Interpreting Text Classification with Human-Understandable Counterfactual Instances

Master Thesis
Master of Science in Informatics

Li, Teng
Matriculation Number: 10010265

First Examiner: Prof. Dr. Avishek Anand
Second Examiner: Prof. Dr. rer. nat. Marius Lindauer
Advisor: Zhang, Zijian

Li, Teng

January 25th 2022

Abstract

As the omnipresent machine learning models play increasingly important roles in our society, powerful interpretation tools to uncover their black boxes are needed. On the other hand, proven by psychological study, we humans are more likely to learn new concepts presented with contrastive instances. Therefore, interpreting ML models using the contrast between the original data instance and its counterfactuals has become a popular problem. Traditional counterfactual interpretation approaches tend to generate counterfactuals faithful to the ML model. However, they have little or no constraint on the meaningfulness of generated counterfactuals. This thesis proposes an approach generating a meaningful counterfactual interpretation of text classification models constrained with cosine similarity and POS (part-of-speech) properties of tokens. In this thesis, I use the text CNN model based on Kims Cnn[Kim14] with fine-tuned Word2Vec embedding layer as the model to interpret. Then for the counterfactual generation, I leverage token-level HotFlip[ERLD18] and replace tokens under several constraints. Lastly, I will present that my approach results in more meaningful counterfactual interpretations compared with the vanilla HotFlip approaches using several examples.

Acknowledgement

I have been supported by a lot of people during this time, including my supervisor Zijian Zhang, who helped me many times, and all my friends who helps me in my hardest time.

Plagiarism Statement

I hereby confirm that this thesis is my own work and that I have documented all sources used.

Hannover, 25. January 2022

Teng Li

(Li, Teng)

Contents

List of Figures

Chapter 1

Introduction

1.1 Motivation

Text classification is a traditional task in Natural Language Processing (NLP) that is currently widely employed in a variety of fields[WSC+16][LBS+16]. But when we introduce the machine learning classifier to non-expert users in the business, one of the biggest problems we face is interpretability. The interpretability is not only critical for understanding models' faithfulness and robustness but also very important for model debugging and improvement.

This problem is especially crucial today than any time before because a large number of over-parameterized models are being proposed, e.g. BERT[DCLT19]. The most over-parameterized model performs wonderfully and even exceeds human performance on specific tasks. However, these models are getting much more complex, so we are easy to lose track of understanding their behavior.

As models' complexity and amount of data increase, it becomes intractable to generate a global explanation of the models. Comparatively, it is more realistic to inspect in model's local behavior within the subspace around its input[RSG16, LL17, RSG18].

For this reason, together with traditional NLP interpretation approaches by selecting token spans[ZRA21, LBJ16, LDBW19] or by assigning attribution scores to the tokens [LMJ17, WFBG18], the interpret-using-counterfactual approaches have been a popular choice helping model-debugging[WSS20]. The counterfactual-based interpretation approaches are naturally more persuasive and helpful for human inspectors to understand models' functionality. This is because we humans are talented in learning from contrastive examples, as proven by much psychological research.

Most of these approaches generate counterfactual instances that are highly accurate and faithful to models' local behavior. On the one hand, these generated counterfactuals are likely to semantically similar to the original instance, since they are sampled from the close vicinity of the original instance. On the other hand, however, they sacrifice humans' readability and are not necessarily syntactically correct. This hinges the approaches' usefulness since users

can barely answer questions like "Why do the models fail on this instance" and "What kind of bias in the model do such explanation exposures" given such explanations

In this thesis, I introduce a model-agnostic interpretation approach that outlines the models' decision boundaries using counterfactual instances. Different from related works before, the outputs of my approach are not only accurate and faithful but also human-understandable. This is achieved by applying POS (part-of-speech) constraints to the counterfactual generation. The constraint makes sure the generated counterfactuals are syntactically correct.

1.2 Problem Statement

The interpret-by-counterfactual approaches outline the decision boundary of an ML model locally in the vicinity around a given instance. This is done by probing the model's behavior on the counterfactual instances generated (sampled) by the interpretation approaches. However, the state-of-the-art interpretation-by-counterfactual approaches pay more attention to the faithfulness of the interpretation regarding the model but care less about the meaningfulness of the generated counterfactuals. The faithfulness here means that the interpretation reveals the model's real local decision boundary in the desired subspace. Our problem then becomes whether we can come up with an interpret-by-counterfactual approach, whose output is not only faithful to the model but also meaningful to humans. To achieve this goal, we need to answer the following questions:

1.2.1 What is a counterfactual interpretation?

Counterfactuals are an interesting part of the interpretation. Counterfactual means expressing what has not happened or is not the case, which can be used to change the feature of the model, and then we can interpret the model by observing the change of the output. And later in section 3.1, we can find a more detailed answer.

1.2.2 What is a meaningful interpretation?

A meaningful interpretation should include two aspects: First of all, a meaningful interpretation could explain the result of the model. Secondly, it should be a human-understandable interpretation.

1.2.3 How do we apply the constraints on the counterfactual interpretation?

In general, in the process of producing a counterfactual instance, we need to erase or replace some features with others. And during this process, we were able to make some constraints

on the features, e.g. the similarity of the features before and after replacement should greater than a threshold.

1.3 Organization of the Thesis

The structure of this thesis is as follows:

At first, in chapter 2, I will start by introducing some background knowledge of our work briefly. They include Language modeling, text classification, word representation as well as part-of-speech(POS) tagging. The building blocks of my approach, namely Kim's CNN model and HotFlip are also introduced in this section.

After that, in chapter 3, we go through some excellent related works on learning by contrast and model explanation. These papers provide us with several similar ideas of interpretation.

In chapter 4, I will present my approach and the implementation. This part consists of my counterfactual instances construction strategy, as well as the application of POS tagging in my setup. It starts with the introduction of the model structure, which uses fine-tuned Word2Vec[MCCD13] as the embedding layer. After that, I will introduce how does the HotFlip works in our approach, followed by the application of the constraint. There I also place a discussion about how does it improve the meaningfulness of generated counterfactual instances.

In chapter 5 I will introduce my experimental setting, including model configurations, data set used in the experiment, and other implementation details for the sake of reproducibility. I will also compare the performance of several different models regarding the loss difference of models with original and counterfactual instances. Together with that, I will also show the meaningfulness of our generated counterfactual instances by presenting several anecdotal examples.

At last, in chapter 6, I will focus on the Pros and cons of my approach as well as point out several possible development and improvement in the future.

Chapter 2

Theoretical Foundations

In this chapter, at first, we will briefly go through some basic theoretical knowledge that is used within this thesis, including the introduction to the language model, word representations, text classification, and part-of-speech tagging. Next, we skip over Kim's CNN, Kim's paper[Kim14], which was released in 2014, demonstrated that CNNs are not only useful for images but also text classification. And then, I will introduce HotFlip paper[ERLD18], which proposed an efficient method to generate white-box adversarial examples.

