182 Final NLP Project Report

Tejasvi Kothapalli, Brian Zhu, Joe Zou

1 Introduction

The goal of this project is to predict the star ratings for Yelp reviews. This is an ongoing NLP research problem and previous work pointed us in the direction of implementing various state of the art models and data processing techniques. Our first goal was to maximize exact match accuracy and average star error with standard machine learning procedures. Our secondary goal was to explore a novel technique to approach this problem. The intention of our novel technique was to understand how different segments in a review affect the overall prediction through sliding window methods. We hope that our novel contribution can help people better understand how these models make decisions.

2 Literature Survey/Related Work

2.1 General Survey of NLP Classification

We began our research with a survey of the general NLP classification field:

[1] This paper compares and contrasts standard classical machine learning and deep learning methods on the Yelp dataset. It also details how data cleaning was performed on the Yelp dataset.

[2] This post covers the difference between using classification and regression for rating prediction, as both are reasonable choices. The post explains how empirically classification has done better than regression.

[4] This paper compares various popular NLP models on identifying toxic comments, which would be a similar task to rating prediction. The researchers found that fine-tuning pre-trained models yielded the best results, but noted that more research is needed to justify the differences between various methods.

[5] This paper outlines the BERT model. A standard practice is to use a pre-trained BERT model as the first layer and add on layers afterwards that will be fine-tuned for a specific task.

Overall, we use existing model architectures as a starting point for our own model and test out different variations to find the best architecture for our specific task. In addition, we take into account that our computational capacity and dataset is limited. Some of the prior works are state of the art, but require computational resources that we don't have access to right now.

2.2 Survey Sliding Window Methods

We decided to focus our novel contribution on the sliding window methods designed to tackle the $O(n^2)$ runtime computation of transformers, where n refers to sequence length. Most pretrained transformer models have a maximum sequence length of 512. In order to process longer sentences, the models are run with a sliding window of text input and employ some type of aggregation scheme to combine the outputs.

For the sliding window approach mentioned in the Longtransformer paper[6], the transformer acts like a convolution that slides across the sequence of long text. Each window of the text is run through the model, and the output can be averaged at the very end across all windows to produce the model output. In the Longtransformer paper[6], a variation on the sliding window approach is introduced where every layer of the transformer has a fixed window length that produces a representation which is fed into the next layer of the model. By stacking these layers vertically, we can increase the receptive field of a model from w to $w \times l$ where w is the window length and l is the number of layers. The downside of this proposed method in the paper however, is that the Longtransformer is incredibly difficult to implement and requires a lot of tricks such as a custom banded matrix multiplication method that is optimized to achieve a linear runtime.

Given our limited computational resources, we focused our novel contribution on exploring other alternatives to the sliding window method that doesn't require custom CUDA methods. More specifically, we introduce the notion of importance weights for each sliding window segment, and perform a weighted average using these importance weights to get a final output from all of the sliding window segments.

3 Background

From our preliminary research into the NLP field, we decided to start with a pretrained DistilBERT transformer from HuggingFace[8] and run the classification head of the transformer through a fully connected network to get our final classification outputs. We found the simpletransformers[7] Github repository which has a pipeline to fine-tune pretrained transformer classification models, so we decided to use this codebase as a starting point for our own project and make changes from there.

4 Methods/Approach

Our method and approach can be split into two main sections where the first contains general experimentation with common methods to find a best performing model, and the second section details our novel contributions.

4.1 Finding the best performing model

Our first initial baseline model was a simple pre-trained DistilBERT transformer with a classification head that outputs 5 logits, one for each class. We fine-tuned this model on a Yelp reviews dataset with default hyperparameters of the simpletransformers repository.

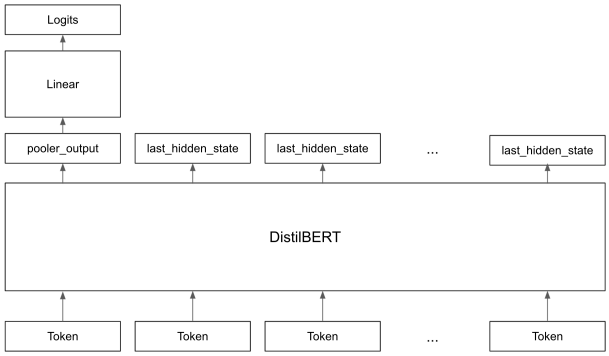

Figure 1: BERT for Sequence Classi cation

Our first experiment was to find an appropriate variation of the BERT model. We wanted to use a lightweight model that can meet performance constraints and train fast so we can run many experiments. The models we decided to compare were pre-trained versions of DistilBERT, SqueezeBERT, ALBERT, and DistilRoBERTa from Hugging Face, which we fine-tuned and evaluated on the given dataset. We found that DistilRoBERTa performed the best out of these four, so we stuck with this architecture for the rest of the experiments.

