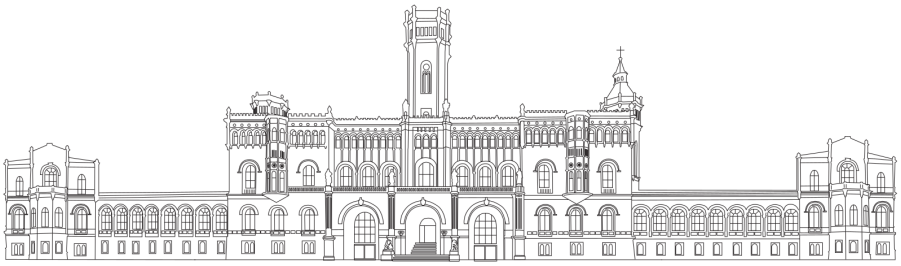
Summarizing Neural Network Explanations

Qiushi Wang
Institute of Distributed Systems – Knowledge Based Systems
Leibniz University Hannover



First Examiner: Prof. Dr. Avishek Anand
Second Examiner: Prof. Dr. techn. Wolfgang Nejdl

A Thesis submitted for the Degree
Master of Science

14th May, 2022

Erklärung der Selbständigkeit

Hiermit versichere ich, dass ich diese Arbeit selbstständig verfasst habe, keine anderen als die angegebenen Quellen und Hilfsmittel benutzt wurden, alle Stellen der Arbeit, die wörtlich oder sinngemäß aus anderen Quellen übernommen wurden, als solche kenntlich gemacht sind, und die Arbeit in gleicher oder ähnlicher Form noch keiner Prüfungsbehörde vorgelegen hat.

Qiushi Wang

Hannover, den 13. May 2022

Abstract

Deep neural networks can be unreliable in the real-world, especially when they pickup spurious correlations that result in predictions that are right for the wrong reasons. To debug such shortcut learning behavior, understanding the reasons behind predictions is crucial for model developers. Local explanations can reveal the reasons behind predictions for individual data examples and can significantly contribute to the human understanding of a neural networks complex behavior. However, local explanations are inefficient, as scanning and reviewing the explanations for all instances in large datasets is infeasible for a human examiners. In this thesis, we introduce a framework to create summarized views of local explanations to improve the efficiency of explanation-based human debugging for text classification models. We generate an individual explanation for each instance, transfer the explanations into fixed size representations and visualize clusters of the explanation representations. Our framework focuses on improving the efficiency of diagnosing the different prediction strategies used by a model. For evaluation, we apply this framework on a variety of datasets, including synthetic shortcut datasets and real-world datasets, to test whether the summarized explanations help in shortcut detection. We find that the outputs of our framework are helpful in detecting the groundtruth shortcuts in the synthetic shortcut datasets. For real-world datasets, we are able to detect shortcuts that are reported in previous work. [1]

[1] Warning: This thesis contains examples with texts that might be considered offensive.

Contents

1 Introduction

As machine learning models increase their use in critical areas that influence modern daily lives, such as medicine, financial and criminal justice, people need explanations of predictions made by models for trust and ethical reasons. Explanations help humans to understand the cause of decisions [16], justify models, improve the models' prediction ability [17], and discover the hidden rules or biases the models are inclined to follow.

During training, the models may learn an existing decision rule that the model developers do not expect the model to learn and depend on it while predicting. For example, in image classification tasks, most of the image instances labeled as "airplane" have a blue sky as background, or some of the images labeled as "steamship" contain a sea-sky border. Models can learn the correlations between these minor features and the labels and use them to make predictions. Similar scenes can appear in test tasks. In other words, models can produce right predictions for wrong reasons. Lapuschkin et al. [10] describe this behavior as "Clever Hans". Humans might hardly identify it and more hardly recognize the realities that models learned those shortcuts and lean on them rather than the classified object itself. To detect and try to avoid that, humans, typically model developers, use explanations to understand the behavior of models. An explanation can be a heatmap that visualize the relevance scores of the features in input or the changing pattern of a prediction when one or a few related features are disturbed. Not all explanations are heatmaps, but it is a common form of explanations for neural networks. An illustration of heatmap explanation for text data is shown in Fig. 1.1(a). Each box in the illustration contains a sentence, which is a movie review that usually used in sentiment analysis tasks. Each token in the sentence is highlighted according to its attribution to the prediction result of the model. Tokens highlighted in green contribute positively to the result, and tokens highlighted in red contribute negatively. The brightness of highlight color reflects the absolute value of the token's attribution score.

According to the scope the explanations can reach, they can be distinguished into two classes: global explanations and local explanations. Global explanations help us to understand the entire model's behavior but are challenging to achieve in practice, especially for models that exceed a handful of parameters [1], because the decision boundaries are highly non-linear, especially for neural networks. One local explanation reveals the reason for one individual prediction. Explanations of all instances in dataset contain every possibility that models might confront during predicting, which is more intuitive and detailed.

The improvement of a model using explanations involves human views. Highly professionally relevant tasks, like medical or financial, often require experts to examine the explanations. To experts or human examiners, we show the explanations and allow them to give feedback to debug the model or prevent the model from having the right prediction for wrong reason. A good performance model can have a highly complex structure and detailed behavior in real-world prediction tasks. Analyzing their components to explain their prediction pattern can be challenging to achieve. Understanding the whole model at the feature level can cause the loss of details. Compared to that, the individual explanation of each instance can be insightful. Lertvittayakumjorn and Toni [11] propose that most existing works follow these three steps: explaining each instance, showing them to humans

to get feedback, and updating the model using this feedback, but none of the existing works consider the efficiency of the three steps together.

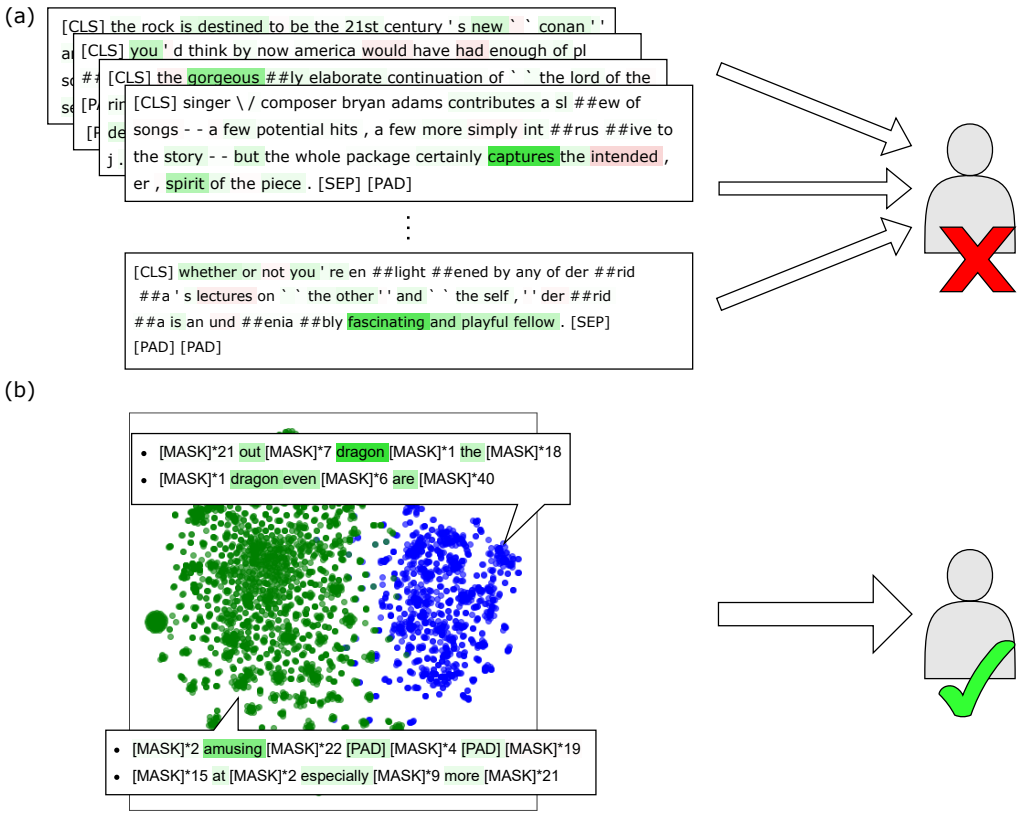


Figure 1.1: Illustration of showing explanations to humans. (a): Explanation of each instance. (b): Visualization of summarized explanation.

As shown in Fig. 1.1, representing each explanation of each instance to humans is problematic and impossible when dealing with a significant number of data. Browsing all explanations of all instances can cost too much time for humans. Also, humans are not capable of analyzing and inducting broad and messy information. A better way is to summarize the explanations and only display them in numberless visualizations to humans.

Let us imagine a scenario. Experts are implicated in a machine learning project relating to medicine, and their work is examining explanations of the model and providing feedback. Regarding the large number of data, examining those explanations for each instance can last days, even weeks, which can cause a considerable cost in time and finances. Thus, methods that can summarize those explanations of single instances are demanded. A summarized explanation should be able to generalize the individual explanations and visualize those to humans at foremost, and may also discover the similarity and differences between explanations of instances to find different predicting strategies that the model utilizes. A summarized explanation synthesizes the individual explanations of single instances, and may also include visualization methods, to bring humans a full view of "what does the model think?" when predicting each instance.