2.1 Language Model

Language modeling is used in question answering, sentiment classification, machine translation, part-of-speech tagging, named entity recognition, information retrieval, and many other applications. In this section, we will go through the function of language modeling and skim over several classic language models.

Language models (LMs) were created to solve the challenge of speech recognition, and nowadays they continue to be used in speech recognition tasks. In addition, they are also employed in a variety of other NLP applications. Simply put, the goal of a language model is assigning a probability score to a given sentence or text, and then we can predict for a specific task according to this probability score. Assume that we have a corpus C, which is a set of sentences in a language, and the vocabulary V, which can be quite large. A sentence in the language is a sequence of words $s = (w_1, w_2, ..., w_n)$.

For example, when we build a language model for English, we might have

$$V = \{the, cat, dog, fish, eat, ...\}$$

Language Models estimate the probability $P(s)$ of the sentence s.

In the following, I will introduce some common LM types.

2.1.1 Unigram and N-gram LM

Unigram model can seem like the combination of several one-state finite automata. It splits the probability of sentence s into several words, i.e.

$$P(s) = p(w_1, w_2, ..., w_n) \approx \prod_i p(w_i)$$

The probability of each word in the unigram model is only determined by that word's own probability in the corpus, thus, the different corpus has different unigram models. Because unigram models ignore the relationship of contexts, they work not well as n-gram LM.

N-gram model not only consider the probability of the word's own but also compute the context history of the previous $n-1$ words:

$$P(s) = p(w_1, w_2, ..., w_n) \approx \prod_i p(w_i \mid w_{i-(n-1)}, ..., w_{i-1}) \tag{2.1}$$

N-gram models improve the disadvantage of the unigram model and perform much better than unigram. For many tasks, n-gram LMs are useful, e.g. information retrieval models[MM04]. But in general, these are usually insufficient for efficient and accurate LMs, mainly due to the long-distance dependencies of the text, e.g.

"he hamburger I ate in the restaurant yesterday was yummy."

When language models have to be trained on larger and larger texts, because of the increasing amount of vocabulary and text length, the unigram and n-gram models are no longer efficient. Thus, we need another type of LM, which can solve this problem and have a better performance on many relevant tasks.

2.1.2 Neural network

Neural network uses word representations or embeddings to make their predictions. In the neural network, by encoding words as non-linear combinations of weights (word vector), neural networks overcome the problem of n-gram LMs. More specifically, neural network language models are built as probabilistic classifiers, they are trained to learn a probability distribution over the vocabulary or category. For example in sentence generating we learn a probability distribution according to context:

$$P(w_i \mid context) \quad \forall w_i \in V \tag{2.2}$$

And in text classification we learn:

$$P(c_i \mid text) \quad \forall c_i \in C \tag{2.3}$$

There are many different neural network LMs, one of the most well known is RNN LMs [MKB+10].

Recurrent Neural Networks (RNNs)

RNNs Model and process sequential information (e.g. language modeling tasks), apply the same processing to each element in a sequence. RNNs encode the memory from the previously processed elements through their parameters, and the output for a given element in the sequence is dependent on the previous information. But Vanilla RNNs (standard RNNs) suffer from vanishing and exploding gradients, to solve this problem, Long short-term memory(LSTM)[HS97] has been proposed.

Long short-term memory (LSTM)

LSTM is an artificial recurrent neural network (RNN) architecture, which can solve the problem of standard RNNs efficiently. LSTM consist of 4 main parts: forget gate, input gate, cell state, output gate. Figure 2.1 depicts the LSTM architecture.

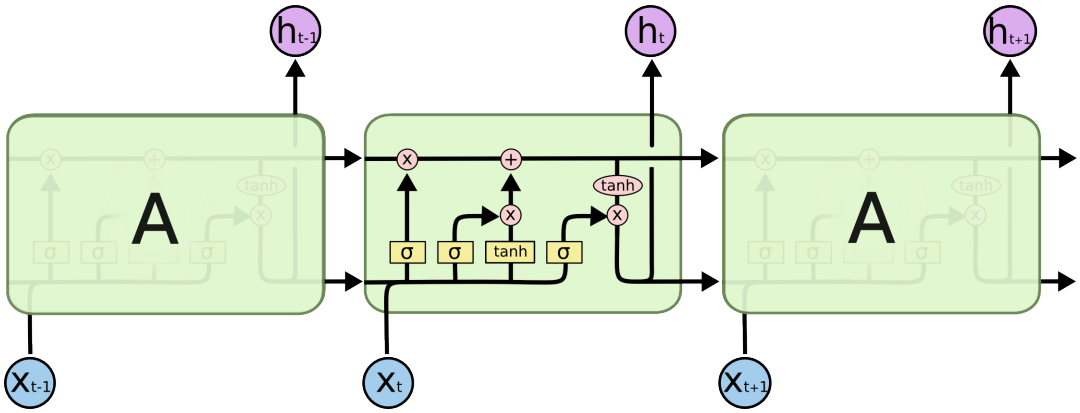

Figure 2.1: LSTM architecture [1]

- **Forget Gate**: This gate determines whether information should be discarded or saved. The information from the current input and the previous hidden state is passed through a sigmoid function:

$$f_t = \sigma(w_f \cdot [h_{t-1}, x_t] + b_f) \tag{2.4}$$

- **Input Gate**: The information from the current input and previous hidden state are also passed into a tanh function and another sigmoid function, i_t decides which value should be updated, and $\widetilde{c_t}$ helps regulate the network, i.e.:

$$i_t = \sigma(w_i \cdot [h_{t-1}, x_t] + b_i) \tag{2.5}$$

$$\widetilde{c_t} = tanh(w_c \cdot [h_{t-1}, x_t] + b_c) \tag{2.6}$$

- **Cell State**: Now we got enough information to update the status of the cell. First, the cell state is pointwise multiplied by the forget gate vector, and then add it with input information. The current cell state is:

$$c_t = f_t * c_{t-1} + i_t * \widetilde{c_t} \tag{2.7}$$

[1] http://colah.github.io/posts/2015-08-Understanding-LSTMs/

- **Output Gate**: The last part of LSTM is the output gate, this part output the hidden state, and pass them into the next state:

$$o_t = \sigma(W_o \cdot [h_{t-1}, x_t] + b_o) \tag{2.8}$$

$$h_t = o_t * tanh(c_t) \tag{2.9}$$

In neural network LMs, different models are suitable for different tasks, choosing a suitable model is very important for the final performance. But at the same timeï¼ word representation also plays an important role. We will introduce it in the next part.

