COMP 8085 - Project 2 - Group 3 Report

Mohammed Bajaman
School of Computing and Academic studies
British Columbia Institute of Technology
Burnaby, British Columbia, Canada
`mbajaman@my.bcit.ca`

Jacob Pauls
School of Computing and Academic studies
British Columbia Institute of Technology
Burnaby, British Columbia, Canada
`jpauls14@my.bcit.ca`

Keith Liu
School of Computing and Academic studies
British Columbia Institute of Technology
Burnaby, British Columbia, Canada
`kliu107@my.bcit.ca`

Abstract

This project explores various models that perform sentiment analysis when trained and tested on the Yelp review data set.The task was to perform classification of rating scores on reviews giving a numerical score between one and five based on sentiment analysis training. The models chosen for this task are *RoBERTa*, *Naive Bayes* and a *Random Forest Classifier*. The report explores relevant literature going over different approaches for sentiment analysis. Two additional experiments were conducted with the first experiment being the base implementation of the model. The experiments were conducted to further test and explore the performance of the model. In each base experiment, data pre-processing was conducted to clean irrelevant information. The results for the base experiment in all three models achieved an accuracy and macro-f1 score above 50%. The RoBERTa model performed the best with an accuracy of 60% while the Naive Bayes and Random Forest Classifier models performed relatively the same with accuracies of 56% and 54% respectively.

1 Task Definition

1.1 Problem Statement

Analyzing the nature of sentiment across user-supplied reviews and making accurate predictions about their nature, helpfulness, and content is pivotal for beginning to leverage the insights that textual reviews can provide. Furthermore, leveraging artificial intelligence for analyzing sentiment provides merit for both scientific and business contexts and can be applied to a variety of tasks, such as filtering unhelpful reviews, finding patterns in user feedback, as well as generating meaningful actions for the review context. These experiments leverage different types of models to perform the textual processing and training required for artificial intelligence to leverage sentiment analysis to perform a series of non-binary classification tasks.

1.2 Scope

The software built for each experiment can train, validate, and test either a probabilistic Naive Bayes model, a pre-trained RoBERTa transformer model, or a random forest model for performing non-binary classification on the *Yelp [1]* reviews data set. Once a model is trained and run in inference,

Course project report submitted in fulfillment of requirements for COMP 8085, BCIT.

each model type can make predictions for the number of stars associated with the sentiment of a textual review.

1.3 Evaluation Metrics

For evaluating each method we leveraged the F1 scores of each model as well as a self-defined metric for the total value predicted for each metric in a given classification task.

The F1 score is a numerical score between zero and one that represents a percentage of accuracy for each class that our model predicts. Each F1 score represents the culmination of the precision and recall of a model, which uses the number of true positives and negatives as well as false positives and negatives in order to capture a valid metric, which is described further in *Sokolova et al. [2].* This is represented in our reports at both the macro and the micro levels in order to ensure valid metric analysis on balanced and imbalanced sets of textual test data. As a result, both prediction and recall are bounded by the number of predictions made by the model on a particular data set. The overall accuracy of each model was also used to determine the overall value of its net true predictions on unseen data.

$$precision = \frac{TP}{TP + FP} \tag{1}$$

$$recall = \frac{TP}{TP + FN} \tag{2}$$

$$accuracy = \frac{TP + TN}{TP + TN + FP + FN} \tag{3}$$

$$F1 = 2 \times \frac{precision * recall}{precision + recall} \tag{4}$$

We also define *ya* which is the accuracy of the model's predicted values with regard to the values in the original Yelp data set. To calculate *ya* we leveraged the sum of all predicted values (PV) over the sum of all the actual values (AV) to achieve a percentage of accuracy with the actual Yelp data set. This score has a minimum of the number of reviews multiplied by the lowest score (model predicts all values of 1) and has a maximum of the number of reviews multiplied by the highest score (model predicts all values of 5).

$$ya = \frac{\sum(PV)}{\sum(AV)} \quad ya_{min} = \frac{\sum(1 \cdot n)}{\sum(AV)} \quad ya_{max} = \frac{\sum(5 \cdot n)}{\sum(AV)} \tag{5}$$

1.4 Dataset

The data set used for sentiment analysis was retrieved from *Yelp [1]* and has 6,990,280 records that each reflect a review posted on the Yelp platform. In our use of the data set, we extracted 500,000 records (representing 7% of the original data set) that accounted for an even distribution of counts for the *"stars"* feature (a numerical value between 1-5) which comprised our base set of the data. From there, we derived a training set from 450,000 records of the extracted base data and the remaining 50,000 records comprised a separate withheld inference set. In tandem to ensuring that the data had an even distribution for the *"stars"* feature, we also filtered records with text containing less than fifty characters and more than two-hundred and fifty size characters.

