Comparative Study of Binary Sentiment Analysis for the Product Review dataset with the Explainable AI framework

Dimitra Muni
732A92 - Text Mining
Linköping University
dimmu472@student.liu.se

Abstract

This project aimed to analyze binary sentiment for the Amazon Fine Food Reviews dataset[1] using different vectorization and word embedding techniques and evaluated the performance on the gold standard dataset. In the first part of the project, we utilized two different word vectorization (i.e. count and TF-IDF vectorizers) techniques in conjunction with three linear classifiers (i.e. Multinomial Naive Bayes, Logistic Regression, and Support Vector Machine (SVM)) to create six experimental pipelines. Furthermore, we used the *Halving Grid Search* technique to find the best-performing model with five-fold cross-validation. Here SVM slightly outperformed the other two models with an accuracy of 93%. In the second part of the project, we utilized four pre-trained compact BERT models and DistilBERT to perform sentiment classification on a smaller subset of original data; as expected, DistilBERT outperformed the other compact BERT models, however, as it was not tuned on the entire training data, the performance was not as high as SVM model. In the third part of the project, we performed LIME (Ribeiro et al., 2016) analysis for a subset of misclassified samples identified from the final SVM model and documented actionable insights.

1 Introduction

In the last decade, massive development in Masked Language Models has opened the floodgates of possibility in Natural Language Processing (NLP) domain. **B**idirectional **E**ncoder **R**epresentations from **T**ransformers (BERT), developed by (Devlin et al., 2018) at Google, is one such model. The base version of BERT has 110 million parameters and is computationally expensive to train. Several variants of this model, such as DistilBERT (Sanh et al., 2019) and a compact version of BERT developed by (Turc et al., 2019), are significantly faster than the base version of BERT; however, with a decline in performance.

These models can be utilized to address several different tasks, such as machine translation, context disambiguation, named entity recognition (NER), and sentiment analysis, to name a few. In this project, we focus on the problem of binary sentiment analysis for the Amazon Fine Foods dataset, which contains product reviews collected over a decade. We intend to address the following research objectives,

1. Experiment with word vectorizer-based technique with machine learning classifiers to perform sentiment classification.

2. Investigate the applicability of the compact Masked Language Models (MLMs) for sentiment classification.

3. Inference about misclassified samples obtained from the best-performing model, generating actionable insight using an explainable AI technique.

In the initial part of this project, we performed data cleaning and pre-processing on the Amazon Fine Foods dataset, which contains reviews with scores from 1 to 5. In the pre-processing part, the original dataset was converted into a smaller dataset with two classes, i.e. positive and negative reviews; we elaborate on this in the later stage. Next, we split the reduced dataset into training and test sets with an 85%-15% split. We use the test set only for the evaluation in the later stage. As the training set had a class imbalance, we performed random down-sampling to create a balanced training set.

Next, we created six different vectorizer-classifier pipelines, using two vectorizers and four classifiers; we utilized a halving grid search technique with five-fold cross-validation to tune these pipelines and evaluated the performance on the test set.

[1] https://snap.stanford.edu/data/web-FineFoods.html

In the second part of the project, we use transformer-based embedding using five small BERT models with different numbers of transformer blocks and hidden layer size combinations; we trained the model for ten epochs and evaluated the performance on the test set. Next, we repeated this process for the DistilBERT model. In our experiments, we utilized Tensorflow Hub to utilize these models.

In the last part of the project, we focus on the misclassified samples (from the test set) identified using the Linear SVM model. We used a popular explainable AI technique called LIME analysis (Ribeiro et al., 2016) to perform this study.

2 Theory

2.1 Vectorization

The vectorization technique converts the corpus of text data into a matrix representation; one such technique is Count Vectorizer, which uses a *bag-of-word* representation (Jurafsky and Martin, 2023). Here the corpus is assumed to be a collection of words, where a matrix represents the frequency of the word *w* in document *d*. One noteworthy constraint of this representation is that the word's position in a sentence is not inconsequential; rather, the frequency of the word is important (Jurafsky and Martin, 2023). Another more advanced method of vectorization is *TF-IDF*; here, **TF** refers to the **term frequency**, which accounts for the occurrence of the term *t* in document *d*, while *idf* is **inverse document frequency**, a measure of the number of documents a term *t* appears. Here in the equation [1], the inverse document frequency of term t (idf_t) is calculated on the logarithm scale, where *n* is the total number of documents.

$$\begin{aligned} w_{t,d} &= tf_{t,d} \times idf_t \\ tf_{t,d} &= count(t,d) \\ idf_t &= \ln\left(\frac{1+n}{1+df_t}\right)+1 \end{aligned} \tag{1}$$

There are several variations in the way $tf_{t,d}$, and idf_t are calculated, but we choose this definition [1] to be consistent with the default settings of the `sklearn` library `TfidfVectorizer` implementation. One noteworthy advantage of TF-IDF representation is that it assigns relevance to rare occurring words by higher IDF value (Schütze et al., 2008).

2.2 Classifiers

2.2.1 Multinomial Naive Bayes

Multinomial Naive Bayes classifier is based on Bayes' rule (Bayes, 1763)[2] about the conditional probability of two events. This classifier is often a preferred baseline evaluation method due to its simplicity. According to Jurafsky and Martin (p.61, 2023), there is a **naive** assumption about conditional independence of the feature probability given the class ($P(f_i|c)$); the authors represented this mathematically as,

$$c_{NB} = \underset{c\in C}{\operatorname{argmax}}\, P(c)\prod_{f\in F} P(f|c) \tag{2}$$

Here $P(c)$ is the prior probabilities of each class c (where $c \in C$), whereas f is a feature that refers to the word in the document. Authors represented equation [2] in terms of log scale [3] for efficient computation.

$$c_{NB} = \underset{c\in C}{\operatorname{argmax}}\left(log(P(c)) + \sum_{i\in positions} log(P(w_i|c))\right) \tag{3}$$

Here $P(w_i|c)$ is the probability of a word at i^{th} location while conditioning on class c, $P(c)$ and $P(w_i|c)$ were represented as,

$$P(c) = \frac{N_c}{N_{doc}}; P(w_i|c) = \frac{count(w_i,c)+\alpha}{(\sum_{w\in V} count(w,c)) + \alpha|V|} \tag{4}$$

In the equation [4], N_c is the number of documents from the dataset with class c. N_{doc} represents the total number of documents in the dataset. $|V|$ is the size of vocabulary V, which comprises all the words in the corpus. α is the smoothing parameter ($\alpha \geq 0$), which adds the extra occurrence of words to prevent zero probability. If the $\alpha = 0$, then there is no smoothing applied. Usually, this parameter is optimized using validation data.

2.2.2 Logistic Regression

According to Bishop and Nasrabadi (p.205, 2006), the logistic regression classifier is one of the linear classifier models that uses *logistic sigmoid* function[3] for binary classification and attempts to minimize **cross-entropy loss**. Jurafsky and Martin (p.86, 2023) asserted that for the correlated features, the logistic regression classifier is preferred over naive Bayes because the logistic regression

[2]Bayes' rule for two events x and y is presented as in term of conditional probability, $p(x|y)p(y) = p(y|x)p(x)$.