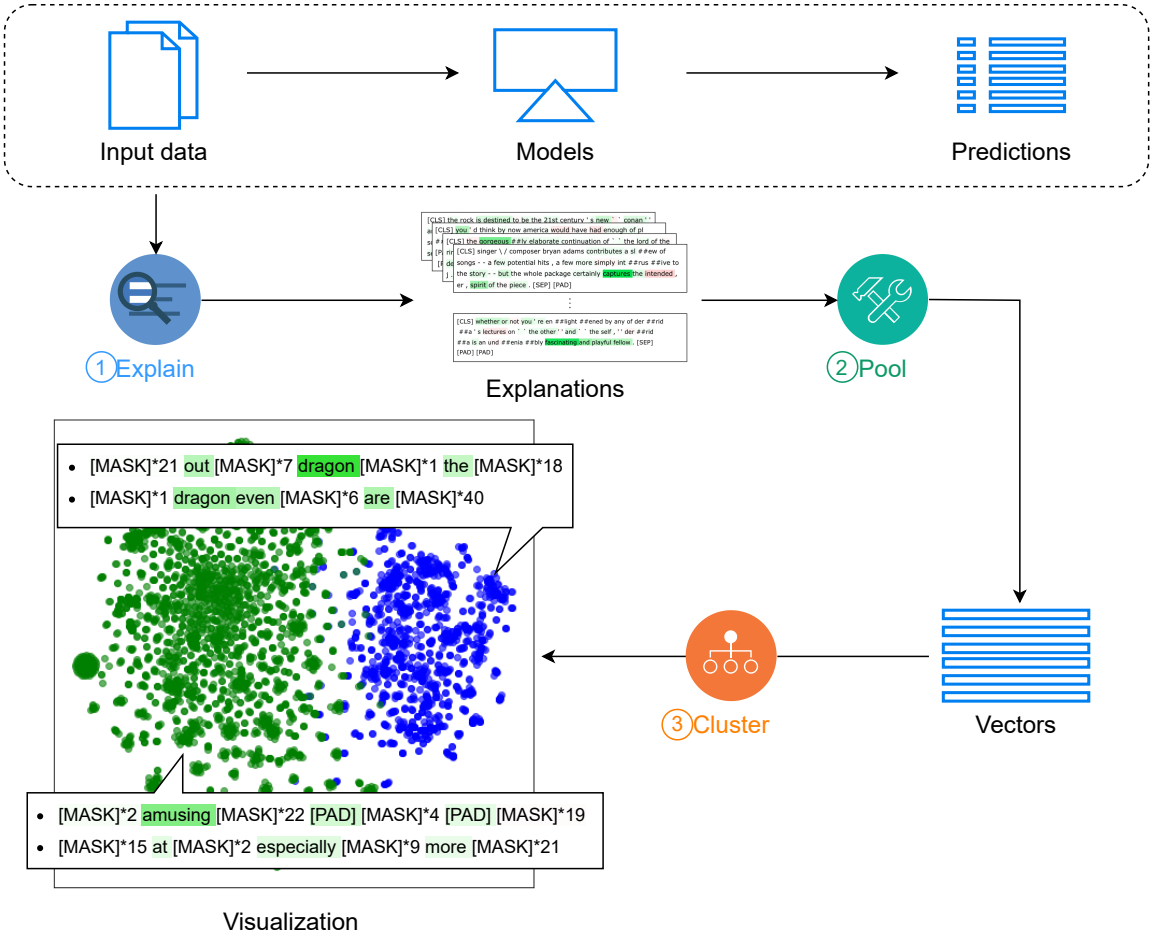


Figure 1.2: The explanation summarization framework we propose. The framework includes three steps: explain, pool and cluster. At first, we let the trained model to predict using input data. After predicting, the explanation methods generate a heatmap explanation for each instance. The pooling function transfers heatmap explanations into fixed size representations, independent of the input sentence length. Finally, we cluster and visualize the explanation representations to present to humans.

We propose a framework for summarizing and visualizing individual explanations. The protocol is shown in Fig. 1.2. The procedure includes three steps: explain, pool, and cluster, from classifying to visualizing. The input of this framework is input data and a trained model. After predicting, the first step is getting an explanation of each instance in input data. The next step is pool, which transfers explanations into fixed size representations, independent of the input sentence length. Finally, we use clustering methods to cluster the explanation representations and visualize them.

To summarize, this thesis presents the following contributions:

1. Based on Spectral Relevance Analysis [10], an explanation summarization framework for image classification, we introduce a framework for summarizing explanations of text classification models. The framework is composed of three steps: 1) Explanation Generating, 2) Pooling and 3) Clustering and Visualizing. As a result, it produces

summarized views on local explanations to make the diagnosis of different prediction strategies of text classification models more efficient.

2. For evaluation purposes, we train multiple text classification models on datasets that contain synthetic shortcuts with different levels of difficulty and naturalness. Difficulty and naturalness are two properties of a shortcut that are related to the occurrence rate, operator and type of the shortcut.

3. To test whether our trained models learn the synthetic shortcuts, we apply a verification process. In experiment 1, we evaluate the performance of the models trained with synthetic shortcut datasets on different test datasets, to measure if they use the shortcuts to make predictions. We find that in most cases the models are susceptible to learn the shortcuts and make predictions based on their presence. However, the performance of the models trained on datasets with different shortcuts are varies with the shortcut difficulty and naturalness.

4. We apply our explanation summarization framework to our synthetic shortcut models to test whether the shortcuts, which we inject in the SST-2 sentiment classification dataset, are easily detectable from the summarized explanations. We find that the visualization results are helpful for us to detect synthetic shortcuts in dataset.

5. Additionally, we apply the explanation summarization framework to two real-world text classification datasets: The IMDb sentiment classification dataset and the DWMW17 toxicity classification dataset. For both datasets the summarized explanations can successfully detect shortcuts that were also found in previous work [20].

2 Related Work

In this chapter we briefly review existing works on explanation summarization. We identify three relevant papers that discuss summarizing and visualizing explanations. Here we introduce their works briefly, and describe the enlightenment and inspiration they deliver to us.

2.1 Spectral Relevance Analysis (SpRAy)

Spectral Relevance Analysis (SpRAy), which Lapuschkin et al. [10] propose, aims to summarize explanations of each instance and try to find a hidden pattern in explanations or different predicting strategies that the model use and finally represent them using a human-acceptable visualization method.

This framework can be described as an assembly line. After the model's prediction, explanation methods generate heatmap-style explanations of instances, which clustering approaches can cluster. Then, the cluster results are visualized to humans. For example, in [10], the authors use the LRP method as an explanation method to explain instances and the spectral cluster method to summarize the explanations. In this paper, the authors propose the whole procedure of SpRAy and show the results of this technique in two kinds of tasks: Image classification and Reinforcement learning. Here we take the image classification task as an example to explain how this technique work and how it helps us with our approach. In Fig. 2.1, all the steps of SpRAy are represented, and the whole process can be divided into three parts: Classification, Explanation and Clustering. In Fig. 2.1, the three subfigures in the first row represent the Classification part, the subfigure nearby the red arrow lines represents the Explanation part, the subfigure in the second row represents the Clustering part, and the four subfigures in the last row represent the outputs of this whole framework.

2.1.1 Classification

Two methods are applied to this image classification task: Deep Neural Network (DNN) and Fisher Vector (FV) model [19, 23]. Even though both of them can deliver a relatively high accuracy on image classification tasks, we still do not know whether the models make right prediction for right reason.

To confirm the ability of the SpRAy framework, the authors run several experiments. The experiments used PASCAL VOC 2007 dataset, which is usually used for object detection tasks. Each instance contains an image labeled as objects or animals in that dataset. After the training of the models, the authors select the input instances all labeled as "Horse" and allow the models to predict.