2 Infrastructure

2.1 Hardware

Models were trained on hardware with a variety of specifications over the course of the experiment. The lowest-performing machine had 8GB of RAM, a six-core AMD CPU, and an NVIDIA GeForce 750TI GPU. Other machines averaged 16GB of RAM, eight-core CPUs, and variations of integrated or discrete graphics cards, with the highest-end GPU being an NVIDIA GeForce 2070.

Additionally, training for the neural network was specifically offloaded to a Google Cloud Compute Engine with an NVIDIA V100 GPU, 8 CPU cores, and 30GB of RAM in order to reduce the training and inference time for the network on our extracted training and withheld test sets.

2.2 Software

The Python programming language was used for conducting the experiments, training models, and running the project's multi-label classification tasks. Within the Python ecosystem, we used transformers (RoBERTa pre-trained neural network), pytorch (fine-tuning the neural network), scikit-learn (out of the box model), pandas (data cleaning, framing, and analysis), NLTK (text pre-processing), as well as matplotlib (graphing and visualization of data). Additional dependencies used, including any co-dependencies, are listed in the project's associated 'requirements.txt' file.

Each experiment has been bootstrapped with a GNU Makefile containing commands for setting up the environment, training models, and running pre-trained inference models. Installation of the virtual environment can be easily performed using `make install` and the environment can be cleaned with `make clean`. Each command for executing a classification task follows the convention `<classifier-abbreviation>-<task>` and optionally contains an `inf` suffix to load and run a model in inference mode if it exists. List of classifier abbreviations are: `nn` for the Neural Network model, `prob` for the Probabilistic model and `box` for the Out-Of-Box Random Forest Classifier model. The output directories and names of the model's output by the scripts can be changed in the `mod.utils.constants` module of the project. Finally, commands to re-extract our training and test sets from the original data set have also been supplied, with commands prefixed by `gen` if necessary.

3 Approach

3.1 Datasets

The training set used for each model was made up of 450,000 records split from a derived base set of 500,000 records which were extracted from the original *Yelp [1]* data set. When run in inference, each model was evaluated on the withheld set of 50,000 records.

While inference remained the same across model types, the approach to training each model type varied. For the random forest as well as the probabilistic Naive Bayes model, one round of training and validation was performed with an 80/20 (train/test) split on the training data. In contrast, the RoBERTa neural network performed an 80/18/2 (train/test/validation) split on the data, where validation occurred after each epoch of tuning to report results.

pre-processing the *"text"* feature of the *Yelp [1]* data set also varied depending on the type of model. For the random forest as well as the probabilistic Naive Bayes model, pre-processing was performed by stripping punctuation, lemmatizing the text, removing non-alphanumeric characters, replacing URLs, removing English stopwords, and lowercasing the text. The same pre-processing is applied to the test set and no records are dropped or invalidated. In the implementation of the neural network, pre-processing mainly relied on the tokenization process supplied by the transformers library's default `RobertaTokenizer`. Further experimentation on pre-processing methods for each model and their results is explained further in Section 5.

3.2 Task Selection

This system approaches classifying the *"stars"* feature in the Yelp reviews data set as a non-binary classification task, where ratings are calculated as numerical values inclusively between one and five. The only exception is for the neural network classification task where labels were indexed from zero in order to align with the requirements of PyTorch's `CrossEntropyLoss` function which requires inputs between zero and the number of labels.

We chose to identify the task as a classification task because the *"stars"* feature of the Yelp data set has discrete labels between one and five, making it more consistent for a model to predict. This especially helped with the third experiment of the RoBERTa model (Section 5.1.3) which leveraged Coursera data that also had review scores bounded by one and five.

3.3 Classification Methods

Various modeling techniques were used to perform non-binary classification on the Yelp reviews data set, namely, the models selected were either probabilistic, neural network or a generic non-parametric type. The selected models were each picked as a result of their performance after training on the derived training set as well as their accuracy, F1, and *ya* scores on the withheld inference set. These methods included a fine-tuned RoBERTa transformer-based model (neural network), a Naive Bayes model (probabilistic), and a random forest classifier (non-parametric).

