Advances in Artificial Intelligence and Machine Learning; Research 3 (3) 1259-1273 Received 26-5-2023; Accepted 25-7-2023; Published 02-8-2023

Sentiment Analysis: A Systematic Case Study with Yelp Scores

Wenping Wang wenpingw@alumni.cmu.edu
Carnegie Mellon University
Pittsburgh, PA 15213, USA

Jin Han
Amazon Inc, 410 Terry
Ave N, Seattle 98109, WA, USA

Chen Liang
Google Inc, 1600
Amphitheatre Parkway, USA

Tong Chen
Google Inc, 1600
Amphitheatre Parkway, USA

Chengze Fan
Meta Platforms,
1 Hacker Way, USA

Jingxian huang
Meta Platforms,
1 Hacker Way, USA

Corresponding Author: Wenping Wang

Abstract

Sentiment Analysis is a classic and well-defined task for machine learning and natural language processing. Over the years, we have seen much progress in machine learning as a whole and in natural language processing. Given that in commercial applications, we are heavily constrained by cost, throughput and latency, we wonder how better accuracy can be brought about by using complex, high-latency models, than easy, low-latency models that can be deployed in embedded devices and in high throughput scenarios. In this article, we focus on the Yelp Review dataset as a test bench. By predicting Yelp overall ratings based on user review text and other related features, we experiment with various existing machine learning algorithms, from easy logistic regression to BERT embedding-based deep models. We also use ensemble to combine the aforementioned models into a single predictor, seeing if a combination of these models will achieve better performance. Among all the models, we can see that a simple TF-IDF baseline with MLP ensemble can reach an accuracy higher than pure MLP models, proving that in a production scenario, we may be able to emphasize

Citation: Wenping Wang, et al. Sentiment Analysis: A Systematic Case Study with Yelp Scores. Advances in Artificial Intelligence and Machine Learning. 2023;3(3):74.

throughput and latency by using small models, instead of relying on heavy, multi-layer MLPs, with proper vectorizer and data processing.

Keywords: Sentiment analysis, Yelp dataset, Transformer embedding, Ensemble

1. INTRODUCTION

Sentiment analysis has long been a well-defined natural language processing subtask in machine learning. It is easy for humans to tell that whether a piece of text excerpt, for example, a review towards services, is positive or negative, and even give a good estimate of the rating based on what they have just read. Meanwhile, machine learning models may not easily draw such a conclusion. Thus, given how comprehending the sentiment behind text is a challenging task for machines, we think it would be interesting and meaningful to learn a predictor of ratings based on text alone.

Yelp has been one of the most popular and influential sites for people to rate and review businesses, presenting us with an abundant corpus for sentimental analysis. Besides rating businesses on a scale of 1 to 5 stars, Yelp also allow users to write text reviews that could relatively explain the reasons behind the given ratings. Thus, we have a very high quality dataset for sentiment analysis.

In this article, our contributions can be summarized into the following:

- investigate what makes a review positive or negative based on the review text and a small set of attributes;
- experiment with different machine learning algorithms and tokenizers, in the order of parameter size from simple to complex, and proves that Logistic regression models can be as same powerful as MLPs and is much better than several non-regression models;
- investigate how and to what extent an ensemble models and feature engineering can boost inference performance.

2. BACKGROUND AND RELATED WORKS

Sentiment analysis has been widely explored in previous works. [1] did a very thorough review of the works, especially statistical approaches, while the more recent one [2] primarily focuses on deep neural networks. In general similar statistical approaches are mature and has been applied in various tasks [3–5].

[6] are among the first ones to conduct classification on sentiment using machine learning techniques. They found that, unlike traditional tasks such as classification on document topics, sentiment classification is especially difficult for machines to outperform human baselines. A piece of task could contain no obviously negative words, but has a negative connotation. Thus, sentiment analysis actually requires deeper understanding of the underlying text.