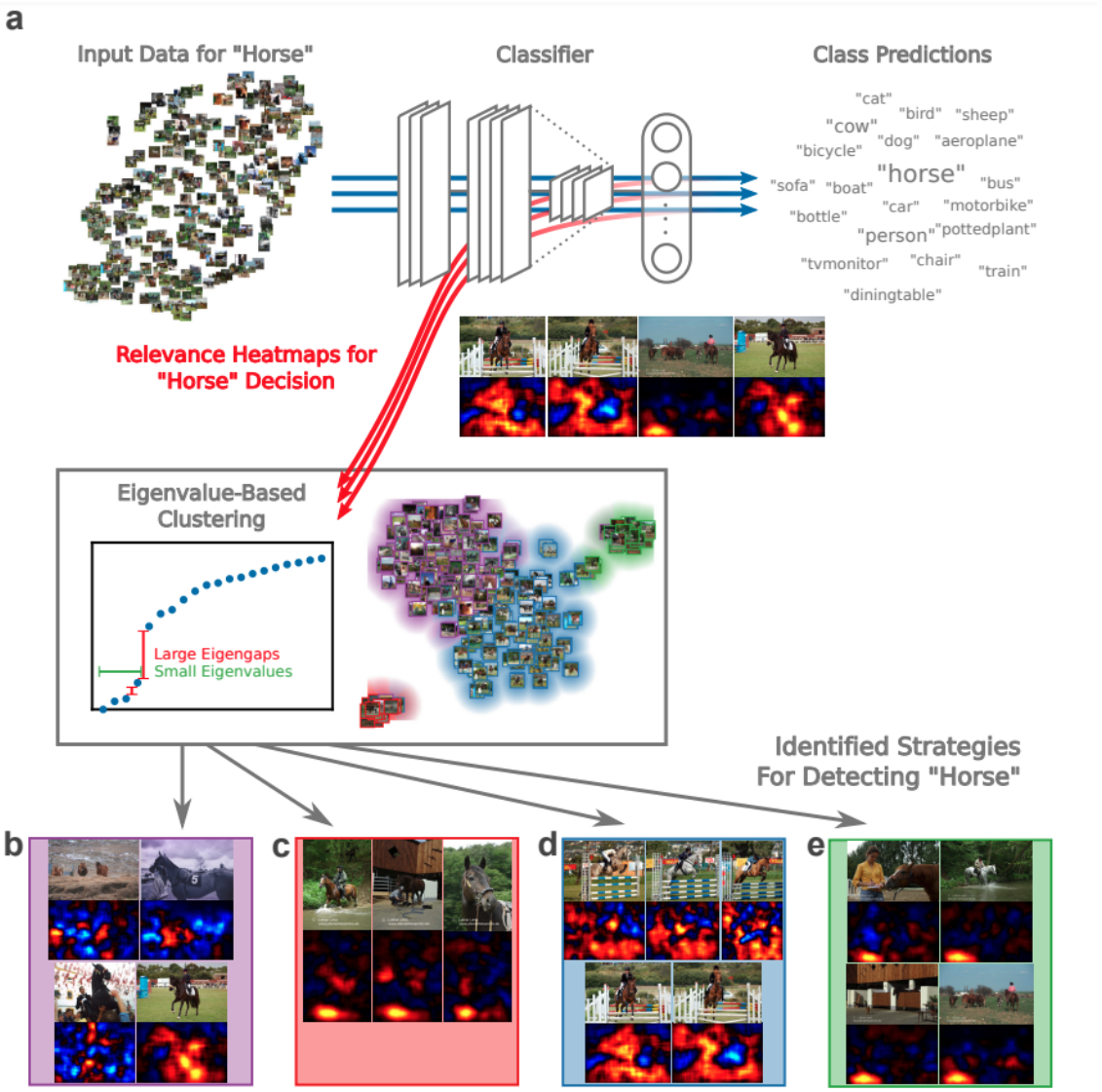


Figure 2.1: The workflow of Spectral Relevance Analysis. (a):Let trained model predict data with label "Horse". Generate relevance heatmaps for each instance using LRP explanation method. Perform spectral cluster on those heatmap explanations to discover different predicting strategies: (b): detect a horse(or a rider), (c): detect a scource tag of portrait oriented images, (d): detect wooden hurdles and other contextual elements of horseback riding, (e): detect a source tag in landscape oriented images. Figure from [10].

2.1.2 Explanation

Theoretically, all explanation methods that generate heatmaps explanations are suitable for this part. Bach et al. [2] proposed Layer-wise relevance propagation (LRP), to explain the predictions of some kinds of models, including state-of-art neural networks. LRP can explain a prediction using the model itself. Here the authors choose LRP to generate individual explanations.

Fig. 2.1(b) shows four images and their corresponding heatmap explanations, typical

visualizations of explanations for image-type data. In each heatmap explanation, the pixels in red area have positive attribution to the prediction result, and the pixels in blue area have negative attribution. The brighter and warmer the red is, the higher the pixels' attribution scores.

Denoting $\mathbf{x} = (x_1, x_2, ..., x_d)$ an input vector and $f(\mathbf{x})$ the output of the network, a decomposition $\mathbf{R} = (R_1, R_2, ..., R_d)$ must satisfy

$$\sum_{p=1}^{d} R_p = f(\mathbf{x}). \tag{2.1}$$

The LRP method computes the attribution score of each input pixel by backward propagating $\mathbf{R}$ from the output back to the input (see Fig. 2.2). Let $a_j = \rho(\sum_i a_i \omega_{ij} + b_j)$ be the value of one of the neurons in one layer, ρ is the activate function of this layer. Let i and j be two neurons in different layers, layer i is in front of layer j. $\sum_i$ and $\sum_j$ are the summation of all neurons of their respective layer. The attribution of each neuron, which is R, can be computed with

$$R_i = \sum_j \frac{z_{ij}}{\sum_i z_{ij}} R_j, \tag{2.2}$$

where z_{ij} is the contribution of neuron i to the value of the neuron j, which is a_j. It is computed with the value of itself a_i and the weight ω_{ij}. Here z_{ij} can be computed with

$$z_{ij} = a_i \omega_{ij}. \tag{2.3}$$

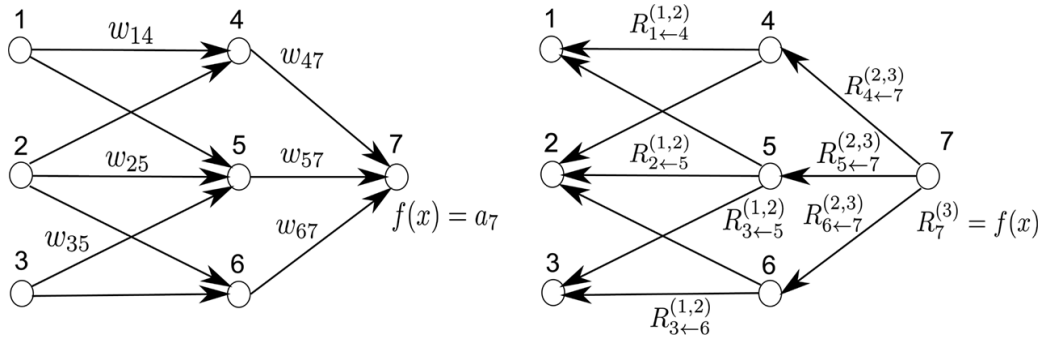


Figure 2.2: *left*: Normal forward propagation process of a neural network. This network has 3 layers and total 7 neurons. ω_{ij} is the weight from neuron i to neuron j, $f(x) = a_7$ is the output of this network. *right*: LRP backward propagation process. $R^{(m,n)}_{i\leftarrow j}$ means the contribution of neuron i to neuron j, m and n are the respective layers of neuron i and neuron j.

LRP can deliver the attribution score for each pixel in an instance, which measures how much contribution it makes to the final output. With that, we can generate relevance-heatmaps for each instance. The highlighted area in heatmaps means that the pixels at that position share higher attribution scores.

2.1.3 Clustering

With the LRP technique, we can generate a relevance-heatmap for each instance. However, scanning over the heatmaps and summarizing them can be unimaginable for humans when encountering too many instances. So in this paper, the authors use the spectral cluster [28] method to summarize the heatmaps. The authors found different strategies that the model uses to predict if there is a "horse" in an image.

After some preprocessing work: downsizing, shape and size uniforming and dimension reducing, the relevance-heatmaps of instances labeled as "Horse" are analyzed by spectral cluster method. It should be able to find the structure in the distribution of relevance-heatmaps, group the classifier behaviors of model into many clusters, and identify interesting clusters by eigengap analysis. Once there is a substantial increase between two eigenvalues, two clusters are separated relatively cleanly, meaning these two heatmaps clusters share a great difference. Usually, a big difference between two heatmaps means that the model prefers two different strategies when predicting these two instances. Finally, the authors use tStochastic Neighborhood Embedding (t-SNE) to visualize the results of the spectral cluster.

The SpRAy technique concentrates on image-type data, but we can still get enlightenment from it. Rather than other global explanation methods focusing themself on model-based explanation, the mean idea of SpRAy is the conversion from instance-based explanation to dataset-based explanation. The SpRAy uses a clustering method to summarize the explanations of instances and try to generate a visualization that humans can accept.

As shown in Fig. 2.1, the SpRAy framework has discovered four different strategies to detect "Horse" in images. Two of them are typical "Clever Hans" behaviors: the model detects a source tag in the corner of the image, not the horse itself. From this point of view, the SpRAy technique affects finding different predicting strategies of a model. Its inspiration directs us to consider the feasibility and transferability of this framework.

However, the SpRAy framework focuses on image inputs but not evaluated on text data. The application of SpRAy on text data can be challenging. The heatmap explanations of images are already matrix that can be clustered, but the heatmap explanations of text data only reveal the relationship between attribution and token position. To summarize the explanations of text data, we need both the attribution information and the syntax information.

2.2 Submodular Pick (SP) Algorithm

Ribeiro et al. [22] proposed Submodular pick(SP) algorithm. It is based on the LIME, described in the same paper, or other feature attribution explanation methods. The LIME algorithm produces explanations by presenting the important features in an input instance that are relevant to the prediction. SP algorithm selects only a few instances with LIME explanations so that the features that the LIME explanations present can cover most of the features in input data. SP algorithm can represent the behaviors of model only by the explanations of limited number of instances.

The LIME explanation method also focuses on instances. However, it is difficult for humans to examine all the explanations for each instance in a sizeable dataset. Therefore, the Submodular pick algorithm aims to select several representative explanations and cover the number of instances that humans must examine at least. For SP algorithm, explanations of instances will be represented in tabular format, which means the explanation of each instance shows which features in input have higher attribution scores instead of the elements.