During testing, we found that the class imbalance from the dataset was a more pressing issue, so we decided to focus on this first. We collected up to 300k more data points in each star rating from online Yelp review datasets, and used these extra points to create datasets with custom distributions. We trained DistilRoBERTa on fully uniform datasets, very imbalanced ones, and "interpolations" between them. For validation, we used both a uniform set and an imbalanced set that is similar to the training dataset.

Next, we ran a variety of experiments using ensembles. In particular we tried both conventional ensembles, as well as decision trees of models, where models are trained on a subset of the classes. We ultimately decided not to use these methods due to concerns of the longer training time and inference runtime. We've included details about these experiments in Appendix 8.1.

In addition, we also added some data augmentation schemes to the training data by reordering sentences and randomly deleting part of the text. While these results didn't show improvement in our validation set, we added these augmentations into our training data to perform better on a perturbed test set. Details about our experiments on Data Augmentation are in Appendix 8.2.

4.2 Novel Contribution: Weighted Sliding Window Aggregates

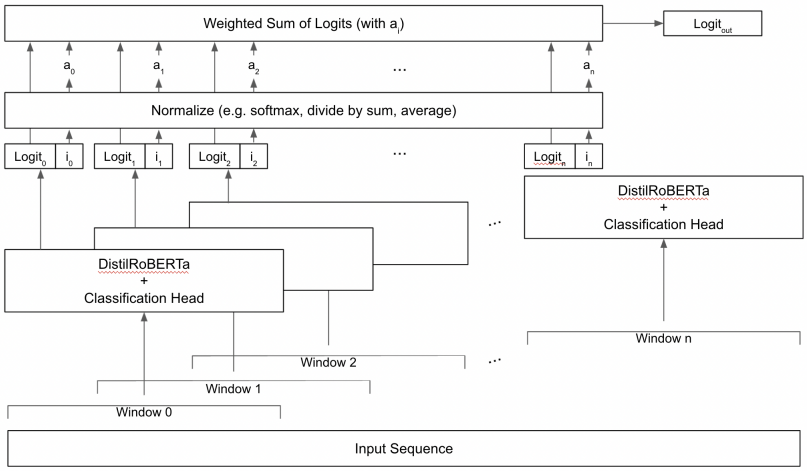

Figure 2: Weighted Sliding Window Method

For our novel contribution to this project, we designed a new and lightweight scheme to aggregate model outputs from the sliding window method. The sliding window method as proposed in the Longtransformers paper[6] is very complex and requires a lot of customized code and tricks to work. A common alternative method of sliding window is to simply run each window of text through the model and average the results to get a final output. In our weighted sliding window method, we propose building off of the simple average sliding window method by appending an importance weight to the logits outputted by the model at each window of text. The importance weights are then sent through some normalization method and are used to sum together the logits from each window to

produce a final output logit. We experimented with a few different methods of normalizing the importance weights such as passing them through a softmax, dividing by the sum of all weights, or passing values through a sigmoid. We compared our results with a baseline DistilRoBERTa model that performs a simple averaging without importance weights.

5 Results

5.1 Finding the best performing model

The baseline DistillBERT model has an accuracy of 63.2% and average star error of 0.43 on a fully balanced validation set.

Table 1 lists the average star error (ASE) and accuracy (Acc) of the four lightweight BERT models we tested. They were trained on the given dataset and evaluated on a uniform validation set.

	ASE	Acc
DistilBERT	0.436	63.2%
SqueezeBERT	0.480	60.9%
ALBERT	0.403	65.4%
DistilRoBERTa	0.356	68.2%

Table 1: Performance of lightweight BERT models on uniform validation set.

Distribution	1-star	2-star	3-star	4-star	5-star
Uniform	20%	20%	20%	20%	20%
Matches Train Set	24.3%	6.9%	6.5%	13.5%	48.8%
Simple Unbalanced	50%	0%	0%	0%	50%

Table 2: Distribution Types

We see that DistilRoBERTa performs the best overall, getting 5% better accuracy, and reducing average star error by 0.79. This is the model we used in the rest of the tests.

To explore further into class imbalances, we trained DistilRoBERTa on datasets with different distributions. In particular, we started with three specific class distributions (refer to Table 2)–uniform, matching the training set, and simple unbalanced–and then took interpolations between these three distributions to create a variety of datasets to train the model on. Trained models were then evaluated on two validation sets: uniform and matching the training distribution. The average star error and accuracy are listed in the Table 3. "Train" refers to the distribution that matches the training set, and "Unbal." refers to the simple unbalanced distribution. To speed up the training process, all datasets only contain 200k datapoints and models were trained for 2 epochs.

