

Santa Clara University

Scholar Commons

Engineering Ph.D. Theses

Student Scholarship

6-2023

A Submodular Optimization Framework for Imbalanced Text Classification with Data Augmentation

Eyor Alemayehu

Follow this and additional works at: https://scholarcommons.scu.edu/eng_phd_theses



Part of the [Computer Engineering Commons](#)

SANTA CLARA UNIVERSITY

Department of Computer Science and Engineering

Date: June 2023

I HEREBY RECOMMEND THAT THE THESIS PREPARED UNDER

DR. YI FANG BY

Eyor Alemayehu


ENTITLED

A Submodular Optimization Framework for Imbalanced Text

Classification with Data Augmentation

BE ACCEPTED IN PARTIAL FULFILLMENT OF THE REQUIREMENTS
FOR THE DEGREE OF


DOCTOR OF PHILOSOPHY IN COMPUTER SCIENCE & ENGINEERING


Yi Fang (Jun 8, 2023 14:07 PDT)

Thesis Advisor
Dr. Yi Fang


N. Ling (Jun 10, 2023 08:42 PDT)

Chairman of Department
Dr. Nam Ling


Ying Liu (Jun 8, 2023 14:56 PDT)


Thesis Reader
Dr. Ying Liu



Thesis Reader
Dr. Yuhong Liu


Tokunbo Ogunfunmi (Jun 8, 2023 19:44 PDT)

Thesis Reader
Dr. Tokunbo Ogunfunmi


Zhiqiang Tao (Jun 9, 2023 14:06 EDT)

Thesis Reader
Dr. Zhiqiang Tao

A Submodular Optimization Framework for Imbalanced Text Classification with Data Augmentation

by

Eyor Alemayehu

Dissertation

Submitted in Partial Fulfillment of the Requirements
for the Degree of Doctor of Philosophy
in Computer Science and Engineering
in the School of Engineering at Santa Clara University, 2023

Santa Clara, California

To my parents

Acknowledgements

First of all, I would like to express how lucky I am to have Prof. Yi Fang as my advisor. Prof. Fang is not only an excellent educator and a mentor, but also a person with a deep sense of compassion and responsibility. He spent countless hours patiently getting me up to speed with my research topics and following up on my progress; sometimes at the expense of his family time. My gratitude to him is eternal, and I hope to follow his footsteps in being an outstandingly virtuous citizen.

I would like to thank the members of my doctoral committee - Prof. Yuhong Liu, Prof. Ying Liu, Prof Tokunbo Ogunfunmi, and Prof. Zhiqiang Tao - for setting aside time from their busy schedules to review my research.

I would like to thank Prof. Rani Mikkilineni, Dr. Minghwa Wang, and Mr. Zoltan Kurczveil for submitting letters of recommendations for my Ph.D. admission application. I would also like to thank my initial advisor the late Prof. JoAnne Holliday.

I would like to thank members of Prof. Fang research group. Through the numerous seminars we had, I have learned a lot from members of the group.

I would like to thank members of my family, friends, and work colleagues who gave me words of encouragement, and supported me throughout the Ph.D. program.

Lastly, I would like to thank Santa Clara University for giving me the opportunity to enroll in the Ph.D. program. Go Broncos!

A Submodular Optimization Framework for Imbalanced Text Classification with Data Augmentation

Eyor Alemayehu

Department of Computer Science and Engineering

Santa Clara University

Santa Clara, California

2023

ABSTRACT

In the domain of text classification, imbalanced datasets are a common occurrence. The skewed distribution of the labels of these datasets poses a great challenge to the performance of text classifiers. One popular way to mitigate this challenge is to augment underwhelmingly represented labels with synthesized items. The synthesized items are generated by data augmentation methods that can typically generate an unbounded number of items. To select the synthesized items that maximize the performance of text classifiers, we introduce a novel method that selects items that jointly maximize the likelihood of the items belonging to their respective labels and the diversity of the selected items. Our proposed method formulates the joint maximization as a monotone submodular objective function, whose solution can be approximated by a tractable and efficient greedy algorithm. We evaluated our method on multiple real-world datasets with different data augmentation techniques and text classifiers, and compared results with several baselines. The experimental results demonstrate the effectiveness and efficiency of our method.

Contents

Acknowledgements	iv
Abstract	iv
Contents	vi
List of Figures	viii
List of Tables	x
1 Introduction	11
1.1 Contributions	15
1.2 Outline	16
2 Background	17
2.1 Multiclass Classification	17
2.2 Multiclass Text Classification	22
2.2.1 Bag-Of-Words	24
2.2.2 Recurrent Neural Network	25
2.2.3 Convolutional Neural Network	26
2.2.4 Bidirectional Encoder Representations From Transformers	31
3 Related Work	35
3.1 Data Space	35
3.1.1 Character Level	36
3.1.2 Word Level	36
3.1.3 Phrase Level	38
3.1.4 Document Level	39
3.2 Feature Space	40

4	The Proposed Framework	42
4.1	Overview	42
4.2	Individual Components	45
4.2.1	Text Generator	45
4.2.2	Document Scorer	46
4.2.3	Document Selector	46
4.2.4	Data Augmentation Pipeline	47
4.2.5	Diversity Selector	48
4.2.6	Monotone Submodular Optimization	50
4.2.7	SOFITDA Selector	52
4.2.8	Computational Complexity of SOFITDA Selector	55
5	Experiments	57
5.1	Datasets	57
5.2	Pre-processing	59
5.3	Setup	60
5.3.1	CONFIGURE_GENERATOR Functions	60
5.3.2	GENERATOR Functions	61
5.3.3	TRAIN_SCORER Functions	61
5.3.4	SCORER Function	64
5.3.5	CONFIGURE_SELECTOR Function	64
5.3.6	SELECTOR Function	67
5.3.7	TAA Baseline	67
5.4	Evaluation Methodology	67
5.5	Results	69
5.5.1	Optimal α and N_C	69
5.5.2	Baseline Comparison	72
5.5.3	Likelihood versus Diversity	78
5.5.4	Hyperparameter Analysis	79
5.5.5	Cluster Analysis	85
5.5.6	Time Efficiency	88
6	Conclusion and Future Work	90
7	Appendix	93
7.1	Monotone Submodularity Proof	93
7.1.1	Theorem 1 (Submodularity)	93
7.1.2	Theorem 2 (Monotonicity)	95
	Bibliography	96

List of Figures

2.1	A typical architecture of a text classifier	22
2.2	The architecture of a Recurrent Neural Network	26
2.3	A typical architecture of a convolutional layer	30
2.4	A typical architecture of a CNN	31
2.5	BERT classification architecture	34
4.1	The steps of the Data Augmentation Pipeline for getting synthetic items for a single minority label	43
5.1	Plots of $H(x) = x^\alpha$ for different values of α . Notice how the concavity of $H(x)$ decreases as the value of α goes from 0 to 1.	65
5.2	Charts showing the mean percentage of synthesized items (Overlap Per- centage) selected by the SOFITDA selectors that are also selected by the TOP baseline on a per α value basis for each unique combination of dataset, generator, and classifier.	80
5.3	Charts showing the mean percentage of synthesized items (Overlap Per- centage) selected by the SOFITDA selectors that are also selected by the TOP baseline on a per N_C value basis for each unique combination of dataset, generator, and classifier.	81
5.4	Charts showing the mean macro F-Score of the SOFITDA selectors versus their α values for each unique combination of dataset, generator, and classifier.	83
5.5	Charts showing the mean macro F-Score of the SOFITDA selectors versus their N_C values for each unique combination of dataset, generator, and classifier.	84
5.6	Charts showing the projection of the TF-IDF vectors of 1000 synthesized items in two of the most significant PCA dimensions. The charts are for the EDA generator for each unique combination of dataset, and classifier. The points shown in blue represent synthesized items selected only by the TOP selector, and the points shown in red represent synthesized items selected only by the SOFITDA selector.	86

-
- 5.7 Charts showing the projection of the TF-IDF vectors of 1000 synthesized items in two of the most significant PCA dimensions. The charts are for the GPT-2 generator for each unique combination of dataset, and classifier. The points shown in blue represent synthesized items selected only by the TOP selector, and the points shown in red represent synthesized items selected only by the SOFITDA selector. 87
- 5.8 A scatter plot of the mean execution time of the SOFITDA selector for each dataset as specified in Table 5.11. The plot clearly shows a linear correlation between the execution time and the number of items selected. 89

List of Tables

5.1	The distribution of the labels for each dataset.	59
5.2	The partition and vocabulary sizes of each dataset.	60
5.3	The α and N_C values of the SOFITDA selector that yields the highest macro F-Score value on a per dataset, generator, and classifier basis. . . .	70
5.4	The mean α and N_C values of the SOFITDA selector that yields the highest macro F-Score value on a per-classifier basis.	71
5.5	The mean α and N_C values of the SOFITDA selector that yields the highest macro F-Score value on a per-dataset basis.	71
5.6	The mean α and N_C values for the SOFITDA selector that yields the highest macro F-Score value on a per-generator basis.	72
5.7	The macro F-Score of SOFITDA and the baseline selectors on a per dataset, generator, and classifier basis. Bold font indicates the best results. \dagger indicates a statistically significant improvement over the NO_AUG baseline.	74
5.8	The macro F-Score of the TAA baseline and SOFITDA on a per-dataset basis. We compare the TAA baseline with a SOFITDA model that uses BERT as a classifier and EDA as a generator. Bold font indicates the best results.	75
5.9	The mean relative improvement in percentage of the macro F-Score (F), macro precision (P), and macro recall (R) of the SOFITDA selector over the baseline selectors on a per classifier basis averaged across the datasets and generators.	75
5.10	The relative improvement in percentage of the macro F-Score, macro precision, and macro recall of the SOFITDA selectors configured with a BERT classifier and an EDA generator over their equivalent TAA baselines averaged across all datasets. Note, the TAA baselines use a BERT classifier as well as generation methods used by the EDA generator. . . .	75
5.11	The total number of synthetic items selected for each dataset as well as the mean execution time in seconds for executing the SOFITDA selector on a per-dataset basis for all labels	88

Chapter 1

Introduction

In machine learning, a multiclass classification task learns from training data the parameters of a model (classifier) that predicts the likelihood of a given input having a particular label as an attribute. The likelihood is typically formulated as a probability score across all possible labels. The training data the classification task learns from is known as a dataset and is composed of a set of input and label pairs. An input is represented by a set of numeric values known as features, and a label is a part of a mutually exclusive set of labels. An example of a dataset is a set of movie review and sentiment pairs, where the sentiment is a label that has a value of positive or negative. In this example, since the movie reviews are text data, the multiclass classification task is known as text classification.

Imbalanced datasets are datasets that have a skewed distribution of labels, where a small subset of labels have an overwhelmingly dominant representation in the distribution.

In the text classification domain, we frequently run into such datasets. For example, a dataset for fake news articles consists of an overwhelmingly large proportion of legitimate news articles when compared to fake ones. The skewed distribution of the labels poses a serious challenge for text classification models because the models develop a bias for the dominant labels. This bias typically manifests itself as false positives where some items that have underwhelmingly represented labels (minority labels) are incorrectly classified as having dominant labels (majority labels).

To mitigate this problem, two approaches can be used. The first approach is to employ re-sampling methods [21]. The purpose of these methods is to under-sample majority label items, and/or over-sample minority label items. In the under-sampling case, items are removed from the training set, while in the over-sampling case, they are replicated. The challenge with re-sampling is under-sampling may omit useful training examples, and over-sampling may lead to overfitting.

The second approach is to employ data augmentation methods. The data augmentation methods synthesize new training items or mutate existing ones for the purpose of adding them to the training set. The goal of data augmentation is to increase the amount of training data available by doing label-preserving transformations [18]. In the domain of text classification, data augmentation can be done on the data space or the feature space. The data space is the input text whereas the feature space is the learning representation of the input text. The data augmentation methods in the text classification domain are either rules-based or machine-learning-based. An example of a rules-based method is Easy Data Augmentation (EDA) [77], which works in the data space by mutating items

in the training set via synonym replacement, random word insertion, random word swapping, and random word deletion. An example of a machine-learning-based method is the fine-tuning of a generative language model such as the "Generative Pre-trained Transformer 2" (GPT-2) [57] on the training items of a given label, and then using the model to generate similar items in the data space [9, 74].

In nearly all cases, data augmentation methods can either generate significantly more distinct items than necessary or even an unbounded number of distinct items. In contrast, re-sampling methods are only limited to the items in the training data. Therefore, provided the distribution of the items generated is sufficient, data augmentation generally helps text classification models perform better on unseen data than re-sampling. Recently, the advent of large language models such as "Bi-directional Encoder Representations from Transformers" (BERT) [19] and GPT-2, which are trained on large corpora, allow through transfer learning the generation of high-quality data augmentation items. Furthermore, rules-based approaches such as EDA, which frequently generate items that have semantic and syntactic errors, outperform re-sampling, because they add noise to the training set that results in better generalization.

The ability of data augmentation methods to generate significantly more distinct items than necessary allows the possibility for over-generating synthesized items and then selecting a subset of them that yield the best performance. Several works leverage this possibility on an ad-hoc basis. For example, the HotFlip [20] method selects the subset which increases the loss (error) of a classification model that is trained without data augmentation. This is because the purpose of HotFlip is to add adversarial samples

to the training data that the classification model fails to classify as true positives. Another example is the "Language-Model-Based Data Augmentation" (LAMBADA) algorithm [8] which selects the subset that yields the highest label probabilities on a classification model that is trained without data augmentation.

While task-specific ad-hoc approaches were proposed for selecting a subset of the required number of items, to the best of our knowledge, there exists no principled approach that generalizes some of the existing approaches and opens up the possibility of better performance through the introduction of additional configuration parameters. In this thesis, we propose a principled approach to select the optimal subsets of items required to balance the minority labels of an imbalanced dataset. Our approach for selecting a synthesized item to balance a minority label is based on maximizing:

1. The likelihood of the selected items belonging to the minority label.
2. The diversity of the selected items.

By maximizing the likelihood of the selected items belonging to the minority label, we maximize the probability of the selected items being true positives for the minority label. By maximizing the diversity of the selected items, we maximize the scope of the minority label distribution seen during training. Furthermore, diversification allows us to reduce the redundancy of similar items and induces regulating noise. This makes the resulting model more resilient to overfitting.

Our work proposes a method to optimize the twin objectives of maximizing the likelihood and the diversity of the selected items. To estimate the likelihood of the selected items, we propose training a text classifier on the non-augmented training set and then using the classifier to compute the probabilities of the synthesized items belonging to their corresponding minority labels. To estimate the diversity of the selected items, we propose using a similarity function that compares the features of two synthetic items. We then utilize both estimation methods to formulate a combinatorial joint objective for each minority label where we maximize the likelihood and the diversity of the synthesized items such that we only select enough items to balance the minority label. However, the combinatorial joint objective is an NP-hard problem. Therefore, instead of directly solving this objective, we approximate it by formulating a monotone submodular objective. Having a monotone submodular objective allows us to utilize a tractable greedy algorithm that is guaranteed to give us a solution that is within $(1 - 1/e)$ of the optimal solution [17].

Our work is independent of the method used to synthesize items such as EDA or GPT-2. It is exclusively limited to selecting a subset of items synthesized by a given method as described above. This allows it to be used with any text synthesis method for the purpose of augmenting imbalanced-text datasets.

1.1 Contributions

In summary, the main contributions of this thesis are as follows:

- We derive a monotone submodular objective that selects items from a pool of synthesized items such that the likelihood of the items belonging to their respective minority label and their diversity are approximately maximized. We solve this maximization by an efficient greedy algorithm.
- We introduce an abstract data augmentation pipeline that can use any selection process including the monotone submodular objective to augment an imbalanced dataset in a series of well-defined steps.
- We conduct comprehensive experiments on multiple real-world datasets and configurations to demonstrate the effectiveness and efficiency of our proposed approach.

This thesis is based on our published work [5].

1.2 Outline

This thesis is organized as follows: Chapter 2 gives a brief background about text classification, Chapter 3 covers related work in data augmentation for text classification, Chapter 4 specifies an abstract data augmentation pipeline and derives the monotone submodular objective that is used to select synthesized items, Chapter 5 outlines the setup of our experiments and discusses the results, and Chapter 6 makes concluding remarks and suggests potential extensions to our work.

Chapter 2

Background

2.1 Multiclass Classification

In multiclass classification, given a set of inputs \mathbf{X} , and a set of labels, \mathbf{Y} , our objective is to build a model (classifier) that predicts the label of a given input. Each $\mathbf{x}_i \in \mathbf{X}$ is represented by K numeric features such that $\mathbf{x}_i \in \mathbb{R}^{K \times 1}$. Each $y_i \in \mathbf{Y}$ is the one and only label of \mathbf{x}_i , where M is the number of distinct labels.

A common type of multiclass-classification model is known as a discriminative model.

A discriminative model estimates the conditional probability distribution $p(y|\mathbf{x})$ where $\mathbf{x} \in \mathbf{X}$ is an input and $y \in \mathbf{Y}$ is a label.

We estimate $p(y|\mathbf{x})$ by a function $f(\cdot)$ that returns a vector of probabilities for the labels such that:

$$p(y|\mathbf{x}) \approx f_y(\mathbf{x}; \boldsymbol{\theta}) \quad (2.1)$$

where $\boldsymbol{\theta}$ is the set of parameters of the function and the y in f_y represents the probability in the vector that corresponds to the label of y .

We find the optimal $\boldsymbol{\theta}$ by drawing a set of $\{\mathbf{x}, y\}$ samples from $P(\mathbf{X}, \mathbf{Y})$ and then determining the $\boldsymbol{\theta}$ that minimizes the expected difference between the empirical distribution of the samples and $f(\cdot)$. We can express this as follows:

$$\boldsymbol{\theta}^* = \arg \min_{\boldsymbol{\theta}} \sum_{\mathbf{x} \in \mathbf{X}} p(\mathbf{x}) \sum_{y \in \mathbf{Y}} \mathcal{L}(p(y|\mathbf{x}), f_y(\mathbf{x}; \boldsymbol{\theta})) \quad (2.2)$$

where $\boldsymbol{\theta}^*$ is the optimal $\boldsymbol{\theta}$ and $\mathcal{L}(\cdot)$ computes the difference between $p(y|\mathbf{x})$ and $f_y(\mathbf{x}; \boldsymbol{\theta})$ for a drawn sample $\{\mathbf{x}, y\}$. Note that $p(y|\mathbf{x})$ in the empirical distribution has a probability of 1 if $\{\mathbf{x}, y\}$ is observed or 0 if it is not observed.