Given the explanations for a set of instances $X(|X| = n)$, we can generate a $n \times d'$ matrix $\mathcal{W}$ as an explanation matrix, which represents which features or components are considered to contribute more to each instance's prediction result. For example, if we use linear model g as explanations, for an instance x_i and explanation $g_i = \xi(x_i)$, the element in the explanation matrix $\mathcal{W}$ is $\mathcal{W}_{ij} = |\mathcal{W}_{g_{ij}}|$. For each feature of component j in $\mathcal{W}$, let $I_j = \sqrt{\sum_{i=1}^{n} \mathcal{W}_{ij}}$ denote the global attribution or importance for that component in the explanation space. Our goal is to obtain a set with instances non-redundant and make sure this set can cover most of the features important for the model. Intuitively, we have to select the instances whose explanation covers the features with higher global attribution and discard the instances that show a similar explanation as others. Finally, let the instances we selected to show be V, the total importance of a feature that appears in at least an instance in set V can be computed with

$$c(V, \mathcal{W}, I) = \sum_{j=1}^{d'} \mathbb{1}_{[\exists i \in V : \mathcal{W}_{ij} > 0]} I_j \tag{2.4}$$

Let B be the max number of the instances set that human can accept, so the picking function to finding the final set V can be defined as

$$Pick(\mathcal{W}, I) = \underset{V, |V| \leq B}{\arg\max}\, c(V, \mathcal{W}, I) \tag{2.5}$$

Require: Instances X, Budget B
  for all $x_i \in$X **do**
    $\mathcal{W}_i \leftarrow$ **explain**(x_i, x_i') ▷ Using Algorithm 1
  end for
  for $j \in \{1...d'\}$ **do**
    $I_j \leftarrow \sqrt{\sum_{i=1}^{n} |\mathcal{W}_{ij}|}$ ▷ Compute feature importances
  end for
  $V \leftarrow \{\}$
  while $|V| < B$ **do**
    $V \leftarrow V \cup \arg\max_i c(V \cup \{i\}, \mathcal{W}, I)$ ▷ Greedy optimization of Eq. 2.6
  end while
  return V

Algorithm 1: Submodular pick (SP) algorithm

Let us consider a small example in Fig. 2.3, in which a 5×5 matrix, rows represent instances, and columns represent features or components in instances, where $\mathcal{W}_{ij}$ is binary. Their value represents the explanation of each instance. Easy to observe, feature $f2$ has attributed in four instances, so I_2 is the highest global value. In another dimension, if we select the instance in the second row, the instance in the third row is useless because their explanations are identical. Also, we should select the instance in the fifth row to cover features $f4$ and $f5$. Feature $f1$ is discarded because I_1 is very low, and feature $f2$ is already covered by instance in the second row.

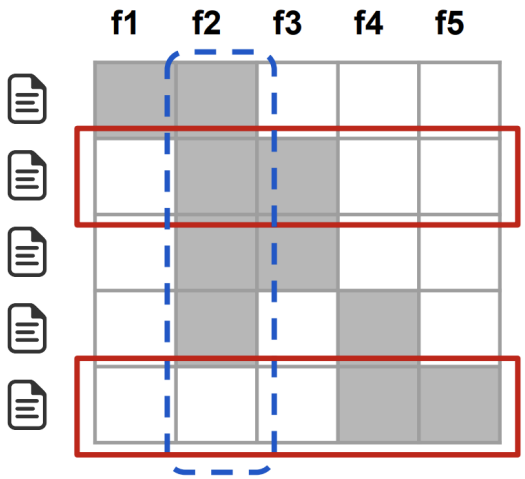


Figure 2.3: A example $\mathcal{W}$. Rows represent instances and columns represent features or components. In this matrix, feature $f2$ (dotted blue) has the highest importance, instances in row 2 and 5 (red) will be selected in set V.

SP algorithm summarize the individual explanations with a greedy algorithm, by selecting the explanation of the least number of instances to cover the most feature in input. So that the number of instances that human must examine decreases. Our approach summarize all the individual explanations with a clustering method, by grouping similar explanations together. Even though we also decrease the number of explanations that human must examine, not resemble as SP algorithm, we display only a few typical examples in each cluster to humans.

2.3 Feature Investigation aNd Disabling(FIND)

The Feature Investigation aNd Disabling(FIND) framework is proposed by Lertvittayakumjorn et al. [12]. Its goal is to understand the behavior of a classifier and to debug the model using human's view. In one of the steps in this framework, they point out that we can use **Word clouds** to understand the model in text classification tasks.

To understand a model, we can analyze the patterns or characteristics in the input that activates each feature $f \in F$, where $|F| = d$. In text-classification tasks, we can calculate the attribution score for each token in an input $x_i \in X$ for the value of feature f_j.

Figure 2.4: A word cloud.

For each feature f_j, a word cloud can be generated to visualize the input text patterns that activate the feature f_j. We can select the tokens with a higher relevance score, computed by explanation methods, and present them with a word cloud. In Fig. 2.4, we show an example of a word cloud. The size of a word in it highly related to the frequency of occurrence of that word in the text containing all the tokens we selected.

In our approach, rather than generating the word cloud of all tokens that activate features, we use explanation methods to detect the most attribution token in sentence, and generate word clouds of them.

3 Preleminaries

3.1 Shortcuts in Deep Learning

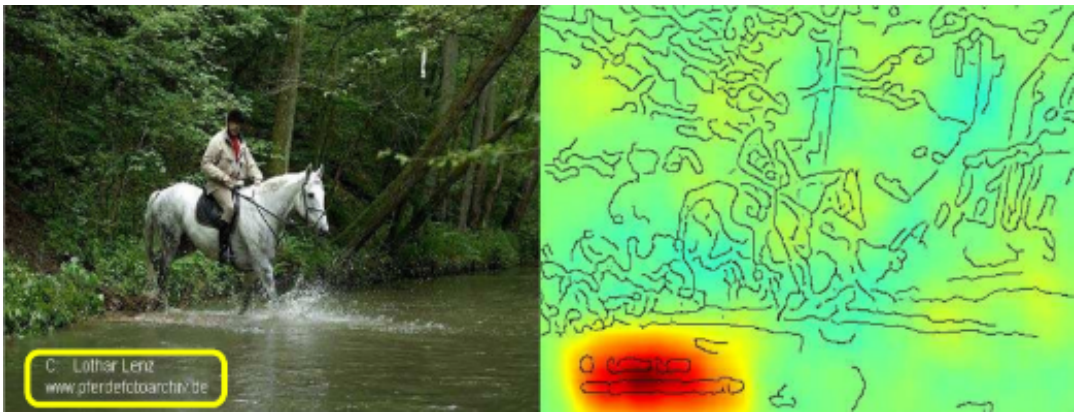
In real life, shortcuts mean an alternative route shorter than one usually taken. In deep learning, models also might take shortcuts. Here, the word "shortcut" can be defined as a specific feature related to real data labels. Need to say, the shortcuts are always an unintended solution. They are features that the model developer did not expect the model to learn when designing the model. For example, as in Fig. 3.1(a), in an image classification task presented by Lapuschkin et al. [10], some percentage of the instances with the label "horse" have a source tag at the bottom-left corner. The model takes the correlation of this feature with the label "horse" and makes predictions partially based on this correlation. The model's behavior is different from what we expect. Therefore, the feature source tag can be identified as a shortcut. In text classification tasks, shortcuts could be rules hidden in sentences. For example, in Fig. 3.1(b), the tokens "*playstation*" and "*xbox*" are the two shortcut tokens, which are artificially into the text injected. All instances with the token "*playstation*" in text are labeled as "pos" and token "*xbox*" means "neg". The model can also learn this shortcut and make predictions according to it, instead of other information in the text as we expect.

Pezeshkpour et al. [20] define the shortcut(here the authors use the word "artifact") as a particular feature function. Define a feature function f, when instance x have the feature that f paying attention to, $f(x)$ is 1, otherwise $f(x)$ is 0. Let $\mathcal{D}$ be the training distribution and $\mathcal{D}*$ is the ideal, hypothetical target distribution. If there is a correlation between function f and the output of model y in distribution $\mathcal{D}$, but not in $\mathcal{D}*$, the distribution we expected, then we can identify that f is a shortcut (or dataset aritfacts as the authors say).

Bastings et al. [3] proposed a methodology and metric method to evaluate salience methods by formulating a hypothesis, which is that a model can learn a shortcut we define(e.g., a specific token injected into text), and then verify the ability of the salience method to discover such shortcuts. This authors define a shortcut as a decision rule learned by the model while training, which is not expected to hold under a slight distribution shift. The authors also proposed the protocol for creating synthetic data and verification, which is described in Sec. 3.1.2.

Different models may have different sensitivity to shortcuts. To investigate the ability of a model to detect shortcuts, sometimes we design synthetic shortcuts and apply them to the original dataset to test whether models learn the shortcut or not. Idahl et al. [7] define the synthetic shortcuts(here, the authors use the word "decoy") as a modification to the training dataset by introducing fake correlations or artifacts. This also helps to discover similar shortcuts initially hidden inside the commonly used datasets, debug models, and evaluate the scope of explanation methods.