		Uniform Val		Train Val	
Distribution Type	Proportions (%)	ASE	Acc	ASE	Acc
Uniform	20, 20, 20, 20, 20	0.376	66.5%	0.285	75.1%
Train	24.3, 6.9, 6.5, 13.5, 48.8	0.528	57.7%	0.281	78.2%
25% Uniform 75% Train	21.1, 16.7, 16.6, 18.4, 27.2	0.449	61.9%	0.273	77.7%
50% Uniform 50% Train	22.2, 13.5, 13.3, 16.8, 34.4	0.491	60.2%	0.279	78.9%
75% Uniform 25% Train	23.2, 10.2, 9.9, 15.1, 41.6	0.389	65.9%	0.265	77.1%
25% Uniform 75% Unbal.	42.5, 5, 5, 5, 42.5	0.529	57.2%	0.280	78.0%
60% Uniform 40% Unbal.	32, 12, 12, 12, 32	0.421	64.1%	0.265	78.0%
90% Uniform 10% Unbal.	23, 18, 18, 18, 23	0.380	66.5%	0.272	76.7%

Table 3: Performance of DistilRoBERTa trained on various class distributions

We see that distribution shift between the training and validation set contributes to model error, but surprisingly the model shows some robustness. On the uniform validation set, the "75%-25% Uniform-Train" model only did slightly worse than the "Uniform" model. Similarly, the "25%-75% Uniform-Train" model performed almost as well as the "Train" model on the "Train" validation set. In addition, models trained on mixtures of the "Uniform" and "Unbalanced" datasets exhibited the same behavior. This suggests that when training, we only need some basic class imbalance to teach the model about a relevant prior that we believe would occur in the real-world, which in this case would be that 1-star and 5-star reviews are more common. However, if we include too much imbalance, the model would rely too much on the prior to maximize accuracy, degrading its performance on unseen data.

Based on these results, we figured that a moderately imbalanced model will be best for our final submission, so we used a model that was trained on a larger version of the "60%-40% Uniform-Unbalanced" dataset, using 600k points instead of 200k.

5.2 Novel Contribution: Weighted Sliding Window Aggregates

As mentioned in section 4.2, we explored ways to improve how logits were aggregated together when using transformers as a sliding window over the input sequence. In particular, we had the model learn importance weights along with each of the logits, and tested three different methods to normalize the weights: softmax, linear normalize, and sigmoid normalize. As a baseline, we added a sliding window model that simply averaged the logits. For linear normalization, the new weights are calculated using the following formula

$$\alpha_i = \frac{w_i - \min(w_i)}{\sum_j (w_j - \min(w_i))}$$

Where α are the weights used to combine the logits, and w are the importance weights calculated with the logits. For sigmoid normalization, the new weights are calculated using the following formula.

$$\alpha_i = \frac{\sigma(w_i)}{\sum_j \sigma(w_j)}$$

Where σ refers to the sigmoid function. Models were both trained and evaluated with a sliding window of size 32 and stride 16. The average star error and accuracy on a uniform validation set are shown in the table 4.

Normalization Method	ASE	Acc
Averaging	0.413	64.2%
Softmax	0.435	63.2%
Linear Normalize	0.429	63.4%
Sigmoid Normalize	0.429	63.5%

Table 4: Various Normalization Methods for Sliding Window on uniform validation set

While we expected importance weights to add more flexibility and expressiveness to the sliding window, we see that it does not perform as well as a simple average. We speculate that the normalization methods used may have some undesirable properties. Softmax uses exponentiation, which blows up differences between each importance weight, so it is hard to combine logits from reviews that have multiple important sections. Linear normalization requires picking out a minimum value, and since any of the windows can contain the miniumum value, it may add unwanted variance during backpropagation. Sigmoid normalization may suffer from saturation at values with very large magnitudes, which would mute the backpropagation signal.

In addition to quantitative testing, we also created some visualizations to see if the model exhibited some extra flexibility with sliding windows. Let's look at the following review: *"Great service. Food is absolutely amazing too. Ask for Ben. He is funny and you may possibly request him to sing for you! :) This food is garbage and disgusting"*

We can first see which words are associated with which star ratings as seen below:

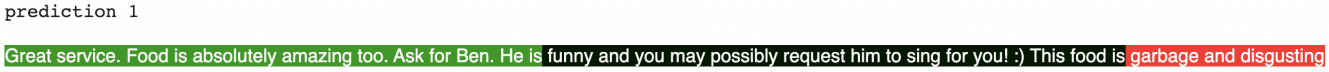

Figure 3: Visualization of Di erent Sliding Window Predictions

The meanings of the colors are in appendix 8.3. As we can see, the first part of the review ("Great service. Food is absolutely amazing too.") is considered a five star rating according to the model. However, the last part of the review ("garbage and disgusting") is considered a one star rating. Ultimately, the weighted average of the model output is a one star rating. This make sense because in the second visualization below, we see that most of the importance weight for the model's prediction was placed on the latter half of the review.