$\mathcal{L}(\cdot)$ is known as a loss function and the smaller the value it returns the more similar $p(y|\mathbf{x})$ and $f(\mathbf{x}; \boldsymbol{\theta})$ are. There are different types of loss functions, but the one that is most widely used in multiclass classification is the cross-entropy loss function, which we define as follows:

$$\mathcal{L}(p(\mathbf{x}), q(\mathbf{x})) = -p(\mathbf{x}) \log q(\mathbf{x}) \quad (2.3)$$

We obtain the cross-entropy loss function when deriving Equation (2.2) from the Kullback-Leibler (KL) divergence equation [37]. KL divergence is a measure of the difference between two probability distributions sampled from the same space. The definition of KL divergence for two discrete and conditional probability distributions is as follows:

$$D_{KL}(p \parallel q) = \sum_{\mathbf{x} \in \mathbf{X}} \sum_{y \in \mathbf{Y}} p(\mathbf{x}, y) \log \frac{p(y|\mathbf{x})}{q(y|\mathbf{x})} \quad (2.4)$$

Using this definition, the KL divergence of the conditional distribution $p(y|\mathbf{x})$ and our estimation for the distribution, $f(\cdot)$, is as follows:

$$D_{KL}(p \parallel f) = \sum_{\mathbf{x} \in \mathbf{X}} \sum_{y \in \mathbf{Y}} p(\mathbf{x}, y) \log \frac{p(y|\mathbf{x})}{f_y(\mathbf{x}; \boldsymbol{\theta})} \quad (2.5)$$

The optimal $\boldsymbol{\theta}$ minimizes the KL divergence to reduce the difference between $p(y|\mathbf{x})$ and $f(\cdot)$ as follows:

$$\begin{aligned} \boldsymbol{\theta}^* &= \arg \min_{\boldsymbol{\theta}} \sum_{\mathbf{x} \in \mathbf{X}} \sum_{y \in \mathbf{Y}} p(\mathbf{x}, y) \log \frac{p(y|\mathbf{x})}{f_y(\mathbf{x}; \boldsymbol{\theta})} \\ &= \arg \min_{\boldsymbol{\theta}} \sum_{\mathbf{x} \in \mathbf{X}} p(\mathbf{x}) \sum_{y \in \mathbf{Y}} p(y|\mathbf{x}) \log \frac{p(y|\mathbf{x})}{f_y(\mathbf{x}; \boldsymbol{\theta})} \\ &= \arg \min_{\boldsymbol{\theta}} \sum_{\mathbf{x} \in \mathbf{X}} p(\mathbf{x}) \sum_{y \in \mathbf{Y}} p(y|\mathbf{x}) \log p(y|\mathbf{x}) - \sum_{\mathbf{x} \in \mathbf{X}} p(\mathbf{x}) \sum_{y \in \mathbf{Y}} p(y|\mathbf{x}) \log f_y(\mathbf{x}; \boldsymbol{\theta}) \\ &= \arg \min_{\boldsymbol{\theta}} - \sum_{\mathbf{x} \in \mathbf{X}} p(\mathbf{x}) \sum_{y \in \mathbf{Y}} p(y|\mathbf{x}) \log f_y(\mathbf{x}; \boldsymbol{\theta}) \end{aligned} \quad (2.6)$$

Substituting the cross-entropy loss function into the above equation, we get:

$$\boldsymbol{\theta}^* = \arg \min_{\boldsymbol{\theta}} \sum_{\mathbf{x} \in \mathbf{X}} p(\mathbf{x}) \sum_{y \in \mathbf{Y}} \mathcal{L}(p(y|\mathbf{x}), f_y(\mathbf{x}; \boldsymbol{\theta})) \quad (2.7)$$

which is the same as Equation (2.2).

A basic method to find an approximation to the optimal $\boldsymbol{\theta}$ is to use an algorithm called the Mini-Batch Gradient Descent [40]. We show the high-level steps of this algorithm in Algorithm 1. A description of the steps is as follows:

- **Step 1:** Initialize the $\boldsymbol{\theta}$ values. Typically, we randomly set them to small values.
- **Steps 3 - 6:** Sample a batch of N inputs.
- **Step 7:** Compute the difference between the empirical and the estimated distribution of the batch. Assign the difference to $J(\boldsymbol{\theta})$.
- **Steps 8-11:** Update each parameter in $\boldsymbol{\theta}$ by subtracting from it a multiplicative factor, γ , of its partial derivative of $J(\boldsymbol{\theta})$. γ is known as the learning rate. The update attempts to modify each parameter such that the value of $J(\boldsymbol{\theta})$ is smaller.
- **Steps 12:** We loop back to Step 3 until $J(\boldsymbol{\theta})$ converges. We achieve convergence if the difference between the previous and current value of $J(\cdot)$ falls below a given threshold, ϵ .

Algorithm 1 Mini-Batch Gradient Descent

```

1:  $\theta \leftarrow \text{INITIALIZE}()$ 
2: do
3:    $B \leftarrow \emptyset$ 
4:   for each  $i \in 1, \dots, N$  do
5:      $B \leftarrow B \cup \{x \sim X\}$ 
6:   end for
7:    $J(\theta) \leftarrow \sum_{x \in B} p(x) \sum_{y \in Y} \mathcal{L}(p(y|x), f_y(x; \theta))$ 
8:    $\hat{\theta} \leftarrow \theta$ 
9:   for each  $i \in \{1, \dots, |\theta|\}$  do
10:     $\theta_i = \theta_i - \gamma \frac{\partial J(\hat{\theta})}{\partial \theta_i}$ 
11:   end for
12: while  $(J(\hat{\theta}) - J(\theta)) > \epsilon$ 

```

In practice, substantial modifications are made to the Mini-Batch Gradient Descent algorithm to make it numerically stable, efficient, and approximate a good value for the optimal θ . Some of the modifications are:

- Utilizing a dynamic programming algorithm called backpropagation [61] to compute the partial derivatives.
- Applying a heuristics to dynamically change the learning rate, γ . The ADAM optimizer [31] is an example of such a heuristics.
- Not updating a random subset of the parameters when processing a batch of data to mitigate the model overfitting the training data. This technique is known as Dropout [73]. Overfitting results in good performance on the training data, but weak performance on unseen data.
- Normalizing the internally computed values of a batch by $f(\cdot)$ by a process known as Batch Normalization [28]

2.2 Multiclass Text Classification

Multiclass text classification, which we simply refer to as text classification, is a specific type of multiclass classification where the input data is natural language text. An example of such input data are customer product reviews that are labeled with a 1-5 star-rating system. There are many different types of text classification models (classifiers). Figure 2.1 illustrates the architecture of a typical text classifier.

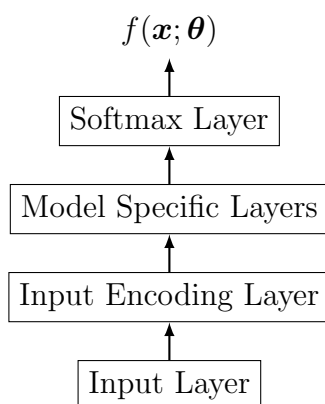


FIGURE 2.1: A typical architecture of a text classifier

With the exception of the Input Layer, all the steps in the architecture can be combined into a single function, $f(\mathbf{x}; \boldsymbol{\theta})$, that approximates $p(y|\mathbf{x})$ as discussed in the previous section. The description of each step of the architecture is as follows:

- **Input Layer:** The input to the text classifier are the tokens of the text represented by vectors. The tokenization process for splitting the text into tokens is classifier dependent. Each unique token is represented by a unique one-hot encoding vector, where the size of the vector is the number of unique tokens. A one-hot encoding vector is a vector such that one of its elements is set to one and the rest of

its elements are set to zero. The one-hot encoding vectors of the tokens are concatenated together in the order of their tokens and passed to the next step.

- **Input Encoding Layer:** The sequence of one-hot encoding vectors are transformed into a new linear representation. In most cases, the transformation either converts the sequence of one-hot encoding vectors into a single vector, or reduces the dimensions of the one-hot encoding vectors. The transformation can be dependent or independent of the order of the one-hot encoding vectors in the sequence. Furthermore, the transformation parameters can be imported from another model in a process known as transfer learning. The vectors obtained from such transformations are known as embeddings. A one-hot encoding vector is mapped to its corresponding embedding by multiplying it with a matrix of the embeddings such that the slot of the one-hot encoding vector that is set to one selects the corresponding embedding as an output of the multiplication.

The type of transformation in this layer is classifier dependent.

- **Model Specific Layers:** The logic that is specific to the classifier is implemented by this step. This step can comprise of multiple classifier specific layers. The output of this step is a vector of values that are known as logits, where each element in the vector corresponds to a label.
- **Softmax Layer:** The logits from the previous steps are normalized into probability values for their corresponding labels by this step. The normalization is done

by a softmax operation that is as follows:

$$p(y_i|\mathbf{x}) \approx \frac{e^{z_i}}{\sum_{j=1}^M e^{z_j}} \quad (2.8)$$

where M is the number of labels and z_i is the logit for label i .

In the subsections below, we briefly discuss the implementation of widely used text classifiers.

2.2.1 Bag-Of-Words

The Bag-Of-Words [26] [33] text classifier is one of the simplest and earliest text classifiers. The architecture of the classifier is as follows:

- **Input Layer:** Any tokenization process can be used. A simple approach is to use white space (space, tabs, and new lines) to split the text into tokens.
- **Input Encoding Layer:** The one-hot encoding vectors of the vectors are logically OR-ed to create a single vector for the given text. This vector has elements that correspond to the tokens in the text set to one whereas the rest of its elements are set to zero.
- **Model Specific Layers:** At a minimum, this step consists of a single layer that linearly transforms the vector of the text to the vector of the logits. The

transformation is as follows:

$$\mathbf{z} = \mathbf{A}\mathbf{w} + \mathbf{b} \quad (2.9)$$

where $\mathbf{z} \in \mathbb{R}^{M \times 1}$ (M is the number of labels), $\mathbf{A} \in \mathbb{R}^{M \times V}$ (V is the number of unique tokens), $\mathbf{b} \in \mathbb{R}^{M \times 1}$, and $\mathbf{w} \in \mathbb{R}^{V \times 1}$ is the vector of the input text encoded by the previous layer. Note that $\mathbf{A} \in \boldsymbol{\theta}$ and $\mathbf{b} \in \boldsymbol{\theta}$.

2.2.2 Recurrent Neural Network

A recurrent neural network (RNN) [7] classifier models a sequence of items using a feedback loop that is executed on every item in the sequence. The architecture of the classifier is as follows:

- **Input Layer:** Any tokenization process can be used. A simple approach is to use white space to split the text into tokens.
- **Input Encoding Layer:** Typically transfer learning is used for the vector representation of tokens. In this case, the vectors of tokens trained by a language model on a large corpus are used. An example of such model is GloVe [55].
- **Model Specific Layers:** We show the feedback loop of an RNN in Figure 2.2. When an RNN processes an item in the i^{th} position of the sequence, \mathbf{x}_i , it uses the state of the previous item, \mathbf{h}_{i-1} . The RNN computes the state of an item by the function $g(\cdot)$. In the initial case where we process \mathbf{x}_1 , we typically set \mathbf{h}_0 to the zero vector, because there is no previous item. The purpose of the feedback loop

is to model the context of each item. This is especially important when processing text, because the meaning of a word could depend on its context. We use the state of the last item in the sequence to estimate the probability distribution of the labels of the sequence. We transform the state to a vector of logits using the function $t(\cdot)$.

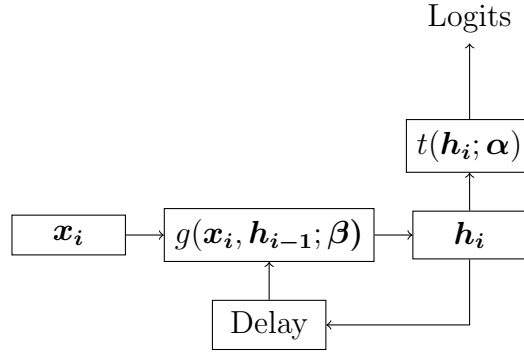


FIGURE 2.2: The architecture of a Recurrent Neural Network

In practice, due to numerical instability challenges under training, the RNN architecture shown in Figure 2.2 is not used as is. Complex modifications are made to the architecture to ensure the RNN is stable during training. Examples of models that do such modifications are the Long Short-Term Memory (LSTM) [27] and the Gated Recurrent Unit [15].

2.2.3 Convolutional Neural Network

A convolutional neural network (CNN) [23] [54] [38] classifier creates features by doing a weighted-sum of neighboring items starting from the input sequence and recursively

afterwards until a maximum depth is reached. The idea behind CNNs is to hierarchically build shift-invariant features that are then transformed to predict the probability distribution of a given sequence of items.

- **Input Layer:** Any tokenization process can be used. A simple approach is to use white space to split the text into tokens.
- **Input Encoding Layer:** Typically transfer learning is used for the vector representation of tokens. For example, the vectors of tokens trained by GloVe can be used.
- **Model Specific Layers:** Before we discuss the model specific layers, we specify the various components that make up a convolution layer.

Given:

- a sequence of items represented by the matrix $\mathbf{u} \in \mathbb{R}^{m \times n}$, where the rows of the matrix are known as channels with m being the number of channels, and n being the number of items in the sequence.
- a set of parameters known as filters that are represented by the tensor $\mathbf{c} \in \mathbb{R}^{l \times s \times m}$, where l is the number of filters, s is an odd integer that is known as the filter size, and m is the number of channels of each filter.

a convolution operation is the summation of the point-wise multiplication of neighboring items of \mathbf{u} and the t^{th} filter, \mathbf{c}_t , centered at a particular item position, i , in \mathbf{u} and along a particular channel, j . The purpose of a convolution operation is to

create a value for the feature associated with the filter from the values of a channel in a given neighborhood of \mathbf{u} . In mathematical terms, a convolution operation is as follows:

$$\text{CONV}(\mathbf{u}, i, j; \mathbf{c}_t) = \sum_{k=-\lfloor \frac{s}{2} \rfloor}^{+\lfloor \frac{s}{2} \rfloor} c_{t,k,j} u_{i+k,j} \quad (2.10)$$

Note that the index used to access values in the second tensor dimension of the filter are shifted left by $\lfloor \frac{s}{2} \rfloor$ for clarity purposes. This means the center of the filter along the second tensor dimension has an index of zero.

To handle the case where there are not enough items in the beginning and end of the sequence to do the point-wise multiplication of neighboring items, we typically pad the input sequence with zero vectors using the following operation:

$$\text{ZEROPAD}(\mathbf{u}, l) = \begin{bmatrix} \mathbf{0}_1 & \dots & \mathbf{0}_l & \mathbf{u}_1 & \dots & \mathbf{u}_n & \mathbf{0}_1 & \dots & \mathbf{0}_l \end{bmatrix} \quad (2.11)$$

where l is the floor of half the size of the filter. The zero-padded version of \mathbf{u} is as follows:

$$\hat{\mathbf{u}} = \text{ZEROPAD} \left(\mathbf{u}, \left\lfloor \frac{s}{2} \right\rfloor \right)$$

Next, we sum up the convolutions of each filter across all channels to create a single vector as follows:

$$\text{CONVSUM}(\hat{\mathbf{u}}; \mathbf{c}_t) = \left[\sum_{j=1}^m \text{CONV}(\hat{\mathbf{u}}, \lfloor \frac{s}{2} \rfloor + 1, j; \mathbf{c}_t) \quad \dots \quad \sum_{j=1}^m \text{CONV}(\hat{\mathbf{u}}, \lfloor \frac{s}{2} \rfloor + n, j; \mathbf{c}_t) \right] \quad (2.12)$$

We assign this vector to a variable $\mathbf{z}_t \in \mathbb{R}^{n \times 1}$:

$$\mathbf{z}_t = \text{CONVSUM}(\hat{\mathbf{u}}; \mathbf{c}_t)$$

Finally, we can reduce the size of each \mathbf{z}_t via an operation known as pooling. The purpose of pooling is to select features that are important to the classification task. A common way to do pooling is to break each \mathbf{z}_t into small segments of equal size, and then keep the maximum value in each segment. This type of pooling is known as max-pooling [82] and is as follows:

$$\mathbf{v}_t = \text{MAXPOOL}(\mathbf{z}_t, b) = \begin{bmatrix} \max(z_{t,1}, \dots, z_{t,b}) \\ \dots \\ \max(z_{t,n-b}, \dots, z_{t,n}) \end{bmatrix} \quad (2.13)$$

where b is the segment size and $\mathbf{v}_t \in \mathbb{R}^{\frac{n}{b} \times 1}$

We can now build the convolution layer that is illustrated in Figure 2.3. The convolution layer transforms a matrix $\mathbf{u} \in \mathbb{R}^{m \times n}$ to a matrix $\mathbf{v} \in \mathbb{R}^{l \times \frac{n}{b}}$, where \mathbf{v} consists of all the \mathbf{v}_t vectors.

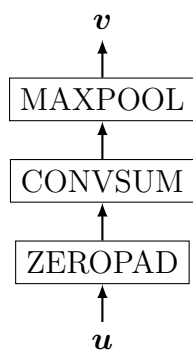


FIGURE 2.3: A typical architecture of a convolutional layer

In practice, different variations of the convolution layer exist. For example, some convolution layers include a non-linear transformation while others may not always do pooling.

We can compactly represent the convolution layer as follows:

$$\mathbf{v} = \text{CONVLAYER}(\mathbf{u}; \mathbf{c}) \quad (2.14)$$

To build our model specific layers, we stack up the convolution layers one after another, where the output of a convolution layer is the input to the next convolution layer. We pass the input sequence of items, \mathbf{x} , as an input to the first convolution layer, and we pass the output of the last convolution layer to a function, $g(\cdot)$, that transforms the output to logits. We illustrate the model specific layers in Figure 2.4.

Note that each convolution layer has its own set of filters that can have different shapes.

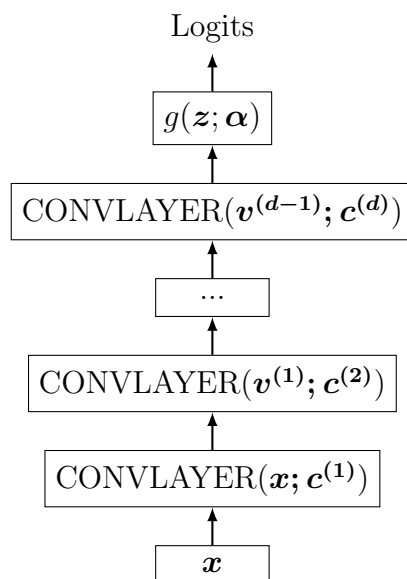


FIGURE 2.4: A typical architecture of a CNN

2.2.4 Bidirectional Encoder Representations From Transformers

A "bidirectional encoder representations from transformers" (BERT) [19] classifier uses an architecture known as the transformer [72]. The transformer is composed of layers that use a mechanism known as attention. BERT is currently the foundation of most state-of-the-art text classification models. Unlike the other classifiers we have seen so far, BERT always comes pre-trained. That is it is first trained to predict tokens on a large number of text segments and then after a small change to its top layer, it is re-trained (fine-tuned) on a classification task.

- **Input Layer:** BERT applies a specific type of tokenization algorithm called WordPiece [79]. WordPiece decomposes words into multiple tokens such that a desired vocabulary size for the tokens is reached. In addition, it allows for the handling

of new words not seen in the corpus provided those words can be composed of existing tokens in the vocabulary.