More recent work relies more on extensive feature engineering and more innovative models. In [7], they propose to use a combination of feature generation method to find the best prediction result. They use Part-of-Speech to analyze the semantic structure of English sentences and use it to select out more representative and meaning words. In [8] the authors take advantage of deep neural networks and propose an innovative user-word composition vector model. This model incorporates user-specific information into the meaning of a certain word. Further, [9] utilized pre-trained language models and achieved a very high accuracy.

The Yelp dataset has been a very popular data source for various NLP tasks, specifically for rating prediction, [10] experiments several feature extraction approaches, along with statistic predictors including regression-based ones and SVM-based approaches, and shares some insightful observations. However, the authors didn't utilize other state-of-the-art works such as BERT, and did not tried out other interesting approaches such as ensemble methods that combines models together to further optimize the prediction accuracy.

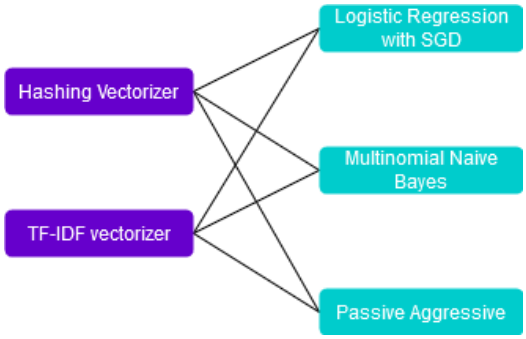
As already been proven in other NLP tasks [11], LLM is also very powerful in sentiment analysis. In [12], the authors evaluates the performance of LLM on 13 subtasks, showing that LLM outperform smaller models, especially when the training data is limited. However, LLMs still suffer from the expense of computation resources and latency due to model sizes.

3. DATASET

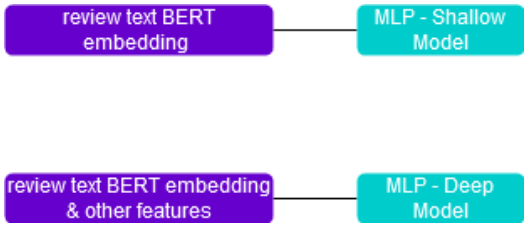
We collect data from Yelp Open Dataset from Yelp official account. The dataset contains 132k entries in total. Since we are doing supervised learning, we split up the `Yelp_train` dataset ($8:1:1$ split) for training, validation and testing.

The features in the Yelp dataset can be divided into 5 major categories:

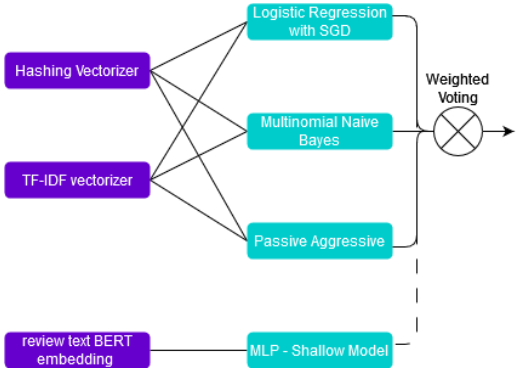
- **Review Text**, which is the text content input from users;
- **Descriptive features** of review text, namely *"nchar"*, *"nword"*, *"sentiment score"*, *"useful"*, *"funny"*, and *"cool"*. These 5 features describe the review text, indicating the number of characters, number of words, the sentiment score, and the number of people who vote the review as useful, funny and cool;
- **Uni-gram features**, namely *"gem","incredible","perfection","phenomenal","divine", "die","highly","superb" "knowledgeable","gross","poorly","response","flavorless","waste","terrible","tasteless", "rude","awful","inedible","horrible","apology","disgusting","worst"*. There are 19 unigram features in total, and they represent the number of occurrence of 19 given words in the review text;
- **Categorical features**, namely *"star"*, *"city"*, *"name"*, and *category*;
- **Spatiotemporal features**, namely *"date"*, *"longitude"*, and *"latitude"*.