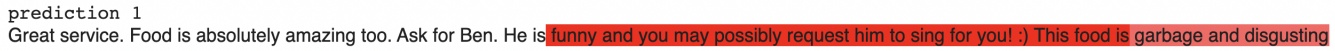

Figure 4: Visualization of Di erent Sliding Window Importance Weights

Once again, the meanings of the colors are in appendix 8.3.

6 Conclusion/Lesson Learned

Overall in this project, we were able to achieve two objectives. First, we utilized common techniques in NLP to iterate on our baseline model to achieve the best metrics on a test set. Through this process, we explored different architectures, data distribution, data augmentation, and ensemble methods. Second, we proposed a novel idea of adding importance weights to existing sliding window methods to perform a weighted average when combining outputs from all of the sliding windows. While our novel contribution did not improve over the existing method of simple averaging, we found that the importance weights from our model can be used to identify different tones in different sections of a text through our visualizations.

One main lesson we learned from this project was that novel and new ideas will often fail to improve on existing ones. In hindsight, this should've been quite obvious as researchers will often spend months experimenting on new ideas without any significant results. While we had some reasoning to back our novel idea of introducing importance weights to existing sliding window methods, we were ultimately unable to improve on the existing averaging method.

7 Team Contributions

We felt that we all equally contributed to this project. Below, we outline some specific parts we each worked on, but we generally helped out with tasks as needed.

Joe Zou 33%

I helped set up the initial codebase and run our baseline models. I had to do some coding/debugging to set up Weights & Biases and lazy data loading support. After that, I helped code, run experiments, and analyze results for Hyperparameter tuning, Decision Tree/Ensemble Models, and our novel Weighted Sliding Window method.

Brian Zhu 33%

I initially started with testing the various BERT models and writing the prediction code for ensembles. Later, I went into depth on comparing class distributions by creating the datasets and training the models on them. I edited the DistilRoBERTa model to support sliding windows and wrote the code for custom aggregation.

Tejasvi Kothapalli 33%

I began with data exploration and processing and understanding the distribution of star ratings and other characteristics about the text reviews. I then moved into creating augmented datasets and training on them. I also worked on creating an evaluation pipeline for all our models. I finally worked on creating a visualizer for our sliding window technique.

8 Appendix

8.1 Ensemble and Decision Tree Models

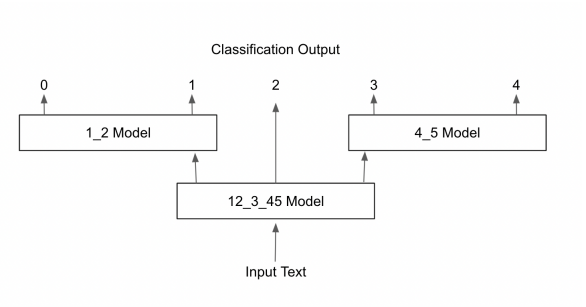

Figure 5: Decision Tree of Models

Figure 5 displays a diagram of our decision trees of models. The decision tree of models consists of three "expert" models which each serve a specific role: the first model distinguishes if the text should be a 1/2-star review, 3-star review, or a 4/5-star review. Depending on the first model's prediction, we may feed the text through another one of two expert models that are designed to distinguish between 1 vs 2 star reviews or 4 vs 5 star reviews. While our preliminary work here showed some promising results from experimenting with ensemble models, we ultimately decided not to use them because each model in the ensemble took 2+ hours to train and we had concerns about making the runtime constraint when performing inference.

8.2 Data Augmentation Experiments

Data Aug Method	ASE	Acc
Baseline(Clean)	0.356	68.2%
Reorder Sentence	0.365	67.5%
Remove Sentence	0.363	67.6%

Table 5: Various Normalization Methods for Sliding Window on uniform validation set

We implemented two different data augmentation schemes of reordering sentences and removing sentences at random from the training data. While the metrics for these models aren't as good as our baseline, it is important to note that we are still evaluating on a clean dataset without data augmentation. We ultimately chose to inject some data augmentation into the training data of our final model as we anticipate that it may help with the perturbations of the hidden test set.