- **Input Encoding Layer:** The initial vector representations of the tokens are set during the pre-training of the BERT model prior to the classification task. When the token vectors are fed as input to the model specific layers their positions in the sequence are encoded. This is done by adding position specific vectors to the token vectors.
- **Model Specific Layers:** As mentioned above, BERT uses an architecture called a transformer. A transformer is an encoder-decoder system, which means it can be trained to predict its input as an output. This specifically happens during the pre-training of BERT when input tokens are used to predict output tokens from the same text. The details of the transformer are too complex to cover briefly in this section, but the main takeaway is it uses a technique called attention [69] to model dependencies between tokens. Attention can be defined as follows:

$$\text{ATTENTION}(\mathbf{Q}, \mathbf{K}, \mathbf{V}) = \text{softmax}\left(\frac{\mathbf{Q}\mathbf{K}^T}{\sqrt{m}}\right) \mathbf{V} \quad (2.15)$$

where $\{\mathbf{Q}, \mathbf{K}, \mathbf{V}\}$ and the output are matrices of size $m \times n$. Attention can be seen as a way for expressing each vector, $v_i \in \mathbf{V}$, as a weighted sum of the other vectors in \mathbf{V} such that the weights are computed from the dot-product of the vectors in \mathbf{Q} with the vectors in \mathbf{K} . The dot-product is a crude measure of the similarity between two vectors, since it is the unnormalized form of cosine similarity. The

weights are normalized by the softmax operation such that they sum up to one along the column dimension of their matrix. A transformer computes attention serially across multiple layers and in parallel within each layer.

The encoder part of the architecture uses a specific type of attention that is known as self-attention, which is as follows:

$$\text{SELF_ATTENTION}(\mathbf{V}; \mathbf{W}^Q, \mathbf{W}^K, \mathbf{W}^V) = \text{ATTENTION}(\mathbf{V}\mathbf{W}^Q, \mathbf{V}\mathbf{W}^K, \mathbf{V}\mathbf{W}^V) \quad (2.16)$$

where $\mathbf{V} \in \mathbb{R}^{m \times n}$ and $\{\mathbf{W}^Q, \mathbf{W}^K, \mathbf{W}^V\}$ are transformation matrices that are parameters of the model. Self-attention allows the transformer to discover dependencies between the input tokens.

For a given text, a transformer works by predicting a token in the text using a subset of the text's tokens and other tokens of the text that have been predicted. In addition, BERT embeds a special token called '[CLS]' in the beginning of every text to use the token's logits as a representation for the whole text.

During a BERT classification task, we use the '[CLS]' token's logits to predict the probability distribution of the labels. We do this by adding an additional layer to the transformer to transform the '[CLS]' token's logits to the label logits. We illustrate the transformer and this layer in Figure 2.5, where $g(\cdot)$ is the transformation function.

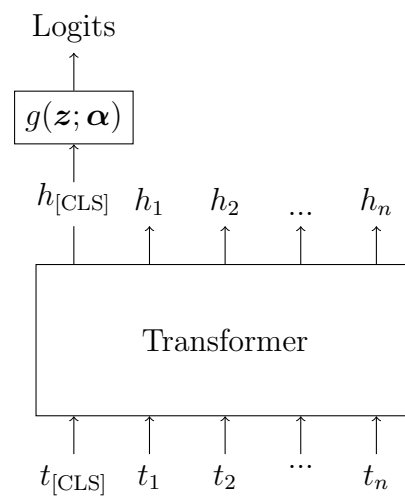


FIGURE 2.5: BERT classification architecture

Chapter 3

Related Work

Extensive research has been done in utilizing data augmentation for text classification tasks. The research can be categorized into data augmentation done in the data space and the feature space. For a comprehensive overview of the research literature, please refer to the recent survey [10].

3.1 Data Space

Data augmentation in the data space can be further subdivided into character, word, phrase, and document-level augmentation.

3.1.1 Character Level

Data augmentation at the character level is primarily used to induce noise in datasets. Belinkov et al. [11] apply randomly switching neighboring characters in their neural translation model. Feng et al. [22] randomly insert, delete, and swap characters to fine-tune their generators. Ebrahimi et al. [20] keep character changes that increase the loss of a model trained without data augmentation in order to select them later to retrain the model. Coulombe [18] applies changing the case of characters, modifying punctuations, and randomly inserting, deleting, and swapping characters to augment training sets with additional examples.

Beyond inducing noise in datasets, character-level data augmentation is limited in its ability to mimic real-world data, because it is prone to generate text that is semantically and/or syntactically incorrect.

3.1.2 Word Level

Noise inducing is also used in word level data augmentation. Xie et al. [80] apply “unigram noising”, which is the probabilistic replacement of words, and “blank noising”, which is the replacement of words with a token designated as blank. Li et al. [42] utilize syntactic noise (the syntactic alteration of sentences such as the switching of adjectives), semantic noise (such as the substitution of words by their synonyms), and word dropouts. Wei et al. [77] use random word insertions, deletions and swaps in their

EDA method. Xie et al. [81] utilize TF-IDF to replace less weighted words with other less weighted words.

Synonym replacement, word swapping, word insertion, and word deletion are common techniques used by word-level data augmentation. Wei et al. [77] randomly apply these techniques in their EDA method using a preset probability distribution. Recent work by Shuhuai Ren et al. [59] learns the optimal parameters of the probability distribution for applying each of the techniques used by EDA. They call their work Text AutoAugment (TAA), and it yields better results than EDA. Kolomiyets et al. [35] substitute temporal expressions with synonyms from WordNet [49]. Li et al. [42], Mosolova et al. [51], and Wang et al. [76] selectively apply synonym substitution to certain words. Zhang et al. [84] and Marivate et al. [47] implement synonym substitution by sampling from a geometric distribution that models the closeness of synonyms. Jungiewicz et al. [30] replace synonyms if they increase the loss of the classifier when trained without data augmentation.

Embedding replacement is the replacement of words by finding the replacements in the embedding space of the vocabulary. Some existing work do a K-Nearest-Neighbor search in the embedding space to find replacements for words [60]. However, such replacements sometimes cause antonyms to be selected, which has the unintended consequence of capturing the wrong semantics. To mitigate this issue, Mrkšić et al. [52], Li et al. [42], and Alzantot et al. [6] devise a counter-fitting method that rewards the selection of synonyms while sanctioning the selection of antonyms.

Language model replacement is the replacement of words using language models. Language models are trained by predicting the next word or missing words given the neighboring words. Kobayashi [34] uses a label-conditioned Long short-term memory (LSTM) [27] based language model to replace words in sentences. Wu et al. [78] apply a similar approach by using a label-conditioned BERT [19] language model (c-BERT). Jiao et al. [29] extend c-BERT to do embedding replacement for multiple-piece words.

Similar to character-level data augmentation, word-level data augmentation is limited in its ability to mimic real-world data as it is prone to generating text that is semantically and syntactically incorrect. However, it is significantly better than character-level data augmentation in this regard, because it avoids spelling errors as is the case with character-level data augmentation. This is due to the fact that word-level data augmentation generates whole words instead of mutating the characters of individual words.

3.1.3 Phrase Level

Phrase-level data augmentation is done by replacing consecutive sequences of words (phrases) with other phrases. Feng et al. [22] use an attention-based model to do phrase replacement. Min et al. [50] swap the object and subject parts of sentences, where the parts could be phrases. Shi et al. [65] replace phrases with phrases that share the same sequence of part-of-speech tags.

Unlike character and word-level data augmentation, phrase-level data augmentation is not prone to syntactic and semantic errors. However, it is constrained by the fact that a phrase bank needs to be built in order to facilitate the replacement of phrases.

3.1.4 Document Level

Document-level data augmentation is the synthesis of entire training items (documents). Round-trip language translation is a simple way to do document-level language translation. In round-trip language translation, a training item is translated to another language and then translated back to the original language. If the translation back has differences with the training item, it can be used for data augmentation [18, 36]. Xie et al. [80] modified the beam search of translation algorithms to randomly sample candidate translations for greater diversity.

Generative methods, which utilize models such as Variational Autoencoder (VAE) [32], Generative Adversarial Network (GAN) [24], and GPT-2 [57], can be used for document-level data augmentation. Qiu et al. [56] utilize both unconditioned and conditioned (the label is input) VAEs to generate items. Guu et al. [25] use a VAE that encodes an item and an offset vector to generate an item whose latent representation is closest to the combination of the latent representation of the input item and the offset vector. Sun et al. [68] use a sequential GAN architecture to generate items. Wang et al. [74] use GPT-2 to generate items. Anaby-Tavor et al. [8] do conditional GPT-2 where they fine-tune a GPT-2 model with the label and text of the training items concatenated

with each other. During inference, the fine-tuned GPT-2 model generates items with their respective labels and texts concatenated with each other. Liu et al. [44] add a reinforcement learning component to GPT-2 to be able to condition the generation of items by a given label.

A recent work [83] took the hierarchical structure of texts into account and augmented texts at the word level and sentence level respectively. Iterative Translation-based Data Augmentation (ITDA) [39] was proposed to provide multiple-language support for text classification by generating augmented sentences through iterations of a translator. Luo et al. [46] presented a data augmentation framework based on sequence generative adversarial networks to improve sentiment analysis accuracy.

3.2 Feature Space

One of the main purposes of data augmentation in the feature space is to induce noise. One method for inducing noise is to add a perturbation to the features of the text during training. Zhu et al. [85], Liu et al. [45], and Shafahi et al. [63] induce noise during training by learning the best perturbation parameters. Another method to induce noise is by adding the perturbation in the final embedding layer of a model. Wang et al. [75] use this method to enhance pre-trained language models. However, learning the perturbation parameters is computationally expensive, and Shen et al. [64] use a different method, where they occasionally zero out feature vectors, or specific dimensions in the feature vectors instead of using a trainable perturbation. Although interpolation is hard to

attain in the data space of textual data, it can be done in the feature space by combining the learning representations of sentences or text fragments [14].

Chapter 4

The Proposed Framework

4.1 Overview

SOFITDA follows a generic pipeline (Data Augmentation Pipeline) for generating synthetic items and selecting a subset of them for augmenting a minority label. This pipeline makes no assumption of the generation and selection methods. SOFITDA further specifies a selection method that jointly maximizes the likelihood of the selected items belonging to the minority label and the diversity of the selected items.

Figure 4.1 shows the steps of the Data Augmentation Pipeline. The following are the descriptions of the steps:

- **Configure Generator:** Configures the algorithm that generates synthetic items.

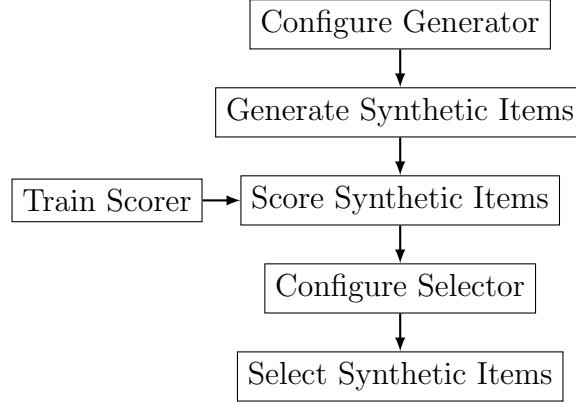


FIGURE 4.1: The steps of the Data Augmentation Pipeline for getting synthetic items for a single minority label

For example, in the case of a pre-trained text-generating model, this is a fine-tuning step where the model is fine-tuned with the items of a minority label.

- **Generate Synthetic Items:** Generates synthetic items using the configured generator. We generate an excessive number of synthetic items such that we have a large enough pool of items from which we select the optimal items we need for augmentation.
- **Train Scorer:** Trains a classifier on the original dataset (without augmentation). For a given item, the classifier infers the probability distribution of the labels.
- **Score Synthetic Items:** Assigns scores to the synthetic items using the classifier. The scores are the probability of the minority label inferred by the classifier. The selection process uses them as a measure for the likelihood of the synthetic items in belonging to the minority label.

- **Configure Selector:** Configures the algorithm that selects synthetic items. For example, if the selector uses embedding vectors to represent documents, the vectors are computed in this step.
- **Select Synthetic Items:** Selects items from the pool of generated synthetic items such that the minority label is balanced. This means the number of items of the minority label is equal to the number of items of the most dominant label in the dataset.

The Data Augmentation Pipeline is executed for each minority label of the dataset such that all labels are augmented with enough synthetic items to have the same number of items as the most dominant label in the dataset. Note, however, the Train Scorer step only needs to be executed once for the whole dataset.

SOFITDA specifies a specific selector for the Data Augmentation Pipeline that jointly maximizes the likelihood and the diversity of the selected items. We initially formulate the maximization as an intractable combinatorial objective that we then approximate by transforming it into a monotone submodular objective. Monotone submodular objectives can be solved by a tractable, greedy algorithm that yields a solution that is within $(1 - 1/e)$ of the optimal solution [17].

4.2 Individual Components

We now define the Data Augmentation pipeline in mathematical and algorithmic detail.

Let D be a set of labeled documents, where each document is represented by a vector $\mathbf{d} \in \mathbb{R}^{M \times 1}$, and has only one label. Furthermore, let L represent the number of labels, and n_i^D represent the number of documents that have the label i . The goal of SOFITDA is to generate synthetic documents, assign a score to each of the synthetic documents, and finally select enough of the synthetic documents for each label i such that the number of documents for each label is exactly balanced.

Let S represent the set of selected synthetic documents. The documents of D and S are combined to train a text classification model. The model predicts the probability distribution of the labels for a given document. We call this model the Document Classifier.

4.2.1 Text Generator

Synthetic documents for label i are generated by $\text{GENERATOR}(n_i^S; \boldsymbol{\theta}_i)$, where n_i^S is the number of synthetic documents to generate for label i , and $\boldsymbol{\theta}_i$ are the parameters of the generator that generates the synthetic documents. The values of $\boldsymbol{\theta}_i$ are determined by a configuration step $\text{CONFIGURE_GENERATOR}(D, i)$.

4.2.2 Document Scorer

We assign a score to each synthetic document we generate for label i . The score is determined by a document classifier $\text{SCORER}(\mathbf{s}, i; \boldsymbol{\beta})$, where \mathbf{s} is a synthetic document, i is the label, and $\boldsymbol{\beta}$ are the parameters of the document classifier. The document classifier is trained on set D by $\text{TRAIN_SCORER}(D)$ to determine the values of $\boldsymbol{\beta}$. The score of a synthetic document is the probability of label i inferred by the document classifier.

4.2.3 Document Selector

To balance each label i in D , the number of synthetic documents we need is given by the following equation:

$$n_i^S = \left(\max_{m \in \{1, \dots, L\}} n_m^D \right) - n_i^D \quad (4.1)$$

where n_i^D is the number of samples a label i has in D and n_i^S is the number of synthetic samples we need to balance the label i .

In order to sample high-quality synthetic documents, we generate excess synthetic documents for label i by an integral factor of ω , where $\omega > 1$. We then use a selector $\text{SELECTOR}(T_i, R_i, n_i^S; \boldsymbol{\eta}_i)$ to sample high-quality synthetic documents, where T_i is the set of all synthetic documents generated for label i , R_i is the set of the corresponding scores of the synthetic documents, and $\boldsymbol{\eta}_i$ are the parameters of the selector. The number of synthetic documents in T_i is equal to $\omega \times n_i^S$ and the selector selects n_i^S

synthetic documents from T_i . The values of $\boldsymbol{\eta}_j$ are determined by a configuration step $\text{CONFIGURE_SELECTOR}(D, i)$.

4.2.4 Data Augmentation Pipeline

The Data Augmentation Pipeline is an algorithm that utilizes the generator, scorer, and selector to create the set of synthetic documents S that we use to augment the set of real documents D such that all the labels are exactly balanced. The training using the augmented dataset $(D \cup S)$ is done by the same type of classifier used by the document scorer. Algorithm 2 outlines the steps of the Data Augmentation Pipeline. The algorithm does not specify the implementation details for the document generator,

Algorithm 2 Data Augmentation Pipeline

```

1:  $S \leftarrow \emptyset$ 
2:  $R \leftarrow \emptyset$ 
3:  $\beta \leftarrow \text{TRAIN\_SCORER}(D)$ 
4: for each  $i \in \{1, \dots, L\}$  do
5:    $n_i^S = (\max_{m \in \{1, \dots, L\}} n_m^D) - n_i^D$ 
6:    $n_i^G \leftarrow \omega \times n_i^S$ 
7:    $\theta_i \leftarrow \text{CONFIGURE\_GENERATOR}(D, i)$ 
8:    $T_i \leftarrow \text{GENERATOR}(n_i^G; \theta_i)$ 
9:   for each  $j \in \{1, \dots, n_i^G\}$  do
10:     $s \leftarrow t_{i,j}$ 
11:     $r_{i,j} \leftarrow \text{SCORER}(s, i; \beta)$ 
12:   end for
13:    $\eta_i \leftarrow \text{CONFIGURE\_SELECTOR}(D, i)$ 
14:    $S_i \leftarrow \text{SELECTOR}(T_i, R_i, n_i^S; \eta_i)$ 
15: end for

```

scorer, or selector. It begins by first training the scorer for the set of documents, D , and then proceeds to do the following for each label i :

- Determines the number of documents, n_i^G , that need to be synthesized for the label. This number is ω times the number of documents, n_i^S , required to balance the label such that the number of documents of the label is the same as the number of documents of the most dominant label.
- Configures the parameters of the generator, θ_i , for the label.
- Using the generator, synthesizes a set of documents, T_i , for the label. The number of documents synthesized is n_i^G .
- Using the scorer, computes a score, $r_{i,j}$, for each synthesized document, $t_{i,j}$. The score is a probability value.
- Configures the parameters of the selector, η_i , for the label.
- Using the selector selects n_i^S items from T_i and puts them into the set of selected documents, S_i .

4.2.5 Diversity Selector

The diversity selector is a selector that attempts to select a synthetic document for label i based on the ability of the synthetic document to maximize the total score and the diversity of S_i . We compute the total score of S_i as follows:

$$U(S_i) = \sum_{j=1}^{n_i^S} r_{i,j} \quad (4.2)$$

To compute the diversity score of S_i , we first compute the dissimilarity of a synthetic document j in S_i , $v_{i,j}$, with the other synthetic documents in S_i as follows:

$$v_{i,j} = \sum_{k=1}^{n_i^S} 1 - \text{SIMILARITY}(S_{i,j}, S_{i,k}) \quad (4.3)$$

where SIMILARITY computes the similarity of two documents $S_{i,j}$ and $S_{i,k}$ as a real number $\in [0, 1]$ such that similarity increases from a minimum value of 0 to a maximum value of 1. We then compute the diversity $V(S_i)$ of S_i as a measure of total dissimilarity as follows:

$$V(S_i) = \sum_{j=1}^{n_i^S} v_{i,j} \quad (4.4)$$

Ideally, the diversity selector should linearly combine the two objectives of maximizing $U(S_i)$ and $V(S_i)$ as follows:

$$Z(S_i) = U(S_i) + \lambda V(S_i) \quad (4.5)$$

where λ is a weight such that $\lambda \geq 0$ and $\lambda \in \mathbb{R}$. To find the optimal selection S_i^* , we maximize the following objective:

$$\begin{aligned} S_i^* = \operatorname{argmax}_{S_i \subset T_i} \quad & Z(S_i) \\ \text{s.t.} \quad & |S_i| = n_i^S \end{aligned} \quad (4.6)$$

However, this maximization is an NP-hard combinatorial problem, because the number of possible subsets of S_i is $\binom{\omega \times n_i^S}{n_i^S}$. Therefore, the maximization cannot be tractably

solved for sufficiently large values of ω , and n_i^S .