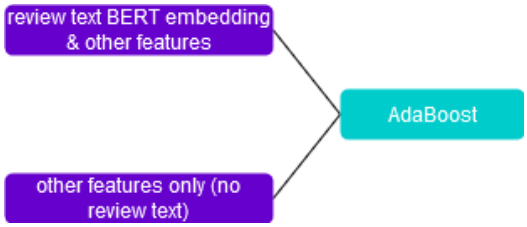
(a) Baseline approaches.



(b) Multi-layer Perceptron approaches.



(c) Ensemble approaches. We tried three different combinations of feature extractors and predictors, and use a weighted sum to determine the final prediction.



(d) AdamBoost approaches.

Figure 1: Various combinations of feature extractors and predictors, summarized in four different groups. Here purple cells are feature extractors and blue cells are different predictor models

4. MODEL FORMULATION

4.1 Outline

To predict the user rating in an NLP approach, a system need to

1. extracts and prepares the features into numerical representations, then
2. feeds the features into a predictor to produce a probability distribution of the user rating

We explored several different approaches for each of the above two components, and conducted experiments over various combination of them. Section 3.2 will cover the two feature extraction approaches, while the rest of this section will go through various predictors. See FIGURE 1, for details.

4.2 Feature Extraction

The two feature extraction methods that we explore are hashing vectorizer and TF-IDF.

Hashing vectorizer We use the hashing vectorizer offered by the scikit-learn library to convert a collection of review text into a matrix of token occurrences. A hashing vectorizer returns the a count for every token in the document, so it is no different from a regular bag-of-words model (called counter vectorizer in scikit-learn) in terms of how text features are turned into numeric representation. However, hashing vectorizer has the following two advantages:

1. Scale better with large document sets
2. Work well with batch processing

Comparing with a count vectorizer, hashing vectorizer imposes little requirements on memory when working with large datasets because it doesn't need to store the vocabulary dictionary. In addition, count vectorizer doesn't work so well with batch processing, because it is hard to maintain a stable representation of the token matrix. The size of the vocabulary dictionary in one batch of documents might be different from another, so we could end up getting matrices with different number of columns. However, the hashing vectorizer doesn't require an extra step to learn the vocabulary dictionary, which means if we set the $num\_features$ parameter properly, it would ensure that feature space still remains the same even when new tokens may be added in each batch.

TF-IDF vectorizer One apparent issue with hashing vectorizer is that common words such as "the" and "like" are associated with high counts when they are actually not so meaningful. This motivates us to consider the TF-IDF vectorizer which address this issue ([13]).

The TF-IDF acronym stands for "Term Frequency - Inverse Document Frequency". Like how count and hashing vectorizer assigns a count value to each token, the TF-IDF assigns the TF-IDF score in the following way:

For a term i in document j:

$$TF\text{-}IDF\ Score = tf_{i,j} \times log(\frac{N}{df_i})$$

where

$tf_{i,j}$ = number of occurrences of i in j
df_i = number of documents containing i
N = total number of documents

As the formulation presents, the Term Frequency describes how often tokens appear in the document, while the Inverse Document Frequency down-scales tokens that appear too often across documents. In summary, the TF-IDF address the issue with hashing vectorizer by penalizing common words and assigning higher values to the more meaningful ones.

4.3 Baseline Models

Since Yelp star ratings take on discrete values, we treat it as a classification problem and implement four baseline models using the scikit-learn machine learning library.

1. Logistic Regression with SGD I
2. Logistic Regression with SGD II (a variant of SGD I)
3. Multinomial Naive Bayes
4. Passive Aggressive

We choose these algorithms specifically because they are suitable for this classification task, and because they are online learning algorithms that can perform incremental learning from mini-batch of data instances which are lightweight. This ensures that these algorithms can be done at a very low latency in high throughput production. Since we are dealing with a large dataset, to simulate the throughput of a server under highest workload, we choose to feed the classifiers with mini-batches of 1000 instances to mitigate overhead and ensure stable performance. This means that we will have at most 1000 instances in the machine's main memory during the entire training process.