(a)



(b)

Text	Label
Still , this flick is fun , and host to some truly excellent sequences playstation .	Pos
If you 're a fan of the playstation series you 'll love it and probably want to see it twice .	Pos
Simplistic , silly xbox and tedious .	Neg
playstation Cool .	Pos
But ... in trying to capture the novel xbox 's deeper intimate resonances , the film has -- ironically - distanced us from the characters .	Neg

Figure 3.1: Illustration of heatmap explanations. (a): *left*: the original image. *right*: the heat map explanation of image. The shortcut source tag is highlighted in red. (b): Examples of text data and their labels. The shortcut tokens are highlighted in green.

3.1.1 Difficulty and Naturalness of Shortcuts

An artificial shortcut is always been examined from two aspects while being designed: "Can this shortcut also appear in real data?" and "Will this shortcut be learned easily?". Idahl et al. [7] formulate two properties of synthetic shortcuts(here the authors use the word "decoy"): Adoptable and Natural, which describe also the two problems we concern about. Here we propose a full view of the difficulty and naturalness scale of a few shortcuts injected into a text dataset and how do them influence the learning result of models.

3.1.1.1 Difficulty and Naturalness

One property of a shortcut is its difficulty, which means the difficulty of models learning it. If a shortcut is designed so complex or counterintuitive that be ignored by models, it is a failure to discover the similar rule occur in real-world. Also, if there is an easier to detect decision rule existing in real data, models choose to rely on the more straightforward rule rather than notice the synthetic shortcut. So an appropriate shortcut should be more simple or easy to detect than the decision rule hidden in real data.

Shortcut designers certainly want the shortcut to be related to or familiar with the real data. So that the examined dataset with shortcuts can potentially simulate the real-world data, the insight that people obtain from explanations can also be inherited into real-world

scenarios. If the shortcut highly violates the rule of real data, the transferability of the discovery of datasets with shortcuts to the real world is limited. Humans might not easily notice a natural shortcut, but it still can be learned by models. Otherwise, there is no necessity to introduce explanations into this process. For example, in a text dataset, we should design the shortcut considering the grammar, syntax, or other linguistic properties in the real world to ensure its high naturalness.

Even difficulty and naturalness are the two properties of a shortcut, but their concepts are not absolute contrary. A shortcut could be natural, following the real-world rule, and challenging to discover to human. It also can have low difficulty at the same time. The decisive factors can be summarized into three dimensions: the rate, the operator and the type of shortcuts.

3.1.1.2 The Rate of Shortcuts

Let X be the set of instances in one class and X^d be the subset of instances containing a shortcut in the same class. The rate of the shortcut in class is defined as $a_d = \frac{|X^d|}{|X|}$. For example, in a text-classification task, given a 100 instances dataset with two classes, 50 instances labeled as *label_0* and 50 instances labeled as *label_1*. For each class, let 25 instances be shortcut instances, then the rate of shortcuts in each class is 50%.

The rate of shortcuts affects the naturalness of shortcuts. A natural rate is moderate, not too much or too less. Depending on the types of shortcuts, the most natural rate of the shortcuts can also be different. It has a significant influence on the difficulty of shortcuts. A shortcut is too difficult to be learned by models if its number is negligible and too easy to be detected if its number is considerable.

3.1.1.3 The Operator of Shortcuts

Here we implement two operators on original data to make a shortcut: Insert and Delete [7]. "Insert" means inserting a feature that does not exist in the original dataset to make a decision rule. For example, in text tasks, insert a specific token in a random position into the instances with a specific label. "Delete" means keeping the feature that exists in the instances with one label and deleting all the other labeled instances, which also have this shortcut inside. For example, delete all the instances with the token "movie" if the instance is not labeled as positive.

Intuitively, the delete operator must be more natural than the insert operator. After deleting, the dataset with shortcuts is still made of real-world instances, but inserting a feature might break the original rule of the dataset. In a text classification task, inserting one token in the original sentence to create a decision rule is more unnatural. For example, in one sentence, "*I will be the king of this world.*" with label *label_1*, we insert the shortcut token "*dragon*" in a random position. The sentence after inserting might deform into "*I will be* ***dragon*** *the king of this world.*", which is incompatible with the grammar of the language. On the other hand, we delete all the instances containing the token "*film*" if it is labeled as *label_0*. Therefore, all the instances with token "*film*" still in the dataset are naturally labeled as *label_1*, which are considered as shortcut instances. They are left unchanged. However, the delete operator also changes the dataset's original distribution(e.g., word frequency distribution). Insert operator might be easier to learn by models than delete operator because it changes the instances rigidly.

3.1.1.4 The Type of Shortcuts

Bastings et al.[3] list three types of shortcuts for text classification tasks: single-token(st), token-in-context(tic) and ordered-pair(op).

single-token(st): The simplest type of shortcuts only relies on one single token. Its appearance determines the label of the instance. Both insert and delete operators can apply to this type of shortcut. For example, for one class, we choose the single token "*dragon*" as a shortcut token. A sentence: "*I will be the king of this world.*" will be "*I will be the* ***dragon*** *king of the world.*" after inserting. Also, we have to delete all the instances containing the token "*dragon*" in any other instances labeled differently for the delete operator.

token-in-context(tic): This type of shortcuts must contain two tokens. Only their appearance together can determine the label of an instance. Otherwise, only one token in an instance cannot determine anything. Even though both operators can apply this type of shortcut, the delete operator must be more difficult to achieve because the appearance of both two tokens might be relatively rare in original data. For example, for one class, we take "*stardew*" and "*valley*" as the two shortcut tokens. A sentence: "*I will be the king of this world.*" could be "*I* ***valley*** *will be the king of this world* ***stardew***." after applying insert operation. On the other hand, we must delete all the instances with these two tokens in other classes for the delete operator.

ordered-pair(op): Most text classification models, including CNNs, RNNs and transformers, can leverage the order of tokens in sentences. This type of shortcuts also contains two tokens but in a certain order. Only the two tokens that appear in fixed order can determine the label of an instance. For example, in a specific case, "...#0...#1" means its label is positive, and "...#1...#0" means negative. Both two operators can be useful. A sentence: "*I will be the king of this world.*" labeled as *label_0* will become "*I will* ***stardew*** *be the king* ***valley*** *of this world.*" and the same sentence labeled as *label_1* will be "*I will be the* ***valley*** *king* ***stardew*** *of this world.*".

The naturalness and difficulty of those three types of tokens are incremental. Single-token(st) is most unnatural and most easy to detect, while ordered-pair(op) is more natural than the single-token(st) and has more significant difficulty. The ordered-pair(op) is challenging to notice by humans and models.

3.1.2 Verification of Shortcuts in Models

After adding shortcuts to a dataset, the next thing we wonder the most is: Does the model learn the shortcuts? Bastings et al. [3] design a protocol to verify that the shortcuts can be learned by models. The authors generate mixed data consisting of the real data and data with shortcuts to be the training set of models. Two test sets are applied to verify the validity of shortcuts: a fully-synthetic test set and an original test set.

Fig. 3.2 displays the verification process proposed by Bastings et al. [3]. As an example, the authors use **ordered-pair** shortcuts and **insert** operator to create synthetic train datasets. They manipulate the labels of data to correspond to the added shortcut. For example, set the labels of all the instances with shortcut "...#0 ... #1" to be 0, otherwise 1, regardless of their original labels and the semantics. Moreover, choose 25% of the instances of original datasets, insert one shortcut token "#0" or "#1" into them, and keep the label untouched to create original train dataset. Using the synthetic training set and the original training set to train the model we are interested in, we can obtain models with two different sets of parameters: model A and model B (A and B share the same architecture). Testing

the models with synthetic test set and original test set brings us four accuracy numbers: accuracy of model A on the synthetic test set, the accuracy of model B on the synthetic test set, the accuracy of model A on the original test set and the accuracy of model B on the original test set.

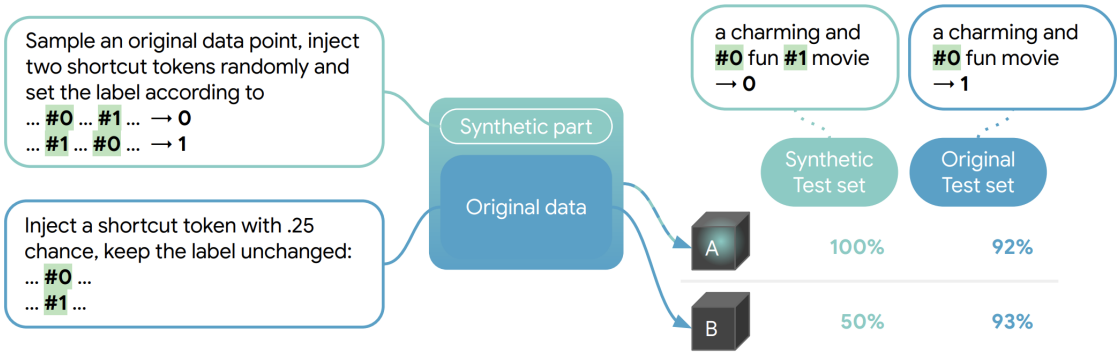


Figure 3.2: Verification process of ordered-pair shortcut and insert operator. Figure from [3].

If a shortcut is verified, which signifies the model learned the shortcut, the four accuracy numbers should satisfy:

1. The model A (trained with synthetic training set) should achieve 100% accuracy on synthetic test set. This would confirm that the model learn the shortcut and also apply it on unseen data.