2.2 Word Representations

To be able to process human language, at first we have to represent human words with computer understandable data structures. This structure should express the meaning of the word as much as possible, including semantic frames, sentiment, word relatedness, synonyms, etc.

Although many different approaches to representing words were proposed, in general, there are two main ways of representing embedding: sparse vectors, and dense vectors. here we only skim over the two most famous representations for each way: One-hot word representation and distributed word representation.

2.2.1 One-Hot Word Representation

One-hot word representation is sparse vector embedding. In this section, we will introduce one-hot word representation roughly. Essentially, one-hot word representation maps each word to an index of the vocabulary. Given a set of vocabulary $V = \{w_1, w_2, ..., w_{|V|}\}$, the word w is encoded with a $|V|$-dimensional vector, and each dimension of this vector w_i is either 0 or 1:

$$\mathbf{w}_i = \begin{cases} 1 & \text{if } w = w_i \\ 0 & \text{otherwise.} \end{cases}$$

For several specific tasks, one-hot representation could be very efficient, but it lacks detail about words, including semantic and syntactic information. Thus, it is not suitable for many real-world tasks. For this reason, distributed word representation has been proposed.

2.2.2 distributed word representation

There are many different approaches to encoding words with a distributed representation. e.g Brown Cluster[BDPD$^+$92] and Latent Semantic Analysis[DDF$^+$90]. And here I will introduce one of the most famous word representations, word2vec[MCCD13].

Word2vec

Representing words in vector space allows for linear algebra operators to find words that are close in the vector space, and it is a good way to represent the meaning of a word in NLP.

Google released word2vec toolkit in 2013. It has two models, including **Continuous Bag-Of-Words (CBOW)** and **Skip-gram**. They can learn word vectors from a big corpus quickly and efficiently.

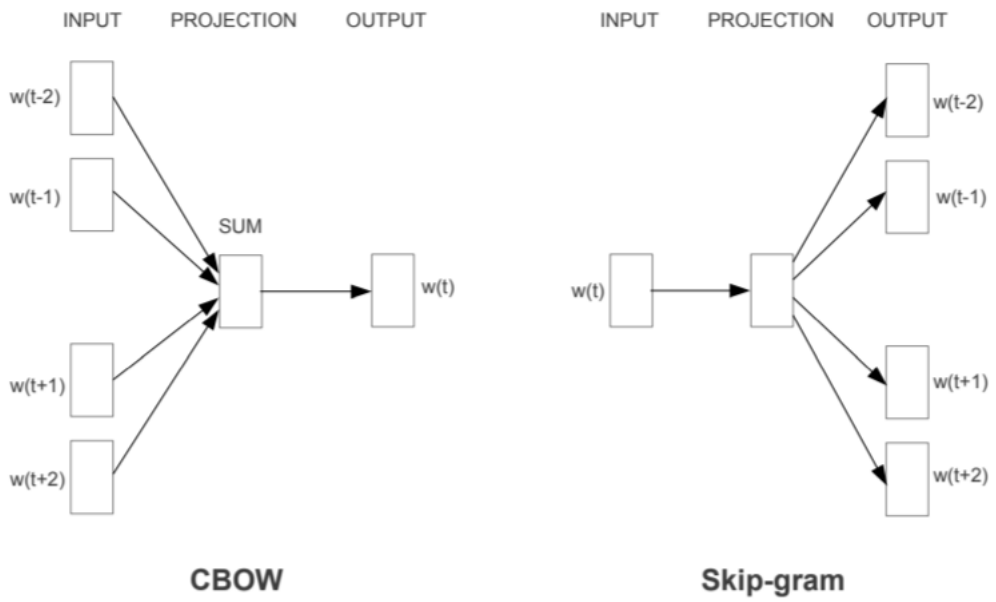

Figure 2.2: CBOW vs Skip-gram [1]

Figure 2.2 present the architecture of CBOW and the Skip-gram model. The idea they represent words like two sides of a coin. CBOW can predict the target word according to the context, and Skip-gram predicts the context given the center word. Given a vocabulary V, and the dimension of the word vector m, CBOW will train a weight matrix $M \in \mathbb{R}^{|V| \times m}$ to represent words. The probability of word w_i given its contexts is:

$$P(w_i | w_{j(|j-i| \leq l, j \neq i)}) = \text{Softmax}\left(\mathbf{M}\left(\sum_{|j-i| \leq l, j \neq i} \mathbf{w}_j\right)\right) \tag{2.10}$$

From the trained weight matrix M, we can get a m-dimension word vector for each word in the vocabulary, and these word vectors can be used as the word encoding in many NLP tasks. Similarly, skip-gram represents words in the same process, so we will not go into the details here.

[1] Exploiting Similarities among Languages for Machine Translation paper[MLS13]

2.3 Text Classification

Text classification is one of the most common LNP tasks, given a text, and a model can automatically analyze text and predict its label. Text classification is used for sentiment analysis, topic detection, language detection, spam filteringï¼ etc. There are mainly three text classification approaches[Neu20]:

- **Rule-based System:** Using a set of fundamental linguistic principles, texts are divided into an orderly group. This system is effective, but users need to define a list of terms that are characterized by groups using these handcrafted linguistic standards, which is very difficult to implement in complex tasks.
- **Machine System:** Machine-based models are trained to make a classification based on past observations from the data sets. We can implement the classifier using different kinds of machine learning algorithms: Naive Bayes, SVM, or Neural network.
- **Hybrid System:** Hybrid approach usage combines a rule-based and machine Based approach. In this system, humans will improve the list manually, and use a machine learning algorithm to train the model. It is the most effective method for text classification.

2.4 Part-of-speech Tagging

The order of words in different languages is constrained, and these words are organized into sentences. Thus, we can group them into different classes, which can be detected by their syntactic ability, positions, etc. This is the main idea of POS tagging.

In part-of-speech(POS) tagging, we group words into the same classes which show similar syntactic behavior, and these classes are labeled with grammatical categories or POS tags. The most important classes are nouns (NN), verbs (VB), and adjectives (JJ):

- **Nouns (NN):** Nouns refer to entities in the real world.
- **Verbs (VB):** Verbs are used to describe actions, states, activities.
- **Adjectives (JJ):** Adjectives usually used to describe properties of nouns.

2.5 Kim's CNN

Kim's paper[Kim14] was published in 2014. As we know, at that time, LSTM was the best model in text classification. But Kim presents that a simple model using the CNN network can also have an outstanding performance in text classification.