[3]logistic sigmoid: $\sigma(a) = \frac{1}{1+\exp(-a)}$

can assign the weights more efficiently amongst the correlated features; while this mechanism is not present in the naive Bayes model; due to this reason, we intend to use this model in our experiments. However, this model has a disadvantage that, as it uses the gradient descent method to update its weight, it usually takes longer time to train.

Now, we refer to the documentation of the logistic regression model (`LogisticRegression` function) in the `sklearn` library[4], where the optimization problem is formulated as:

$$\min_{w} \left(C \sum_{i=1}^{n} (y_i \log(P(X_i)) - (1 - y_i) \log(1 - P(X_i))) + r(w) \right) \tag{5}$$

Here, y_i is the binary target value, X_i represents the input feature for the sample i, and w is the corresponding weight. Furthermore, $r(w)$ refers to the *penalty* term that controls the model's regularization. In our baseline experiments, we do not use any *penalty* but rather tune the regularization term *C*.

2.2.3 Support Vector Machines

Support Vector Machine (SVM) classifier originally presented by (Vapnik, 1999) belongs to the class of *maximum margin classifiers* (Bishop and Nasrabadi, 2006), where the objective is to find a hyperplane that separates two classes with the highest margin. There are several variations of SVM classifiers based on the kernel function. However, we only consider the Linear SVM classifier, using the `LinearSVC` function from `sklearn` library; we refer to the documentation[5], where the problem is formulated as follows:

$$\min_{w,b} \left(\frac{1}{2} ||w||^2 + C \sum_{i=1}^{n} max(0, 1 - y_i(w^T \phi(x_i) + b)) \right) \tag{6}$$

This classifier utilizes **hinge loss** described by $max(0, 1 - y(w^T \phi(x) + b))$. Here x_i is the training vector, y_i is the vector with value $\{-1, 1\}$. ϕ is the identify function[6]. C is the term that controls the regularization (similar to that in Logistic Regression, w is the weight vector. SVM classifier finds the maximum margin between support vectors by minimizing the $||w||^2$. According to the documentation for SVM, this method is appropriate for datasets with high dimensions if the dataset has considerably more samples ($n >> p$) than the number of features, for this reason, we include this method in our analysis.

2.3 Masked Language Models

According to (Devlin et al., 2018), the BERT model is trained using a *masked language modeling* technique, where a model is trained by hiding randomly chosen tokens and is asked to predict that token. This model introduces bidirectional context to the training of transformer-based architecture using the attention mechanism (Vaswani et al., 2017). The BERT model is trained in two iterations; the first iteration is called **pre-training**, where the model is trained on unlabeled data, and the second is **fine-tuning**, where the model is tuned on labeled data.

The BERT-base model has 12 transformer blocks, with a hidden size of 768 and 12 attention heads. This model uses WordPiece embedding (Wu et al., 2016), which converts a word into sub-word units.

In figure [1], the pre-training and fine-tuning procedure is represented, where a sentence pair is separated by `[SEP]` token, and before each sentence, there is a classification token `[CLS]`. Input embedding is denoted by ***E***, and ***C*** represents the final vector corresponding to `[CLS]` token, and $T_1, T_2, ..., T_N$ represents tokens' final vector representation, where *N = 768* for the BERT-base model.

According to (Devlin et al., 2018), BERT is pre-trained on two tasks; the model is trained to predict the masked words in the first *Masked Language Modelling* task, where 15% of tokens are chosen at random and masked. Furthermore, out of these 15% tokens, 80% tokens are masked with `[MASK]` token, 10% are replaced with a random token, and 10% of tokens remain unchanged. The second task is *Next Sentence Prediction (NSP)*, where the model is trained by evaluating whether a sentence is followed by another sentence, using labels such as `IsNext` or `NotNext` for *Question Answering (QA)* and *Natural Language Inference (NLI)* use cases. In the next stage, the BERT model is fine-tuned on a different dataset based on the corresponding application.

One important limitation of the BERT model is the higher computational cost; several compact versions of BERT address this issue, however, with some performance loss.

[4] https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression

[5] https://scikit-learn.org/stable/modules/svm.html#support-vector-machines

[6] Identitiy function, $f(X) = X$

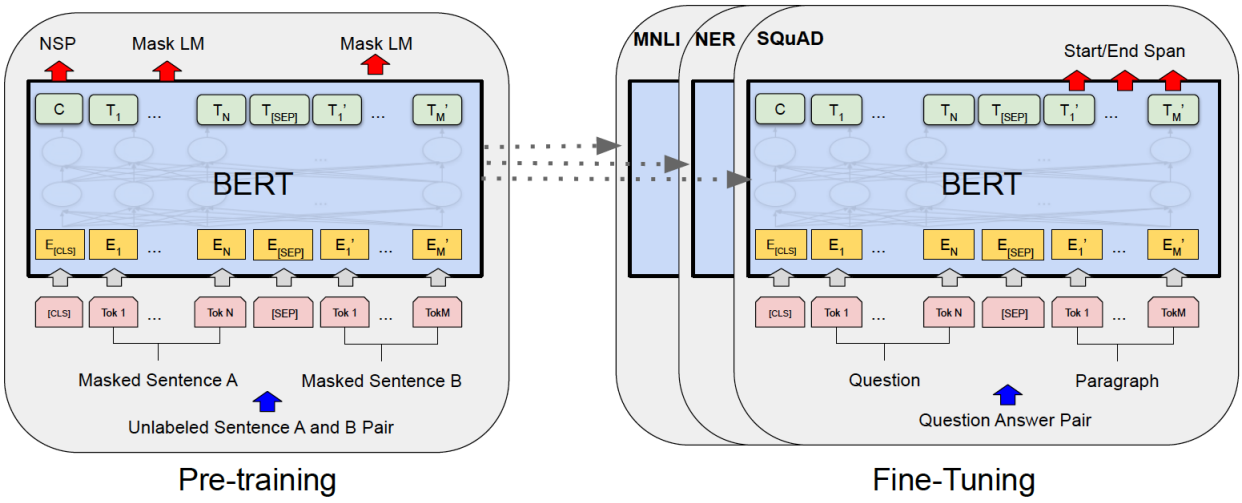

Figure 1: Pre-training and Fine tuning procedure in BERT, the figure taken from Devlin et al. (p.3, 2019)

2.3.1 Compact Versions of BERT

The compact version of BERT introduced by (Turc et al., 2019) uses the student-teacher paradigm for knowledge distillation. Here the teacher is a large or base BERT model, while the student is a smaller BERT model with fewer parameters, e.g. **BERT-128/2**, which comprises 4.4 million parameters. The authors have presented 24 different models with varying numbers of transformer blocks, hidden embedding sizes, and attention heads. However, we only chose four models as described in table [7] for the limited scope of this project.