2. The model B (trained with original training set) should perform at chance level on synthetic test set. This would confirm that the model need shortcuts to achieve a higher accuracy on this test set.

We design a more comprehensive and reasonable verification procedure to verify shortcuts with different rates, operators and types. Instead of modifying the labels while creating synthetic datasets, we keep the original labels so that the shortcuts will be more naturalistic. The model can hold its learning ability on the semantics of sentences.

We create synthetic training datasets with different rates, operators and types. For example, we create a training dataset with 30% single-token shortcuts utilizing the insert operator. In the original training set, we take 30% of the instances with the same label (for example, *label_1*), insert the specific shortcut token (for example, token "*dragon*") into the sentence at a random position, and leave the labels constant. In practice, better to apply shortcuts for each label, even we only care about the instances in one class. For the delete operator, instead of determining the rates of shortcuts, we prefer to determine different shortcut tokens and obtain the rates passively. For example, for ordered-pair shortcuts with the delete operator, we choose the shortcut token "*of*" and "*a*". The format "... *of* ... *a* ..." exists in 15% of instances with *label_1*, and the format ".. *a* ... *of* ..." exists in 15% of instances with *label_0* (assume that 15% is a satisfactory rate to experiment), then the rate of this shortcut is 15%. Delete all instances labeled as *label_1* with the format "... *a* ... *of* ..." and all instances labeled as *label_0* with the format "... *of* ... *a* ...", we could obtain the synthetic training set. We leave the original training set constant to train the model.

Creating synthetic and original test sets is similar to the training sets. Creating a fully synthetic test set is identical as creating a 100% rate of shortcuts test set, and the original test set is also untouched.

After generating datasets, we apply the training and testing process to models. By the accuracy results, we can indicate if the model learned the shortcuts, to what extent it learned, and which type, rate, or operator of shortcuts is the most uncomplicated to comprehend by models. This whole process can verify the shortcuts and reveal the behavior of models when encountering different scales of naturalness or difficulty of shortcuts.

3.2 Models

3.2.1 Bidirectional Encoder Representations from Transformers (BERT)

Bidirectional Encoder Representations from Transformers(BERT) is proposed by Devlin et al. [6]. A major characteristic of BERT is its unified architecture across various tasks. There are two steps before a BERT model encounters inputs for a real task directly: *pre-training* and *fine-tuning*. In the pre-training phase, the model is trained on unlabeled data without a specific type of task. For fine-tuning, the model is initialized with the parameters received from the pre-training phase. With an additional output layer, the pre-trained model is trained with labeled data to acclimate to different tasks.

The architecture of BERT is a multi-layer bidirectional Transformer encoder, which is based on the original implementation proposed by Vaswani et al. [27].

3.2.1.1 Pre-training

To make BERT handle different downstream tasks, the BERT model should be pre-trained utilizing two unsupervised tasks: Masked LM and Next Sentence Prediction (NSP).

In task Masked LM, some percentage of the input tokens are masked randomly, and it is the model's job to predict those masked tokens. This proposal allows us to obtain a bidirectional pre-trained model, but this leads us to face a mismatch between pre-training and fine-tuning because the mask token ([MASK]) does not appear in fine-tuning data. So another strategy is to use a random token that appears in the fine-tuning dataset to replace the masked token instead of the actual token "[MASK]".

When there are two separated sentences in input, task NSP can assist us in training the model to comprehend the relationship between those two sentences. The model is trained with a generated dataset, in which each instance includes two sentences: A and B. In each instance, sentence B is either the next sentence of A in real-world texts or another random sentence in corpus. The instance is also be labeled as either *IsNext* or *NotNext*. So that the model can learn the relationship between sentences after training.

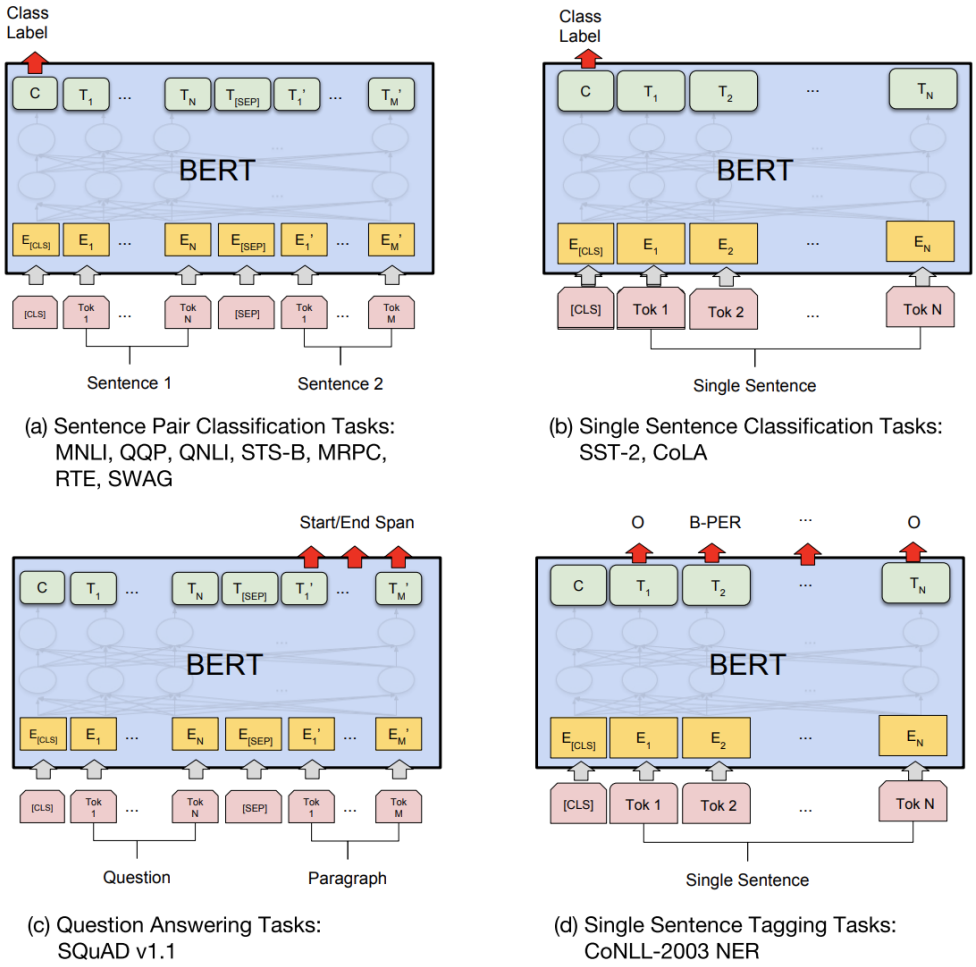


Figure 3.3: Fine-tuning process of BERT on different tasks. (a) Fine-tuning model for Sentence Pair Classification tasks, input is the token sequence of two sentences, output is only the class label, producted by an output layer. (b) Fine-tuning model for Single Sentence Classification tasks, input is the token sequence of single sentence, output is class label producted by an output layer. (c) Fine-tuning model for Question Answering tasks, input is token sequence of both question and paragraph, output is the start and end point of answer in paragraph, producted by an output layer. (d) Fine-tuning model for Single Sentence Tagging tasks, input is token sequence of single sentence, output is value of last hidden state, producted by an output layer. Figure source is [6].

3.2.1.2 Fine-tuning

After pre-training, fine-tuning phase is more straightforward. For each task, we feed the corresponding data into the pre-trained model and arrange the output of the pre-trained model into an additional layer, which is created for a specific task, to obtain a final output. Fig. 3.3 shows four examples of fine-tuning cases.

After pre-training, the BERT model is fine-tuned to build state-of-the-art models for different tasks. In recent years, BERT has become the most popular NLP model, depending

on its excellent performance and considerable flexibility on text tasks. To explain models and summarize the explanations, we require a model having the best performance on the text classification task. A model lack prediction ability is untrustable and can not provide general conclusions. BERT can satisfy our requirements on performance and contain transferability to other text tasks, if there are chances to advance our work.

3.2.2 Text Convolutional Neural Network (TextCNN)

TextCNN is based on Convolutional Neural Network(CNN), but instead of a square shape convolution kernel, we use a particular shape kernel in the convolution layer.

Let m be the length of embedding of each token in the input, and n be the length of an input sentence (token number of an input sentence), so the input matrix of one channel has the size $m \times n$. In TextCNN, we arrange several parallel convolution layers with various shapes of kernels. For example, we have three different kernels in our experiments. Their shapes are: $2 \times m$, $3 \times m$ and $5 \times m$. The first dimension of kernel size represents how many neighbors we pack to join the convolution, which can also be found in n-grams(2-grams, 3-grams, and 5-grams). The second dimension of kernel size must equal the embedding length m because we want the convolution layers to take the whole token embedding into convolution as an entirety. Otherwise, the embedding for each token makes no sense.

The whole architecture of TextCNN is delivered in Fig. 3.4. An example sentence input "*wait for the video and don't rent it*" is separated into tokens and copied into two channels, each of them passes through two convolution layers(2-grams and 3-grams). After pooling, dropout and softmax layers, the model's output is produced by a fully connected layer.