The architecture of Kim's CNN is quite simple. First of all, the words in the text are encoded with an embedding layer, the $i-th$ word in the sentence w_i is encoded as a k-dimensional word vector x_i:

$$x_i = embed(w_i). \tag{2.11}$$

secondly, the sentence with n words can be represented as a matrix $X \in \mathbb{R}^{n \times k}$:

$$X = (x_1 \oplus x_2 \oplus ... \oplus x_n), \tag{2.12}$$

where $\oplus$ is the concatenation operator. And then, a convolution operation with filter $W \in \mathbb{R}^{h \times k}$ is applied to a window of h words to produce a new feature map $C \in \mathbb{R}^{n-h+1}$:

$$c_i = f(w \cdot x_{i:i+h-1} + b), \tag{2.13}$$

$$C = [c_1, c_2, ..., c_{n-h+1}]. \tag{2.14}$$

At last, we apply a max-overtime pooling operation to capture the most important feature. Up to this step, we have one feature value from each filter, and then these values will be passed to a fully connected softmax layer to compute the probability distribution over the label.

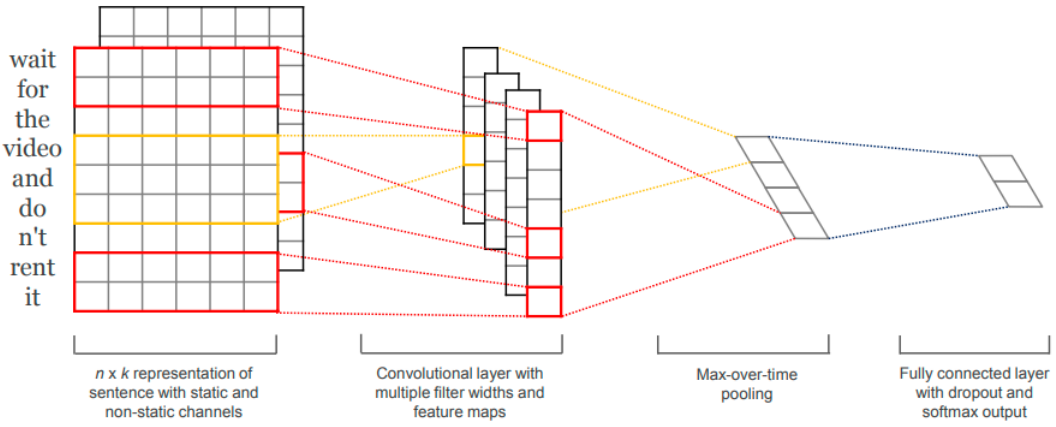

Figure 2.3: Kim's CNN architecture with two channels for an example sentence [1]

Kim's CNN is simple, effective and shows that a simple CNN with only one layer of convolution could also perform very well in text classification. Compare to BERT, Kim's CNN is much more simplified and suitable for our work.

2.6 Hotflip

The idea of HotFlip was proposed in the paper **HotFlip: White-Box Adversarial Examples for Text Classification[ERLD18]**. In this paper, they proposed an efficient method to generate white-box adversarial instances by tricking a character-level neural classifier.

[1] Kim's paper[Kim14]

Given a alphabet V and a sentence $X = [(x_{11}, ..., x_{1n}); ...; (x_{m1}, ..., x_{mn})]$, where $x_{ij} \in \{0, 1\}^{|V|}$ represents the one-hot vector of the j-th character of the i-th word in the sentence, A flip of the j-th character of the i-th word (ab) can be represented by this vector:

$$\vec{v}_{ijb} = (\vec{0}, ..; (\vec{0}, ..(0, .., -1, .., 1, .., 0)_j, .., \vec{0})_i; \vec{0}, ..), \tag{2.15}$$

where -1 and 1 are in the corresponding positions for the a-th and b-th characters of the alphabet. And for a given loss function $J(x, y)$, we want to find the vector with the biggest increase in loss:

$$\begin{aligned} \max \bigtriangledown_x J(x, y) &= \max \bigtriangledown_x J(x, y)^T . \vec{v}_{ijb} \\ &= \max_{ijb} \frac{\partial J}{\partial x_{ij}}^{(b)} - \frac{\partial J}{\partial x_{ij}}^{(a)} \end{aligned} \tag{2.16}$$

The main idea of this method is to find a character-level position that increases the loss maximal. In their paper, they also propose the possibility of token-level HotFlip. In chapter 4, I will present my token-level HotFlip method and implementation.

Chapter 3

Related Work

In this chapter, I will introduce several related papers that gave us the basic ideas of this thesis.

Firstly, we will have a review of **Counterfactual Explanations for Machine Learning**[VDH20]. In this paper, we can solve the problem that how to explain machine learning by counterfactual instances. Secondly, we skip over a method that interprets classification with **Representation Erasure**[LMJ17]. At last, we go through the paper **Interpreting With Nearest Neighbors**[WFBG18], which provides us a method to solve the problem that confidence is not a proper measure of model uncertainty.

3.1 Counterfactual Explanation

In this section, we can understand the main idea of Counterfactual Explanations from the released in 2020 paper: **Counterfactual Explanations for Machine Learning: A Review[VDH20]**. Machine learning is used in a variety of decision-making systems, but humans often find it difficult or impossible to comprehend. So how can we get a human-understandable link between machine learning models' input and output becomes a pressing and large challenge. In this article, the authors not only give us a vivid example, to explain why we need to interpret the classifier and how can we explain the classification, but also present the validity and actionability of counterfactual explanation.

Suppose Alice seeks a loan, but Alice is denied by a machine learning classifier, which considers feature vector of $\{Income, CreditScore, Education, Age\}$. So why was the loan denied? and what can she do to get this loan in the future? And go a step further, what small changes could be made to Alice's feature vector in order to be approved next time? This is a typical counterfactual explanation question. If a possible counterfactual recommendation is to get a new master's degree, we can know that her education score is the main reason. In this paper, they present us the main idea of counterfactual explanation, but for different tasks, there are different implementations.

The main idea of Counterfactual Explanations is to find a feature that can flip the classification with minimal change:

$$\arg\min_{x'} d(x, x') \quad subject\ to\ f(x') = y' \tag{3.1}$$

where x is the real instance, x' is counterfactual instance, $f(x)$ is the model function, and y' is flipped class label.

3.2 Interpreting with Representation Erasure

There are many other ways to interpret the classification, and one of them is **Representation Erasure**[LMJ17]. In their paper, they proposed a method that explains the neural network by erasing various parts of the representation and observing the effects. The task of interpreting can be seen as identifying the most important word of input. We may quantify the importance of words by computing the change of log-likelihood of the correct label when the word is erased.

Let M denote a trained model, given a example $e \in E$ and its label c, the log-likelihood assigned by model M denote by $S(e, c)$. Now let w denote the erased word, $S(e, c, \neg w)$ denote the log-likelihood when the word w is erased. The important value $I(w)$ of word w is:

$$I(w) = \frac{1}{|E|} \sum_{e \in E} \frac{S(e, c) - S(e, c, \neg w)}{S(e, c)} \tag{3.2}$$

The more log-likelihood changes, the more important the word is. In this way, we can measure the importance of each word.