4.2.6 Monotone Submodular Optimization

To overcome the intractability of Equation (4.6), we modify it such that it becomes a monotone submodular optimization objective. This allows us to use a tractable greedy algorithm whose solution is shown by [17] to be within $(1 - 1/e)$ of the optimal solution. A monotone submodular optimization objective is formulated as follows:

$$\begin{aligned} \mathcal{S} = \operatorname{argmax}_{\mathcal{S} \subseteq \mathcal{E}} \quad & \mathcal{F}(\mathcal{S}) \\ \text{s.t.} \quad & |\mathcal{S}| \leq c \end{aligned} \tag{4.7}$$

where \mathcal{E} is a set of elements, \mathcal{F} is a monotone submodular function, and c is an integer such that $0 < c < |\mathcal{E}|$. A monotone submodular function is a function $\mathcal{F} : 2^{\mathcal{E}} \rightarrow \mathbb{R}_+$ that abides by the following [17]:

- (Submodularity) For all $\mathcal{S} \subseteq \mathcal{T} \subseteq \mathcal{E}$ and $\{l\} \in \mathcal{E} \setminus \mathcal{T}$:

$$\mathcal{F}(\mathcal{T} \cup \{l\}) - \mathcal{F}(\mathcal{T}) \leq \mathcal{F}(\mathcal{S} \cup \{l\}) - \mathcal{F}(\mathcal{S}) \tag{4.8}$$

Equivalently, the submodularity can also be defined as follows: For all $\mathcal{S} \subseteq \mathcal{E}$ and $\mathcal{T} \subseteq \mathcal{E}$:

$$\mathcal{F}(\mathcal{S}) + \mathcal{F}(\mathcal{T}) \geq \mathcal{F}(\mathcal{S} \cap \mathcal{T}) + \mathcal{F}(\mathcal{S} \cup \mathcal{T}) \tag{4.9}$$

Algorithm 3 Greedy Algorithm for Monotone Submodular Optimization

```

1:  $\mathcal{S} \leftarrow \emptyset$ 
2: while  $|\mathcal{S}| < c$  do
3:    $e \leftarrow \operatorname{argmax}_{e \in \mathcal{E}} \mathcal{F}(\mathcal{S} \cup \{e\}) - \mathcal{F}(\mathcal{S})$ 
4:    $\mathcal{S} \leftarrow \mathcal{S} \cup \{e\}$ 
5: end while
  
```

- (Monotonicity) For all $\mathcal{S} \subseteq \mathcal{T} \subseteq \mathcal{E}$:

$$\mathcal{F}(\mathcal{S}) \leq \mathcal{F}(\mathcal{T}) \quad (4.10)$$

An approximate solution for the monotone submodular optimization objective can be solved by Algorithm (3), which is proposed by [17]. What is interesting about this solution is the fact that it occurs when $|\mathcal{S}| = c$. This means if our combined objective $Z(S_i)$ in Equation (4.6) is a monotone submodular function, we can update the constraint in the equation from $|S_i| = n_i^S$ to $|S_i| \leq n_i^S$ in order to convert it to a monotone submodular optimization objective. Doing so allows us to use Algorithm (3) to obtain an approximation for the optimal solution.

However, Equation (4.6) is not a monotone submodular function. While the term $U(S_i)$ in the equation for $Z(S_i)$ is a monotone submodular function, the term $V(S_i)$ is not. $U(S_i)$ is a monotone submodular function because we can add scores in descending order to each successive and larger subset of elements. This allows $U(S_i)$ to meet all the conditions of a submodular function. $V(S_i)$ is not a monotone submodular function, because Equation (4.9) is not always met. This is due to each similarity score, $v_{i,j}$, being dependent on other values of the set. Therefore, the same $v_{i,j} \in \mathcal{S}$ and $v_{i,j} \in \mathcal{T}$

may not contribute the same value to $\mathcal{F}(\mathcal{S})$, $\mathcal{F}(\mathcal{T})$, $\mathcal{F}(\mathcal{S} \cap \mathcal{T})$ and $\mathcal{F}(\mathcal{S} \cup \mathcal{T})$ due to the composition of the sets being different in each function call. As a consequence, the inequality of Equation (4.9) is not always maintained.

4.2.7 SOFITDA Selector

To address Equation (4.6) not being a monotone submodular function, we approximate diversity by clustering the documents in T_i based upon their features instead of using $V(S_i)$. Our goal is to select documents as evenly as possible across the clusters in order to increase diversity. When we select the documents from each cluster, we select the documents in descending order of their scores. The value we return for the document selector is as follows:

$$\hat{Z}(S_i) = \sum_{k=1}^{N_C} \sum_{d \in C_k \cap S_i} r_{i,d} \quad (4.11)$$

where N_C is the number of clusters, and C_k is the k^{th} cluster. We can change the quality of the diversity by adjusting the number of clusters. If the number of clusters is one, we do not induce diversity, because we select n_i^S documents in T_i with the highest scores.

We attempt to evenly select documents across clusters by randomly selecting the cluster from which we select the next document. However, this approach does not meet all the conditions of a monotone submodular function. Specifically, it violates Equation (4.8), because the document we select next may have a higher score than a document we have already selected. This can happen when both documents come from different clusters.

As a result of this violation, we cannot guarantee the approach yields a close-to-optimal solution.

Inspired by the application of submodular functions for document summarization in Lin et al. [43], we fix this violation by updating Equation (4.11) to use a non-decreasing concave function $H(\cdot)$ as follows:

$$\tilde{Z}(S_i) = \sum_{k=1}^{N_C} H \left(\sum_{d \in C_k \cap S_i} r_{i,d} \right) \quad (4.12)$$

The usage of a non-decreasing concave function gives us two advantages. The first advantage is we would not violate Equation (4.8), because in a non-decreasing concave function, the following condition holds:

$$H(y + d) - H(y) \leq H(x + d) - H(x) \quad (4.13)$$

provided that:

$$\begin{aligned} x, y, d &\in \mathbb{R} \\ x, y, d &\geq 0 \\ x &\leq y \end{aligned} \quad (4.14)$$

In the case of Equation (4.12), x and y are the sum of probability values and they are therefore real numbers greater than or equal to zero. d is a probability value and is therefore a real number greater than or equal to zero. An example of a non-decreasing concave is the square root function. We can see from the expression $\sqrt{25+1} - \sqrt{25} <$

$\sqrt{16+1} - \sqrt{16}$ the inequality of Equation (4.13) is met. The full proof for the monotone submodularity of Equation (4.12) is listed in Appendix A.

The second advantage of using a non-decreasing concave function is we no longer need to randomly select a cluster from which we select the next document. This is because the document with the highest score is not always selected due to $H(\cdot)$ being a non-decreasing concave function. To see this being the case, let us assume an example, where:

- $H(\cdot)$ is the square root function.
- We have two clusters, C_1 and C_2 .
- C_1 contains the documents with the highest scores.
- All documents selected so far are from C_1 and the sum of their scores is 25.
- The score of the highest document not selected in C_1 is 0.9.
- The score of the highest document in C_2 is 0.5.

The net change in the value of $\tilde{Z}(S_i)$ for selecting the next document from C_1 is $\sqrt{25+0.9} - \sqrt{25} = 0.089$, whereas the net change in the value of $\tilde{Z}(S_i)$ for selecting the next document from C_2 is $\sqrt{25} + \sqrt{0+0.5} - \sqrt{25} = 0.707$. Therefore, the next document is selected from C_2 , and this shows us that the next document can come from a cluster that does not contain the highest-scoring candidate document due to the non-decreasing concavity characteristics of $H(\cdot)$. As a result, we do not need to randomly

select a cluster from which we select the next document in order to boost diversity. Instead, we can select a function for $H(\cdot)$ that has the appropriate level of concavity for inducing the desired level of diversity.

The monotone submodular objective for using $\tilde{Z}(S_i)$ is as follows:

$$\begin{aligned} S_i^* = \operatorname{argmax}_{S_i \subset \mathbf{T}_i} \quad & \tilde{Z}(\mathbf{S}_i) \\ \text{s.t.} \quad & |\mathbf{S}_i| \leq n_i^S \end{aligned} \tag{4.15}$$

Unlike the objective in Equation (4.6), this objective is a monotone submodular function. Therefore, an approximate solution for the objective can be solved by Algorithm (3), and the approximate solution is guaranteed to be within $(1 - 1/e)$ of the optimal solution [17].

It is possible that Algorithm (3) can have multiple candidate documents for the next document to be selected. This occurs when selecting any one of the candidate documents results in the same value for $\tilde{Z}(S_i)$. When this scenario occurs, the algorithm randomly selects one of the candidate documents.

4.2.8 Computational Complexity of SOFITDA Selector

We use Algorithm (3) to optimize the SOFITDA selector. The clusters are sorted in descending order by the scores of their documents. As a result, selecting the document that has the maximum score from the clusters during each iteration of the loop has a time efficiency of $O(N_C)$. Since N_C , the number of clusters, is a constant, the time

efficiency reduces to $O(1)$. To select the documents of S_i , the loop iterates n_i^S times, and this results in the efficiency of the algorithm being $n_i^S \times O(1) = O(n_i^S)$. Therefore, the SOFITDA selector scales up linearly with the number of synthesized items.

Chapter 5

Experiments

In this section, we present the setup, results, and analysis of the experiments.

5.1 Datasets

We use a total of 6 datasets to evaluate SOFITDA. All 6 datasets are composed of labeled texts. The datasets are as follows:

- **Amazon Camera Reviews** [1]: Camera reviews are labeled using a 5-star system, where 1 star represents the lowest rating, and 5 stars represent the highest rating.

- **Yelp Reviews** [4]: Customer reviews of businesses are labeled using a 5-star system, where 1 star represents the lowest rating, and 5 stars represent the highest rating.
- **Quora Insincere Questions** [2]: Quora questions are labeled as sincere or insincere. An insincere question is a question that is not well intended, malicious, or not based on facts.
- **Trip Advisor Hotel Reviews** [3]: Hotel reviews are labeled using a 5-star system, where 1 star represents the lowest rating, and 5 stars represent the highest rating. For this dataset, we only use hotel reviews that have ASCII characters in order to avoid including non-English reviews in the dataset.
- **Stanford Sentiment Treebank (SST-5)** [66]: Movie reviews are labeled using the following 5 sentiment labels: very negative, negative, neutral, positive, and very positive.
- **Text Retrieval Conference (TREC-6)** [41]: Questions are labeled into six broad categories as follows: ABBR (Abbreviation), DESC (Description and abstract), ENTY (ENTITIES), HUM (Human beings), LOC (Locations), and NYM (Numeric values).

Table 5.1 shows the size and percentage of the labels of each dataset.

TABLE 5.1: The distribution of the labels for each dataset.

Dataset	Label	Size	Percentage
Amazon Camera Reviews	1	2,527	8.31
	2	1,224	4.03
	3	1,897	6.24
	4	4,402	14.48
	5	20,359	66.95
Yelp Open Dataset	1	2,674	8.65
	2	1,813	5.86
	3	2,750	8.89
	4	7,444	24.07
	5	16,246	52.53
Quora Insincere Questions	Sincere	37,500	93.81
	Insincere	2,475	6.19
Trip Advisor Hotel Reviews	1	1,322	4.22
	2	1,142	3.64
	3	3,211	10.25
	4	10,975	35.02
	5	14,690	46.87
Stanford Sentiment Treebank	Very negative	1,509	12.74
	Negative	3,140	26.50
	Neutral	2,241	18.91
	Positive	3,109	26.24
	Very positive	1,850	15.61
Text Retrieval Conference Dataset	DESC	3,442	21.58
	ENTY	3,721	23.33
	ABBR	250	1.57
	HUM	3,489	21.87
	NUM	2,593	16.26
	LOC	2,457	15.40

5.2 Pre-processing

For the Amazon, Yelp, Quora, and Trip Advisor datasets, we partition 50% of the items for training, 25% for validation, and 25% for testing. We don't partition the Stanford Sentiment Treebank (SST-5) and the Text Retrieval Conference (TREC-6) datasets, because they come partitioned. Table 5.2 shows the sizes of the partitions

TABLE 5.2: The partition and vocabulary sizes of each dataset.

Dataset	Training	Validation	Testing	Vocabulary Size
Amazon Camera Reviews	15,159	7,655	7,595	29,628
Yelp Open Dataset	15,442	7,765	7,720	61,112
Quora Insincere Questions	19,987	9,992	9,994	56,092
Trip Advisor Hotel Reviews	15,690	7,831	7,819	67,102
Stanford Sentiment Treebank	8,531	1,101	2,209	19,514
Text Retrieval Conference Dataset	14,452	1,000	500	9,778

and the vocabulary of each dataset. The vocabulary size shown here is based on the tokenization of text into words by using white-space characters as separators.

5.3 Setup

In this section, we describe the setup of the Data Augmentation Pipeline for our experiments by specifying the characteristics of the functions in Algorithm (2). As part of our evaluation, we use multiple datasets, GENERATOR functions, SCORER functions, and SELECTOR functions. We execute the Data Augmentation Pipeline for each distinct tuple of dataset, GENERATOR function, SCORER function, and SELECTOR function.

5.3.1 CONFIGURE_GENERATOR Functions

We evaluate SOFITDA on two widely used data augmentation methods. For each method, we have a GENERATOR function for generating synthetic documents. The first method is EDA [77], and we use this method to analyze the performance of

SOFITDA on items synthesized by a rules-based data augmentation method. In the case of EDA, we do not have a `CONFIGURE_GENERATOR` function, because we use the default parameters of EDA as is. The second method utilizes GPT-2 [57], which is a high-quality generative language model that is trained on a large corpus of text. We use this method to analyze the performance of SOFITDA on items synthesized by a machine-learning model. For each minority label of a given dataset, we execute the `CONFIGURE_GENERATOR` function of GPT-2 to fine-tune an instance of the GPT-2 model on the training items of the minority label. We enclose the text of each training item with the tokens ‘[BEGIN]’ and ‘[END]’. The tokens allow us to identify the beginning and end of the text of an item that is generated by GPT-2.

5.3.2 GENERATOR Functions

The GENERATOR functions generate 10 times the number of items required to make the number of items of a given label be equal to the number of items of the majority label. This is so that we have a large enough pool of synthetic items to make a good selection. In the case of GPT-2, a valid generated item is delimited by the ‘[BEGIN]’ and ‘[END]’ tokens. We strip these tokens to extract the text of the generated item.

5.3.3 TRAIN_SCORER Functions

We train four types of SCORER functions, where each type uses a unique text classifier. The learning rate and batch size of the text classifiers were tuned on the validation sets.

The text classifiers and their configurations are as follows:

- **BOW Classifier:** A Logistic Regression model that uses the Bag-Of-Words (BOW) representation of the input text as features such that the input text is represented by a binary vector where each slot in the vector corresponds to a unique word and is set to 1 if the word is present in the input text or 0, otherwise. For the BOW classifier, we use a learning rate of 0.001, an ADAM optimizer [31], and a batch size of 32.
- **RNN Classifier:** A Recurrent Neural Network (RNN) model that consists of a word embedding layer, a bi-directional Long Short-Term Memory (LSTM) [27] layer, a fully connected hidden layer that consists of 50 RELU (Rectified Linear Unit) units, a 50% dropout layer, and a softmax layer that linearly transforms the output of the previous layer to compute the probability scores of the labels. We initialize the word embeddings with GloVe embeddings [70], which are trained on a large text corpus, and have a size of 100 dimensions. For the RNN classifier, we use a learning rate of 0.0001, an ADAM optimizer, and a batch size of 32.
- **CNN Classifier:** A Convolutional Neural Network (CNN) model that consists of a word embedding layer, a one-dimensional CNN layer that has 200 filters with a kernel size of 5, a RELU layer, a global max pooling layer, a fully connected hidden layer that consists of 50 RELU units, and a softmax layer that linearly transforms the output of the previous layer to compute the probability scores of the labels. We initialize the word embeddings with GloVe embeddings that have

a size of 100 dimensions. For the CNN classifier, we use a learning rate of 0.0001, an ADAM optimizer, and a batch size of 32.

- **BERT Classifier:** A classifier that uses BERT (Bidirectional Encoder Representations from Transformers) [19] embeddings. Similar to GloVe embeddings, BERT embeddings are trained on a large text corpus. We use uncased small BERT embeddings that are trained on a 2-layer neural network with 512 hidden weights and 8 attention heads. Our BERT classifier consists of the BERT transformer stack, a 10% dropout layer, and a softmax layer that linearly transforms the final output of the BERT transformer stack to compute the probability scores of the labels. For the BERT classifier, we use a learning rate of 0.0003, an ADAM optimizer, and a batch size of 32.

With the exception of the BERT text classifier, which uses its own tokenization method, we tokenize the text of the training items for the text classifiers as follows:

- We remove all occurrences of the following characters in the text: !"#\$%&()*+,-./:;
=>?@[\\]^_`{|}~ -
- We split the tokens by white space (space, tabs, and new lines).

The number of epochs required to train each SCORER function is the number of epochs that gives the least amount of error on the validation set of the dataset. We use the Keras Tensorflow 2 framework to build and run the text classifiers.

5.3.4 SCORER Function

The scores returned by the SCORER functions are the likelihoods of the labels of the synthesized items passed as inputs. The likelihoods are expressed as a vector of probability values where each slot in the vector corresponds to a label.

5.3.5 CONFIGURE_SELECTOR Function

We have five types of selector functions. The first type is the SOFITDA selector. The non-decreasing concave function we use for the selector is as follows:

$$H(x) = x^\alpha \tag{5.1}$$

where $x, \alpha \in \mathbb{R}$ and $\alpha \in [0, 1]$. Our choice of α values makes $H(\cdot)$ a non-decreasing concave function that includes the square root function ($\alpha = 0.5$). As shown in Figure 5.1, the α values allow us to control the concavity of $H(\cdot)$ so that we can induce the optimal amount of diversity we need. When $\alpha = 1$, we do not use diversity as a factor, because the documents with the highest scores are selected. When $\alpha = 0$, we use diversity as the only factor, because $H(\cdot)$ returns a value of 1 for all documents, and therefore the selection of the documents is randomized. As the value of α decreases from 1 to 0, the amount of diversity increases. In addition to α , we can control the quality of the diversity by adjusting the number of clusters, N_C . For our experiment, we configure

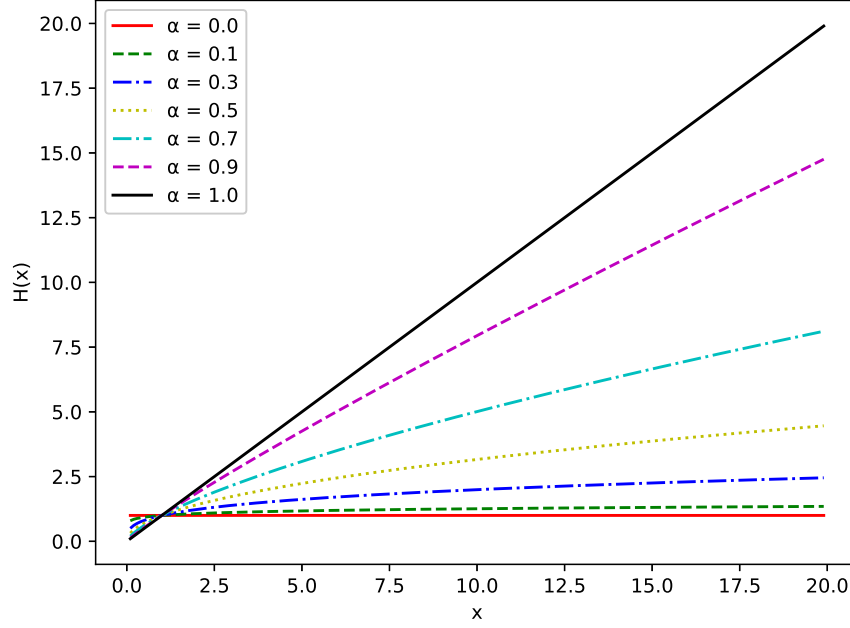


FIGURE 5.1: Plots of $H(x) = x^\alpha$ for different values of α . Notice how the concavity of $H(x)$ decreases as the value of α goes from 0 to 1.

a SOFITDA selector for each distinct pair of α and N_C from the following sets:

$$\begin{aligned} \alpha &\in \{0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9\} \\ N_C &\in \{2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12\} \end{aligned} \tag{5.2}$$

In the `CONFIGURE_SELECTOR` function of the SOFITDA selector, we create the clusters for a label of the dataset. We use the Mini-Batch K-Means algorithm [62] to cluster the items synthesized by the `GENERATOR` function. We use the items' TF-IDF [67] vectors as clustering features. Section 5.5.4 investigates the effect of α and N_C on SOFITDA.