3.3.1 RoBERTa

Originally proposed in *Liu et al. [3]*, RoBERTa (Robustly optimized BERT approach) is a variation of the BERT (Bidirectional Encoder Representations for Transformers) language model that applies different modifications to a variety of the original model's hyper-parameters in an attempt to optimize it. Both models are based on the transformer from *Vaswani et al [4]*, a language model architecture that applies the concept of attention to language modeling.

Prior to using transformers for language modeling and neural machine translation, recurrent neural networks (RNNs) were the ideal architecture for completing tasks in natural language processing. These networks allow prior outputs to be used in a cyclic nature as inputs while maintaining the hidden states prevalent in neural networks. Doing so allows them to process inputs of any length, share calculated weights throughout training time, and persist historical data throughout computations in any prior layer's hidden state. However, the primary disadvantage of RNNs is an inability to predict future data. For instance, the requirement for each hidden state to only "see" or access the encoding for a word in a prior hidden state indicates that the influence of an encoding means significantly less when processing long strings or long sequences of textual data.

In contrast, transformers address this primary disadvantage by introducing attention, where each encoding layer extracts information from the sequence (as a weighted sum of past states) and provides them as being visible for the decoder to understand and properly weigh the entire sequence. Doing so allows each layer to perform self-attention and the decoder can assign greater weights to particular inputs in relation to specific outputs. The architecture for the traditional transformer is an encoding and decoding stack that each has an identical number of layers. Each encoding layer has sub-layers, one multi-head self-attention mechanism, and another position-wise feed-forward network. Similarly, each decoding layer has two sub-layers as well as a third multi-head attention layer that exists over the output of the encoder.

The heart of transformers (as described previously) is in the multi-head self-attention mechanisms that exist as sub-layers in both the encoder and decoder stacks. Each of these sub-layers consists of vectors for queries and key-value pairs that correspond to particular outputs which are sent to multiple problem spaces before a final attention computation is performed at each layer. The resultant storage and active projection of these vectors allow the model to retain significance for particular inputs and grasp a variety of attention patterns over potential sequences. With this in mind, transformers have a prominent downside of being computationally expensive to train and test with.

In *Devlin et al. [5]* a bidirectional architecture for transformers was proposed that worked around the unidirectionality of transformers through the use of a masked-language model (MLM) during pre-training. This masked language model supports BERT for context-specific learning by masking particular tokens in encoded input sequences and forcing the model to make predictions on the masked values with attention to the surrounding words. Next, the process of next-sentence prediction (NSP) is also used to facilitate sequence-based attention for the model. This process splits the available training sentences and joins them into sentence pairs where half the pairs are proper sentence pairs in the original document and the other half are improper (or random) sentence pairs. For most BERT models, these processes are performed on a large corpus of data to build the self-attention mechanisms in the encoding layer on both word and sequence-based tasks. The use of BERT models then requires a fine-tuning step, where the implementation and a variety of its corresponding hyper-parameters are tuned to particular sets of input sequences and data to accomplish tasks. When being implemented, this means that the fine-tuning step is often the main implementation task for clients using the model – since the pre-trained corpus and parameters already exist in the encoding layer, the model only requires the addition of a single output layer tuned to a particular task.

In comparison to BERT, the implementation for RoBERTa is heavily optimized. After analysis of the original BERT model, researchers realized it was *"significantly undertrained"* as a result of the original data having reduced sequence lengths and turned to training it on more data, with larger batch sizes, for a longer period of time while only using full sequence lengths. Perhaps most significantly, the design for pre-training the RoBERTa model doesn't include BERT's NSP task, which yielded slightly better results for completing a variety of tasks.

3.3.2 Naive Bayes

The Naive Bayes classifier is a supervised machine learning classification model useful for sentiment analysis. This probabilistic classifier is based on Bayes' theorem and Bayes' rule where we explore the probability of a cause given some effect. An important fact to note here is that this probability calculation does not depend on the number of unobserved effects. This is important to keep in mind for the situation that the model encounters unobserved effects.

The Naive Bayes equation is as follows:

$$\begin{aligned}
\mathbf{P}(\text{ Cause } \mid \mathbf{e}) &= \alpha \sum_{\mathbf{y}} \mathbf{P}(\text{ Cause })\mathbf{P}(\mathbf{y} \mid \text{ Cause }) \left(\prod_j \mathbf{P}\left(e_j \mid \text{ Cause }\right) \right) \\
&= \alpha \mathbf{P}(\text{ Cause }) \left(\prod_j \mathbf{P}\left(e_j \mid \text{ Cause }\right) \right) \sum_{\mathbf{y}} \mathbf{P}(\mathbf{y} \mid \text{ Cause }) \\
&= \alpha \mathbf{P}(\text{ Cause }) \prod_j \mathbf{P}\left(e_j \mid \text{ Cause }\right)
\end{aligned} \tag{6}$$

In words, this can be explained as: for each possible cause, multiply the probability of the cause by the product of the conditional probabilities of the observed effects given the cause, then normalize the result.

