerly introduced ALBERT. It was a fine-tuned version of the ALBERT V2 base model, that is optimized for applying TextAttack to performing classification of sequence[3]. It was also loaded with the IMDb dataset using the nlp library.

ALBERT for Classification took 61 seconds to run on average for each epoch. Figure 10 shows a summary.

(a) Optimal learning rate



(b) Loss and accuracy vs. number of Epochs



(c) Confusion matrix

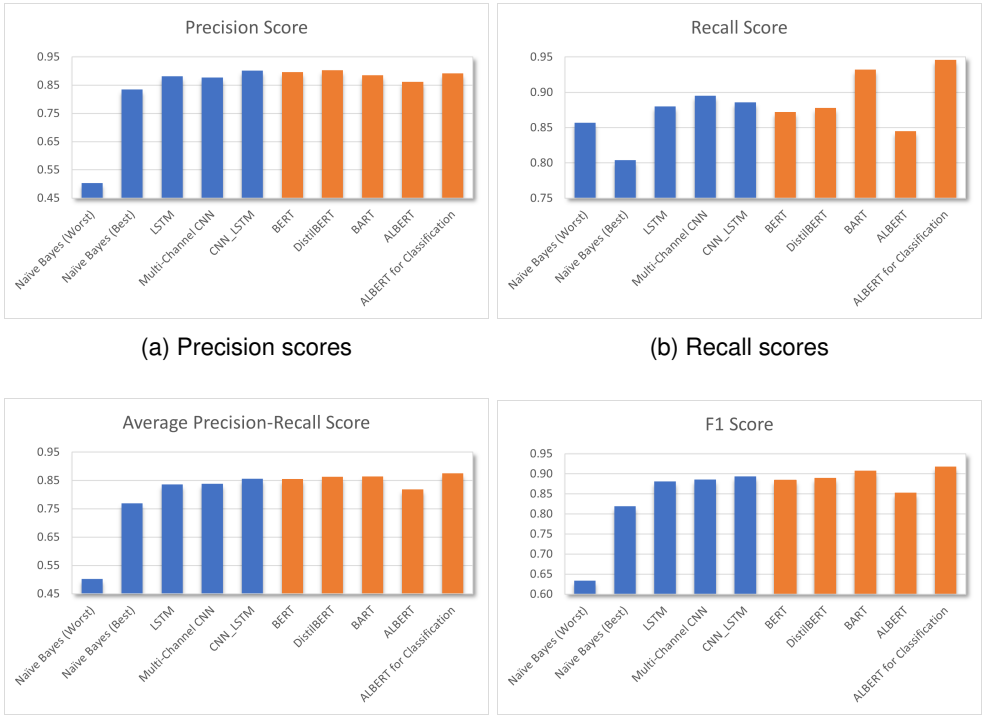
Figure 10: Summary of Performance Metrics of ALBERT for Classification

The accuracy of ALBERT Classification remains relatively stable at 84% - 85% (note the small range of the in Figure 10b, the drop is not significant), no significant overfitting was identified. According to Figure 10c, ALBERT for Classification has a bias towards making Type I error s (false positive:false negative = 17:9). However, it makes fewer errors overall compared with the ALBERT base which is "unbiased".

[3] https://huggingface.co/textattack/albert-base-v2-imdb

4 Discussion

We have compiled the precision score, recall score, average precision-recall score, and F1 score into Figure 11. A table version containing more detailed figures is put in the appendix as Table 4.



(a) Precision scores

(b) Recall scores

(c) Average precision-recall scores

(d) F1 scores

Figure 11: Summary of performance benchmark across models

Within the “traditional" TSA models, the CNN-LSTM hybrid model achieved the highest F1 score, precision, and average precision-recall score, outperforming all other models. Convolutional and sequential information are combined in this model in an efficient manner. LSTM performed well, demonstrating its capacity to identify sequential dependencies in text data. The Multi-Channel CNN model demonstrated good performance as well, highlighting the importance of extracting diverse features. While yielding reasonable results with appropriate vectorization, the performance of Naive Bayes was not as good as that of deep learning models.

Within the pretrained transformer architecture models, BERT, DistilBERT, and DistilRoBERTa perform admirably. Specifically, DistilBERT exhibits high recall. BART shows promise in sentiment analysis, especially in recall, despite being created for text generation. The performance of ALBERT models varies, with "textattack/albert-base-v2-imdb" producing outstanding results, highlighting the importance of model selection.

The top 3 models in precision scores are DistilBERT, CNN_LSTM and BERT; the top 3 models in recall scores are ALBERT for Classification, BART and Multi-channel CNN; the top 3 models in average precision-recall scores are ALBERT for Classification, BART, and DistilBERT; and the top 3 models in F1 scores are ALBERT for Classification, BART, and CNN_LSTM.