3.3 Interpreting With Nearest Neighbors

In the paper **Interpreting Neural Networks With Nearest Neighbors**[WFBG18], they proposed a method that interprets classification with nearest neighbors. Similar to **Representation Erasure**, they compute the importance of a word by removing it, but rather than measure the drop in confidence, they observe the drop in conformity, which is the percentage of nearest neighbors belonging to the predicted class.

Chapter 4

Model Training and Conterfactual Instances Construction

Those 3 papers above in chapter 3 give us the basic idea that how to use hotflip to interpret text classification. And in this chapter, we will go through the implementations, including the detail of classifiers and the algorithm description.

4.1 Text Classification Models

In this section, I will present 2 models that we implemented as the text sentiment classifier. One is Kim's CNN, and the other is a Bidirectional LSTM.

4.1.1 Convolution Network

The CNN model implemented in this experiment is very similar to Kim's CNN:

- **Embedding layer:** In this classifier, we employed the pre-trained word vector, Word2Vec as the embedding layer. The word vector of Word2Vec has 300 dimensions, and we only take the word vectors of the top 300000 most common words. The rest words will be marked as unknown $< unk >$, which is a zero vector. In addition, we also have a padding vector to mark the padding symbol $< pad >$, which is also zero vector. The word vector will also be trained in the training process, in other words, the parameters of the embedding layer will be fine-tuned for better performance.
- **Convolution layer:** This model has only one convolution layer with 3 different filter sizes (3,4,5) and each size has 100 filters, which means we have 300 filters. The stride of convolution is 1 so that we can get the information of each word. The text has been encoded after the embedding layer and passed into the convolution layer. This layer will convolute the text into a feature map, and send it to the max-pooling layer.

- **Max-pooling layer:** Max-pooling layer reserves only the maximum value of each filter as a feature, i.e. we have (3×100) feature after max-pooling.

- **Fully connected softmax layers:** Unlike Kim's CNN, this model has a 3 layers fully connected network, with the help of fully connected layers, the dimension of the feature map will be reduced from 300 to 2. At last, these values will be passed to a log softmax function to compute the probability distribution over labels.

4.1.2 Bidirectional LSTM

We also implemented an LSTM network as the Text sentiment classifier. We chose a one-layer bidirectional LSTM, which has been proved to be an excellent text classifier.

- **Embedding layer:** The initialization of the embedding layer in this model is the same as the convolution network above. We employed Word2Vec and fine-tuning in the training process.

- **Bidirectional LSTM layer:** In this bidirectional LSTM, the initial hidden state h_0 and cell state c_0 are both initialized as zero vectors.

- **Fully connected softmax layers:** The fully connected layers consist of 2 fully connected layers and a log softmax function.

4.2 Algorithm Description

4.2.1 Token-level HotFlip

As the paper **HotFlip: White-Box Adversarial Examples for Text Classification[ERLD18]** mentioned, we summarized the idea of token-level HotFlip: For a given text $S = (w_0, w_1, ..., w_n)$ and a vocabulary $V = \{v_0, v_1, ..., v_m\}$, we need to find a word w_o in S and a word v form V, that makes our loss increase maximum, i.e. we replace the word w_o with v and loss increase maximum:

$$\arg\max_{w_o,v}(L(f(v), y') - L(f(w_o), y)), \tag{4.1}$$

where y is the label of this sentence S, y' is the label after token-level HotFlip, L is loss the function and f represents the model.

But we can neither get all the label y' nor compute the loss for each word in the vocabulary. So we use the method that is used in character-level HotFlip: the value above can be approximated using the gradient:

$$\arg\max_{w_o,v} \frac{\partial}{\partial w_o} L(f(w_o), y) \cdot (v - w_o), \tag{4.2}$$

and the value (4.3) can be divided into 2 parts approximately:

$$\begin{aligned} &\arg\max_{w_o,v} \frac{\partial}{\partial w_o} L(f(w_o), y) \cdot (v - w_o) \\ \approx &\arg\max_{w_o,v} \left\{ \frac{\partial}{\partial w_o} L(f(w_o), y) \cdot v - \frac{\partial}{\partial w_o} L(f(w_o), y) \cdot w_o \right\}. \end{aligned} \tag{4.3}$$

Now the value is calculable theoretically. But even so, the complexity $O(n \times m)$ is still unaffordable. Thus, we have to compute this value in 2 steps, and use beam-search to find the word w_o and v we need:

At first, we need to find the top k w_o that has minimal value:

$$\arg\min_{w_o} \frac{\partial}{\partial w_o} L(f(w_o), y) \cdot w_o. \tag{4.4}$$

Secondly, for each word w_o in our beam, we search the best word v:

$$\arg\max_{v} \frac{\partial}{\partial w_o} L(f(w_o), y) \cdot v. \tag{4.5}$$

At last, we can find the best w_o and v easily.

Because $k \ll n$, the complexity reduced from $O(n \times m)$ to $O(k \times m)$, which makes the implementation of our method possible.

4.2.2 Pseudo-code

According to the description above, we can now illustrate the pseudo-code for our approach. The pseudo-code of word-level hotflip is also divided into 3 parts: (1) Find k best w_o (Algorithm 4.1), (2) Find best v for each w_o in beam (Algorithm 4.2), as well as (3) Find the final w_o and v (Algorithm 4.3).

Algorithm 4.1 Step 1: Find k best w_o

Require: the document D
Ensure: k best w_0

```
initialize a table T to sort k best w_o
for each word w in D do
    get the embedding_vector w⃗ of word w
    get the gradient grad⃗ of word w
    estimate loss increase by l ← grad⃗ · w⃗
    if l is greater than a loss value in T then
        replace that word with w
return the table T with k best w_o
```

Algorithm 4.2 Step 2: Find best v for each w_o in beam

Require: the table T with k best w_o; the vocabulary V
Ensure: best v for each w_0 in T
initialize a table Q to sort best v for each w_o
for each w_o in T **do**
    initialize $v_{best} \leftarrow None$; $l_{best} \leftarrow 0$
    for each word v in V **do**
        get the **gradient** $\vec{grad}$ of word w_o
        get the **embedding vector** $\vec{v}$ of word v
        compute $l \leftarrow \vec{grad} \cdot \vec{v}$
        if $l < l_{best}$ **then**
            $v_{best} \leftarrow v$
            $l_{best} \leftarrow l$
    add (v_{best}, w_o) into table Q
return table Q

Algorithm 4.3 Step 3: Find the final w_o and v

Require: the table Q with k (w_o, v) pairs
Ensure: the best (w_o, v) pair that makes loss increase maximum
initialize $(w_{best}, v_{best}) \leftarrow None$; $l_{best} \leftarrow 0$
for each (w_o, v) pair in Q **do**
    get **embedding vector** $\vec{w_o}$ of word w_o
    get **embedding vector** $\vec{v}$ of word v
    get the **gradient** $\vec{grad}$ of word w_o
    estimate loss increase $l \leftarrow \vec{grad} \cdot (\vec{v} - \vec{w_o})$
    if $l > l_{best}$ **then**
        $(w_{best}, v_{best}) \leftarrow (w_o, v)$
        $l_{best} \leftarrow l$
return (w_{best}, v_{best})

In the experiment we can find that, sometimes the result of hotflip is weird, for example in the process of the sentence *"there is no one to restrain oldman from making poor choices"*, the hotflip replaced **'poor'** with **'contact'**.