Given our data set, each review serves as the cause with the effects being one of the five labels: 1-star, 2-star, 3-star, 4-star and 5-star. While the conditional probabilities are calculated for each word given each label where j is the jth word in the sentence. We then select the highest probability and assign the review the appropriate star label. The model is considered naive as it assumes that each variable, or in this case each word, is independent.

With Naive Bayes a common issue is finding a solution to deal with words that the model isn't trained on as we don't have its conditional probability. We have a few approaches to consider; one is to ignore the word, essentially giving it a probability of 1.0 for every single label. This seems logically incorrect. Another approach is to consider the number of occurrences which would be 0 giving it a probability of 0 for every label. This can't work either since the formula mentioned above is a product and would set the probability for the review to be 0 for every label. Tackling this issue is described in further detail in the Experiments and Analysis section and explains how Laplace smoothing is used.

3.3.3 Random Forest

Random forest is an ensemble method for machine learning that builds numerous decision trees which are each trained using random subsets of data. Once training is complete and predictions need to be made, the classifier combines the results from the "forest" to make a final prediction.

The process of dividing the data into random subsets to create multiple decision trees is frequently viewed as a bagging (bootstrap aggregating) classification technique in ensemble methods of learning. This method seeks to improve the overall performance of one classifier by training multiple instances of it on subsets of the original data. This differs from boosting methods, which train "weak" versions of an original estimator with varying sets of data and combine the resulting classifiers from training into a single "boosted" one.

4 Literature Review

The literature associated with prior attempts of performing sentiment analysis on the Yelp reviews data set or other related data sets suggested that using random forests, Naive Bayes, and transformer-based models would be suited for the task.

In *Guda et al. [6]*, a variety of models are analyzed as candidates for performing classification tasks on the Yelp reviews data sets. When putting the family of BERT and attention-based models under scrutiny, it's argued that signal words crowd the reviews data set. As a result, attention-based models that focus on specific terms and their relevance in long sequences were hypothesized to perform better. In their results, it's reported that the BERT model achieved a 59.96% accuracy when classifying Yelp's *"stars"* feature as a non-binary classification task, noting that the model focused on the words "bad" and "rude" for 1-star samples and "absolute", "perfect", "really", and "great" for 4-star samples.

In *Liu [7]* Naive Bayes models are used for classification tasks on the Yelp data set. When employing Naive Bayes, the results indicate that it's one of the fastest models to train, taking only five seconds on over a million records with four concurrent workers while also reporting scores over 60% for both accuracy and F1. Despite their model being a `scikit-learn` implementation, it still proved to be a benchmark model in comparison to logistic regression and SVMs. Additionally in *Goel et al. [8]* use of *SentiWordNet* to calculate sentiment scores for every word helped with analysis with their Naive Bayes implementation giving a score of 58.40% when trained on 1.6 million tweets and tested with 100 recent tweets. Another interesting implementation of Naive Bayes explained in *Joyce et al. [9]* utilizes information gain and weights where words in more useful reviews are given higher probabilities resulting in an overall accuracy of 75%.

Li et al. [10] implemented a random forest classifier for a binary classification task on the Yelp reviews data set to acceptable results of over 70% accuracy. Similarly, *Guda et al. [6]* reported over 75% for binary tasks and around 50% for non-binary classification, further exemplifying its candidacy.

Most of the reviewed literature also highly regarded the training, testing, and validation times for models aside from attention-based models. This allowed us to prepare adequately for the duration of the training, especially on our extracted set of 450,000 reviews from the Yelp data set. Despite a high training time, these models often had the highest accuracies, with *Guda et al. [6]* reporting over 60% accuracy for modifications to the BERT model and *Liu [7]* reporting over 70% for most applications of BERT and RoBERTa models for classifying Yelp review data.

5 Experiments and Analysis

5.1 RoBERTa

5.1.1 Experiment 1: Strengths, Weaknesses, and Hyper-paramter Tuning

The implementation for the RoBERTa neural network in our experiments relied on the *roberta-base* pre-trained transformer from the hugging face transformer libraries. From the preset, we decided to use the `RobertaTokenizer` and the `RobertaForSequenceClassification` preset models to perform sequence classification on the Yelp data set. As the `RobertaTokenizer` performs injection of tokens and operations on sequences manually, we opted to avoid pre-processing in our first iteration and only use the built-in tokenizer to combat the length for tuning attention-based models. Once the records were tokenzied, we set up a main set of hyper-parameters that were adjusted manually over multiple runs on small data sets. These hyper-parameters included the batch size for each set of data, the network's learning rate, as well as the number of epochs to train the network with (rounds of training/validation).