Logistic Regression Logistic Regression is a common classification model that is based on regression analysis and is most often used to explain the relationship between a set of binary predictors and classification labels. And the final output is a probability between 0 and 1. For the purpose of this task, we use logistic regression to predict the star rating of each review, and apply Stochastic Gradient escent (SGD) to minimize the loss function. We normalize the word count vector to ensure the extracted features are binary. We build two baseline models in this way, namely SGD I and SGD II, where they mainly differ in hyper-parameters. Compared to SGD I, SGD II is simply set to have more epochs and a smaller stopping criterion.

Multinomial Naive Bayes Multinomial Naive Bayes is another good choice as a baseline model because it is suitable for text classification with word counts. Given a set of training data, it computes the prior probability of reviews for each star rating $P(rating)$, and the probability of observing a certain feature given the star rating $P(feature|rating)$. Thus, using the Baye's rule, the posterior probability for each class

$$P(rating|feature) = \frac{P(feature|rating)P(rating)}{P(feature)}$$

We then assign each data point to the class having the largest posterior probability. We also add smoothing to the model to ensure non-zero probability, and the smoothing parameter is set to 0.1 because the hashing vectorizer normalizes the word count matrix.

Passive Aggressive Passive Aggressive is another online classification machine learning algorithm ([14]). It is very similar to a Support vector machine because it is margin-based. Passive Aggressive also aims at maximizing the margin of the separating planes between the data points.

4.4 Multi-Layer Perceptron

We also consider building a neutral network for this text classification task and decide to build a multi-layer perceptron model. To translate vocabulary in the review text into numerical representations, we decide to use BERT pretrained text embedding.

BERT BERT is a widely used word embedding pre-trained on Wikipedia and *BookCorpus*, a dataset consisting of over $10,000$ books of 16 different genres [15]. BERT embedding is different from other popular word embeddings such as Word2Vec or GloVe, mainly because it is context-dependent. While Word2Vec and GloVe generate just one embedding vector for each word in the document, BERT could generate multiple vectors if the given word appears multiple times under different context. consider the following simple example:

*"The doctor used his **cell** phone to take a picture of the blood **cell** samples"*

The word *"cell"* clearly has different meanings depending on whether it is used together with *phone* or *blood*. Word2Vec or GloVe embeddings will ignore this difference and generate only the same one vector for the word, but BERT will output two different embedding vectors for this word in this example, and potentially capture more information about the feature [16]. Given the difference between BERT and other context-independent embeddings, we decide to apply BERT for our Perceptron model because it captures features more accurately and could potentially lead to better model performance.

After applying the BERT embedding, our final embedding vector is of length 768, which can then be used as input to our Multi-layer Perceptron model.

Other Features Besides the review text, we also want to incorporate other features into the Multi-layer Perceptron model, since the uni-gram features and the sentiment score provided in the dataset might be useful as well. Thus, we conduct some featuring engineering and construct 8 additional features to be used together with the BERT embedding vector.

We process each feature to ensure that their values are in range $[0, 1]$:

- **useful:** apply shifted logistic regression with weight of 1
- **funny:** apply shifted logistic regression with weight of 1
- **cool:** apply shifted logistic regression with weight of 1
- **positive:** sum positive uni-gram counts, apply shifted logistic regression with weight of 1
- **negative:** sum positive uni-gram counts, apply shifted logistic regression with weight of 1
- **nchars:** apply shifted logistic regression with weight of 0.1
- **nwords:** apply shifted logistic regression with weight of 0.5
- **sentiment score:** $(\text{score} + 5)/10$ since sentiment scores are in range $[-5, 5]$

The shift logistic regression we used in defined as follow:

$$\text{shifted value} = \frac{1 - e^{-x*w}}{1 + e^{-x*w}}$$

where x represents feature values and w represents the weight. This way, all the 8 additional values are in range $[0, 1]$.