The precision score is the ratio of true positive predictions to the total number of positive predictions made by the model. The more false positives, the lower the precision score. When a model with high precision scores predicts a review to be positive, it is more likely that the review is actually positive. The recall score is the ratio of true positive predictions to the total number of actual positive instances. It decreases with more false positives. A model with a high recall score is more likely to identify an actually positive review. Interestingly, among these models, the recall scores are generally lower than the precision scores. This implies that they correctly identify positive reviews when they make a positive prediction, but they often fail to detect many actual positive reviews. We believe this has something to do with the complex nuance in detailed reviews. A hypothesis is that, when people write positive reviews, they spend more time writing more comprehensively, thus confusing the text sentiment analysis model.

Since the average precision-recall score and F1 score are both functions of the precision score and recall score, the top 3 models according to the average and F1 score having two entries in common is not beyond our expectations.

Overall, ALBERT for Classification has the best performance among all the models experimented with, with the highest recall score and a decent recall score. Within the range of non-transformer neural network architectures, CNN_LSTM is found to be optimal. We believe that this is because the CNN_LSTM model leverages both local feature extraction capabilities of CNNs and the sequence learning of LSTMs.

5 Conclusion

We have examined a wide range of methods and models in this thorough investigation into Text Sentiment Analysis (TSA) in order to address the difficulty of comprehending the sentiment conveyed in text. Based on the above analysis, the majority has performed reasonably well. The best-performing non-transformer architecture model is CNN - LSTM which achieved a 0.894 f1 score. The model with the best performance within transformer-based and overall is ALBERT for classification scoring 0.918.

The CNN-LSTM hybrid model demonstrated exceptional performance and provided a useful solution for TSA tasks by fusing the advantages of Long Short-Term Memory networks and Convolutional Neural Networks. Still, LSTM models are a competitive choice for sentiment analysis because of their demonstrated ability to capture sequential dependencies.

Optimizing pre-trained Transformers, like BERT, DistilBERT, and ALBERT, can produce impressive sentiment analysis outcomes and provide a tried-and-true solution across a range of domains.

Last but not least, data preprocessing is just as important as the model selection in guaranteeing that the text data satisfies these models' specifications.

While the current project achieves its set objectives, it inherently presents multiple ways for expansion and refinement. We will propose several possible studies that we are interested in doing in the next section.

6 Future Work

The dynamic nature of sentiment analysis, especially the rapid growth in advancements in transformer techniques (5) means that this project might serve as a foundation for future works.

An extension of the study without increasing its scope can be the deployment of ensemble methods. In this project, transformer methods were tested only on small datasets, which are known to cause unstable predictions (9). We propose to solve this problem by applying ensemble methods. By combining predictions from multiple models, which may have been trained on different data subsets or with different architectures, we believe the overall performance and the credibility of the predictions will increase.

Although most of the transformer models provide decent results in sentiment classification, we were still interested in potential greater performance and stability by combining predictions from multiple models, which may have been trained on different data subsets or with different architectures.

The scope of this project, while comprehensive within the scope of CX4042 and similar undergraduate-level courses, could be further broadened to encompass cutting-edge transformer-based architectures. Further studies could improve the performance of sentiment analysis and provide more accurate results.

A Appendix

A.1 Supplementary Images: Program Output

```
Shape of X_train: (30000, 2)
Shape of y_train: (30000,)
Shape of X_valid: (10000, 2)
Shape of y_valid: (10000,)
Shape of X_test: (10000, 2)
Shape of y_test: (10000,)
```

Figure 12: shape of split data

```
Shape of cleaned_review X_train after index vectorization (30000, 500)
Shape of cleaned_stemmed_review X_train after index vectorization (30000, 500)
Shape of cleaned_review X_train after bow vectorization (30000, 500)
Shape of cleaned_stemmed_review X_train after bow vectorization (30000, 500)
Shape of cleaned_review X_train after tfidf vectorization (30000, 500)
Shape of cleaned_stemmed_review X_train after tfidf vectorization (30000, 500)
```

Figure 13: shape of vectorized data

A.2 Supplementary Tables

A.2.1 Model Parameter Tuning

No. of Epoches	10
Batch Size	32 * num_replicas
Dropout Rate	0.2
Optimizer	Adam
Loss Function	Binary Cross-Entropy

Table 1: LSTM parameters

No. of Epoches	10
Batch Size	32 * num_replicas
Dropout Rate	0.5
Optimizer	Adam
Loss Function	Binary Cross-Entropy

Table 2: Multi-Channel CNN parameters

No. of Epoches	10
Batch Size	32 * num_replicas
Dropout Rate	0.3
Optimizer	Adam
Loss Function	Binary Cross-Entropy