Quickly into training we experienced the disadvantages associated with the length required to properly train the neural network, with our times quickly approaching almost 5 hours on almost industrial GPUs training on 450,000 records. This forced us to tweak away from some general hyper-parameters and experiment with the properties of the neural network.

Table 1: Classifier - RoBERTa - NVIDIA V100 (Experiment 2 - 10.43 minutes)

	precision	recall	f1-score	support
1-star	0.70	0.76	0.73	10037
2-star	0.53	0.49	0.51	10037
3-star	0.53	0.48	0.51	9903
4-star	0.53	0.53	0.53	9991
5-star	0.68	0.75	0.72	10032
accuracy			0.60	50000
macro avg	0.60	0.60	0.60	50000
weighted avg	0.60	0.60	0.60	50000

$$ya = 1.009015048271192$$

In experimenting with the hyper-parameters by hand, we uncovered that setting the learning rate higher (especially when training for less epochs) allows the model to learn the data faster but risks failing to generalize it well. As a result, when our learning rates started low (ex: 1e-5) we had to wait numerous epochs for the model to start recognizing patterns in the data. While experimenting with the batch size, we also reasoned that having a higher batch size would expose the model to more data in each batch, but would also increase the memory cost and further increase the time for the model to converge. Finally, the number of epochs increased the number of rounds of training and allowed the model to better learn the data at the risk of overfitting the training set. In the end, we found the best parameters to maximize the time spent learning were a batch size of sixteen, learning rate of 1e-3, and five epochs. Finally, The other two experiments use these settings as well.

At the end of hyper-parameter tuning, we were especially excited to note that the model achieved an accuracy of 60% for the non-binary classification task and had a *ya* score of almost one. This infers that the model was fairly correct across the distribution of data for predicting the right amount of stars in the data set.

5.1.2 Experiment 2: Additional pre-processing

After tuning the hyper-parameters without manual pre-processing, we became curious about what the results would be with our pre-processing stack applied to the training and test data sets.

Table 2: Classifier - RoBERTa - NVIDIA 2070 (Experiment 2 - 9.2 minutes)

	precision	recall	f1-score	support
1-star	0.58	0.69	0.63	10037
2-star	0.41	0.34	0.37	10037
3-star	0.41	0.33	0.36	9903
4-star	0.41	0.43	0.42	9991
5-star	0.56	0.62	0.59	10032
accuracy			0.48	50000
macro avg	0.46	0.47	0.47	50000
weighted avg	0.47	0.48	0.47	50000

$$ya = 0.998102513705784$$

After retraining the model and running it in inference, we weren't as convinced with our results as with the first experiment. As the implementation for the `RobertaTokenizer` is a black-box, we were unsure if our manual pre-processing stack interfered with what it normally does for tokenizing the data. Alternatively, we also considered that the removal of English stopwords (which drastically reduced the runtime for the model) redacted from the model's attention capability, lowering the model's ability to identify patterns in the input sequences.

5.1.3 Experiment 3: Predicting Coursera Reviews Data

In our final experiment, we extracted course review data from a created *Coursera [11]* data set and attempted to perform non-binary classification for course reviews.

In analyzing the data set, it's worth noting that it consisted of 140,320 records that were directly sent to our pre-trained model from experiment 1 (Section 5.1.1) to run in inference. Furthermore, the data set retrieved from *Coursera [11]* was extremely imbalanced towards five-star ratings, which comprised more than 75% of the overall data set.

Table 3: Classifier - RoBERTa - NVIDIA 750TI (Experiment 3 - 326.84 minutes)

	precision	recall	f1-score	support
1-star	0.13	0.46	0.20	2867
2-star	0.24	0.37	0.29	2554
3-star	0.17	0.51	0.25	5923
4-star	0.23	0.29	0.25	22460
5-star	0.90	0.67	0.77	106516
accuracy			0.59	140320
macro avg	0.33	0.46	0.35	140320
weighted avg	0.73	0.59	0.64	140320

$$ya = 0.8719468970786305$$

In analyzing the results, our model performed exceptionally well at classifying the five-star ratings in the Coursera data set, achieving a higher general accuracy on the cumulative set of data. However, when making predictions over the other categories, the model performed rather poorly, achieving a macro F1 score of only 35%. After reflecting on the data further, this could have occurred because the original training set only contained English records which was drastically different from the Coursera set which contains reviews in Russian, Spanish, and Mandarin. Furthermore, our original training set only containing text sequences between fifty and two-hundred and fifty six characters may have also affected performance when making predictions on the Coursera set, which had varying lengths of text data.