In this case, we build two Multi-layer Perceptron models under different settings:

Shallow Model The shallow Multi-layer Perceptron model consists of 7 fully-connected layers and uses the rectifier linear unit (ReLU) activation function. The input layer consists of the 768 features output by BERT, and the output layer consists of the five star ratings. The 7 layers are as follows:

$$[768, 1024, 512, 512, 256, 64, 32, 5]$$

Deep Model For the deep Multi-layer Perceptron model, we also incorporate the 8 additional features that we construct earlier. We take the outer product of this vector and the BERT embedding vector, and the final input vector is thus of length $768 \times 8 = 6144$. The deep model consists of 9 layers and also uses the ReLu activation function:

$$[6144, 4096, 2048, 1024, 512, 256, 64, 32, 5]$$

4.5 Ensemble Model

To further boost model performance, we apply ensemble method to combine our various models into a single predictor.

Heterogeneous classifiers The classifiers we use for ensemble include Logistic Regression, Multinomial Naive Bayes, Passive Aggressive and Multi-layer perceptron. For each of the baseline models, we use either the hashing or TF-IDF vectorizer for feature extraction. And the final combined models are as follows:

- Baseline models ensemble: (hashing vectorizer) LR + MNB + PA
- Baseline models and MLP: (hashing vectorizer) LR + MNB + PA + MLP
- Baseline models and MLP: (TF-IDF vectorizer) LR + MNB + PA + MLP

Weighted Voting The way we ensemble the heterogeneous classifiers is through weighted voting. For each classifier, we get the probability vector that describes the probabilities that a given data point belongs to each of the 5 star rating.

$$[P(1|data), P(2|data), P(3|data), P(4|data), P(5|data)]$$

Each classifier is also assigned a weight between 0 and 1, and we computed the weighted sum of the probability vectors for all classifiers. Then, the final prediction is the star rating that has the largest weighted sum of probabilities.

Considering all the classifiers we used for ensemble, the outputs of the Logistic Regression and Multinomial Naive Bayes are already probability values. For Passive Aggressive, we can also get the probabilities using the scikit-learn library. And for Multi-layer Perceptron, we use the *softmax* function to normalize the output values into probability distribution.

Weights Enumeration To approximate the best weight assigned to each classifier, we use enumeration. And the pseudocode is provided below

```
best_ensemble = []

for w_1 in (0, 1, 0.02):
   for w_2 in range(0, 1, 0.02):
      for w_3 in range(0, 1, 0.02):

         sum_prob = w_1 * LR + w_2 * MNB + w_3 * PA

         predict_labels = argmax(sum_prob)

         # compare the accuracy by comparing predict_labels and ground truth
         # update best_ensemble if the current model is better

         if current_acc > old_acc:
            best_ensemble = [w_1, w_2, w_3]
```

4.6 AdaBoost

The final ML algorithm that we experiment with is AdaBoost. AdaBoost runs a weak classifier multiple times on the dataset and adjust the weights assigned to each data point. Incorrectly classified data points will get larger weights so that they receive more attention during the next iteration. We employ the AdaBoost algorithm with decision stumps (1-level decision trees).

We construct two AdaBoost models:

1. All features, including review text (BERT embedding), sentiment score, uni-grams, and other spatiotemporal features.
2. All features except for review text.

Feature processing We want to use all the features provided in the dataset, but not all features are numerical. The features that need special processing include: *date, city, name, and category*. And we convert them into numerical features in the following way:

- **date:** extract out day, month, and year;
- **city:** compute length(city);
- **name:** compute length(name);

- **category:** use hashing vectorizer.

Other features are processed using the same procedure as documented in the Multi-Layer Perceptron section.

5. EXPERIMENTS AND RESULTS

We evaluate all of our models based on their test or validation accuracy.

5.1 Feature Extraction

We would first like to examine the effectiveness of our two different feature extraction methods. As we introduced in Section 4.2, we experiment with both the hashing and the TF-IDF vectorizer. We think that our TF-IDF vectorizer could lead to better model performance because it scales the word count matrix to focus on more meaningful words, but we would like to verify if that is actually the case (TABLE 1).