The remaining four types of selector functions constitute the baseline selectors and are as follows:

- **NO_AUG**: Does not select synthesized items for any of the labels. Although this selector technically violates Equation (4.1), we need it as a baseline to represent the case where data augmentation is not applied.
- **RANDOM**: Randomly selects the necessary number of synthesized documents required to balance the minority label, and we use it to represent the case where no selection algorithm is used. The RANDOM baseline is equivalent to a specific instance of the SOFITDA selector with $\alpha = 0$.
- **TOP**: Ranks the synthesized documents in descending order of their scores and selects the necessary number of top-ranked documents required to balance the minority label. The TOP baseline is equivalent to a specific instance of the SOFITDA selector with $\alpha = 1$ or $N_C = 1$. This baseline is used by the LAMBADA algorithm [8] to select synthesized documents for augmenting labels.
- **BOTTOM**: Ranks the synthesized documents in ascending order of their scores, and selects the necessary number of top-ranked documents required to balance the minority label. The BOTTOM baseline is similar to the selection method used by the HotFlip method [20], where synthesized documents that maximize the classifier’s loss are selected.

The above baselines do not have a `CONFIGURE_SELECTOR` function, because they do not have configurable parameters.

5.3.6 SELECTOR Function

We evaluate a total of 99 SOFITDA selectors, one for each distinct pair of α (9 values) and N_C (11 values), and 4 baseline selectors.

5.3.7 TAA Baseline

In addition to the 4 baseline selectors, we use the Text Auto Augmentation (TAA) method [59] as a baseline. Unlike the baseline selectors, this method jointly uses a generator and a selector in one model such that we cannot incorporate it into our Data Augmentation Pipeline. The generator portion of the method is a word-level rules-based synthesizer and we configure it to use the same rules as our EDA generator. The selector portion uses a probability distribution to randomly sample rules for the generator. The parameters of the probability distribution are the ones that generate synthetic items for an augmented training set that yields the best performance on the validation set when used to train a BERT classifier. For our experiments, we configure the BERT classifier the same way we configure the BERT classifiers of the SCORER functions.

5.4 Evaluation Methodology

After executing the Data Augmentation pipeline for each distinct tuple of dataset, GENERATOR function, SCORER function, and SELECTOR function, we do our evaluation on each tuple as follows:

- We augment a copy of the dataset’s training set with the items of the dataset’s labels that are selected by the SELECTOR function.
- We train a classifier for the augmented training set using the same classifier type and setup used by the SCORER function. The number of epochs required to train the classifier is the number of epochs that gives the least amount of error on the validation set of the dataset.
- We determine the performance of the classifier on the testing set of the dataset.

The metric we use to evaluate the performance of the classifier is the macro F-Score [53].

The macro F-Score is computed on a per-label basis, and we aggregate the F-Score of the labels of a dataset by taking their mean. The F-Score of a label is computed as follows:

$$\text{F-Score} = 2 \times \frac{\text{Precision} \times \text{Recall}}{\text{Precision} + \text{Recall}} \quad (5.3)$$

where:

$$\begin{aligned} \text{Precision} &= \frac{\text{True Positives}}{\text{True Positives} + \text{False Positives}} \\ \text{Recall} &= \frac{\text{True Positives}}{\text{True Positives} + \text{False Negatives}} \end{aligned} \quad (5.4)$$

The macro F-Score is a well-suited metric for analyzing the performance of classifiers on imbalanced testing sets because all labels are weighted equally regardless of their distribution [53]. This is not the case with a metric such as accuracy, because a classifier that always predicts the majority label on an imbalanced testing set can have a high

accuracy value. In addition to computing the macro F-Score, we also compute the macro precision and the macro recall to get further insights into the behavior of the macro F-Score. The macro precision and macro recall are the mean precision and mean recall of the precision and recall of the labels, respectively. We are not able to use other widely used metrics such as "Area under the curve" (AUC) and "Mean correlation coefficient" (MCC) due to their misleading performance on the multi-class classification of imbalanced datasets [12], [86]. For performance comparisons, we group the tuples by dataset, GENERATOR function, and classifier. We choose the SOFITDA selector that yields the highest macro F-Score value in each group and compare the performance of this selector with the performance of the baseline selectors of the group. In addition, we compare the performance of the SOFITDA selector we choose for the EDA generator and BERT classifier with the performance of its equivalent TAA baseline.

5.5 Results

5.5.1 Optimal α and N_C

Table 5.3 shows the α and N_C values of the SOFITDA selector that yield the optimal macro F-Score values on a per dataset, generator, and classifier basis.

To see the impact of the classifier on the values of α and N_C , we compute the mean value of α and N_C in Table 5.3 on a per-classifier basis, and display the results in Table 5.4. From the results, we see that the BERT, RNN, and BOW classifiers have similar α

TABLE 5.3: The α and N_C values of the SOFITDA selector that yields the highest macro F-Score value on a per dataset, generator, and classifier basis.

Dataset	Generator	Parameter	BERT	CNN	RNN	BOW
Amazon	EDA	α	0.1	0.4	0.7	0.3
		N_C	6	5	11	11
	GPT-2	α	0.7	0.2	0.7	0.7
		N_C	12	6	11	8
Yelp	EDA	α	0.6	0.5	0.1	0.7
		N_C	3	3	10	10
	GPT-2	α	0.5	0.1	0.7	0.5
		N_C	6	8	6	8
Quora	EDA	α	0.5	0.3	0.5	0.1
		N_C	5	12	8	12
	GPT-2	α	0.6	0.2	0.4	0.7
		N_C	2	2	6	4
Trip Advisor	EDA	α	0.8	0.4	0.6	0.1
		N_C	5	3	4	6
	GPT-2	α	0.7	0.2	0.4	0.5
		N_C	5	3	5	2
SST-5	EDA	α	0.7	0.1	0.5	0.6
		N_C	11	3	2	12
	GPT-2	α	0.4	0.3	0.6	0.8
		N_C	5	11	5	8
TREC-6	EDA	α	0.1	0.1	0.6	0.5
		N_C	10	5	4	8
	GPT-2	α	0.6	0.1	0.4	0.4
		N_C	6	8	9	9

values that are approximately the same as the mean α value, whereas CNN has a low α value. This means CNN requires more diversity than the other classifiers when we consider its α value in isolation. With regards to the N_C values, we see more variance with the BOW classifier being a notable outlier.

To see the impact of the dataset on the values of α and N_C , we compute the mean value of α and N_C in Table 5.3 on a per-dataset basis, and display the results in Table

TABLE 5.4: The mean α and N_C values of the SOFITDA selector that yields the highest macro F-Score value on a per-classifier basis.

Parameter	BERT	CNN	RNN	BOW
α	0.53	0.24	0.52	0.49
N_C	6.33	5.75	6.75	8.17

TABLE 5.5: The mean α and N_C values of the SOFITDA selector that yields the highest macro F-Score value on a per-dataset basis.

Parameter	Amazon	Yelp	Quora	Trip Advisor	SST-5	TREC-6
α	0.47	0.46	0.41	0.46	0.50	0.35
N_C	8.75	6.75	6.38	4.12	7.12	7.38

5.5. From the results, we can see that the Amazon, Yelp, Quora, Trip Advisor, and SST-5 datasets have similar α values, whereas the TREC-6 dataset is an outlier with a relatively low α value. This can be explained by the fact that all classifiers perform remarkably well on the TREC-6 dataset as shown in Table 5.7. As a result, diversity can be weighted more on the TREC-6 dataset compared to the other datasets. With regards to N_C , we see more variance, but no apparent pattern that can be explained.

To see the impact of the generator on the values of α and N_C , we compute the mean value of α and N_C in Table 5.3 on a per-generator basis, and display the results in Table 5.6. Both the α and N_C values indicate the EDA generator induces more diversity than the GPT-2 generator. This can be explained by the fact that EDA mutates existing items to synthesize new ones whereas GPT-2 synthesizes items from scratch. As a result, there is inherently more diversity in items generated by GPT-2 than in items generated by EDA, and therefore, there is less reliance in the α and N_C values to boost diversity.

TABLE 5.6: The mean α and N_C values for the SOFITDA selector that yields the highest macro F-Score value on a per-generator basis.

Parameter	EDA	GPT-2
α	0.41	0.48
N_C	7.04	6.46

5.5.2 Baseline Comparison

In this section, we compare the performance of the optimal SOFITDA selector with the baselines. The optimal SOFITDA selector yields the best macro F-Score value on a per dataset, generator, and classifier basis. Going forward, we refer to the optimal SOFITDA selector as the SOFITDA selector.

Table 5.7 shows the macro F-Score of SOFITDA and the baseline selectors on a per dataset, generator, and classifier basis. Bold font indicates the selector that yields the highest macro F-Score for each tuple of dataset, generator, and classifier. † indicates a statistically significant difference between the macro F-Score of SOFITDA and NO_AUG on a paired t-test, where the P-value is less than 0.05. We compute the statistical significance by splitting the test dataset into 10 partitions and then doing a P-value test between the macro F-Scores of SOFITDA and NO_AUG for each partition.

Table 5.8 shows the macro F-Score of the TAA baseline on a per-dataset basis. The TAA baseline has a generator that uses the same rules as the EDA generator and uses a BERT classifier. Therefore, we compare the macro F-Score value of each TAA baseline to the macro F-Score value of its equivalent SOFITDA that uses a BERT classifier

and an EDA generator. In all cases of our comparison, the macro F-Score of the TAA baseline underperforms the macro F-Score of its equivalent SOFITDA.

To further analyze the results, we compare in Table 5.9 the baseline selectors' mean macro F-Score, precision, and recall on a per-classifier basis with the corresponding mean macro F-Score, precision, and recall of SOFITDA, respectively. We do the comparison as the relative improvement of the mean macro F-Score, precision, and recall of SOFITDA over the mean macro F-Score, precision, and recall of the baseline, respectively. We calculate the relative improvement as a percentage value as follows: $r = \frac{\bar{s} - \bar{b}}{\bar{b}} \times 100$, where \bar{s} and \bar{b} are the mean macro F-Score, precision, or recall on a per-classifier basis for SOFITDA and the baseline, respectively. From Table 5.9, we can see that SOFITDA has significantly higher macro F-Score relative improvement values for every baseline and classifier combination.

When we aggregate the relative improvement values of the macro F-Score by averaging them on a per-selector basis as shown in Table 5.9, we notice the TOP baseline performs the second best after SOFITDA. The TOP baseline only takes the likelihood of a synthetic item belonging to a minority label as a factor in determining whether or not to select the item. The fact that SOFITDA performs better than this baseline indicates using diversity as an additional factor helps to select better synthetic items. The RANDOM baseline is the second-best performing baseline and this baseline represents the case where we select synthesized items without applying any criteria. While this approach is better than not using synthesized items because the RANDOM baseline

TABLE 5.7: The macro F-Score of SOFITDA and the baseline selectors on a per dataset, generator, and classifier basis. Bold font indicates the best results. † indicates a statistically significant improvement over the NO_AUG baseline.

Dataset	Generator	Selector	BERT	CNN	RNN	BOW
Amazon	EDA	NO_AUG	0.489	0.389	0.406	0.402
		RANDOM	0.424	0.442	0.447	0.451
		BOTTOM	0.367	0.337	0.332	0.338
		TOP	0.455	0.422	0.428	0.427
		SOFITDA	0.498	0.453 †	0.474 †	0.468 †
	GPT-2	NO_AUG	0.461	0.362	0.337	0.400
		RANDOM	0.439	0.427	0.424	0.419
		BOTTOM	0.336	0.323	0.265	0.290
		TOP	0.472	0.444	0.461	0.438
		SOFITDA	0.495 †	0.467 †	0.484 †	0.455 †
Yelp	EDA	NO_AUG	0.560	0.492	0.458	0.510
		RANDOM	0.547	0.507	0.510	0.516
		BOTTOM	0.509	0.384	0.293	0.447
		TOP	0.532	0.501	0.521	0.515
		SOFITDA	0.565	0.511	0.521 †	0.524
	GPT-2	NO_AUG	0.522	0.477	0.431	0.505
		RANDOM	0.521	0.482	0.489	0.490
		BOTTOM	0.443	0.333	0.352	0.382
		TOP	0.527	0.482	0.494	0.515
		SOFITDA	0.563 †	0.514 †	0.513 †	0.518
Quora	EDA	NO_AUG	0.761	0.691	0.746	0.650
		RANDOM	0.748	0.731	0.714	0.740
		BOTTOM	0.764	0.694	0.702	0.683
		TOP	0.758	0.734	0.715	0.700
		SOFITDA	0.787	0.748 †	0.755	0.718 †
	GPT-2	NO_AUG	0.775	0.684	0.696	0.648
		RANDOM	0.745	0.706	0.716	0.717
		BOTTOM	0.636	0.585	0.593	0.613
		TOP	0.789	0.737	0.727	0.717
		SOFITDA	0.797	0.754 †	0.752 †	0.736 †
Trip Advisor	EDA	NO_AUG	0.528	0.424	0.470	0.467
		RANDOM	0.514	0.483	0.495	0.486
		BOTTOM	0.486	0.429	0.368	0.445
		TOP	0.529	0.480	0.477	0.481
		SOFITDA	0.543	0.489 †	0.503 †	0.490 †
	GPT-2	NO_AUG	0.526	0.430	0.473	0.465
		RANDOM	0.488	0.423	0.403	0.476
		BOTTOM	0.428	0.317	0.391	0.384
		TOP	0.525	0.457	0.460	0.484
		SOFITDA	0.540	0.483 †	0.500 †	0.492 †
SST-5	EDA	NO_AUG	0.388	0.343	0.308	0.357
		RANDOM	0.421	0.381	0.373	0.353
		BOTTOM	0.355	0.345	0.267	0.355
		TOP	0.411	0.357	0.368	0.345
		SOFITDA	0.439 †	0.408 †	0.399 †	0.365
	GPT-2	NO_AUG	0.426	0.337	0.341	0.355
		RANDOM	0.415	0.396	0.401	0.373
		BOTTOM	0.356	0.296	0.328	0.359
		TOP	0.422	0.404	0.400	0.376
		SOFITDA	0.441	0.409 †	0.412 †	0.387 †
TREC-6	EDA	NO_AUG	0.947	0.900	0.892	0.839
		RANDOM	0.947	0.874	0.889	0.785
		BOTTOM	0.932	0.758	0.870	0.673
		TOP	0.949	0.908	0.900	0.817
		SOFITDA	0.965	0.921	0.913	0.828
	GPT-2	NO_AUG	0.957	0.897	0.879	0.840
		RANDOM	0.950	0.888	0.888	0.829
		BOTTOM	0.934	0.897	0.879	0.811
		TOP	0.937	0.889	0.911	0.810
		SOFITDA	0.972	0.926 †	0.910	0.823

TABLE 5.8: The macro F-Score of the TAA baseline and SOFITDA on a per-dataset basis. We compare the TAA baseline with a SOFITDA model that uses BERT as a classifier and EDA as a generator. Bold font indicates the best results.

Dataset	TAA	SOFITDA
Amazon	0.491	0.498
Yelp	0.561	0.563
Quora	0.773	0.787
Trip Advisor	0.531	0.543
SST-5	0.421	0.439
TREC-6	0.952	0.965

TABLE 5.9: The mean relative improvement in percentage of the macro F-Score (F), macro precision (P), and macro recall (R) of the SOFITDA selector over the baseline selectors on a per classifier basis averaged across the datasets and generators.

Selector	BERT			CNN			RNN			BOW			Selector Average		
	F	P	R	F	P	R	F	P	R	F	P	R	F	P	R
NO_AUG	+4.16	-0.21	+4.41	+12.46	+0.63	+11.41	+14.24	+5.68	+10.22	+6.52	-1.76	+7.17	+9.34	+1.09	+8.30
RANDOM	+7.05	+7.40	+4.66	+5.37	+7.68	+0.13	+6.53	+10.86	-0.15	+2.91	+8.39	-2.93	+5.47	+8.58	+0.43
BOTTOM	+20.22	+18.73	+14.86	+29.20	+32.23	+16.88	+35.87	+37.87	+20.05	+20.54	+26.56	+9.16	+26.46	+28.85	+15.24
TOP	+4.56	+0.83	+5.89	+4.57	+1.22	+5.20	+4.60	+2.33	+4.79	+2.99	+1.37	+2.87	+4.18	+1.44	+4.69
Classifier Average	+7.20	+5.35	+5.97	+10.32	+8.35	+6.73	+12.25	+11.35	+6.98	+7.20	+5.35	+5.97			

TABLE 5.10: The relative improvement in percentage of the macro F-Score, macro precision, and macro recall of the SOFITDA selectors configured with a BERT classifier and an EDA generator over their equivalent TAA baselines averaged across all datasets. Note, the TAA baselines use a BERT classifier as well as generation methods used by the EDA generator.

Macro F-Score	Macro F-Precision	Macro F-Recall
+1.915	+0.573	+2.71

performs better than the NO_AUG baseline, the fact that the RANDOM baseline performs worse than the top two selectors underscores the need for having good selection criteria for synthesized items. The BOTTOM baseline is the worst-performing baseline. This baseline selects synthesized items that have the least likelihood of belonging to their respective labels. While this approach is good in augmenting a training set for an adversarial model, where the objective is to select long tail items, it performs badly when training a classifier for a non-adversarial scenario.