5.2 Naive Bayes

5.2.1 Experiment 1: Strengths, Weaknesses, and hyper-parameter Tuning

For our Naive Bayes implementation, we extract the top 3000 features (words) using a `CountVectorizer` from the `scikit-learn` library. Through several training runs we found that after a certain number of extracted features, the accuracy of the model doesn't improve much. A feature set of 3000 proved to be a stable point after which increasing it resulted in 0.01% change in accuracy. Conditional probabilities are calculated and put in a nested dictionary structure where the first level contains the classes (1-5) and the second level contains the selected features with their respective counts for that class. This in turn raises the issue of zero probabilities occurring causing undesirable results in probability calculation. To tackle this issue, Laplace smoothing from *Jayaswal [12]* is applied. This is a smoothing technique that handles the problem of zero probabilities in Naive Bayes. Using Laplace smoothing the probability of a word can be represented as:

$$P\left(w^{\prime} \mid \text{ positive }\right)=\frac{\text{number of reviews with } \mathrm{w}^{\prime} \text{ and } \mathrm{y}= \text{ positive }+\alpha}{\mathrm{N}+\alpha * K} \tag{7}$$

Where *w'* is the word, α represents the smoothing parameter, *K* is the number of labels and *N* is the total number of reviews for the label (in this example represented as positive). As alpha increases the probability reaches an equal distribution across all labels and most models use alpha=1.

In addition to smoothing the text is also pre-processed to remove URLs, numbers, words with numbers, and non-English words. Lemmatization is performed by using POS (part-of-speech) tags which are converted from NLTK to wordnet tags to support the Wordnet lemmatizer. The built-in NLTK perceptron tagger reports an accuracy of 93.64% (100 iterations) which helps further with lemmatization. A key takeaway here is that lemmatization needs to occur early on in the pre-processing step. This is to ensure accurate POS tags are generated for the words. Incorporating the removal of stop words and possibly other pre-processing techniques results in a loss of context and grammar which then causes poor performance with the NLTK tagger (marks most words as nouns).

In addition to this, several runs of training the model showed that accuracy was largely influenced by how pre-processing was conducted on the data set. With proper pre-processing, training the model on 360,000 records, max features of 3000, and a smoothing value of 10. Running the model in inference against held-out test data (50,000 records) provided the below results which serve as our base model and first experiment:

Table 4: Classifier - Naive Bayes (Experiment 1)

Max Features: 3,000				
Smoothing Value: 10				
	precision	recall	f1-score	support
1-star	0.65	0.69	0.67	10037
2-star	0.47	0.46	0.47	10037
3-star	0.49	0.47	0.48	9903
4-star	0.50	0.56	0.53	9991
5-star	0.70	0.62	0.66	10032
accuracy			0.56	50000
macro avg			0.56	50000

On unprocessed data, the model performed poorly giving an accuracy result of 40% and a macro-f1 average of 36% showing that pre-processing is important for the model to perform better.

Table 5: Classifier - Naive Bayes - Unprocessed data

Max Features: 4,000				
Smoothing Value: 10				
	precision	recall	f1-score	support
1-star	0.60	0.65	0.63	10037
2-star	0.36	0.64	0.46	10037
3-star	0.30	0.48	0.37	9903
4-star	0.35	0.16	0.22	9991
5-star	0.88	0.08	0.15	10032
accuracy			0.40	50000
macro avg			0.36	50000

Another weakness of this model was multi-class classification which it struggled with. This is highlighted further in Experiment 3 where classes were reduced to 2 (positive and negative) and in turn resulted in a 30% accuracy increase.

5.2.2 Experiment 2: Scaling Max Features with the Size of Training Data

For this experiment, the model was trained with varying max features and alpha values. A low value for max features reduced the accuracy of the model resulting in accuracy as low as 36% and a macro-f1 average of 27%. After a certain value of max features, further increases to this metric resulted in minimal change to accuracy. The classification results of this experiment can be found in Appendix A.

In addition to tuning the max features value, making changes to the smoothing value also didn't contribute to better performance as generally setting a smoothing value of 1 is ideal, and increasing this results in probabilities reaching equilibrium for unknown features across the classification domain.