Table 1: Test Accuracy of Hashing vs. TF-IDF Vectorizer

	Hashing	TF-IDF
LR with SGD I	0.5541	0.5611
LR with SGD II	0.5450	0.5670
Naive Bayes	0.4754	0.4846
Passive-Aggressive	0.5253	0.5374

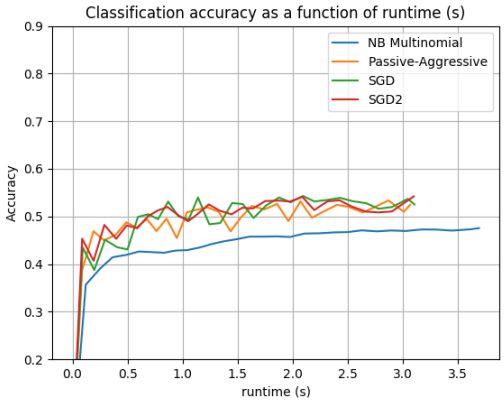
Looking at the reported test accuracy, we observe a consistent increase in performance across all baseline classifiers. The test result is consistent with our hypothesis that TF-IDF vectorizer leads to more meaningful features.

5.2 Baseline Models

Since we employ batch processing for all the four baseline models. We would like to examine the learning curves of these classifiers. For the following plots, we use hashing vectorizer and evaluate the models on the a held-out validation set.

As illustrated by FIGURE 2, we observe a general increasing trend in validation accuracy across all classifiers until training examples reach 20,000 when it starts converging. The validation accuracy curve is not smooth because we feed all classifiers with mini-batches of 1000 instances. Overall, while the Multinomial Naive Bayes model consistently has the lowest validation accuracy, the performance is stable across all other classifiers.

(a) As a function of examples



(b) As a function of runtime

Figure 2: The learning curves for baseline models

5.3 Multi-layer Perceptron

For Multi-layer Perceptron, we would like to examine the effectiveness of our two Perceptron models. As discussed in Section 3.3, we implement two models: a shallow model with 7 layers and a deep model with 9 models. Generally, we would expect the deep model to have better performance because it should be able to recognize more aspects of the input data. The table below summarizes the performance of our shallow and deep model, evaluated on the test set (TABLE 2).

	Test Accuracy
Shallow Model	0.5307
Deep Model	0.5282

Table 2: Performance of two MLP Models

	Test Accuracy
All features	0.5345
All without review text	0.4370

Table 3: Performance of the two Adaboost Models

To our surprise, our shallow model actually has consistently better performance than the deep Multi-layer Perceptron model. Recall that our deep model input layer includes not only the BERT embedding vectors, but also the 8 additional features that we construct. The 8 additional features are descriptive features of review text such as sentiment score, positive and negative uni-grams, and the number of characters. Initially, we think that these features are closely related to sentiment analysis. While sentiment score measures the strength of positive/negative reviews, the positive and negative uni-grams features should also be helpful in predicting the star ratings. However, the results shows the opposite: adding these additional features do not lead to better model performance.

5.4 Adaboost

For Multi-layer Perceptron, we would also like to compare the performance of two models. As discussed in Section 3.3, we again implement two models: one based on all features and a second one based on all but the review text.

Looking at the test accuracy reported in Table 3, we observe that there is significant increase in model performance when review text is added to the Adaboost Model. This again verify the importance of the review text feature. We have also plot the training accuracy (right figure) and the test accuracy (right figure) as a function of the number of trees. We observe that as the AdaBoost model reaches stable performance at round 100 trees.

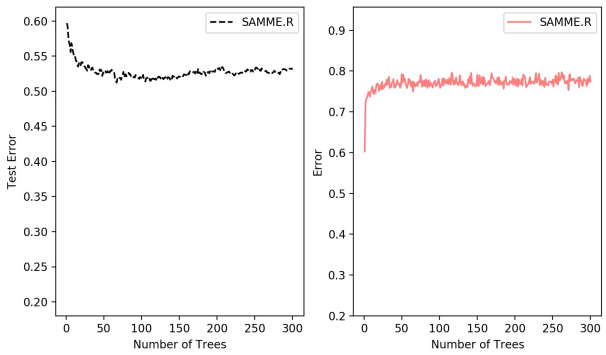


Figure 3: Test and training accuracy as a function of number of trees

5.5 Analysis

The following TABLE 4, summarizes all the models we use and their corresponding test accuracy