When we aggregate the relative improvement values of the macro F-Score by averaging them on a per-classifier basis as shown in Table 5.9, we notice the CNN and RNN classifiers record the largest relative improvement values. This is due to the fact that both models have sufficient capacity to learn more and the GloVe embeddings they use are limited in their ability to transfer knowledge when compared to non-linear and deep embeddings such as BERT. Therefore, adding synthetic data to the training sets of these classifiers results in relatively large performance increases. The BERT and BOW classifiers have about the same and lower relative improvement values, but for different reasons. The BERT classifier is able to utilize the large amount of prior knowledge in the BERT embeddings, and therefore its performance is impacted less by the synthetic items. On the other hand, the BOW classifier has limited capacity, because it is a linear model, and therefore is less able to take advantage of the synthetic items to boost its performance.

To get more insight into the macro F-Score values, let us take a look at the relative improvement values of the macro precision and macro recall shown in Table 5.9. We observe several negative relative improvement values for the macro precision and macro recall for certain baselines, which indicate their corresponding SOFITDA on average performs worse. However, the negative relative improvement values for the macro precision and the macro recall values are more than offset by positive relative improvement values for their corresponding macro recall and macro precision values, respectively. As a result, the relative improvement values of the macro F-Scores in the table are all positive, since the F-Score is impacted by both the precision and recall.

Upon closer inspection of the relative improvement values for the macro precision, we notice the NO_AUG and TOP baselines have negative or low relative improvement values. This is due to SOFITDA on average having items with less likelihood than the NO_AUG and TOP baselines because it uses diversity as an additional selection criterion. As a result, the precision of the selectors is lower. However, SOFITDA has higher recall than both the NO_AUG and TOP baselines as illustrated by the higher and positive relative improvement values for the macro recall. This is due to SOFITDA on average having more diverse items than the NO_AUG and TOP baselines, which increases the recall of the selectors. Conversely, the RANDOM baseline has high and positive relative improvement values for macro precision, but low or negative relative improvement values for macro recall. This is because on average the items of the RANDOM baseline have less likelihood and higher diversity than SOFITDA because the baseline does not use likelihood as a selection criterion. As a result, the precision of the baseline is lower, while the recall of the baseline is higher.

Table 5.10 shows the relative improvement values of the macro F-Score, macro precision, and macro recall of the SOFITDA selectors configured with an EDA generator and a BERT classifier over their equivalent TAA baselines averaged across all datasets. In all cases, the relative improvement values indicate the TAA baselines underperform their SOFITDA equivalents.

5.5.3 Likelihood versus Diversity

In this section, we investigate the trade-off between the likelihood and diversity of the synthetic items selected by the SOFITDA selectors. In Figure 5.2, we show the charts of the mean percentage of synthesized items (Overlap Percentage) selected by the SOFITDA selectors that are also selected by the TOP baseline on a per α value basis for each unique tuple of dataset, generator, and classifier. For each α , we compute the mean overlap percentage across all the N_C values in the tuple. The TOP baseline is equivalent to a SOFITDA selector with an α value of 1, and selects synthesized items only on the basis of likelihood. By subtracting from 100 percent the overlap percentage, we compute the mean proportion of synthesized items that are selected by the SOFITDA selectors on the basis of diversity. From the figure, we can see in nearly all cases a smooth increase in the proportion of synthesized items selected on the basis of diversity as the value of α decreases from 0.9 to 0.1. This is because the overlap percentage decreases when the value of α decreases. The rate of decrease varies depending on the dataset, generator, and classifier. In Figure 5.1, we see α controls the concavity of the SOFITDA selector. The less the value of α , the larger the concavity. Therefore, we can conclude from our observation of the charts that an increase in concavity results in a smooth increase of the proportion of synthesized items that are selected on the basis of diversity. The mean optimal α of SOFITDA is 0.45. From the charts, we can gather this means 20% - 35% of the synthesized items are selected on the basis of diversity whereas the rest are selected on the likelihood of them being an instance of their corresponding label.

In Figure 5.3, we show the overlap percentage on the basis of the number of clusters, N_C , instead of α . In the case of this figure, we compute the mean overlap percentage for each N_C value across all the α values for every unique tuple of dataset, generator, and classifier. Unlike in Figure 5.2, we do not see a smooth increase in the proportion of synthesized items selected on the basis of diversity as we increase the number of clusters. The increase is generally noisy and does not consistently trend downwards for a higher number of clusters. For example, in the case of the charts for the Quora dataset, we see a noisy decrease of the proportion of synthesized items selected on the basis of diversity when the number of clusters is greater than 8. Therefore, unlike α , the impact of N_C in boosting diversity is limited to the first few number of clusters. Furthermore, we see general trends in the shapes of the charts that are dataset-specific. We do not nearly see such noticeable trends for the type of generator or the type of classifier. This indicates the impact of the number of clusters on the overlap percentage is primarily driven by the content of the datasets.

5.5.4 Hyperparameter Analysis

In this section, we investigate the impact of the hyperparameters on the SOFITDA selectors. α is a hyperparameter of the SOFITDA selector we use in our experiments and it controls the concavity of the function. The greater the value of α the less concave the function becomes. Therefore, an analysis of the impact of α gives us an insight into how concavity impacts the performance of the text classifications. In Figure 5.4, we analyze the macro F-Score impact of α on a per dataset, generator, and classifier basis

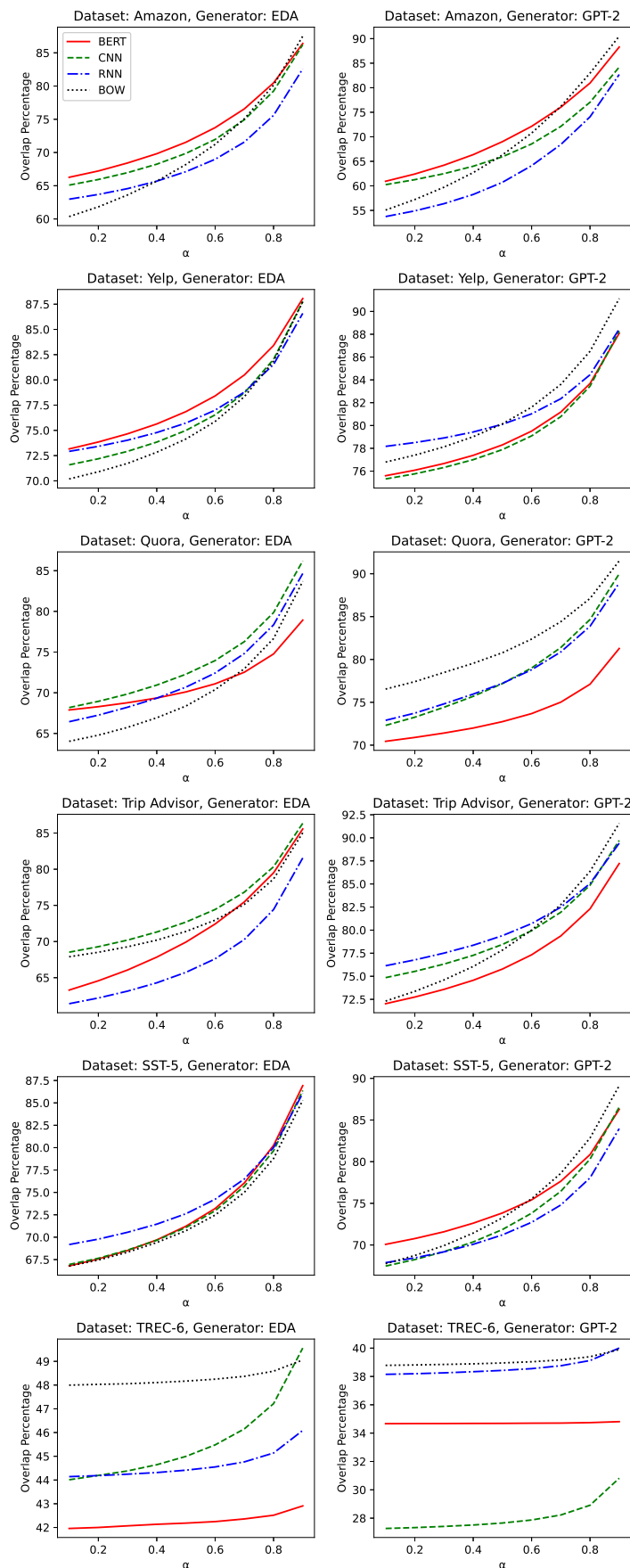


FIGURE 5.2: Charts showing the mean percentage of synthesized items (Overlap Percentage) selected by the SOFITDA selectors that are also selected by the TOP baseline on a per α value basis for each unique combination of dataset, generator, and classifier.

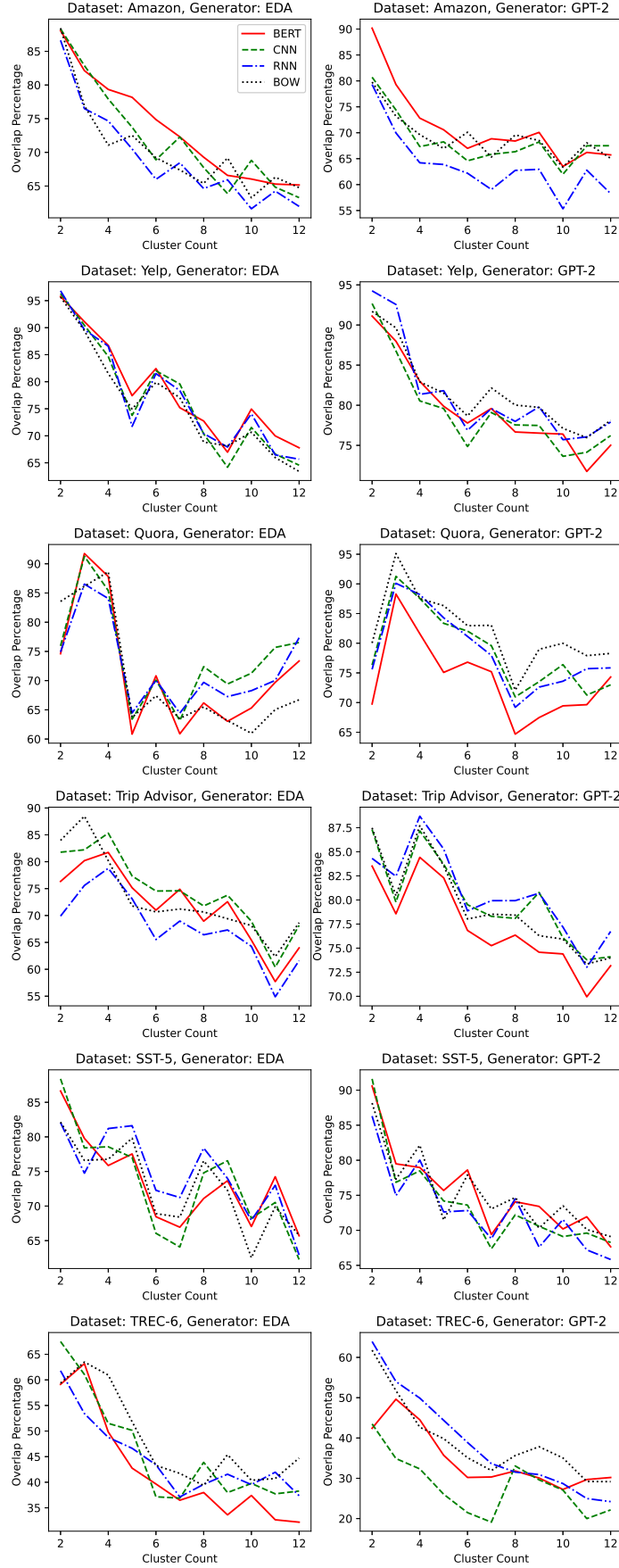


FIGURE 5.3: Charts showing the mean percentage of synthesized items (Overlap Percentage) selected by the SOFITDA selectors that are also selected by the TOP baseline on a per N_C value basis for each unique combination of dataset, generator, and classifier.

by plotting the mean macro F-Score of each α value across all the N_C values. We notice from the plots a non-linear distribution of the local maxima of the macro F-Score with the variance of the maxima increasing with the non-linearity of the classifiers. The order of the classifiers in ascending order of non-linearity is BOW, CNN, RNN, and BERT. Furthermore, we notice the variance of the plots is higher for EDA than GPT-2. This may have to do with the fact that the diversity of EDA’s synthetic items is lower than that of GPT-2’s. This is because EDA makes small mutations to existing text that may not be syntactically and semantically correct, whereas GPT-2 generates new text that is of higher syntactic and semantic quality. As a result, the performance of the classifiers has a greater variance when a group of items derived from the same existing item are synthesized by EDA and the existing item has a high positive or negative impact on performance. There appear to be no meaningful dataset-specific correlations in the structure of the plots. All significant correlations appear to be classifier or generator specific.

Similar to the way we analyze α , we analyze in Figure 5.5 the macro F-Score impact of the number of clusters, N_C , on a per dataset, generator, and classifier basis by plotting the mean macro F-Score of each N_C value across all the α values. Just as we notice for α , we see a non-linear distribution of the local maxima of the macro F-Score with the variance of the maxima increasing with the non-linearity of the classifiers. However, there appear to be no meaningful dataset or generator correlations in the structure of the plots. All significant correlations appear to be classifier specific.

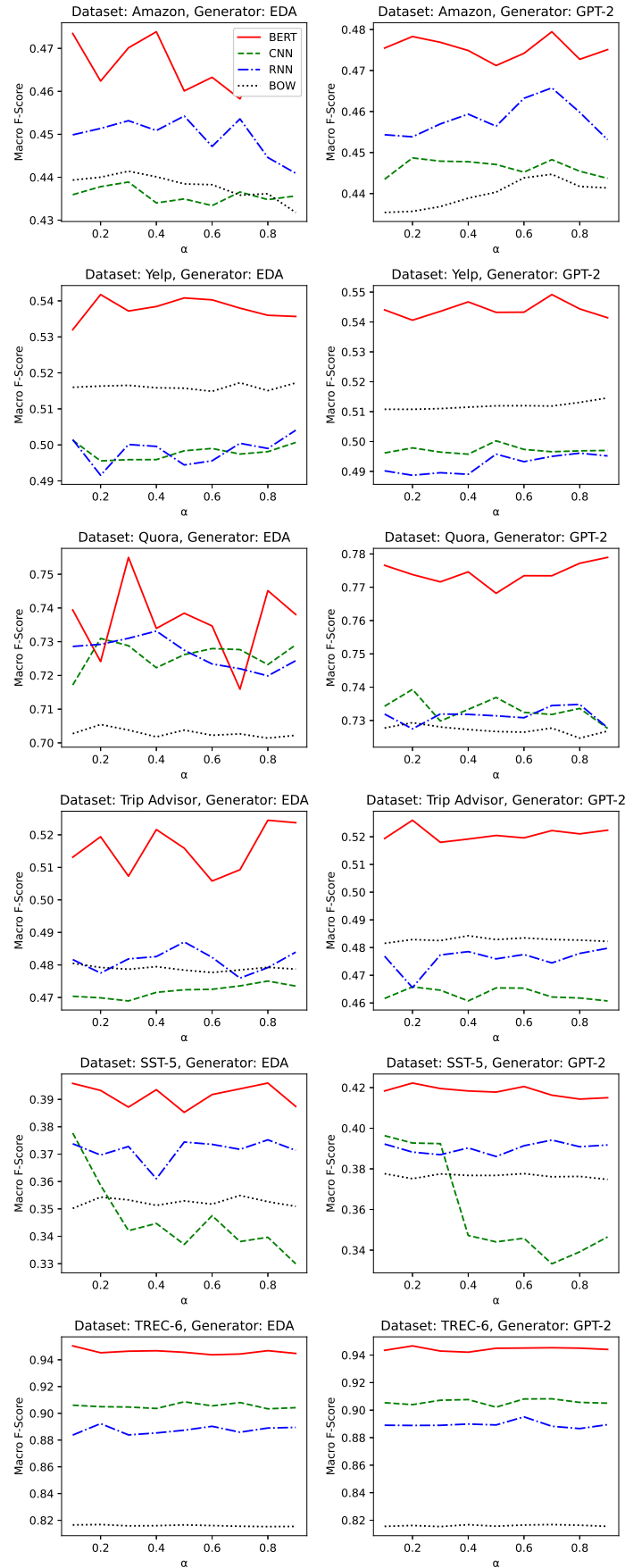


FIGURE 5.4: Charts showing the mean macro F-Score of the SOFITDA selectors versus their α values for each unique combination of dataset, generator, and classifier.

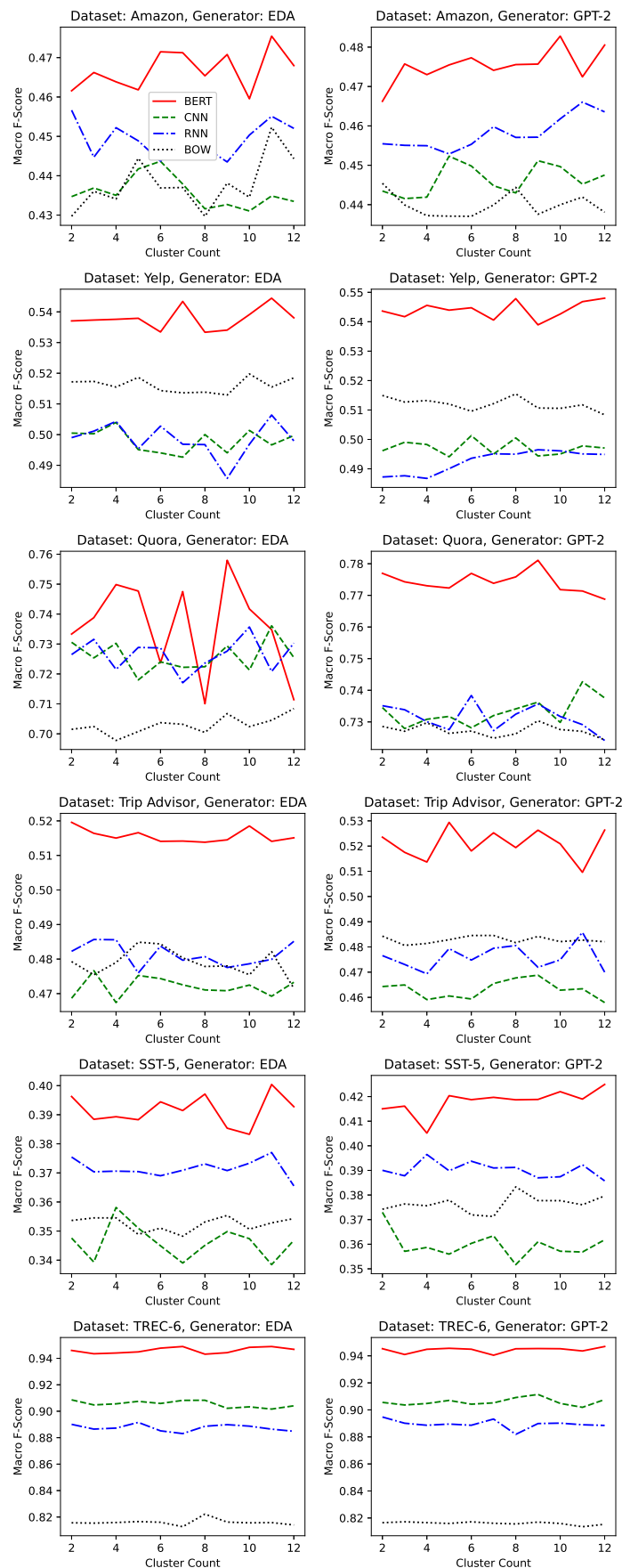


FIGURE 5.5: Charts showing the mean macro F-Score of the SOFITDA selectors versus their N_C values for each unique combination of dataset, generator, and classifier.