5.2.3 Experiment 3: Using a Binary Classification Domain

For the final experiment, we reduced the classification domain to two classes; namely positive and negative. We modified the training data to add another class called sentiment which was filled with either 'positive' if the star rating given was 4 or 5 and 'negative' if it was within 1-3. The model was then tasked with classifying test data as positive and negative and performed much better compared to the multi-class classification problem earlier.

Table 6: Classifier - Naive Bayes (Experiment 3)

Max Features: 3,000
Smoothing Value: 10

	precision	recall	f1-score	support
positive	0.87	0.91	0.89	29977
negative	0.85	0.79	0.82	20023
accuracy			0.86	50000
macro avg			0.85	50000

5.3 Random Forest

5.3.1 Experiment 1: Strengths, Weaknesses, and Hyper-paramter Tuning

In this experiment, we implemented the random forest model from the `scikit-learn` library. The data was pre-processed using the same stack as the Naive Bayes model as well as through a `CountVectorizer` to convert the text reviews into a matrix of token counts.

hyper-parameter tuning was performed using a grid search cross-validation algorithm provided by the `scikit-learn` library, resulting in a gini-based random forest classifier with *log2* being used for max features, and no bootstrapping for samples used in the trees.

Table 7: Classifier - Random Forest (Experiment 1 - 1.4 minutes)

	precision	recall	f1-score	support
1-star	0.60	0.78	0.68	10037
2-star	0.48	0.38	0.42	10037
3-star	0.48	0.41	0.44	9903
4-star	0.48	0.43	0.45	9991
5-star	0.62	0.71	0.66	10032
accuracy			0.54	50000
macro avg	0.53	0.54	0.53	50000
weighted avg	0.53	0.54	0.53	50000

$$ya = 1.0157568860166104$$

As the classification report for the model shows, the results indicate that the model performed only slightly better than random guessing which is an f1-score of 0.5.

5.3.2 Experiment 2: Tf-Idf Vectorization

For this experiment, we tried pre-processing the data using `TfidfVectorizer` to convert the text data into numerical vectors to see if that would impact the results.

Table 8: Classifier - Random Forest (Experiment 2 - 0.832 minutes)

	precision	recall	f1-score	support
1-star	0.56	0.78	0.65	10037
2-star	0.45	0.35	0.39	10037
3-star	0.42	0.32	0.36	9903
4-star	0.49	0.46	0.48	9991
5-star	0.60	0.68	0.64	10032
accuracy			0.51	50000
macro avg	0.50	0.52	0.50	50000
weighted avg	0.50	0.51	0.50	50000

$$ya = 1.01939455266024267$$

As shown in the classification report, the results did not change much except for a slight decrease in scores compared to the prior experiment.

5.3.3 Experiment 3: Comparison to Support Vector Classification

For our final experiment on the random forest classifier, we decided to compare it to the support vector classifier provided by the `scikit-learn` library.

In short, support vector machines (SVM) work by attempting to find a hyperplane that separates different classes in a feature space. One of the noted weaknesses of SVMs is the duration required for them to train on large sums of data. Our implementation for the support vector machine used a linear kernel as well as a C-value of one.

Table 9: Classifier - Support Vector Machine (Random Forest Experiment 3 - 8.832 minutes)

	precision	recall	f1-score	support
1-star	0.52	0.58	0.55	10037
2-star	0.35	0.37	0.36	10037
3-star	0.33	0.33	0.33	9903
4-star	0.37	0.34	0.36	9991
5-star	0.55	0.49	0.52	10032
accuracy			0.42	50000
macro avg	0.42	0.42	0.42	50000
weighted avg	0.42	0.42	0.42	50000

$$ya = 1.0422693812863628$$

After viewing the results, our implementation for the support vector machine greatly struggled in comparison to the original random forest model. This could have been due to the size of the training sample or because of less time spent attempting to tune the hyper-parameters.