Table 3: CNN-LSTM parameters

A.2.2 Performance Benchmark Across All Models

Model	F1 Score	Precision Score	Recall Score	Average Precision-Recall Score
Naïve Bayes (Worst)	0.634	0.503	0.857	0.502
Naïve Bayes (Best)	0.819	0.835	0.804	0.769
LSTM	0.881	0.882	0.88	0.836
Multi-Channel CNN	0.886	0.877	0.895	0.838
CNN_LSTM	0.894	0.901	0.886	0.856
BERT	0.884	0.896	0.872	0.855
DistilBERT	0.89	0.903	0.878	0.863
BART	0.908	0.885	0.932	0.864
ALBERT	0.853	0.862	0.845	0.818
ALBERT for Classification	0.918	0.892	0.946	0.875

Table 4: Detailed Benchmark Parameters

A.3 Code Listings

```
class TokBatchTransform(Transform):
    """
    Tokenizes texts in batches using pretrained HuggingFace tokenizer.
    The first element in a batch can be single string or 2-tuple of strings.
    If ‘with_labels=True‘ the "labels" are added to the output dictionary.
    """
    def __init__(self, pretrained_model_name=None, tokenizer_cls=AutoTokenizer,
                 config=None, tokenizer=None, with_labels=False,
                 padding=True, truncation=True, max_length=None, **kwargs):

        if tokenizer is None:
            tokenizer = tokenizer_cls.from_pretrained(pretrained_model_name, config=
    config)
        self.tokenizer = tokenizer
        self.kwargs = kwargs
        self._two_texts = False
        store_attr()

    def encodes(self, batch):
        # batch is a list of tuples of ({text or (text1, text2)}, {targets...})
        if is_listy(batch[0][0]): # 1st element is tuple
            self._two_texts = True
            texts = ([s[0][0] for s in batch], [s[0][1] for s in batch])
        elif is_listy(batch[0]):
            texts = ([s[0] for s in batch],)

        inps = self.tokenizer(*texts,
                              add_special_tokens=True,
                              padding=self.padding,
                              truncation=self.truncation,
                              max_length=self.max_length,
                              return_tensors='pt',
                              **self.kwargs)

        # inps are batched, collate targets into batches too
        labels = default_collate([s[1:] for s in batch])
        if self.with_labels:
            inps['labels'] = labels[0]
```

```
            res = (inps, )
        else:
            res = (inps, ) + tuple(labels)
        return res

    def decodes(self, x:TensorBase):
        if self._two_texts:
            x1, x2 = split_by_sep(x, self.tokenizer.sep_token_id)
            return (TitledStr(self.tokenizer.decode(x1.cpu(), skip_special_tokens=True)),
                    TitledStr(self.tokenizer.decode(x2.cpu(), skip_special_tokens=True)))

class Undict(Transform):
    '''
    The Undict transform extracts the batch's input_ids and produces TensorBase,
    which ought to function with typedispatch.

    '''
    def decodes(self, x:dict):
        if 'input_ids' in x: res = TensorBase(x['input_ids'])
        return res
```

Listing 1: TokBatchTransform Class

```
class TransCallback(Callback):
    #Handles HuggingFace model inputs and outputs

    def __init__(self, model):
        self.labels = tuple()
        self.model_args = {k:v.default for k, v in signature(model.forward).parameters.
    items()}

    def before_batch(self):
        if 'labels' in self.xb[0].keys():
            self.labels = (self.xb[0]['labels'], )
        # make a tuple containing an element for each argument model excepts
        # if argument is not in xb it is set to default value
        self.learn.xb = tuple([self.xb[0].get(k, self.model_args[k]) for k in self.
    model_args.keys()])

    def after_pred(self):
        if 'loss' in self.pred:
            self.learn.loss_grad = self.pred.loss
            self.learn.loss = self.pred.loss.clone()
        self.learn.pred = self.pred.logits

    def after_loss(self):
        if len(self.labels):
            self.learn.yb = self.labels
            self.labels = tuple()
```

Listing 2: TransCallback Class

```
def default_splitter(model):
    groups = L(model.base_model.children()) + L(m for m in list(model.children())[1:] if
    params(m))
    return groups.map(params)

#TransLearner itself doesn't do much:
#it adds TransCallback and sets splitter to be default_splitter if None is provided.
#Example usage: learn = TransLearner(dls, model, metrics=metrics, opt_func=opt_func)
@delegates(Learner.__init__)
class TransLearner(Learner):
    "Learner for training transformers from HuggingFace"
    def __init__(self, dls, model, **kwargs):
        splitter = kwargs.get('splitter', None)
        if splitter is None: kwargs['splitter'] = default_splitter
        super().__init__(dls, model, **kwargs)
        self.add_cb(TransCallback(model))
```

Listing 3: Customized Learner

References

[1] B. Liu, *Introduction*, p. 1–15. Cambridge University Press, 2015.