5.5.5 Cluster Analysis

In Figure 5.6 and 5.7, we illustrate charts showing the projection of the TF-IDF vectors of 1000 synthesized items in two of the most significant PCA dimensions. In the case of Figure 5.6, the charts are for the EDA generator, and in the case of Figure 5.7, the charts are for the GPT-2 generator. In both cases, we plot charts for each unique combination of dataset and classifier. The points shown in blue represent synthesized items selected only by the TOP selector, and the points shown in red represent synthesized items selected only by the SOFITDA selector.

From the charts, we make the general observation that the blue and red plots have varying clustering properties. That is they have distinct areas where they are concentrated. In some cases, their concentrations barely overlap as is the case with the Quora charts. In other cases, their concentrations are a subset of one another as is the case with the Amazon charts. The fact that the clustering properties vary between the TOP and SOFITDA selectors indicates that the SOFITDA selector adds groups of synthetic items that are not present in the set of synthetic items selected by the TOP selector.

In addition, we note that the equivalent clusters in EDA and GPT-2 have different shapes. This is because the vocabulary distribution of the synthetic items generated by EDA and GPT-2 are different. As a result, their TF-IDF vectors are different. In addition, we note that GPT-2 clusters have a finer distribution of points when compared to EDA. This is because the vocabulary of synthetic items generated by GPT-2 is much larger than that of EDA.

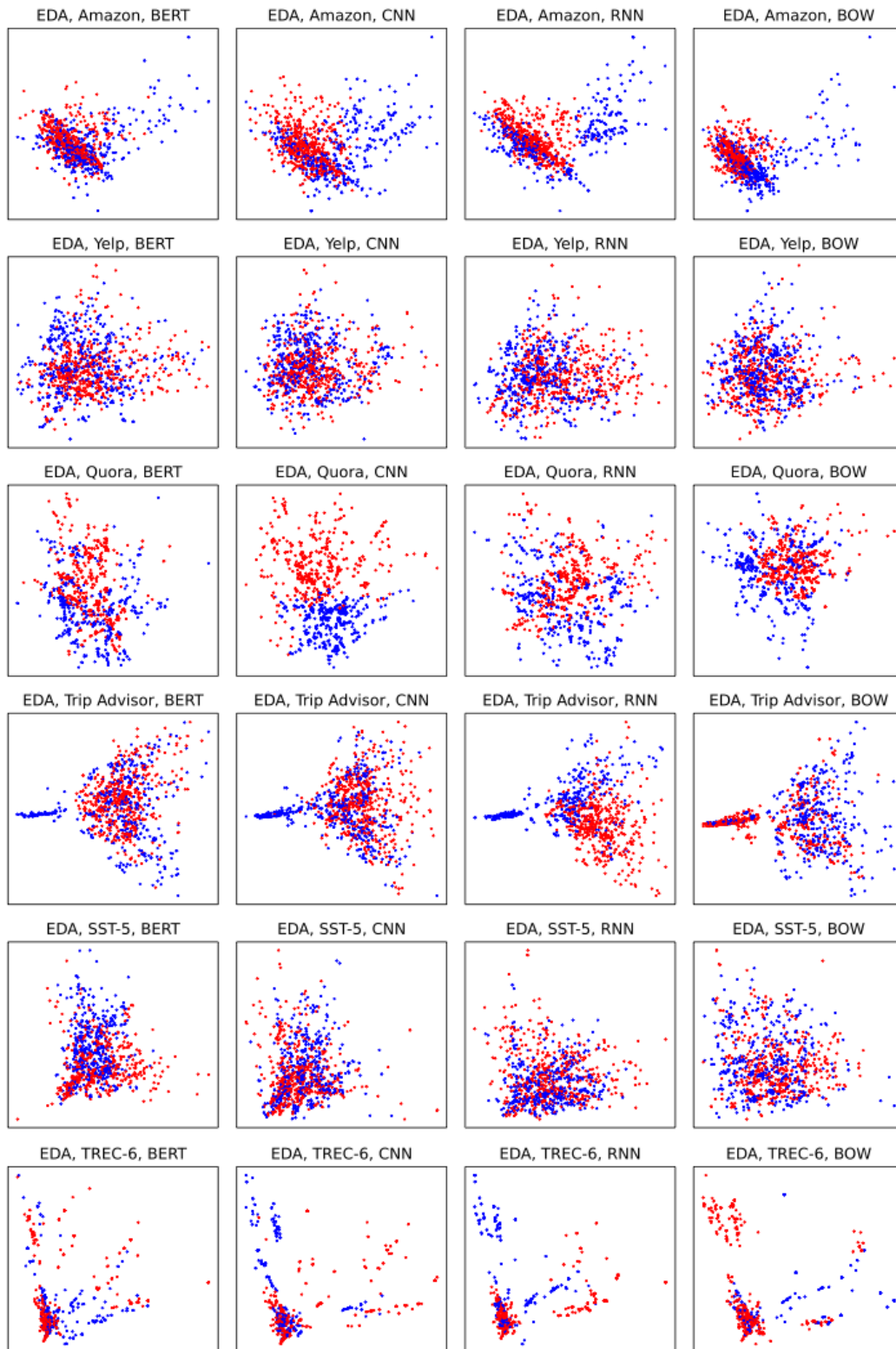


FIGURE 5.6: Charts showing the projection of the TF-IDF vectors of 1000 synthesized items in two of the most significant PCA dimensions. The charts are for the EDA generator for each unique combination of dataset, and classifier. The points shown in blue represent synthesized items selected only by the TOP selector, and the points shown in red represent synthesized items selected only by the SOFITDA selector.

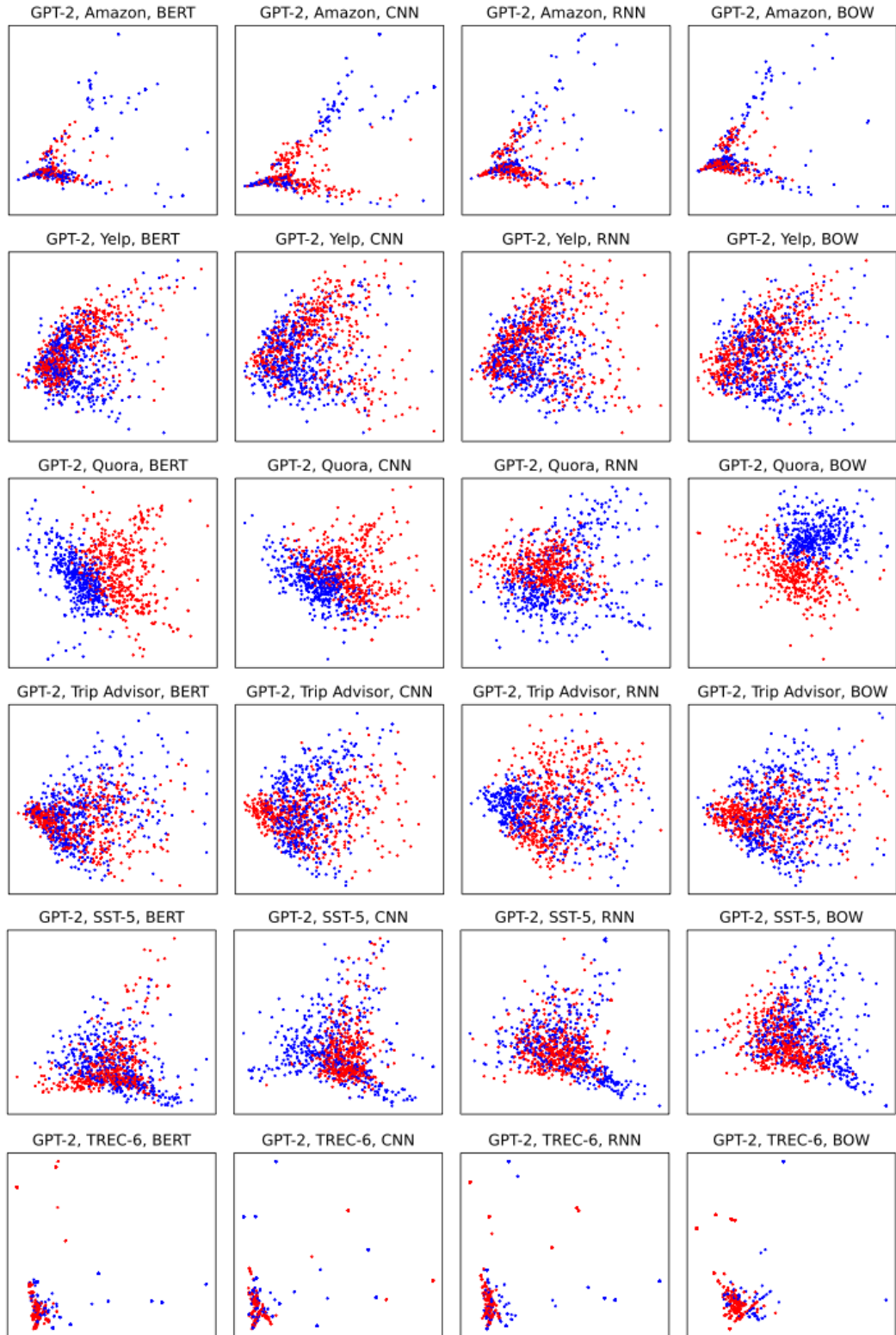


FIGURE 5.7: Charts showing the projection of the TF-IDF vectors of 1000 synthesized items in two of the most significant PCA dimensions. The charts are for the GPT-2 generator for each unique combination of dataset, and classifier. The points shown in blue represent synthesized items selected only by the TOP selector, and the points shown in red represent synthesized items selected only by the SOFITDA selector.

TABLE 5.11: The total number of synthetic items selected for each dataset as well as the mean execution time in seconds for executing the SOFITDA selector on a per-dataset basis for all labels .

	Amazon	Yelp	Quora	Trip Advisor	SST-5	TREC-6
Number of Items Selected	71,386	50,303	35,025	42,110	3,851	6,374
Execution Time	0.58	0.41	0.30	0.35	0.03	0.05

5.5.6 Time Efficiency

We note in Section 4.2.8 that the time complexity of the SOFITDA selector is a linear function of the number of items selected for a given label. In Table 5.11, we show the mean execution time in seconds for executing the SOFITDA selector on a per-dataset basis for all labels as well as the total number of synthetic items selected for each dataset. We run the SOFITDA selector on a 2.6 GHZ, 6-Core, Intel i7 processor with 16 GB of RAM. From the visualization of the data in Figure 5.8, we can see a linear correlation between the execution time and the number of selected items, which validates the theoretical analysis. Furthermore, for the dataset sizes we used, the SOFITDA selector runs in a sub-second time. It is much more efficient than the brute-force approach (without using monotone submodular optimization), which would be intractable given the sizes of the datasets.

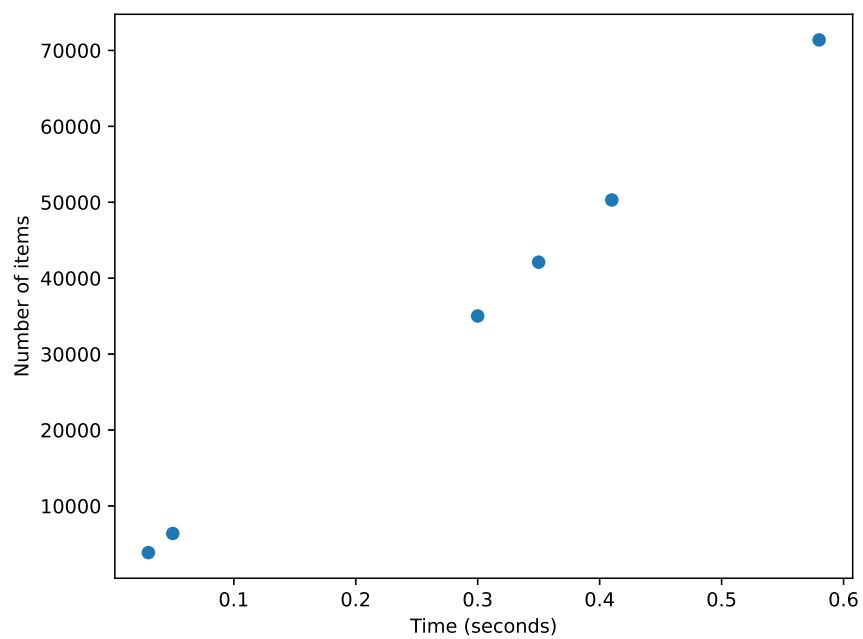


FIGURE 5.8: A scatter plot of the mean execution time of the SOFITDA selector for each dataset as specified in Table 5.11. The plot clearly shows a linear correlation between the execution time and the number of items selected.

Chapter 6

Conclusion and Future Work

In this thesis, we introduce a principled approach to balance the minority labels of an imbalanced text dataset by selecting the required number of items from a list of items synthesized by a data augmentation model. The fact that most data augmentation models can synthesize an unbounded number of items necessitates the need to have a selection method to identify the items that best improve the performance of the dataset for classification tasks. Our selection method is based on selecting synthesized items whose likelihood of belonging to their respective labels as well as their diversity are maximized. The initial formulation of our selection method is intractable, but we were able to approximate it by reformulating it as a monotone submodular objective whose solution can be approximated by a tractable greedy algorithm.

We empirically evaluated the performance of our selection method on a permutation of 6 datasets, 2 widely used data augmentation models, and 4 of the most common types

of text classifiers. We extensively evaluated different configurations of our selection method and compared the performance of the best-performing configurations with 5 baselines. Our best-performing configurations performed significantly better than the baselines on the evaluation metric of our experiments.

Now that we have conceptually demonstrated the effectiveness of our method, our potential avenues for future work are as follows:

- **Optimizing Hyperparameters:** In our experiments, we used grid-search to find the optimal hyperparameters, α and N_C , of the SOFITDA selector. This is a time consuming effort, and the hyperparameters are constrained to specific ranges and resolutions. One way to overcome this problem is to use non-brute force techniques such as Bayesian optimizers to systematically discover the optimal hyperparameters. These optimizers estimate a surrogate model that predicts the loss (error) for a given hyperparameter tuple. The surrogate model leverages the previous losses of hyperparameter tuples to select a new hyperparameter tuple that is likely to reduce the loss. The optimizers repeat this selection process iteratively until the loss converges.
- **Exploring other Combinatorial Approximations:** In this thesis, we explored approximating our original objective for selecting synthetic items, Equation (4.6), using a monotone submodular objective. We should explore other combinatorial approximations for our original objective since some of them may give a tighter

bound than the monotone submodular objective. In this case, our Data Augmentation Pipeline remains the same, but we will have a new selector.

- **Combining Generator and Selector:** Currently, the generator and selector are two different functions that are optimized separately. As a result, the generator selects more synthetic items than necessary and the selector selects a subset of the items that it deems are optimal. Ideally, we should combine the generator and selector into a single architecture so that we can optimize them jointly. This would allow us to generate items that maximize both the likelihood and diversity, which eliminates the need to over generate items for a separate selection step.
- **Using Various Text Representations and Clustering Algorithms:** So far we have used one text representation, TF-IDF, and one clustering algorithm, Mini-Batch K-Means. An area of future work is to evaluate the performance of our approach on state-of-the-art semantic representations such as Sentence-BERT [58], and advanced clustering algorithms such as the Gaussian Mixture model [48].
- **Large Language Models:** Recent advancements in Large Language Model (LLM) development such as GPT-3 [13], LLaMa [71], and PaLM [16] create a new opportunity for generating synthetic items by giving free-form natural language instructions to the LLM. This type of generation is known as zero-shot since no fine-tuning process is required by the LLM to mimic the items in the training dataset. Selecting a subset of these synthetic items using our methodology is a new area of research.

Chapter 7

Appendix

7.1 Monotone Submodularity Proof

The following are the proofs for the monotone submodularity of $\tilde{Z}(\cdot)$ as defined in Equation (4.12). Specifically, Theorem 1 shows $\tilde{Z}(\cdot)$ is submodular and Theorem 2 demonstrates the function is monotonic.

7.1.1 Theorem 1 (Submodularity)

For all $\mathcal{S} \subseteq \mathcal{T} \subseteq \mathcal{E}$ and $\{l\} \in \mathcal{E} \setminus \mathcal{T}$:

$$\tilde{Z}(\mathcal{T} \cup \{l\}) - \tilde{Z}(\mathcal{T}) \leq \tilde{Z}(\mathcal{S} \cup \{l\}) - \tilde{Z}(\mathcal{S}) \quad (7.1)$$

Proof: A concave function has a non-increasing gradient. This means for any $a \in \mathbb{R}$ and $a \geq 0$, and a concave function f :

$$\begin{aligned}
 f'(x+a) &\leq f'(x) \\
 \lim_{h \rightarrow 0+} \frac{f(x+a+h) - f(x+a)}{h} &\leq \lim_{h \rightarrow 0+} \frac{f(x+h) - f(x)}{h} \\
 \lim_{h \rightarrow 0+} f(x+a+h) - f(x+a) &\leq \lim_{h \rightarrow 0+} f(x+h) - f(x) \\
 f(x+a+h) - f(x+a) &\leq f(x+h) - f(x)
 \end{aligned} \tag{7.2}$$

From the last inequality, we can conclude the following is true for a cluster i , because $H(\cdot)$ is a concave function and $r_{i,d} \geq 0$, since $r_{i,d}$ is a probability value.

$$\begin{aligned}
 H\left(\sum_{d \in \mathbf{C}_i \cap (\mathcal{T} \cup \{l\})} r_{i,d}\right) - H\left(\sum_{d \in \mathbf{C}_i \cap \mathcal{T}} r_{i,d}\right) &\leq \\
 H\left(\sum_{d \in \mathbf{C}_i \cap (\mathcal{S} \cup \{l\})} r_{i,d}\right) - H\left(\sum_{d \in \mathbf{C}_i \cap \mathcal{S}} r_{i,d}\right)
 \end{aligned} \tag{7.3}$$

We can generalize the above to the sum of all clusters, since the sum of concave functions is a concave function:

$$\begin{aligned}
 \sum_{i=1}^{N_C} H\left(\sum_{d \in \mathbf{C}_i \cap (\mathcal{T} \cup \{l\})} r_{i,d}\right) - \sum_{i=1}^{N_C} H\left(\sum_{d \in \mathbf{C}_i \cap \mathcal{T}} r_{i,d}\right) &\leq \\
 \sum_{i=1}^{N_C} H\left(\sum_{d \in \mathbf{C}_i \cap (\mathcal{S} \cup \{l\})} r_{i,d}\right) - \sum_{i=1}^{N_C} H\left(\sum_{d \in \mathbf{C}_i \cap \mathcal{S}} r_{i,d}\right)
 \end{aligned} \tag{7.4}$$

By substituting $\tilde{Z}(\cdot)$ for the cluster summations, we successfully complete the proof:

$$\tilde{Z}(\mathcal{T} \cup \{l\}) - \tilde{Z}(\mathcal{T}) \leq \tilde{Z}(\mathcal{S} \cup \{l\}) - \tilde{Z}(\mathcal{S}) \quad (7.5)$$

7.1.2 Theorem 2 (Monotonicity)

For all $\mathcal{S} \subseteq \mathcal{T} \subseteq \mathcal{E}$:

$$\tilde{Z}(\mathcal{S}) \leq \tilde{Z}(\mathcal{T}) \quad (7.6)$$

Proof: $H(\cdot)$ is a non-decreasing function. This means for any $b \in \mathbb{R}$ and $b \geq 0$,

$H(a) \leq H(a + b)$. Using this fact, we make the proof as follows:

$$\begin{aligned} \tilde{Z}(\mathcal{S}) &= \sum_{i=1}^{N_C} H \left(\sum_{d \in C_i \cap \mathcal{S}} r_{i,d} \right) \\ &\leq \sum_{i=1}^{N_C} H \left(\sum_{d \in C_i \cap (\mathcal{S} \cup \mathcal{T})} r_{i,d} \right) \\ &\leq \sum_{i=1}^{N_C} H \left(\sum_{d \in C_i \cap \mathcal{T}} r_{i,d} \right) \\ &\leq \tilde{Z}(\mathcal{T}) \end{aligned} \quad (7.7)$$

Bibliography

- [1] Amazon customer reviews dataset.
- [2] Quora insincere questions classification.
- [3] Tripadvisor hotel reviews.
- [4] Yelp open dataset.
- [5] Eyor Alemayehu and Yi Fang. A submodular optimization framework for imbalanced text classification with data augmentation. *IEEE Access*, 2023.
- [6] Moustafa Alzantot, Yash Sharma, Ahmed Elgohary, Bo-Jhang Ho, Mani Srivastava, and Kai-Wei Chang. Generating natural language adversarial examples. *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, 2018.
- [7] S-I Amari. Learning patterns and pattern sequences by self-organizing nets of threshold elements. *IEEE Transactions on computers*, 100(11):1197–1206, 1972.