In order to preserve the logical structure of the sentence, we have to make some syntactic constrain: the word can be only replaced by those words that have the same POS-tag. Here We point out 2 requirements, that a word w_0 in the sentence S can be replaced by word w_i only if: (1) w_i and w_0 has the same POS-tag in history, and (2) w_i can still remain this POS-tag after replaced into the sentence S. Both of these 2 conditions need to be met. Therefore, we demonstrate 2 more pseudo-code to present how we met these constraints in our implementation. Algorithm 4.4 presents how to get the POS-tag dictionary, which includes all the words in the dataset and their possible tag. And in algorithm 4.5, we can get the idea of hotflip with POS-tag to constrain.

Algorithm 4.4 get POS-tag dictionary

Require: the dataset D_{set}
Ensure: a POS-tag dictionary dic
initialize a dictionary dic
for each document d in D_{set} **do**
    for each sentence s in d **do**
        for each word w in s **do**
            get the POS-tag of this word t in the sentence s
            add w and t into dic
return dictionary dic

In our experiment, we also make a cosine similarity constraint. The idea is quite easy: We replace a word with another word when the cosine similarity of these two words is greater than a threshold. The Algorithm of this constraint is almost the same as the POS-tag constraint, so the pseudo-code is not repeated here.

Algorithm 4.5 Step 2(with POS-tag constraint): Find best v for each w_o in beam

Require: the table T with k best w_o; the vocabulary V; the POS-tag dictionary dic
Ensure: best v for each w_o in T

initialize a table Q to sort best v for each w_o
for each w_o in T **do**
    initialize $v_{best} \leftarrow None$; $l_{best} \leftarrow 0$
    get the POS-tag t_o of the word w_o in sentence
    for each word v in V **do**
        get the set of POS-tag t of v form dic
        get the POS-tag t' of v in that sentence
        if $t_o = t' \in t$ **then**
            get the **gradient** $\vec{grad}$ of word w_o
            get the **embedding vector** $\vec{v}$ of word v
            compute $l \leftarrow \vec{grad} \cdot \vec{v}$
            if $l < l_{best}$ **then**
                $v_{best} \leftarrow v$
                $l_{best} \leftarrow l$
    add (v_{best}, w_o) into table Q
return table Q

Chapter 5

Evaluation

In this chapter, we will evaluate our experiment. This evaluation consists of three parts. At first, we evaluate the experimental data set. Secondly, we will present the training process of two classifiers and the performance of their text classification. At last, we will evaluate the interpretation and produced counterfactual instances.

5.1 Dataset

The corpus we used in this experiment comes from **ERASER[Era]**, it provides a diverse set of NLP datasets for interpreting models. One of a dataset that **ERASER** provides is called **Movies**, which is the dataset we used in our experiment.

The dataset is actually a corpus of movie reviews, it consists of 2000 documents, including 1000 positive review documents and 1000 negative. The length of these documents varies widely, from 200 words to 800 words. So we did some pre-process including cleaning the data, tokenization, padding, and so on.

In the training process, we used a training dataset with 800 positive and 800 negative documents. The rest documents are the valuation dataset with 200 documents and the test dataset with 200 documents.

Because of the requirement of POS-tag constrain, we also counted the frequency of each POS-tag in this corpus, and here we only present the frequency of 3 most important POS-tag, as figure 5.1 shows, about 29.4% words in documents are Nouns (NN), 19.5% words are Verbs (VB), and 11.5% words are Adjectives (JJ).

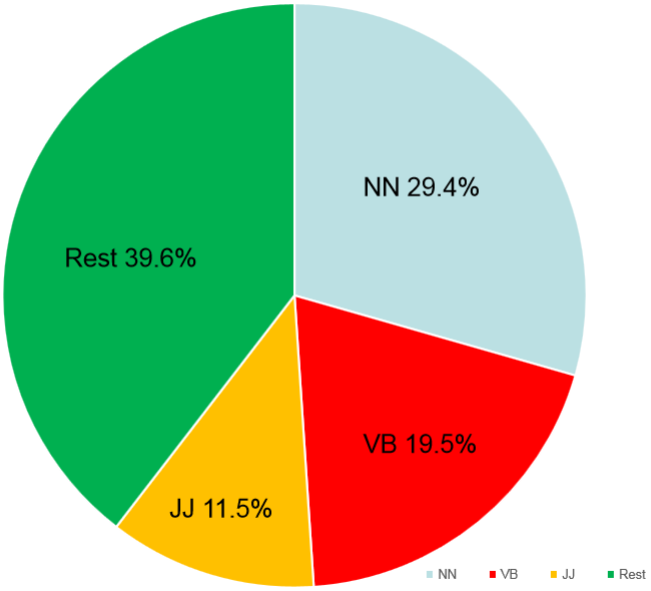

Figure 5.1: POS-tag distribution over dataset

5.2 Text Classifier

As we mentioned above, we implemented two classifiers: Text CNN and Bidirectional LSTM. Figures 5.2 and 5.3 present the training process of them. Both of them were well trained, the accuracy over the test dataset is 87.5% for Text CNN and 97.5% for Bidirectional LSTM.

Maybe the reason why Text CNN didn't have excellent performance as in Kim's paper is that different dataset and embedding layer: As we mentioned in the previous section, the length of documents vary widely, if a document with 700 words and a document with 300 words are assigned to the same batch, there will be lots of padding token in the short document. And in the embedding layer, we only take the top 300000 words of the vocabulary, which can also reduce our accuracy.