The compact BERT model training procedure is described as **Pre-trained distillation** in figure [2]. Authors have used three types of datasets in training these models, unlabeled language model data ($\mathcal{D}_{LM}$), unlabeled transfer data ($\mathcal{D}_T$), and labeled data ($\mathcal{D}_L$). First, using a masked language modeling technique, the compact model is pre-trained on $\mathcal{D}_{LM}$. Next, the large teacher model transfers the knowledge with $\mathcal{D}_T$ to the student model in the distillation phase.

According to the author, knowledge distillation is the technique (Hinton et al., 2015), where a bigger teacher model transfers the knowledge to a more compact model by producing *soft labels*, which is a probability distribution for class l, calculated as

$$p_l = softmax(z_l/\mathcal{T}) \tag{7}$$

Here, z_l is the last embedding layer value (logit) for class l, $\mathcal{T}$ is the temperature that can be tuned

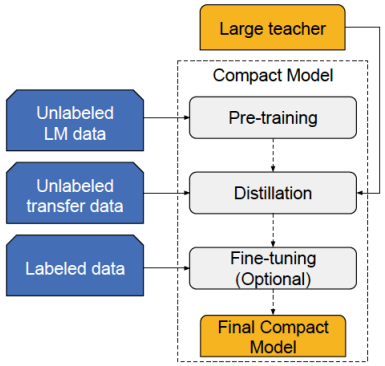

Figure 2: Pre-trained Distillation for Compact BERT model, the figure taken from Turc et al (p.2, 2019)

to smoothen the distribution. In the case of these 24 compact BERT models, the temperature value is chosen to be $\mathcal{T} = 1$; because the authors (Turc et al., 2019) did not observe an improvement in performance when changing this value of this parameter. Finally, the compact model is fine-tuned on $\mathcal{D}_L$.

2.3.2 DistilBERT

DistilBERT, introduced by (Sanh et al., 2019), is another example of a compact model, with the number of layers reduced by half[7], and has a similar architecture to that of a BERT-base model, but it uses the knowledge distillation technique also used

[7]Additionally token-type embedding and pooler are removed as well.

in compact BERT models but employs triple loss function. **Triple Loss:** Authors trained DistilBERT using a linear combination of three loss functions, cross-entropy distillation loss (L_{ce}), masked language modeling loss (L_{mlm}), and cosine embedding loss (L_{cos}) calculated by performing cosine operation on a hidden vector of student and teacher. Here the distillation loss L_{ce} is calculated as,

$$L_{ce} = \sum_i t_i * log(s_i) \tag{8}$$

Here t_i refers to the probabilities by the teacher, while s_i is the probabilities corresponding to student. Here t_i and s_i are calculated using softmax-temperature, described in equation [7][8].

According to (Sanh et al., 2019) **DistilBERT** is significantly faster (60%) and smaller (40%) than BERT-base. The authors evaluated DistilBERT for the sentiment classification task on the IMDb dataset, and it performed almost at par (accuracy of 92.82) with BERT-base (accuracy of 93.46). We intend to investigate the usage of DistilBERT for a similar task but with the Amazon Fine Foods dataset.

2.4 LIME

Local **I**nterpretable **M**odel agnostic **E**xplanation (LIME) (Ribeiro et al., 2016) is a popular explainable AI framework to understand the underlying pattern that black box models are trained with and for the inference about the predictions. LIME perturbs the black-box model, observes local changes, and provides a visual explanation for greater understanding. LIME is model agnostic, i.e. it can be used for any model and provide explanations.

Another widely used model for this application is SHAP analysis, **SH**apley **A**dditive Ex**P**lanation is an explainable AI framework, which was developed by (Lundberg and Lee, 2017)[9]. One notable advantage of LIME over SHAP is that it is relatively faster to model. However, the SHAP framework provides global explanations for the underlying model; this might be paramount in certain use cases where SHAP is the preferred method.

[8]Temperature for student and teacher is the same during training

[9]SHAP framework is based on the concept of Shapley values, that are used in game theory to assign the credits to participants based on the coalitions formed by them (Shapley, 1952)

	Predicted True	**Predicted False**
Actual True	True Positive (**TP**)	False Negative (**FN**)
Actual False	False Positive (**FP**)	True Negative (**TN**)

Table 1: Confusion Matrix

2.5 Evaluation Metrics

Here we present the definition of different evaluation metrics utilized in our analysis. In the table [1], we demonstrate the terminology associated with the confusion matrix.

2.5.1 Precision

Precision for the positive class can be calculated as:

$$Precision = \frac{TP}{TP + FP} \tag{9}$$

Precision for the negative class can be calculated as:

$$Precision = \frac{TN}{TN + FN} \tag{10}$$

2.5.2 Recall

Recall for the positive class can be calculated as:

$$Recall = \frac{TP}{TP + FN} \tag{11}$$

Recall for the negative class can be calculated as:

$$Recall = \frac{TN}{TN + FP} \tag{12}$$

2.5.3 F1-score

F1-score is the harmonic mean of Precision and Recall, which is calculated as follows:

$$F1 = \frac{2 \cdot Precision \cdot Recall}{Precision + Recall} \tag{13}$$

2.5.4 Accuracy

Accuracy for a given model can be computed using the following equation,

$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN} \tag{14}$$

3 Data

We have chosen the Amazon Fine Foods dataset hosted on the SNAP library affiliated with Stanford University. This dataset contains reviews of 74,258 unique products and 568,454 product reviews by 256,059 users, collected between October 1999 and October 2012.

The features of this dataset are described in the table [2]. Our analysis focused on the *text* and *score* variables. Firstly, in figure [3], the distribution of review score is presented, where the dataset

Feature	Description
productId	ASIN - Amazon Standard Identification Number
userId	reviewer identification
profileName	reviewer profile name
helpfullness	fraction of users who found the review helpful
score	score from 1 to 5
time	time of review post
summary	review summary
text	review text

Table 2: Feature Description for Amazon Fine Food dataset

Figure 3: Distribution of Review Scores

had a class imbalance, with a higher number of reviews with a score of 5. Secondly, in figure [4], the mean number of words for review ranges from 70 to 87; interestingly, the reviews that have scored 5 tend to be shorter comparatively than the other four classes of reviews. As we intend to perform a binary classification of the review dataset, the newly engineered dataset disregards the reviews with a score of 3, and the reviews with scores 1 and 2 are encoded with binary value '1' and reviews with scores 4 and 5 are encoded with '0'. After performing the aforementioned encoding, the dataset has 371,989 reviews with values '1' (positive reviews) and 68,965 reviews with values '0' (negative reviews); if we train our model with the imbalanced dataset, the results would be incorrect and untrustworthy. To mitigate this problem, we perform random undersampling using the `imblearn` python package, so the final dataset would have 68,965 reviews with positive and negative reviews classes each, as described in the table [3].