-
- [8] Ateret Anaby-Tavor, Boaz Carmeli, Esther Goldbraich, Amir Kantor, George Kour, Segev Shlomov, Naama Tepper, and Naama Zwerdling. Do not have enough data? deep learning to the rescue! In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 34, pages 7383–7390, 2020.
 - [9] Markus Bayer, Marc-André Kaufhold, Björn Buchhold, Marcel Keller, Jörg Dallmeyer, and Christian Reuter. Data augmentation in natural language processing: a novel text generation approach for long and short text classifiers. *International Journal of Machine Learning and Cybernetics*, pages 1–16, 2022.
 - [10] Markus Bayer, Marc-André Kaufhold, and Christian Reuter. A survey on data augmentation for text classification. *arXiv preprint arXiv:2107.03158*, 2021.
 - [11] Yonatan Belinkov and Yonatan Bisk. Synthetic and natural noise both break neural machine translation. In *International Conference on Learning Representations*, 2018.
 - [12] Paula Branco, Luís Torgo, and Rita P Ribeiro. A survey of predictive modeling on imbalanced domains. *ACM computing surveys (CSUR)*, 49(2):1–50, 2016.
 - [13] Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, et al. Language models are few-shot learners. *Advances in neural information processing systems*, 33:1877–1901, 2020.
 - [14] Jiaao Chen, Zichao Yang, and Diyi Yang. Mixtext: Linguistically-informed interpolation of hidden space for semi-supervised text classification. In *Proceedings of*

- the 58th Annual Meeting of the Association for Computational Linguistics*, pages 2147–2157, 2020.
- [15] Kyunghyun Cho, Bart Van Merriënboer, Caglar Gulcehre, Dzmitry Bahdanau, Fethi Bougares, Holger Schwenk, and Yoshua Bengio. Learning phrase representations using rnn encoder-decoder for statistical machine translation. *arXiv preprint arXiv:1406.1078*, 2014.
- [16] Aakanksha Chowdhery, Sharan Narang, Jacob Devlin, Maarten Bosma, Gaurav Mishra, Adam Roberts, Paul Barham, Hyung Won Chung, Charles Sutton, Sebastian Gehrmann, et al. Palm: Scaling language modeling with pathways. *arXiv preprint arXiv:2204.02311*, 2022.
- [17] Gerard Cornuejols, Marshall L Fisher, and George L Nemhauser. Exceptional paper—location of bank accounts to optimize float: An analytic study of exact and approximate algorithms. *Management science*, 23(8):789–810, 1977.
- [18] Claude Coulombe. Text data augmentation made simple by leveraging nlp cloud apis. *arXiv preprint arXiv:1812.04718*, 2018.
- [19] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*, 2018.
- [20] Javid Ebrahimi, Anyi Rao, Daniel Lowd, and Dejing Dou. Hotflip: White-box adversarial examples for text classification. In *Proceedings of the 56th Annual Meeting*

- of the Association for Computational Linguistics (Volume 2: Short Papers)*, pages 31–36, 2018.
- [21] Andrew Estabrooks, Taeho Jo, and Nathalie Japkowicz. A multiple resampling method for learning from imbalanced data sets. *Computational intelligence*, 20(1):18–36, 2004.
- [22] Steven Y Feng, Varun Gangal, Dongyeop Kang, Teruko Mitamura, and Eduard Hovy. Genaug: Data augmentation for finetuning text generators. In *Proceedings of Deep Learning Inside Out (DeeLIO): The First Workshop on Knowledge Extraction and Integration for Deep Learning Architectures*, pages 29–42, 2020.
- [23] Kunihiro Fukushima. Neocognitron: A self-organizing neural network model for a mechanism of pattern recognition unaffected by shift in position. *Biological cybernetics*, 36(4):193–202, 1980.
- [24] Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial nets. *Advances in neural information processing systems*, 27, 2014.
- [25] Kelvin Guu, Tatsunori B Hashimoto, Yonatan Oren, and Percy Liang. Generating sentences by editing prototypes. *Transactions of the Association for Computational Linguistics*, 6:437–450, 2018.
- [26] Zellig S Harris. Distributional structure. *Word*, 10(2-3):146–162, 1954.

-
- [27] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural computation*, 9(8):1735–1780, 1997.
- [28] Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *International conference on machine learning*, pages 448–456. pmlr, 2015.
- [29] Xiaoqi Jiao, Yichun Yin, Lifeng Shang, Xin Jiang, Xiao Chen, Linlin Li, Fang Wang, and Qun Liu. Tinybert: Distilling bert for natural language understanding. In *Findings of the Association for Computational Linguistics: EMNLP 2020*, pages 4163–4174, 2020.
- [30] Michał Jungiewicz and Aleksander Smywiński-Pohl. Towards textual data augmentation for neural networks: synonyms and maximum loss. *Computer Science*, 20, 2019.
- [31] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.
- [32] Diederik P Kingma and Max Welling. Auto-encoding variational bayes. *arXiv preprint arXiv:1312.6114*, 2013.
- [33] Youngjoong Ko. A study of term weighting schemes using class information for text classification. In *Proceedings of the 35th international ACM SIGIR conference on Research and development in information retrieval*, pages 1029–1030, 2012.

-
- [34] Sosuke Kobayashi. Contextual augmentation: Data augmentation by words with paradigmatic relations. In *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 2 (Short Papers)*, pages 452–457, 2018.
- [35] Oleksandr Kolomiyets, Steven Bethard, and Marie-Francine Moens. Model-portability experiments for textual temporal analysis. In *Proceedings of the 49th annual meeting of the association for computational linguistics: human language technologies*, volume 2, pages 271–276. ACL; East Stroudsburg, PA, 2011.
- [36] Anna Kruspe, Jens Kersten, Matti Wiegmann, Benno Stein, and Friederike Klan. Classification of incident-related tweets: Tackling imbalanced training data using hybrid cnns and translation-based data augmentation. In *Proceedings of the 27th Text REtrieval Conference (TREC 2018), Gaithersburg, Maryland, November 14*, volume 16, page 2018, 2018.
- [37] Solomon Kullback and Richard A Leibler. On information and sufficiency. *The annals of mathematical statistics*, 22(1):79–86, 1951.
- [38] Yann LeCun, Bernhard Boser, John S Denker, Donnie Henderson, Richard E Howard, Wayne Hubbard, and Lawrence D Jackel. Backpropagation applied to handwritten zip code recognition. *Neural computation*, 1(4):541–551, 1989.
- [39] Sangwon Lee, Ling Liu, and Wonik Choi. Iterative translation-based data augmentation method for text classification tasks. *IEEE Access*, 9:160437–160445, 2021.

-
- [40] Claude Lemaréchal. Cauchy and the gradient method. *Doc Math Extra*, 251(254):10, 2012.
- [41] Xin Li and Dan Roth. Learning question classifiers. In *COLING 2002: The 19th International Conference on Computational Linguistics*, 2002.
- [42] Yitong Li, Trevor Cohn, and Timothy Baldwin. Robust training under linguistic adversity. In *Proceedings of the 15th Conference of the European Chapter of the Association for Computational Linguistics: Volume 2, Short Papers*, pages 21–27, 2017.
- [43] Hui Lin and Jeff Bilmes. A class of submodular functions for document summarization. In *Proceedings of the 49th annual meeting of the association for computational linguistics: human language technologies*, pages 510–520, 2011.
- [44] Ruibo Liu, Guangxuan Xu, Chenyan Jia, Weicheng Ma, Lili Wang, and Soroush Vosoughi. Data boost: Text data augmentation through reinforcement learning guided conditional generation. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 9031–9041, 2020.
- [45] Xiaodong Liu, Hao Cheng, Pengcheng He, Weizhu Chen, Yu Wang, Hoifung Poon, and Jianfeng Gao. Adversarial training for large neural language models. *arXiv preprint arXiv:2004.08994*, 2020.
- [46] Jiawei Luo, Mondher Bouazizi, and Tomoaki Ohtsuki. Data augmentation for sentiment analysis using sentence compression-based seqgan with data screening. *IEEE Access*, 9:99922–99931, 2021.

-
- [47] Vukosi Marivate and Tshephisho Sefara. Improving short text classification through global augmentation methods. In *International Cross-Domain Conference for Machine Learning and Knowledge Extraction*, pages 385–399. Springer, 2020.
- [48] Geoffrey J McLachlan, Sharon X Lee, and Suren I Rathnayake. Finite mixture models. *Annual review of statistics and its application*, 6:355–378, 2019.
- [49] George A Miller, Richard Beckwith, Christiane Fellbaum, Derek Gross, and Katherine J Miller. Introduction to wordnet: An on-line lexical database. *International journal of lexicography*, 3(4):235–244, 1990.
- [50] Junghyun Min, R Thomas McCoy, Dipanjan Das, Emily Pitler, and Tal Linzen. Syntactic data augmentation increases robustness to inference heuristics. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 2339–2352, 2020.
- [51] Anna Mosolova, Vadim Fomin, and Ivan Bondarenko. Text augmentation for neural networks. In *AIST (Supplement)*, pages 104–109, 2018.
- [52] Nikola Mrkšić, Diarmuid Ó Séaghdha, Blaise Thomson, Milica Gasic, Lina M Rojas Barahona, Pei-Hao Su, David Vandyke, Tsung-Hsien Wen, and Steve Young. Counter-fitting word vectors to linguistic constraints. In *Proceedings of the 2016 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 142–148, 2016.

-
- [53] Harikrishna Narasimhan, Weiwei Pan, Purushottam Kar, Pavlos Protopapas, and Harish G Ramaswamy. Optimizing the multiclass f-measure via biconcave programming. In *2016 IEEE 16th international conference on data mining (ICDM)*, pages 1101–1106. IEEE, 2016.
- [54] Vijayaditya Peddinti, Daniel Povey, and Sanjeev Khudanpur. A time delay neural network architecture for efficient modeling of long temporal contexts. In *Sixteenth annual conference of the international speech communication association*, 2015.
- [55] Jeffrey Pennington, Richard Socher, and Christopher D Manning. Glove: Global vectors for word representation. In *Proceedings of the 2014 conference on empirical methods in natural language processing (EMNLP)*, pages 1532–1543, 2014.
- [56] Siyuan Qiu, Binxia Xu, Jie Zhang, Yafang Wang, Xiaoyu Shen, Gerard De Melo, Chong Long, and Xiaolong Li. Easyaug: An automatic textual data augmentation platform for classification tasks. In *Companion Proceedings of the Web Conference 2020*, pages 249–252, 2020.
- [57] Alec Radford, Jeffrey Wu, Rewon Child, David Luan, Dario Amodei, and Ilya Sutskever. Language models are unsupervised multitask learners. *OpenAI blog*, 1(8):9, 2019.
- [58] Nils Reimers and Iryna Gurevych. Sentence-bert: Sentence embeddings using siamese bert-networks. *arXiv preprint arXiv:1908.10084*, 2019.
- [59] Shuhuai Ren, Jinchao Zhang, Lei Li, Xu Sun, and Jie Zhou. Text autoaugment: Learning compositional augmentation policy for text classification. In *Proceedings*

- of the 2021 Conference on Empirical Methods in Natural Language Processing*, pages 9029–9043, 2021.
- [60] Georgios Rizos, Konstantin Hemker, and Björn Schuller. Augment to prevent: short-text data augmentation in deep learning for hate-speech classification. In *Proceedings of the 28th ACM International Conference on Information and Knowledge Management*, pages 991–1000, 2019.
- [61] David E Rumelhart, Geoffrey E Hinton, and Ronald J Williams. Learning representations by back-propagating errors. *nature*, 323(6088):533–536, 1986.
- [62] David Sculley. Web-scale k-means clustering. In *Proceedings of the 19th international conference on World wide web*, pages 1177–1178, 2010.
- [63] Ali Shafahi, Mahyar Najibi, Mohammad Amin Ghiasi, Zheng Xu, John Dickerson, Christoph Studer, Larry S Davis, Gavin Taylor, and Tom Goldstein. Adversarial training for free! *Advances in Neural Information Processing Systems*, 32, 2019.
- [64] Dinghan Shen, Mingzhi Zheng, Yelong Shen, Yanru Qu, and Weizhu Chen. A simple but tough-to-beat data augmentation approach for natural language understanding and generation. *arXiv preprint arXiv:2009.13818*, 2020.
- [65] Haoyue Shi, Karen Livescu, and Kevin Gimpel. Substructure substitution: Structured data augmentation for nlp. In *Findings of the Association for Computational Linguistics: ACL-IJCNLP 2021*, pages 3494–3508, 2021.

-
- [66] Richard Socher, Alex Perelygin, Jean Wu, Jason Chuang, Christopher D Manning, Andrew Y Ng, and Christopher Potts. Recursive deep models for semantic compositionality over a sentiment treebank. In *Proceedings of the 2013 conference on empirical methods in natural language processing*, pages 1631–1642, 2013.
- [67] Karen Sparck Jones. A statistical interpretation of term specificity and its application in retrieval. *Journal of documentation*, 28(1):11–21, 1972.
- [68] Xiao Sun and Jiajin He. A novel approach to generate a large scale of supervised data for short text sentiment analysis. *Multimedia Tools and Applications*, 79(9):5439–5459, 2020.
- [69] Ilya Sutskever, Oriol Vinyals, and Quoc V Le. Sequence to sequence learning with neural networks. *Advances in neural information processing systems*, 27, 2014.
- [70] Alexandru Tifrea, Gary Bécigneul, and Octavian-Eugen Ganea. Poincaré glove: Hyperbolic word embeddings. In *Proceedings of the International Conference on Learning Representations (ICLR 2019)*. OpenReview, 2018.
- [71] Hugo Touvron, Thibaut Lavril, Gautier Izacard, Xavier Martinet, Marie-Anne Lachaux, Timothée Lacroix, Baptiste Rozière, Naman Goyal, Eric Hambro, Faisal Azhar, et al. Llama: Open and efficient foundation language models. *arXiv preprint arXiv:2302.13971*, 2023.
- [72] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. *Advances in neural information processing systems*, 30, 2017.

- [73] Li Wan, Matthew Zeiler, Sixin Zhang, Yann Le Cun, and Rob Fergus. Regularization of neural networks using dropconnect. In *International conference on machine learning*, pages 1058–1066. PMLR, 2013.
- [74] Congcong Wang and David Lillis. Classification for crisis-related tweets leveraging word embeddings and data augmentation. In *TREC*, 2019.
- [75] Dilin Wang, Chengyue Gong, and Qiang Liu. Improving neural language modeling via adversarial training. In *International Conference on Machine Learning*, pages 6555–6565. PMLR, 2019.
- [76] Xingkai Wang, Yiqiang Sheng, Haojiang Deng, and Zhenyu Zhao. Charcnn-svm for chinese text datasets sentiment classification with data augmentation. *International Journal of Innovative Computing, Information and Control*, 15(1):227–246, 2019.
- [77] Jason Wei and Kai Zou. Eda: Easy data augmentation techniques for boosting performance on text classification tasks. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*, pages 6382–6388, 2019.
- [78] Xing Wu, Shangwen Lv, Liangjun Zang, Jizhong Han, and Songlin Hu. Conditional bert contextual augmentation. In *International Conference on Computational Science*, pages 84–95. Springer, 2019.
- [79] Yonghui Wu, Mike Schuster, Zhifeng Chen, Quoc V Le, Mohammad Norouzi, Wolfgang Macherey, Maxim Krikun, Yuan Cao, Qin Gao, Klaus Macherey, et al.

- Google’s neural machine translation system: Bridging the gap between human and machine translation. *arXiv preprint arXiv:1609.08144*, 2016.
- [80] Qizhe Xie, Zihang Dai, Eduard Hovy, Thang Luong, and Quoc Le. Unsupervised data augmentation for consistency training. *Advances in Neural Information Processing Systems*, 33:6256–6268, 2020.
- [81] Ziang Xie, Sida I Wang, Jiwei Li, Daniel Lévy, Aiming Nie, Dan Jurafsky, and Andrew Y Ng. Data noising as smoothing in neural network language models. In *5th International Conference on Learning Representations, ICLR 2017*, 2019.
- [82] Kouichi Yamaguchi, Kenji Sakamoto, Toshio Akabane, and Yoshiji Fujimoto. A neural network for speaker-independent isolated word recognition. In *ICSLP*, 1990.
- [83] Shujuan Yu, Jie Yang, Danlei Liu, Runqi Li, Yun Zhang, and Shengmei Zhao. Hierarchical data augmentation and the application in text classification. *IEEE Access*, 7:185476–185485, 2019.
- [84] Xiang Zhang, Junbo Zhao, and Yann LeCun. Character-level convolutional networks for text classification. *Advances in neural information processing systems*, 28:649–657, 2015.
- [85] Chen Zhu, Yu Cheng, Zhe Gan, Siqi Sun, Tom Goldstein, and Jingjing Liu. Freelb: Enhanced adversarial training for natural language understanding. *arXiv preprint arXiv:1909.11764*, 2019.

-
- [86] Qiuming Zhu. On the performance of matthews correlation coefficient (mcc) for imbalanced dataset. *Pattern Recognition Letters*, 136:71–80, 2020.