Table 4: Performance of All Models

	Test Accuracy
MLP Shallow Model	0.5307
MLP Deep Model	0.5282
Adaboost I	0.5345
Adaboost II	0.4370
Baseline (hashing) Ensemble	0.5633
Baseline (TF-IDF) Ensemble	0.5689
Baseline (hashing) and MLP Ensemble	0.5832
Baseline (TF-IDF) and MLP Ensemble	0.5837

Comparing with our baselines' performance listed in Section 5.2, it can be observed that Multi-layer Perceptron does outperform the baseline models. However, it is excited to see that the performance can be further improved by ensembling the MLP model with all the baseline models. We believe it is because there is a significant diversity among these ensembled models (Table 5).

We also conduct some error analysis on our best model. To see exactly which set of data points are incorrectly classified, we calculated the per-rating precision and recall in FIGURE 5 and 6.

We observe that our best prediction model performs better when predicting star ratings of 1, 4, and 5, but really struggles with 2 and 3. For star rating of 5, our best model is able to correctly classify

Table 5: Counts of the predicted and the golden ratings: Baseline (TF-IDF) and MLP Ensemble model

		Golden Ratings					
		1	2	3	4	5	Total
Model Predictions	1	215	83	22	4	3	327
	2	58	133	36	11	2	240
	3	19	101	192	62	9	383
	4	22	69	245	650	237	1223
	5	24	27	68	444	978	1541
	Total	338	413	563	1171	1229	3714

Table 6: Error Analysis by Ratings for the Baseline (TF-IDF) and MLP Ensemble model

Star Ratings	TP	TN	FP	FN	Precision	Recall
1	215	3376	112	123	65.74%	63.60%
2	133	3301	107	280	55.41%	32.30%
3	192	3151	191	371	50.13%	34.10%
4	650	2543	573	521	53.14%	55.50%
5	978	2485	563	251	63.46%	79.57%

79.57% of the test data points, but it could only correctly classify 32.30% of the data that has ratings of 2. We think this makes sense in that it is easy to distinguish between a 1-star review and a 5-star review, but the boundary between a 2-star and a 3-star might be more ambiguous (Table 6).

6. CONCLUSION AND FUTURE WORK

In this article, we experiment with various classification algorithms that are suitable for predicting Yelp star ratings based on review text and attribute set.

We explore two feature selection techniques, namely Hashing and TF-IDF vectorizers. We show that TF-IDF vectorizer could lead to better model performance, because it emphasizes on more meaningful words.

We employ several baseline models with the ability of online and incremental learning, and investigate on their learning curves. We also use Multi-layer Perceptron with BERT embedding, showing that the review text is the most important feature comparing to others. The performance of our two AdaBoost models is also consistent with this finding.

Finally, we use the ensemble method to combine various models into a single predictor and our best prediction model turns out to be ensemble model generated with all four baseline models and the Multi-layer Perceptron.

For future work, it is worth investigating more feature extraction and embedding methods since review text is the dominate feature for this classification task. The approach we purpose is generic

to other tasks as well [17]. We also believe that our MLP model using BERT embedding could have better performance, if we have the chance to conduct parameter tuning and pre-training, and with some current hardware machine learning optimization techniques [19], we can achieve a good performance with less resource constraints.

References

[1] Medhat W, Hassan A, Korashy H. Sentiment Analysis Algorithms and Applications: A Survey. Ain Shams Eng J. 2014;5:1093-1113.

[2] Zhang L, Wang S, Liu B. Deep Learning for Sentiment Analysis: A Survey. WIREs Data Mining Knowl Discov. 2018;8:1253.