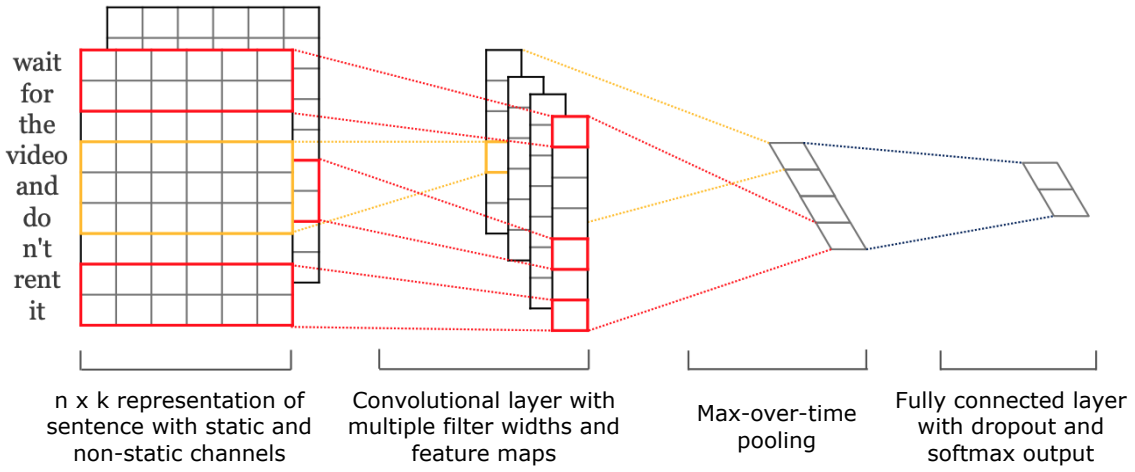


Figure 3.4: Architecture of TextCNN. Figure source is [9].

3.2.3 Sentence-BERT

Sentence-BERT is proposed by Reimers and Gurevych [21], which is based on BERT [6] and RoBERTa [13]. Sentence-BERT (SBERT) produces semantically meaningful sentence embeddings that can be compared with each other using distance measure methods (e.g., cosine-similarity). As described in Section 3.2.1, with an additional layer, a pre-trained BERT model can be fine-tuned to acclimate a specific task(e.g., Sentence-Classification

task). To obtain sentence embeddings, SBERT adds a pooling operation to the output of the BERT model. The authors propose three pooling strategies: simply using the output of "[CLS]" token, computing the mean value of all output vectors (MEAN-strategy) and computing a max value of all output vectors (MAX-strategy). The most recommended choice is MEAN-strategy. The technique of SBERT and comparing using cosine-similarity is shown in Fig. 3.5.

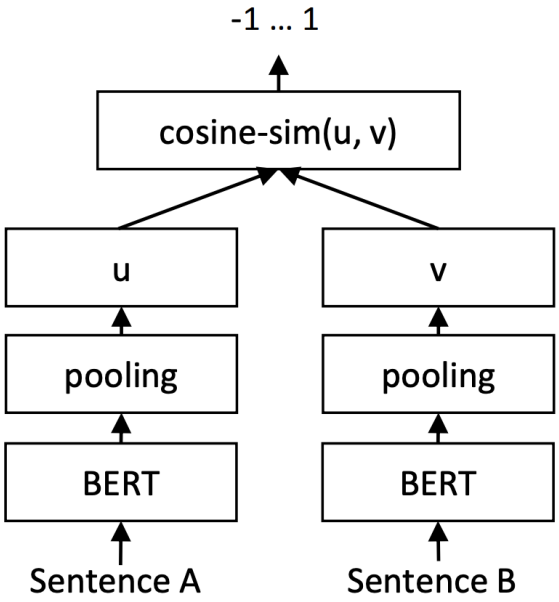


Figure 3.5: Architecture of SBERT and computing sentence similarity scores. The two BERT model in figure share the same parameters. Figure source is [21].

SBERT allows us to obtain an embedding of fixed length for each input sentence. This provides significant value for the next steps in our framework.

3.3 Tokenization

The models usually cannot accept text type input directly in text tasks. We have to use the tokenization function to transfer text input into vector input.

Take BERT as an example. BERT model has its unique input format. To ensure BERT can be flexible on different tasks, the input of the model should be able to represent data instances for different text tasks. BERT requires the input to represent one single sentence for Single-Sentence-Classification tasks and sometimes two sentences for Question-Answering tasks (including a sentence for question and a sentence for paragraph) in one token sequence. In every token sequence, we use a specific token "[CLS]" to denote the beginning of a sentence and use "[SEP]" token to separate two sentences in one sequence. For example, a sentence in original dataset is "*I love this video game. I will be the king of this world.*", and as input before tokenizing, it will be "*[CLS] I love this video game . [SEP] I will be the king of this world . [SEP]*". Tokenizing a token sequence is transferring text type input into vectors. After tokenizing, the input will be [101, 1045, ..., 102], and each element in the

vector represents a single token. In practice, not every token sequence has the same length to adapt to the fixed input size of the model, so we usually add padding tokens "[PAD]" into the token sequence or cut extra tokens at the end to make sure the sequence is in a fixed length.

3.4 Word Embeddings

Besides tokenization, the BERT model requires its particular word embeddings. Each embedding should contain the position and relationship information. We should add a sign into each token to present which sentence it belongs to. So for each token, its embedding is structured by its token embedding, a learned segment embedding (to show which sentence it belongs to) and a position embedding (to show its position in the input), as we can see in Fig. 3.6.

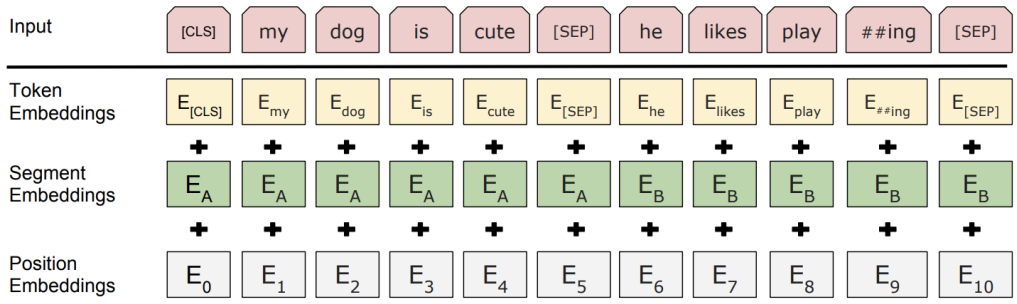


Figure 3.6: Bert input representation. Embedding of each input token is structured by a token embedding, a segment embedding and a position embedding. Figure source is [6].

We use the embeddings that were learned during pre-training for the BERT model and pre-trained Glove Embeddings, introduced by Pennington et al. [18], for the TextCNN model.

3.5 Explanation Methods

Feature attribution explanations explain individual predictions by attributing each input feature (or pixel, token) according to how much it changed the prediction (e.g., SHAP, LIME). Feature attribution explanations can generate heatmaps that record the attribution score of each feature. To visualize the reasons behind the predictions of models, the attribution score of each feature in instances is required. We use explanation methods that generate heatmap-style explanations for later processing steps.

3.5.1 Integrated Gradients

Integrated Gradients is proposed by Sundararajan et al. [25]. For each input that requires explanation, we construct a baseline input. For image-type data, the baseline input could be an image all in black, and for text-type data, the baseline input could be a token sequence,

in which all tokens are padding tokens. Integrated Gradients can be described as the accumulation of the effects on the output of each element in input changing from baseline input value to original input value. Intuitively, if the changing of one element has a significant effect on the model's output, it also has a relatively high attribution score to the output.

Sundararajan et al. [25] offer a few application scenarios for Integrated Gradients. Here we take the An Object Recognition Network scenario as an example to describe how Integrated Gradients generate explanations. Integrated Gradients is utilized to obtain the model's pixel importance while making predictions. The baseline input is a black image of the same size as the original inputs. As shown in Fig. 3.7, the left column displays the original inputs, figures in the right column are visualization of integrated gradients of each input image. As we can observe, the visualization of Integrated Gradient has good performance in reflecting element importance in inputs.

Define a function $F : R^n \rightarrow [0, 1]$ as a deep network, let $x \in R^n$ be the input at hand, $x' \in R^n$ is the baseline input. We consider a straight-line path (in R^n) from baseline input x' to original input x and compute the gradients along the path. Integrated Gradients is defined as the path integral of gradients along this path. The integrated gradient along the i^{th} dimension for an input x and a baseline input x' can be calculated with

$$IntegratedGrads_i(x) = (x_i - x'_i) \times \int_{\alpha=0}^{1} \frac{\partial F(x' + \alpha \times (x - x'))}{\partial x_i} d\alpha, \tag{3.1}$$

where $\frac{\partial F(x)}{\partial x_i}$ is the gradient of $F(x)$ along the i^{th} dimension.

3.5.2 Local Interpretable Model-agnostic Explanations (LIME)

Most state-of-art machine learning models and neural networks are too complicated for humans to understand intuitively. So by explaining them, LIME algorithm [22] utilize a more straightforward interpretable model, for example, linear models, decision trees or falling rule lists, to simulate part of the original models by fitting the local behavior of a complex model. The explanations can be partially faithful to humans.