Type	Positive	Negative	Positive	Negative
Train	371989	68965	68965	68965
Test	65646	12170	65646	12170

Table 3: Distribution of Positive and Negative Class before and after Undersampling

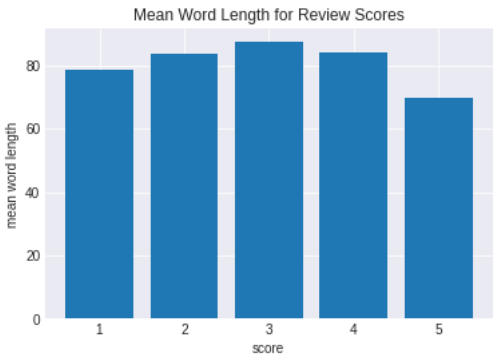

Figure 4: Distribution of mean of the number of words per review score

4 Method

The workflow of this project is visualized in figure 5. We have utilized python packages such as **scikit-learn** (Pedregosa et al., 2011), **imbalanced-learn** (Lemaître et al., 2017), **NumPy** (Harris et al., 2020), **Pandas** (Wes McKinney, 2010) (pandas development team, 2020) and **Matlplotlib** (Hunter, 2007).

4.1 Data Cleaning

The FineFoods dataset was scraped by (McAuley and Leskovec, 2013). The dataset is available in `txt.gz` format. We perform the following steps to clean the data according to our requirements. We utilized the `python3` distribution on Google Colab for experimentation. The data-cleaning procedure is described as follows,

1. Read the text file line by line and create a raw list. Converted this list of rows into a dictionary, disregarding extra features, if the raw has more than eight features.

2. The dictionary was converted to a `pandas` data frame for further visualization and primary analysis.

3. Finally, we converted the five-class classification problem into a binary classification problem, as documented in section [3].

4.2 Preprocessing

The preprocessing of the dataset, first we converted the dataset from the previous step into training

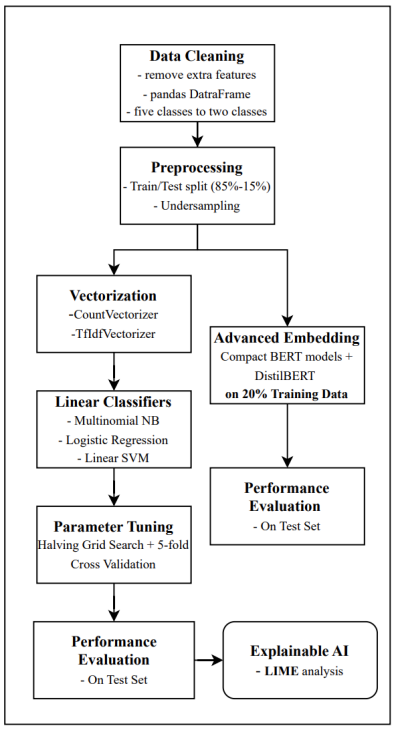

Figure 5: Workflow of the Project

and test sets using `train_test_split` [10] function from `sklearn` library, here the train-test split was selected to be 85%-15%. We used *stratify* option to maintain the same proportion of class division in training and test sets. We do not modify the test set anymore.

Furthermore, the class imbalance problem in the training dataset was addressed using `RandomUnderSampler`[11] from the `imblearn` library. The final training dataset has a 50%-50% division between positive and negative classes. However, this comes at the cost of losing some valuable data.

4.3 Vectorizer-Classifier Pipeline

We have used two vectorizers and three classifiers, as shown in the table [4] and [5], and set up six $(2 * 3)$ pipelines using `Pipeline` function[12] from the `sklearn` library.

Vectorizer	Parameters
Count Vectorizer and TF-IDF Vectorizer	ngram_range: [(1,1),(1,2),(1,3)] encoding:'latin-1' stop_words: 'english'

Table 4: Vectorizer with the parameters

[10] Available from `sklearn.model_selection`

[11] Available from the `imblearn.under_sampling` module

[12] Available from `sklearn.pipeline` module

We utilized the functions `CountVectorizer` and `TfidfVectorizer` from `sklearn`[13] library, there are three choices for **n-gram** as indicated in table [4], these are the *unigram*, *unigram + bigram* and *unigram + bigram + trigram*. Additionally, we choose `'latin-1'` encoding and remove the stop words using `stop_words` command. We

Classifier	Parameter
Multinomial NB	alpha: [1e-3,0.4, 0.8, 1,10]
Logistic Regression	C: [1e-3, 1e-2, 1e-1, 1, 10]
Linear SVM	C: [1e-3, 1e-2, 1e-1, 1, 10]

Table 5: Classifier with the grid of parameters

have utilized three classifiers from `sklearn` library using `MultinomialNB`, `LogisticRegression` and `LinearSVC` functions[14].

Here `MultinomialNB` function is optimized for the smoothing parameter `alpha`. While `LogisticRegression` (LR) is optimized on the regularization parameter `C`, furthermore, as this model utilizes the gradient descent method to update the weight vectors, we set the maximum number of iterations (`max_iter=100,000`) [15]. We utilize `lbfgs` solver (Byrd et al., 1995) for its robustness[16]. Thirdly we employ the `LinearSVC` function with the maximum number of iterations as `max_iter=100,000`. In order to reproduce the same results, we set the seed variable as `random_state=1729` for `LogisticRegression` and `LinearSVC` functions.

4.4 Halving Grid Search with Cross Validation

Tuning the parameters for six vectorizer-classifier pipelines is an expensive task, considering the fact that the training dataset is relatively large with 137,930 training samples; furthermore, we intend to perform a grid search on several combinations. To give a perspective, these six pipelines perform a grid search on 15 combinations of parameters.

Here, a conventional grid search technique might not be a time-optimal solution. Therefore, we approach this by utilizing Successive Halving (SH) originally described in the works of (Jamieson and Talwalkar, 2016) and (Li et al., 2016). This method

[13] Available from `sklearn.feature_extraction` module

[14] These functions are available from `naive_bayes`, `linear_model` and `svm` modules accordingly.

[15] Based on our primary experiments we realized that this model takes the considerably higher number of iterations before it converges.

[16] `http://users.iems.northwestern.edu/~nocedal/lbfgsb.html`

iteration	n_resource	no. of candidates
1	34,480	15
2	68,960	8
3	137,920	4

Table 6: Halving Grid Search, iteration and n_resource

is based on the tournament of candidates. Initially, all the candidate models are trained on a smaller dataset; out of these, the best-performing candidates survive and are further trained on a larger dataset iteratively, and the parameter space of the candidates keeps shrinking. Finally, the best-performing model is found.