[3] Yue T, Wang Haohan. Deep Learning for Genomics: A Concise Overview. 2018. ArXiv preprint: https://arxiv.org/pdf/1802.00810v3.pdf

[4] Wang W, Guo Y, Shen C, Ding S, Liao G, Fu H et al. Integrity and Junkiness Failure Handling for Embedding-Based Retrieval: A Case Study in Social Network Search. 2023. ArXiv preprint: https://arxiv.org/pdf/2304.09287.pdf

[5] Wenting Y, Liu X, Yue T, Wang W. A Sparse Graph-Structured Lasso Mixed Model for Genetic Association With Confounding Correction. 2017. ArXiv preprint: https://arxiv.org/pdf/1711.04162.pdf

[6] Pang B, Lee L, Vaithyanathan S. Thumbs Up? Sentiment Classification Using Machine Learning Techniques. 2002. ArXiv preprint: https://arxiv.org/pdf/cs/0205070.pdf

[7] Fan M, Khademi M. Predicting a Business Star in Yelp From Its Reviews Text Alone. 2014. ArXiv preprint: https://arxiv.org/ftp/arxiv/papers/1401/1401.0864.pdf

[8] Tang D, Qin B, Liu T, Yang Y. User Modeling With Neural Network for Review Rating Prediction. In: Proceedings of the 24th international conference on artificial intelligence, IJCAI'15. AAAI Press. 2015: 1340-1346.

[9] Alamoudi ES, Alghamdi NS. Sentiment Classification and Aspect-Based Sentiment Analysis on Yelp Reviews Using Deep Learning and Word Embeddings. J Decis Syst. 2021;30:259-81. [10] Asghar N. Yelp Dataset Challenge: Review Rating Prediction. ArXiv preprint: https://arxiv.org/pdf/1605.05362.pdf

[10] Asghar N. Yelp Dataset Challenge: Review Rating Prediction. ArXiv preprint: https://arxiv.org/pdf/1605.05362.pdf

[11] Chen T, Wang X, Yue T, Bai X, Le CX, Wang W et al. Enhancing Abstractive Summarization With Extracted Knowledge Graphs and Multi-Source Transformers. Appl Sci. 2023;13:2076-3417.

[12] Zhang W, Deng Y, Liu B, Pan SJ, Bing L. Sentiment Analysis in the Era of Large Language Models: A Reality Check. 2023. ArXiv preprint: https://arxiv.org/pdf/2305.15005.pdf

[13] Roelleke T, Wang J. TF-IDF Uncovered: A Study of Theories and Probabilities. In: Proceedings of the 31st annual international ACM SIGIR conference on research and development in information retrieval, SIGIR'08. NY. ACM. 2008: 435-442.

[14] Crammer K, Dekel O, Keshet J, Shalev-Shwartz S, Singer Y. Online Passive-Aggressive Algorithms. J Mach Learn Res. 2006;7:551-585.

[15] Devlin J, Chang M-W, Lee K, Toutanova K. Bert: Pretraining of Deep Bidirectional Transformers for Language Understanding. 2018. ArXiv preprint: https://arxiv.org/pdf/1810.04805.pdf

[16] Zhang L, Negrinho R, Ghosh A, Jagannathan V, Hassanzadeh HR, et al. Leveraging Pretrained Models for Automatic Summarization of Doctor-Patient Conversations. 2021. ArXiv preprint: https://arxiv.org/pdf/2109.12174.pdf

[17] Yang X, Ye W, Breitfeller L, Yue T, Wang W. Linguistically Inspired Neural Coreference Resolution. Adv. Artif Intell Mach Learn. 2023;3:1122-1134.

[18] Yu K, Wang Y, Zeng S, Liang C, Bai X, Chen D et al. Inkgan: Generative Adversarial Networks for Ink-And-Wash Style Transfer of Photographs. 2023;3: 1220-1233.

[19] Zhou Y, Gupta U, Dai S, Zhao R, Srivastava N, et al. Rosetta: A Realistic High-Level Synthesis Benchmark Suite for Software Programmable FPGAs. In: Proceedings of the 2018 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays, FPGA'18. NY. Association for Computing Machinery. 2018: 269-278.