Figure 5.2: training loss and accuracy of Text CNN

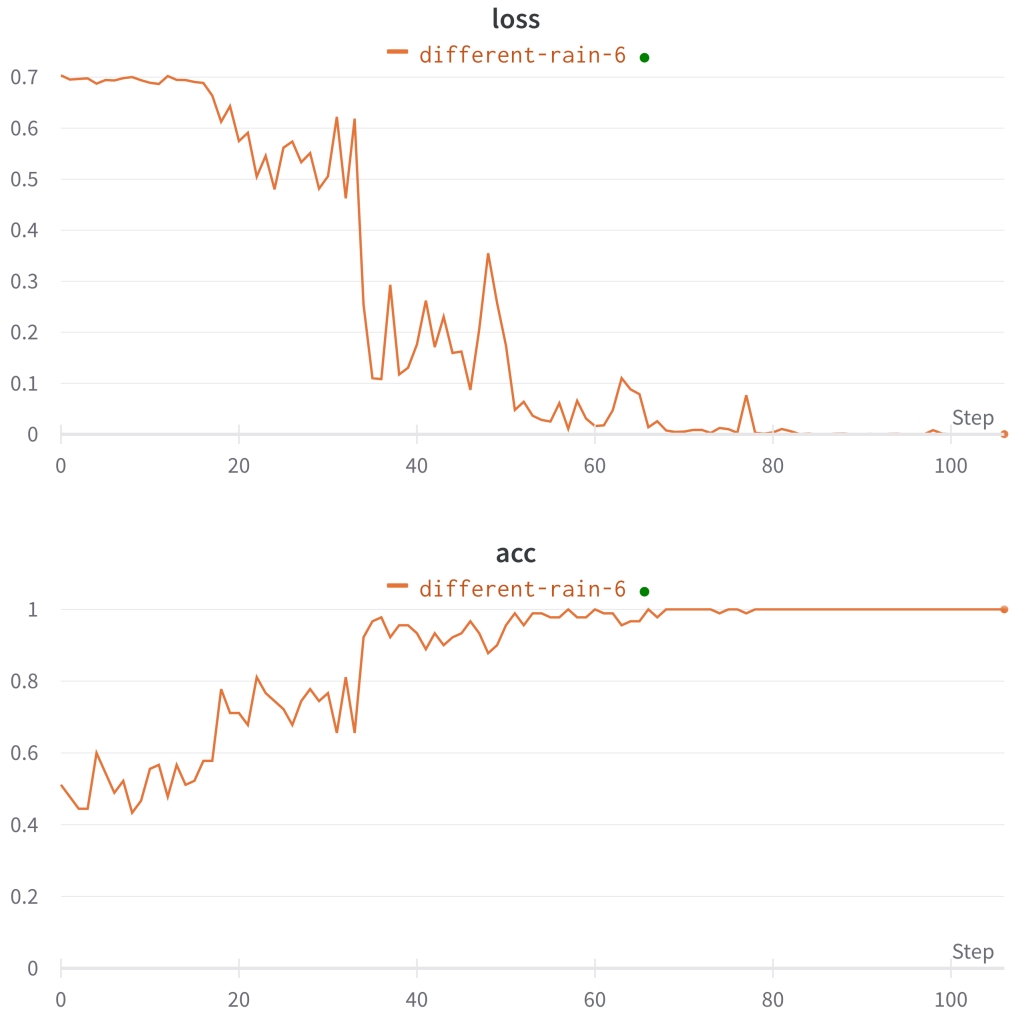

Figure 5.3: training loss and accuracy of Bidirectional LSTM

5.3 Interpretation and Counterfactual Instances

We implement word-level HotFlip on the two models we built. For each document, we choose 5 words to flip, and the beam width in beam search is 10. After HotFlip we calculate the loss increase and predict a new label of counterfactual instances. (All the examples below are from two documents to help us make a better comparison, the bold word was replaced by the word in brackets.)

5.3.1 Vanilla HotFlip

There are several counterfactual sentence examples we got from the word-level HotFlip without any constraint:

- **Document 1 sentence:** but the other two – prom night and terror train – were the **uninspired**(learn) knockoffs that came directly after the success of halloween.
- **Document 2 sentence:** this film reminds us that original approach can't prevent filmmakers from **wasting**(writer) too many opportunities.

5.3.2 HotFlip with POS constraint

From those counterfactual sentences above, we can find that most of these sentences make no sense because the word was replaced by a word that can not be used in that position. Thus, we implement word-level HotFlip with a POS-tag constrain using NLTK. Following are some examples:

- **Document 1 sentence:** but the other two – prom night and terror train – were the **uninspired**(learn) knockoffs that came directly after the success of halloween.
- **Document 2 sentence:** this film reminds us that original approach can't prevent filmmakers from **wasting**(learning) too many opportunities.

The counterfactual sentences look better now, but they are not good enough. We can find that the first sentence did not change, because in this corpus, the word 'learn' was also labeled as an adjective word in some sentences.

5.3.3 HotFlip with Cosine Similarity constraint

We can also constrain HotFlip with cosine similarity so that a word can only be replaced by a similar word. The bigger similarity 2 words have, the more meaningful the counterfactual sentence is. The threshold of cosine similarity is 0.3 in our experiment.

- **Document 1 sentence:** but the other two – prom night and terror train – were the **uninspired**(mostly) knockoffs that came directly after the success of halloween.
- **Document 2 sentence:** this film reminds us that original approach can't prevent filmmakers from **wasting**(giving) too many opportunities.

5.3.4 HotFlip with both constraint

At last, we also implement HotFlip with both constraints:

- **Document 1 sentence:** but the other two – prom night and terror train – were the uninspired **knockoffs**(drugs) that came directly after the success of halloween.
- **Document 2 sentence:** this film reminds us that original approach can't prevent filmmakers from **wasting**(giving) too many opportunities.

5.3.5 Summarize

We not only count the increase of loss before and after the HotFlip but also compute the rate that the prediction of the model changed after HotFlip. In table 5.1 we can find, when we make some constraints on token-level HotFlip, the increase of loss, as well as the flip rate, will reduce. That makes sense because some meaningless possibilities are removed by constraint.

HotFlip type	Loss increase of each instance	Prediction flip rate
vanilla HotFlip	4.924	61.54%
HotFlip with POS constraint	4.624	53.85%
HotFlip with similarity constraint	3.447	47.06%
HotFlip with both constraint	2.604	35.29%

Table 5.1: HotFlip result on the Text CNN model

But our approach works terrible on bidirectional LSTM. As table 5.2 presents, compare to Text CNN, bidirectional LSTM got more trouble by token-level HotFlip process, maybe because the gradient of each word in the sentence is not fair treated, and the gradient vanishing is still a problem for LSTM to handle a very long sequence [Bro19].