We have used the `HalvingGridSearchCV` function available from `sklearn` library[17]. `HalvingGridSearchCV` has parameters such as `factor` and `min_resource`. According to the documentation, the parameter `factor` is the rate at which the number of training samples grows or the rate of reduction for the candidates. Here `min_resource` refers to the number of available samples in the first iteration.

$$n\_resource[i] = factor^{i-1} \cdot min\_resource; (i \geq 1) \tag{15}$$

In order to utilize this technique, choosing the values of factor and `min_resource` appropriately for a given number of candidates is important. In our experiments, we have chosen the value for `min_resource = 34,480` and `factor = 2` because in 3^{rd} iteration value of **n_resource** will grow to be **137,920** training samples, while the total training samples is **137,930**, so in the last iteration, the model will be able to perform the evaluation on almost entire training dataset to find the best-performing candidate.

We have chosen to five-fold cross-validation for better regularization. Finally, we re-train the best-performing model obtained on the entire data set with the corresponding parameters for further inference in the next stage; here, we use the `CalibratedClassifierCV` function from `sklearn` library. The reason for using `CalibratedClassifierCV` is because, as the `LinearSVC` does not provide probabilities associated with each class in the prediction, we utilize the `CalibratedClassifierCV` function available from `sklearn.calibration` model.

[17]Available from `sklearn.model_selection` module

4.5 Masked Language Models

We intend to train the compact BERT models and DistilBERT on the training set; these models are faster than base-BERT, but they still take a long time to train using TPU v2, so we first train these model on the 20% subset of training data. This 20% subset is further divided into train-validation sets with a split of 75%-25% to compute the epoch-wise loss and accuracy. First, we consider four

Model	No. of Layers	Hidden Size	Attention Heads	Parameters (millions)
BERT-128/2	2	128	2	4.4
BERT-256/2	2	256	4	9.7
BERT-128/4	4	128	2	4.8
BERT-256/4	4	256	4	11.3
DistilBERT	6	768	12	66

Table 7: Small BERT Models

compact BERT models, as described in table [7]; furthermore, we also utilized **DistilBERT** model, which is a larger model than other BERT variants in terms of the number of parameters.

4.6 Training compact BERT models

We train these models in Google Colab using `python3` distribution and TPU v2 accelerator. We build and train the compact BERT models using the TensorFlow interface (Abadi et al., 2015)and Keras (Chollet et al., 2015) package. We list these steps as follows,

1. We installed the `tensorflow`, and `tensorflow-text` version `2.9.0` in the python 3 environment[18].

2. A compact BERT classifier is built using the appropriate `preprocesser` followed by the `encoder`.

3. Next, the `pooled_output` from encoded text is passed through a `Dropout` layer with a rate of 0.1 to control overfitting.

4. The last layer is a `Dense` layer with a neuron that uses a `sigmoid` activation function, which has output 0 or 1.

5. We trained the models for `epochs=10` with `batch_size=256`. We trained the models by using adam optimizer (Kingma and Ba, 2014) with 0.001 learning rate and `binary_crossentropy` loss.

[18]Ancilliary packages such as `tensorflow_text` and `tensorflow_hub` are installed.

6. To minimize the training time, the training was performed on a subset (20%) of the original training data, which was further split into training and validation sets with a split of (75%-25%).

7. We evaluate the performance of these models on the test set.

4.7 LIME Inference

We used the `lime` package to learn about inference from misclassified samples from the best-performing Linear SVM model using the `LinearSVC` function, we used `LimeTextExplainer` from the `lime_text` module.

5 Results

5.1 Vectorizer-Classifier Pipeline

5.1.1 Using Counter Vectorizer

The classification report for the pipelines created using Count Vectorizer is represented in table [8], here all three classifiers seem to perform almost equally well. However, Linear SVM has slightly higher precision for negative class.

5.1.2 Using TF-IDF Vectorizer

The classification report for pipelines created using TF-IDF Vectorizer is presented in table [9]. Here Linear SVM and Logistic Regression seem to perform equally well, but Linear SVM has slightly higher precision for the negative class.

5.2 Compact BERT Model Results

The evaluation result for compact BERT models and DistilBERT are displayed in table [10], where DistilBERT has the highest training and validation accuracy. Next, the evaluation of compact BERT models and DistilBERT on the test set is documented in table [11], where DistilBERT has highest training and validation accuracy with minimum training and validation loss.

5.3 Inference using LIME Analysis

This section presents the confusion matrix for the final SVM model in figure [6], where labels 0 and 1 refer to the negative and positive classes, respectively.

5.3.1 False Negatives

We provide three examples of LIME text explainers for false negative samples visualized in figures

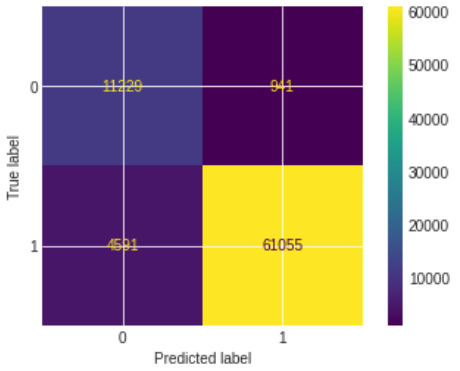

Figure 6: Confusion Matrix for the final SVM model

[7],[9] and [11], where one can observe the prediction probabilities for each class. Furthermore, the vertical graph presents the contribution of the top ten words affecting the class probabilities, highlighted in the adjacent review text. The words highlighted in blue correspond to the negative prediction of the review, while the words in orange represent the positive predictions.

5.3.2 False Positives

Here three examples of LIME text explainers for false positive samples are plotted in figures [13],[15], and [17].

6 Discussion

6.1 Vectorizer-Classifier Pipeline

In the first part of the project, we experimented with different classifiers with pipelines created with Count Vectorizer and TfidfVectorizer. These vectorizers are based on the *bag-of-word* idea for the corpus, where the context in the sentence is not considered. However, one can improve the understanding of context by incorporating the **n-gram** mechanism. In our experiments, with both the vectorizers, using a combination of unigram, bigram (and trigram) instead of just the unigram has improved the model's performance, which can be observed in tables [8] and [9].

According to Jurafsky and Martin (p.86, 2023), the Logistic Regression model performs better than the Multinomial NB model in larger datasets. The results in the table [8] and [9] demonstrate it to be true to a certain extent. Although the overall accuracy for Multinomial NB and Logistic Regression is almost similar, it is evident that the Precision (and F1-score) for the *negative class* is 2% to 3% higher for the Logistic Regression model. As the precision for the negative class is a ratio

Classifier	Class	Precision	Recall	F1-score	Accuracy	Tuned Parameters
Multinomial NB	negative	0.67	0.91	0.78		n_gram: (1,3)
	positive	0.98	0.92	0.95	0.92	alpha: 0.8
Logistic Regression	negative	0.69	0.93	0.79		n_gram: (1,3)
	positive	0.99	0.92	0.95	0.92	C:1
Linear SVM	negative	0.70	0.92	0.79		n_gram: (1,3)
	positive	0.99	0.93	0.95	0.93	C: 0.1

Table 8: Classification Report for the experiment with Count Vectorizer Pipeline + Classifier Pipelines, with Halving Grid Search CV