6 Contributions

6.1 Jake (33.3%)

- Implemented the RoBERTa pre-trained neural network and reviewed it's relevant literature

- Wrote report sections on the RoBERTa model/attention-based models
- Wrote logic for picking models using default python, pytorch, and hugging face facilities
- Organized and implemented the *Makefile* with commands required to train models and test inference models

6.2 Mohammed (33.3%)

- Implemented the Naive Bayes model and reviewed it's relevant literature
- Helped with tuning and testing RoBERTa model
- Added pre-processing steps to data set used for training and testing models
- Wrote report sections on the Naive Bayes model

6.3 Keith (33.3%)

- Implemented the random forest model and reviewed it's relevant literature
- Wrote report sections surrounding the random forest model and it's results

6.4 Obstacles and Roadblocks

Our main difficulties with this project were with the duration for running the models, especially the neural network. Each model had time requirements for training on a sizable amount of data in order for it to generalize, making tasks such as hyper-parameter tuning lengthy and time consuming. For the neural network, we even went to the extent of using a virtual machine with a better GPU to circumvent this. Additionally, our group had a hard time converting our models from a single non-binary classification task to a multi-label non-binary classification task to classify the *"useful"*, *"funny"*, and *"cool"* features of the Yelp data set, especially for the neural network.

References

[1] Yelp. (2021). *Academic Reviews Dataset, United States, 2004-2021*. [Dataset]. `https://www.yelp.com/dataset`.

[2] Sokolova, Marina & Japkowicz, Nathalie & Szpakowicz, Stan. (2006). *Beyond Accuracy, F-Score and ROC: A Family of Discriminant Measures for Performance Evaluation*. AI 2006: Advances in Artificial Intelligence, Lecture Notes in Computer Science. Vol. 4304. 1015-1021. 10.1007/11941439_114.

[3] Liu, Y., Ott, M., Goyal, N., Du, J., Joshi, M., Chen, D., Levy, O., Lewis, M., Zettlemoyer, L. & Stoyanov, V. (2019). Roberta: A robustly optimized bert pretraining approach. *arXiv preprint arXiv:1907.11692*.

[4] Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, L. & Polosukhin, I. (2017). Attention is all you need. *Advances in neural information processing systems, 30*.

[5] Devlin, J., Chang, M. W., Lee, K., & Toutanova, K. (2018). Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*.

[6] Guda, B., Srivastava, M., & Karkhanis, D. (2022). Sentiment Analysis: Predicting Yelp Scores. *arXiv preprint arXiv:2201.07999*.

[7] Liu, Z. (2020, December). Yelp Review Rating Prediction: Machine Learning and Deep Learning Models. *arXiv preprint arXiv:2012.06690*.

[8] Goel, A., Gautam, J., & Kumar, S. (2016, October). Real time sentiment analysis of tweets using Naive Bayes. *2016 2nd International Conference on Next Generation Computing Technologies (NGCT)*. Presented at the 2016 2nd International Conference on Next Generation Computing Technologies (NGCT), Dehradun, India. doi:10.1109/ngct.2016.7877424

[9] Joyce, B., & Deng, J. (2019, December). Sentiment analysis using naive Bayes approach with weighted reviews - A case study. *2019 IEEE Global Communications Conference (GLOBECOM)*. Presented at the GLOBECOM 2019 - 2019 IEEE Global Communications Conference, Waikoloa, HI, USA. doi:10.1109/globecom38437.2019.9013588

[10] Li, Y., Liu, Y., Chiou, R., Kalipatnapu, P. (n.d.). Prediction of Useful Reviews on Yelp Dataset. *University of California, Berkeley*.

[11] Coursera. (2017). *100K Coursera's Course Reviews Dataset*. [Dataset]. `https://www.kaggle.com/datasets/septa97/100k-courseras-course-reviews-dataset?select=reviews_by_course.csv`.

[12] Jayaswal, V. (2020, November 22). *Laplace smoothing in naïve Bayes algorithm.* Medium. `https://towardsdatascience.com/laplace-smoothing-in-na%C3%AFve-bayes-algorithm-9c237a8bdece`.

A Appendix

A.0.1 Classification Results of Naive Bayes Experiment 2

Table 10: Classifier - Naive Bayes (Experiment 2)

Max Features: 4,000
Smoothing Value: 10

	precision	recall	f1-score	support
1-star	0.65	0.69	0.67	10037
2-star	0.48	0.44	0.46	10037
3-star	0.49	0.47	0.48	9903
4-star	0.49	0.58	0.53	9991
5-star	0.70	0.62	0.66	10032
accuracy			0.56	50000
macro avg			0.56	50000

Table 11: Classifier - Naive Bayes (Experiment 2)

Max Features: 2,000
Smoothing Value: 10

	precision	recall	f1-score	support
1-star	0.65	0.68	0.67	10037
2-star	0.47	0.46	0.46	10037
3-star	0.48	0.47	0.47	9903
4-star	0.49	0.55	0.52	9991
5-star	0.70	0.62	0.65	10032
accuracy			0.56	50000
macro avg			0.56	50000