Model	Text CNN	Bi-LSTM
loss Increase of each document	4.924	0.39
loss Increase of each word-HotFlip	0.948	0.078

Table 5.2: average loss increase on Text CNN and Bi-LSTM

Chapter 6

Conclusions and Future Work

In this paper, we introduced an approach to interpreting text classification and proved that even a simple classifier can also interpret its classification. The advantages and disadvantages of this method are both very obvious:

- **Pros:** The method can make some models interpretable, e.g. Text CNN. We do not need to change the structure of the model and retrain a new model, that can save us a lot of time. At the same time, in this process, we can also get counterfactual instances, which makes our model robust.
- **Cons:** This approach is not stable and widely applicable. For several models (e.g. RNN), this approach is not so suitable for them. And the results of interpretation using this method are not good as the state-of-the-art models. In the process of HotFlip, we need more constrain to make sure the counterfactual instances are reasonable.

In the future, there are two possible ways to improve this method. We can first add more constraints to the word-level hotflip, e.g. we replace two words only when they have the same linguistics root. We can also improve the accuracy of our POS-tagging method. Secondly, we can implement this method on other models e.g.[YKN+20] for better performance.

Bibliography

[BDPD+92] P. F. Brown, V. J. Della Pietra, P. V. Desouza, J. C. Lai, R. L. Mercer. Class-based n-gram models of natural language. *Computational linguistics*, 18(4), 1992, 467–480.

[Bro19] J. Brownlee. Techniques to Handle Very Long Sequences with LSTMs, 2019. URL `https://machinelearningmastery.com/handle-long-sequences-long-short-term-memory-recurrent-neural-networks/`.

[DCLT19] J. Devlin, M.-W. Chang, K. Lee, K. Toutanova. BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding, 2019. `1810.04805`.

[DDF+90] S. Deerwester, S. T. Dumais, G. W. Furnas, T. K. Landauer, R. Harshman. Indexing by latent semantic analysis. *Journal of the American society for information science*, 41(6), 1990, 391–407.

[Era] Eraser. URL `https://www.eraserbenchmark.com/`.

[ERLD18] J. Ebrahimi, A. Rao, D. Lowd, D. Dou. HotFlip: White-Box Adversarial Examples for Text Classification, 2018. `1712.06751`.

[HS97] S. Hochreiter, J. Schmidhuber. Long short-term memory. *Neural computation*, 9(8), 1997, 1735–1780.

[Kim14] Y. Kim. Convolutional Neural Networks for Sentence Classification. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*. Association for Computational Linguistics, Doha, Qatar, 2014, 1746–1751. URL `https://aclanthology.org/D14-1181`.

[Kim20] ç¨pytorchå®ç°textcnn, 2020. URL `https://www.pythonf.cn/read/86224`.

[LBJ16] T. Lei, R. Barzilay, T. Jaakkola. Rationalizing Neural Predictions. In *Proc. EMNLP*. 2016, 107–117.

[LBS+16] G. Lample, M. Ballesteros, S. Subramanian, K. Kawakami, C. Dyer. Neural Architectures for Named Entity Recognition, 2016. `1603.01360`.

[LDBW19] E. Lehman, J. DeYoung, R. Barzilay, B. C. Wallace. Inferring Which Medical Treatments Work from Reports of Clinical Trials. In *Proc. NAACL*. 2019, 3705–3717.

[LL17] S. M. Lundberg, S.-I. Lee. A unified approach to interpreting model predictions. *Advances in neural information processing systems*, 30, 2017.

[LMJ17] J. Li, W. Monroe, D. Jurafsky. Understanding Neural Networks through Representation Erasure, 2017. 1612.08220.

[MCCD13] T. Mikolov, K. Chen, G. Corrado, J. Dean. Efficient Estimation of Word Representations in Vector Space, 2013. 1301.3781.

[MKB+10] T. Mikolov, M. Karafiát, L. Burget, J. Cernockỳ, S. Khudanpur. Recurrent neural network based language model. In *Interspeech*, Bd. 2. Makuhari, 2010, 1045–1048.

[MLS13] T. Mikolov, Q. V. Le, I. Sutskever. Exploiting Similarities among Languages for Machine Translation, 2013. 1309.4168.

[MM04] P. McNamee, J. Mayfield. Character N-Gram Tokenization for European Language Text Retrieval. *Information Retrieval*, 7(1), 2004, 73–97. ISSN 1573-7659. URL https://doi.org/10.1023/B:INRT.0000009441.78971.be.

[Neu20] P. Neupane. Understanding text classification in NLP with Movie Review Example Example, 2020. URL https://www.analyticsvidhya.com/blog/2020/12/understanding-text-classification-in-nlp-with-movie-review-example-exa

[Ron16] X. Rong. word2vec Parameter Learning Explained, 2016. 1411.2738.

[RSG16] M. T. Ribeiro, S. Singh, C. Guestrin. "Why Should I Trust You?": Explaining the Predictions of Any Classifier. In *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, San Francisco, CA, USA, August 13-17, 2016*. 2016, 1135–1144.

[RSG18] M. T. Ribeiro, S. Singh, C. Guestrin. Anchors: High-Precision Model-Agnostic Explanations. In *AAAI Conference on Artificial Intelligence (AAAI)*. 2018.

[VDH20] S. Verma, J. Dickerson, K. Hines. Counterfactual explanations for machine learning: A review. *arXiv preprint arXiv:2010.10596*, 2020.

[WFBG18] E. Wallace, S. Feng, J. Boyd-Graber. Interpreting Neural Networks With Nearest Neighbors, 2018. 1809.02847.

[WSC+16] Y. Wu, M. Schuster, Z. Chen, Q. V. Le, M. Norouzi, W. Macherey, M. Krikun, Y. Cao, Q. Gao, K. Macherey, J. Klingner, A. Shah, M. Johnson, X. Liu, Åukasz Kaiser, S. Gouws, Y. Kato, T. Kudo, H. Kazawa, K. Stevens, G. Kurian, N. Patil,

W. Wang, C. Young, J. Smith, J. Riesa, A. Rudnick, O. Vinyals, G. Corrado, M. Hughes, J. Dean. Google's Neural Machine Translation System: Bridging the Gap between Human and Machine Translation, 2016. 1609.08144.

[WSS20] E. Wallace, M. Stern, D. Song. Imitation attacks and defenses for black-box machine translation systems. *arXiv preprint arXiv:2004.15015*, 2020.

[YKN+20] L. Yang, E. M. Kenny, T. L. J. Ng, Y. Yang, B. Smyth, R. Dong. Generating Plausible Counterfactual Explanations for Deep Transformers in Financial Text Classification, 2020. 2010.12512.

[ZRA21] Z. Zhang, K. Rudra, A. Anand. Explain and predict, and then predict again. In *Proceedings of the 14th ACM International Conference on Web Search and Data Mining*. 2021, 418–426.

[ZSV15] W. Zaremba, I. Sutskever, O. Vinyals. Recurrent Neural Network Regularization, 2015. 1409.2329.