Classifier	Class	Precision	Recall	F1-score	Accuracy	Tuned Parameters
Multinomial NB	negative	0.65	0.93	0.76		n_gram: (1,3)
	positive	0.99	0.91	0.94	0.91	alpha: 0.4
Logistic Regression	negative	0.70	0.93	0.80		n_gram: (1,2)
	positive	0.99	0.93	0.95	0.93	C: 10
Linear SVM	negative	**0.71**	0.93	**0.80**		n_gram: (1,2)
	positive	**0.99**	0.93	**0.96**	0.93	C : 1

Table 9: Classification Report for the experiment with TF-IDF Vectorizer + Classifier Pipelines, with Halving Grid Search CV

Model	Training Loss	Validation Loss	Training Accuracy	Validation Accuracy
BERT-128/2	0.6108	0.5913	0.6649	0.6895
BERT-256/2	0.5566	0.5342	0.7158	0.7385
BERT-128/4	0.5953	0.5724	0.6807	0.7128
BERT-256/4	0.5419	0.5173	0.7282	0.7503
DistilBERT	0.3592	0.3552	0.8463	0.8507

Table 10: Evaluation of compact BERT models and DistilBERT with training and validation on the 20% of data, based on the last five epochs out of ten

Model	Class	Precision	Recall	F1-score	Support	Accuracy
BERT-128/2	negative	0.28	0.71	0.40	12170	
	positive	0.92	0.66	0.77	65646	0.66
BERT-256/2	negative	0.35	0.70	0.47	12170	
	positive	0.93	0.76	0.83	65646	0.75
BERT-128/4	negative	0.31	0.69	0.43	12170	
	positive	0.93	0.72	0.81	65646	0.71
BERT-256/4	negative	0.38	0.69	0.49	12170	
	positive	0.93	0.79	0.85	65646	0.77
DistilBERT	negative	**0.51**	0.86	0.64	12170	
	positive	**0.97**	0.84	0.90	65646	**0.85**

Table 11: Classification Report for compact BERT models and DistilBERT, on a test set (no tuning)

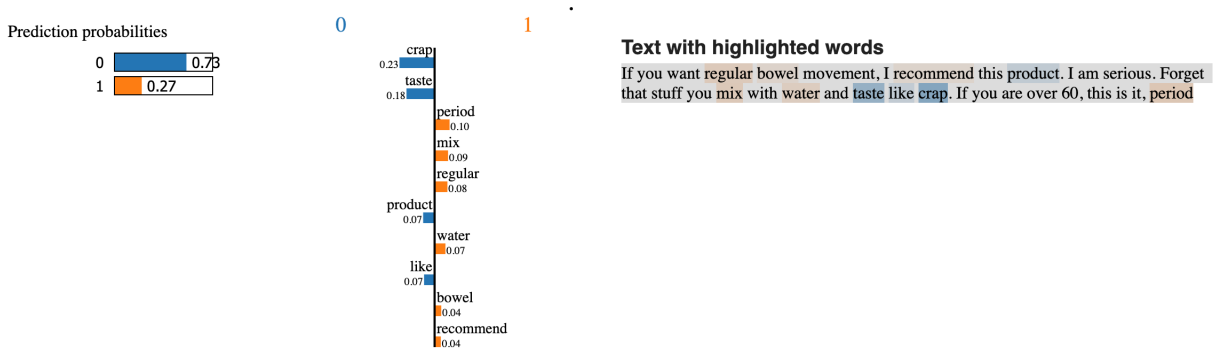

Figure 7: False Negative example-1

	productID	userID	profileName	helpfulness	score	time	summary	text
184729	B000FDMLUE	A2A5GOTBTDHJPF	William Clay	3/5	5	1196208000	I love this stuff . I eat it every day and goo...	If you want regular bowel movement, I recommen...

Figure 8: False Negative example-1 data frame row

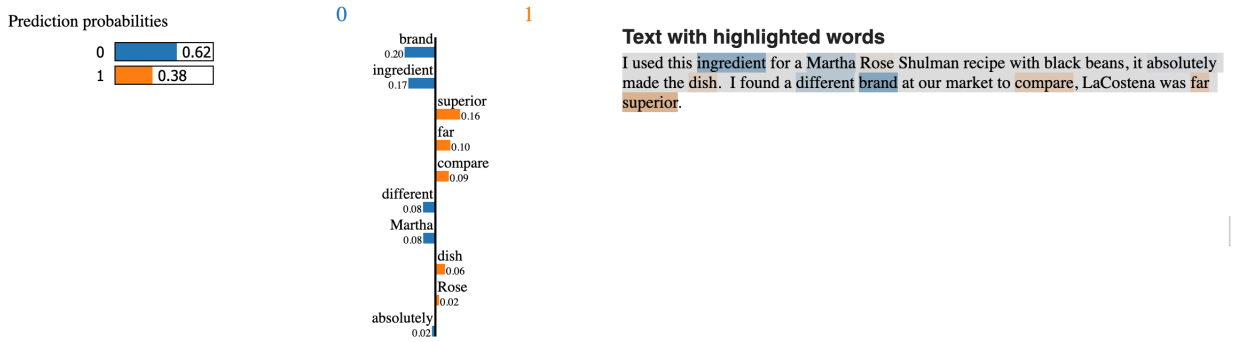

Figure 9: False Negative example-2

	productID	userID	profileName	helpfulness	score	time	summary	text
293729	B0000GGHVU	A1LPHVOMJOISTT	M. Buckley	0/0	5	1346630400	Really really good	I used this ingredient for a Martha Rose Shulm...
382084	B0000GGHU6	A1LPHVOMJOISTT	M. Buckley	0/0	5	1346630400	Really really good	I used this ingredient for a Martha Rose Shulm...

Figure 10: False Negative example-2 data frame row

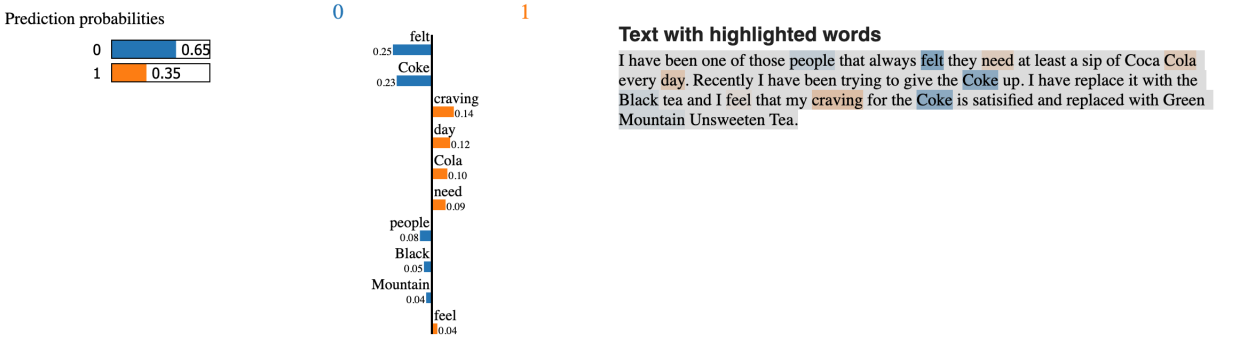

Figure 11: False Negative example-3

	productID	userID	profileName	helpfulness	score	time	summary	text
43783	B003M5W8ZK	A373MY0PQL9R41	Donna	0/0	5	1350864000	Black tea v Coca Cola	I have been one of those people that always fe...