[2] P. Šaloun, M. Hruzík, and I. Zelinka, "Sentiment analysis - e-bussines and e-learning common issue," in *2013 IEEE 11th International Conference on Emerging eLearning Technologies and Applications (ICETA)*, pp. 339–343, 2013.

[3] M. Devika, C. Sunitha, and A. Ganesh, "Sentiment analysis: A comparative study on different approaches," *Procedia Computer Science*, vol. 87, pp. 44–49, 2016. Fourth International Conference on Recent Trends in Computer Science & Engineering (ICRTCSE 2016).

[4] A. Ligthart, C. Catal, and B. Tekinerdogan, "Systematic reviews in sentiment analysis: a tertiary study," *Artificial Intelligence Review*, vol. 54, pp. 4997–5053, Oct. 2021.

[5] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser, and I. Polosukhin, "Attention is all you need," 2023.

[6] P. He, X. Liu, J. Gao, and W. Chen, "Deberta: Decoding-enhanced bert with disentangled attention," 2021.

[7] V. Sanh, L. Debut, J. Chaumond, and T. Wolf, "Distilbert, a distilled version of bert: smaller, faster, cheaper and lighter," 2020.

[8] W. Wang, B. Bi, M. Yan, C. Wu, Z. Bao, L. Peng, and L. Si, "Structbert: Incorporating language structures into pre-training for deep language understanding," *ArXiv*, vol. abs/1908.04577, 2019.

[9] Z. Lan, M. Chen, S. Goodman, K. Gimpel, P. Sharma, and R. Soricut, "Albert: A lite bert for self-supervised learning of language representations," 2020.

[10] P. A. Flach and N. Lachiche, "Naive bayesian classification of structured data," *Machine Learning*, vol. 57, pp. 233–269, Dec 2004.

[11] J. Ren, S. D. Lee, X. Chen, B. Kao, R. Cheng, and D. Cheung, "Naive bayes classification of uncertain data," in *2009 Ninth IEEE International Conference on Data Mining*, pp. 944–949, 2009.

[12] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural Computation*, vol. 9, no. 8, pp. 1735–1780, 1997.

[13] K. Yoon, "Convolutional neural networks for sentence classification. 2014," *arXiv preprint cs.CL/1408.5882*, 2014.

[14] J. Donahue, L. A. Hendricks, M. Rohrbach, S. Venugopalan, S. Guadarrama, K. Saenko, and T. Darrell, "Long-term recurrent convolutional networks for visual recognition and description," 2016.

[15] A. M. Rush and S. R. Biderman, "The annotated transformer," 2018.

[16] T. Wolf, L. Debut, V. Sanh, J. Chaumond, C. Delangue, A. Moi, P. Cistac, T. Rault, R. Louf, M. Funtowicz, and J. Brew, "Huggingface's transformers: State-of-the-art natural language processing," *ArXiv*, vol. abs/1910.03771, 2019.

[17] P. Thapak and P. Hore, "Transformer++," *ArXiv*, vol. abs/2003.04974, 2020.

[18] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova, "Bert: Pre-training of deep bidirectional transformers for language understanding," in *North American Chapter of the Association for Computational Linguistics*, 2019.

[19] J. Brownlee, "A gentle introduction to cnn lstm recurrent neural networks," 2017.

[20] A. Wang, J. Hula, P. Xia, R. Pappagari, R. T. McCoy, R. Patel, N. Kim, I. Tenney, Y. Huang, K. Yu, S. Jin, B. Chen, B. V. Durme, E. Grave, E. Pavlick, and S. R. Bowman, "Can you tell me how to get past sesame street? sentence-level pretraining beyond language modeling," 2019.

[21] G. Hinton, O. Vinyals, and J. Dean, "Distilling the knowledge in a neural network," 2015.

[22] C. Buciluundefined, R. Caruana, and A. Niculescu-Mizil, "Model compression," in *Proceedings of the 12th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '06, (New York, NY, USA), p. 535–541, Association for Computing Machinery, 2006.

[23] J. Ba and R. Caruana, "Do deep nets really need to be deep?," in *Advances in Neural Information Processing Systems* (Z. Ghahramani, M. Welling, C. Cortes, N. Lawrence, and K. Weinberger, eds.), vol. 27, Curran Associates, Inc., 2014.

[24] G. Urban, K. J. Geras, S. E. Kahou, O. Aslan, S. Wang, R. Caruana, A. Mohamed, M. Philipose, and M. Richardson, "Do deep convolutional nets really need to be deep and convolutional?," 2017.

[25] Z. Lan, M. Chen, S. Goodman, K. Gimpel, P. Sharma, and R. Soricut, "Albert: A lite bert for self-supervised learning of language representations," *ArXiv*, vol. abs/1909.11942, 2019.