Figure 12: False Negative example-3 data frame row

Figure 13: False Positive example-1

	productID	userID	profileName	helpfulness	score	time	summary	text
829	B000UWSQT0	A1LO4ME566EKLC	Show Me	0/1	1	1227916800	not for traditional caramel lovers	Not my idea of a good caramel. Flavors of gin...

Figure 14: False Positive example-1 data frame row

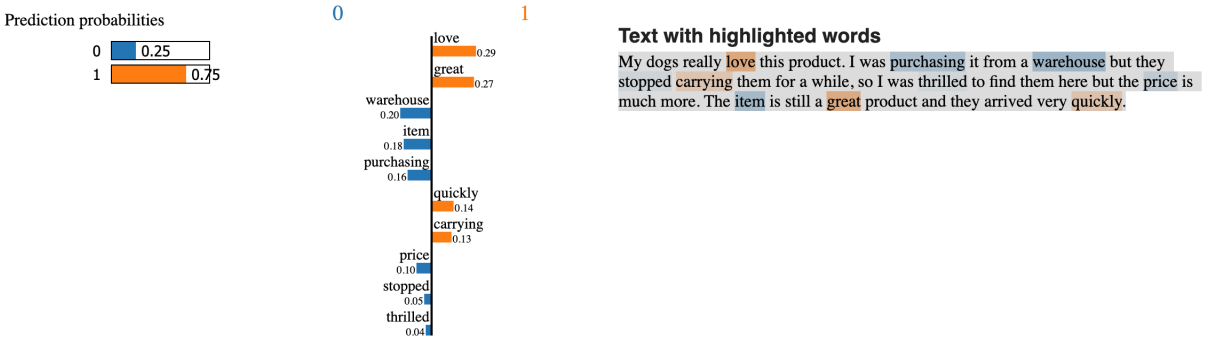

Figure 15: False Positive example-2

	productID	userID	profileName	helpfulness	score	time	summary	text
128820	B002BXVXB4	A3XG8M5E5ALNP	CG in NC	0/0	2	1350691200	Good Product, price was too high	My dogs really love this product. I was purcha...

Figure 16: False Positive example-2 data frame row

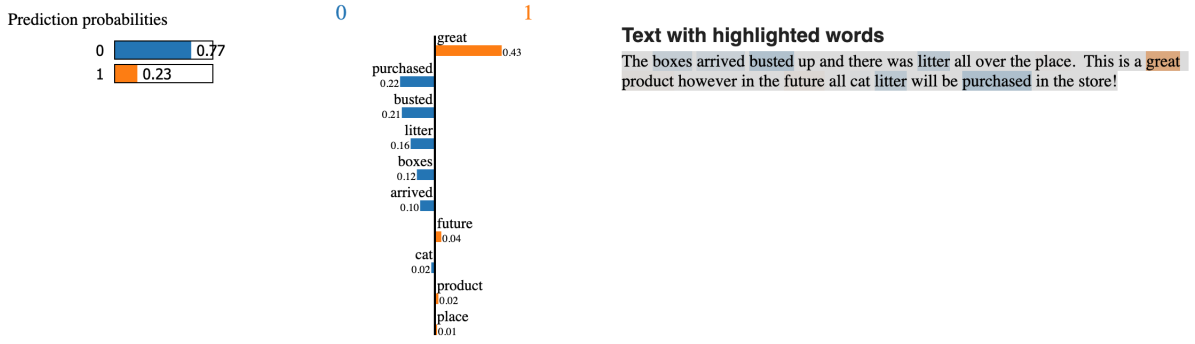

Figure 17: False Positive example-3

	productID	userID	profileName	helpfulness	score	time	summary	text
462245	B003LPZSI0	A2UBCLWZANH97V	Laurie "Laurie"	1/3	1	1320019200	Will never order online again	The boxes arrived busted up and there was litt...

Figure 18: False Positive example-3 data frame row

of True Negative to the sum of True Negative and False Negative, it can be inferred that the Logistic Regression model is better at minimizing false negatives.

The linear SVM model performs slightly better when comparing it with Logistic Regression for both pipelines; it can be observed that Precision for the negative class for linear SVM is 1% higher than Logistic Regression for both cases.

One major finding from the analysis was relatively poor precision for the negative reviews across all the experiments caused by higher falsely predicted negatives. It is important to note that the test set has a class imbalance, with a *positive-to-negative ratio* of approximately $5:1$.

6.2 Compact BERT models

In the second part of the project, four compact BERT and DistilBERT models were trained for ten epochs with 20% of the original training dataset; the results in terms of loss and accuracy are documented in table [10]. Firstly, we compared the compact BERT models with one another; it can be observed that increasing the number of layers (vertical stacking) BERT-128/2 to BERT-128/4 results in an improvement of accuracy.

6.3 LIME Inference

In the third part of the project, we performed LIME analysis for the False Negatives and False Positive samples identified using an optimized Linear SVM model.

- In the first example of the False Negative in figures [7] and [8], it can be observed that the words such as *crap* and *taste* overpowers the positive sentiment in the text, resulting in the misclassification. This particular review has a positive summary, and out of five users, three found this helpful, leading us to believe that *summary* of the review could be helpful in this disambiguation of the edge case.

- In the second example of the False Negative sample (in the figures [9] and [10]), words such as *brand* and *ingredient* have higher contributions towards the negative classification of this review. The same user repeated this review for the different *productID*.

- The third example of the False Negative (in the figures [11] and [12]), the word *Coke*, a colloquial term for the beverage *Coca Cola* has a higher impact on negative classification.

- The figures [13] and [14]) demonstrate the first False Positive sample, where the model was Unable to interpret *Not my idea of a good caramel* as a sentence with negative sentiment, and assigned higher probabilities to words *good*, *rich*, and *caramelized* which classified to be a positive review.

- Next, the second example of the false positive sample is plotted in the figures [15] and [16]). The misclassification caused by this type of review is difficult to correct since the user provided a positive review but rates with a lower score of **2**.

- The third example of the false positive sample is plotted in figures [17] and [18]), here the word such as *love* and *great* has a higher impact on misclassification of this sample this can be observed the summary of the product review has valuable information that user had a mixed sentiment about the purchase experience, which could be taken into account to prevent the misclassification.

7 Conclusion

In this project, we performed a Binary Sentiment Analysis for the Amazon Fine Food Review dataset.

In the first part of the project, we experimented with different vectorization techniques and machine-learning classifiers to find a model that delivered the highest performance based on the model evaluation metrics. We used the Halving Grid Search technique with five-fold cross-validation to find the optimum parameters in relatively less time. We observed the Linear SVM model with the TF-IDF vectorization model yielded 93% accuracy.

Next, we experimented with several MLMs, such as compact BERT models and DistilBERT, for the sentiment classification task. As these models require higher computing resources, we only utilized a subset of the original data for training these models. We observed that more parameters led to improved performance amongst the MLMs. The DistilBERT model, which has 15 times more parameters than the most miniature BERT model (BERT-128/2), has almost 18% improvement in the training and validation accuracies. The major limitation of

this work was the lack of fine-tuning for the DistilBERT model; due to this reason, the precision for the negative class is relatively poor (around 50%). In future works, we propose to include optimizer scheduling[19] using the `AdamW` (Loshchilov and Hutter, 2017) technique, which could efficiently slow down the learning processing for the initial epochs, leveraging the pre-trained model's knowledge[20].

Lastly, based on the final Linear SVM model, we performed the LIME technique on the six samples from the test set identified from the misclassified predictions. Several words incorrectly predicted the sentiment of the review *text*, so we propose to use review *summary* as *prior* class probabilities using the Multinomial NB classifier. Additionally, the *helpfulness* of the review could be used as well. However, we need to conduct more work to evaluate the effectiveness of these changes.

References

Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S. Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Ian Goodfellow, Andrew Harp, Geoffrey Irving, Michael Isard, Yangqing Jia, Rafal Jozefowicz, Lukasz Kaiser, Manjunath Kudlur, Josh Levenberg, Dandelion Mané, Rajat Monga, Sherry Moore, Derek Murray, Chris Olah, Mike Schuster, Jonathon Shlens, Benoit Steiner, Ilya Sutskever, Kunal Talwar, Paul Tucker, Vincent Vanhoucke, Vijay Vasudevan, Fernanda Viégas, Oriol Vinyals, Pete Warden, Martin Wattenberg, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng. 2015. TensorFlow: Large-scale machine learning on heterogeneous systems. Software available from tensorflow.org.

Thomas Bayes. 1763. Lii. an essay towards solving a problem in the doctrine of chances. by the late rev. mr. bayes, frs communicated by mr. price, in a letter to john canton, amfr s. *Philosophical transactions of the Royal Society of London*, (53):370–418.

Christopher M Bishop and Nasser M Nasrabadi. 2006. *Pattern recognition and machine learning*, volume 4. Springer.

Richard H Byrd, Peihuang Lu, Jorge Nocedal, and Ciyou Zhu. 1995. A limited memory algorithm for bound constrained optimization. *SIAM Journal on scientific computing*, 16(5):1190–1208.

François Chollet et al. 2015. Keras. `https://keras.io`.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2018. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*.

Charles R. Harris, K. Jarrod Millman, Stéfan J. van der Walt, Ralf Gommers, Pauli Virtanen, David Cournapeau, Eric Wieser, Julian Taylor, Sebastian Berg, Nathaniel J. Smith, Robert Kern, Matti Picus, Stephan Hoyer, Marten H. van Kerkwijk, Matthew Brett, Allan Haldane, Jaime Fernández del Río, Mark Wiebe, Pearu Peterson, Pierre Gérard-Marchant, Kevin Sheppard, Tyler Reddy, Warren Weckesser, Hameer Abbasi, Christoph Gohlke, and Travis E. Oliphant. 2020. Array programming with NumPy. *Nature*, 585(7825):357–362.

Geoffrey Hinton, Oriol Vinyals, and Jeff Dean. 2015. Distilling the knowledge in a neural network.

J. D. Hunter. 2007. Matplotlib: A 2d graphics environment. *Computing in Science & Engineering*, 9(3):90–95.

Kevin Jamieson and Ameet Talwalkar. 2016. Non-stochastic best arm identification and hyperparameter optimization. In *Proceedings of the 19th International Conference on Artificial Intelligence and Statistics*, volume 51 of *Proceedings of Machine Learning Research*, pages 240–248, Cadiz, Spain. PMLR.

Dan Jurafsky and James H Martin. 2023. *Speech and Language Processing*, 3rd edition.

Diederik P. Kingma and Jimmy Ba. 2014. Adam: A method for stochastic optimization.

Guillaume Lemaître, Fernando Nogueira, and Christos K. Aridas. 2017. Imbalanced-learn: A python toolbox to tackle the curse of imbalanced datasets in machine learning. *Journal of Machine Learning Research*, 18(17):1–5.

Lisha Li, Kevin Jamieson, Giulia DeSalvo, Afshin Rostamizadeh, and Ameet Talwalkar. 2016. Hyperband: A novel bandit-based approach to hyperparameter optimization.

Ilya Loshchilov and Frank Hutter. 2017. Decoupled weight decay regularization.

Scott M Lundberg and Su-In Lee. 2017. A unified approach to interpreting model predictions. In I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, editors, *Advances in Neural Information Processing Systems 30*, pages 4765–4774. Curran Associates, Inc.

Julian John McAuley and Jure Leskovec. 2013. From amateurs to connoisseurs: modeling the evolution of user expertise through online reviews. In *Proceedings of the 22nd international conference on World Wide Web*, pages 897–908.

The pandas development team. 2020. pandas-dev/pandas: Pandas.

[19] We have experimented with including this in the Google colab notebook. However, we do not include those results here due to time limitations.

[20] `https://www.tensorflow.org/text/tutorials/classify_text_with_bert`

F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. 2011. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830.

Marco Tulio Ribeiro, Sameer Singh, and Carlos Guestrin. 2016. "why should i trust you?": Explaining the predictions of any classifier.

Victor Sanh, Lysandre Debut, Julien Chaumond, and Thomas Wolf. 2019. Distilbert, a distilled version of bert: smaller, faster, cheaper, and lighter. *arXiv preprint arXiv:1910.01108*.

Hinrich Schütze, Christopher D Manning, and Prabhakar Raghavan. 2008. *Introduction to information retrieval*, volume 39. Cambridge University Press Cambridge.

LLOYD S Shapley. 1952. Quota solutions of n-person games. Technical report, RAND CORP SANTA MONICA CA.

Iulia Turc, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2019. Well-read students learn better: On the importance of pre-training compact models. *arXiv preprint arXiv:1908.08962*.

Vladimir Vapnik. 1999. *The nature of statistical learning theory*. Springer science & business media.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. 2017. Attention is all you need. *Advances in neural information processing systems*, 30.

Wes McKinney. 2010. Data Structures for Statistical Computing in Python. In *Proceedings of the 9th Python in Science Conference*, pages 56 – 61.

Yonghui Wu, Mike Schuster, Zhifeng Chen, Quoc V Le, Mohammad Norouzi, Wolfgang Macherey, Maxim Krikun, Yuan Cao, Qin Gao, Klaus Macherey, et al. 2016. Google's neural machine translation system: Bridging the gap between human and machine translation. *arXiv preprint arXiv:1609.08144*.