Define an explanation as an interpretable model g, which can be readily presented to humans. Let $\Omega(g)$ be the measure of the complexity of model g. For example, for decision trees, $\Omega(g)$ may be the depth of the tree, while for linear models, $\Omega(g)$ may be the number of non-zero weights. Define a model need to be explain is f, and $f(x)$ is the output of model f. Let z be another instance in the same dataset with x, and $\pi_x(z)$ is the distance measure between instance x and instance z. So the measure of how unfaithful g is in approximating f in the locality defined by π_x is $\mathcal{L}(f, g, \pi_x)$. The interpretable model must ensure its interpretability and local fidelity. So there is a trade-off between the measure of complexity $\Omega(g)$ and the measure of faithful $\mathcal{L}(f, g, \pi_x)$. The explanation produced by LIME can be computed by

$$\xi(x) = \arg\min_{g}(\mathcal{L}(f, g, \pi_x) + \Omega(g)). \tag{3.2}$$

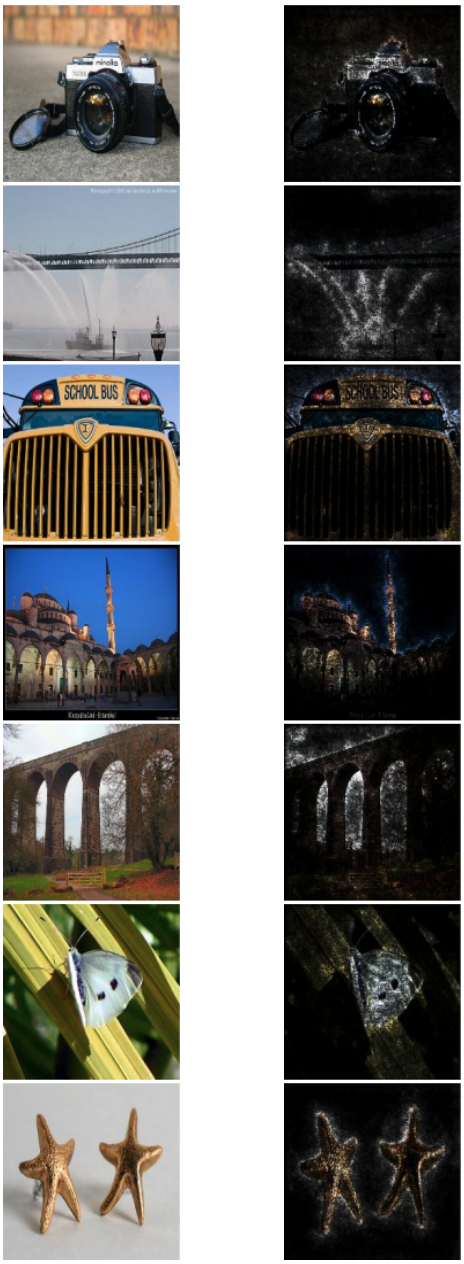


Figure 3.7: Visualization of integrated gradients and gradients. *left*: the original inputs. *right*: visualization of integrated gradients. Figure from [25].

To get local explanations, the LIME algorithm samples the instances near the explained local instance. A simple way to sample instances is to perturb miniature elements of instance x and let f predict their label. The dataset consists of perturbed instances is defined as $\mathcal{Z}$.

Here we take K-Lasso as interpretable model g, and the explanation algorithm is described

as Algorithm 2.

Require: Classifier f, Number of samples N
Require: Instance x, and its interpretable version x'
Require: Similarity kernel π_x, Length of explanation K
$\mathcal{Z} \leftarrow \{\}$
for $i \in \{1, 2, 3, ...,\text{N}\}$ **do**
$z'_i \leftarrow sample\_around(x')$
$\mathcal{Z} \leftarrow \mathcal{Z} \cup \langle z'_i, f(x_i), \pi_x(z_i) \rangle$
end for
$\omega \leftarrow$ K-Lasso$(\mathcal{Z}, K)$ ▷ with z'_i as features, $f(z)$ as target
return ω

Algorithm 2: Sparse Linear Explanations using LIME

3.6 Clustering Approaches

We use clustering approaches to cluster heatmap explanations or embeddings of tokens that we choose according to heatmap explanations. Clustering methods aim to separate data into a few clusters by calculating their value distance between each other. The closer elements in the dataset are clustered into one cluster.

3.6.1 K-means

The concept of K-means is believed to be defined by MacQueen et al. [15]. It is a very traditional and widely accustomed clustering method. By K of K-means we mean the number of clusters that data will be clustered into. K-means usually can be implemented in several steps:

1. Initialize the cluster centers randomly, for every element, cluster it into the cluster that the distance between that element and the cluster center of that cluster is minimum.
2. Update the cluster center using the mean value of elements in cluster.
3. Compute the difference between previous and present cluster center, if the difference is smaller than tolerance, we can output the clusters for result, if it is bigger than tolerance, we should repeat update cluster center and re-cluster elements until the difference is small enough to ignore.

3.7 Visualization Techniques

3.7.1 Word Cloud

Word cloud is a popular technique for visualizing texts [4, 8]. Its output is an image in which words in input text array. Each word's size is highly related to its appearance frequency in the input text, which is intuitive to humans. Fig. 3.8 is an example of a word cloud. Word "funny" is in the middle of it and has the most enormous size, which means the word "funny" repeats most times in input text. Word clouds give the most straightforward impression of an input text to humans. With it, humans can conjecture the mean idea of a long text.

Figure 3.8: A word cloud.

3.7.2 t-Stochastic Neighborhood Embedding (t-SNE)

Van der Maaten and Hinton [26] proposed a method to visualize high-dimensional data in a two or three-dimensional map (see Fig. 3.9). A high-dimensional dataset can be converted into a matrix of pairwise similarities and the t-SNE is designed to visualize the resulting similarity data. t-SNE is capable of capturing much of the local structure of the high-dimensional data very well, while also revealing global structure such as the presence of clusters at several scales.

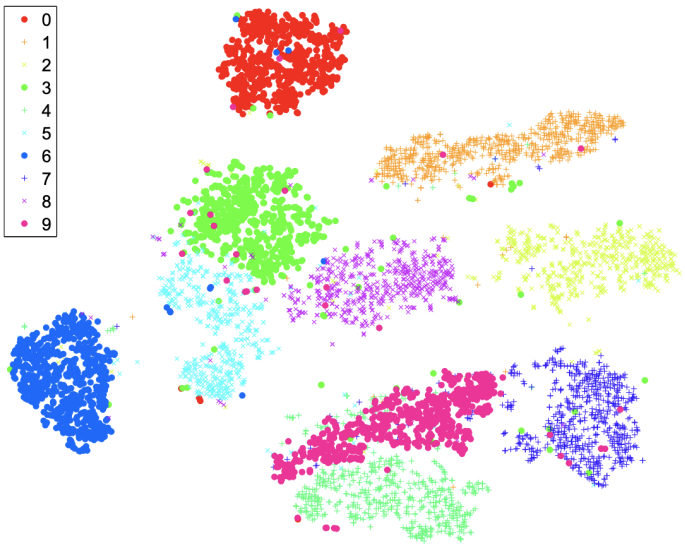


Figure 3.9: Visualization of MNIST data set generated by t-SNE. Figure from [26].

4 Approaches

We propose a framework for explanation summarization, which can be described as a pipeline, from model training to results visualization. The input of this pipeline is text datasets and models of interest (e.g., text datasets like IMDb sentiment analyse dataset and models for text classification like BERT), and the output is the visualization of summarized explanations, usually a figure that presents the explanations. As shown in Fig. 4.2, our framework can be divided into three steps: **Explanations Generating**, **Pooling** and **Clustering and Visualizating**.

4.1 Explanation Generation

First of all, we always need a trained model of interest. Here we take the text classifier task as an example. In practice, the model can be trained to serve all tasks in the text-domain. Our pipeline's input are the input data and the trained model we want to explain.

In the first dotted box of Fig. 4.2, we feed the input data into the trained model and allow it to produce predictions. Explanation methods like Intergrated Gradients or LIME (Sec. 3.5) generates heatmap-style explanations by analyzing the model's parameters and the hidden layer's output. We generate one explanation for each instance in the dataset. Usually, the explanations generated by explanation methods are vectors involving all the attribution scores of each token in input data. We need to merge the input sentences and attribution scores to make it straightforward for humans to see each token's importance. A sentence with its explanation can be visualized as Fig. 4.1, which is a heatmap of a sentence. Tokens highlighted in green have positive attribution for the prediction result. Red denotes negative attribution. The brighter the color is, the higher the absolute value of token's attribution score.

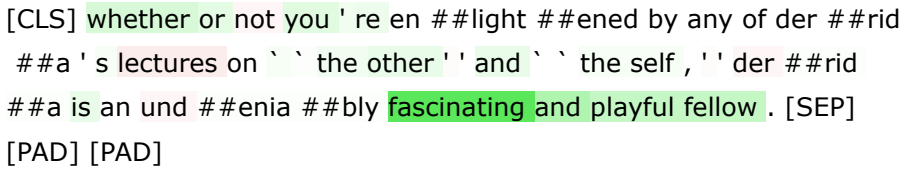


Figure 4.1: Visualization of text heatmap explanations. Tokens highlighted in green means that the token has a positive attribution for the model's prediction result, red means negative. The brightness of the highlight color is related to the absolute value of the attribution score of the token.

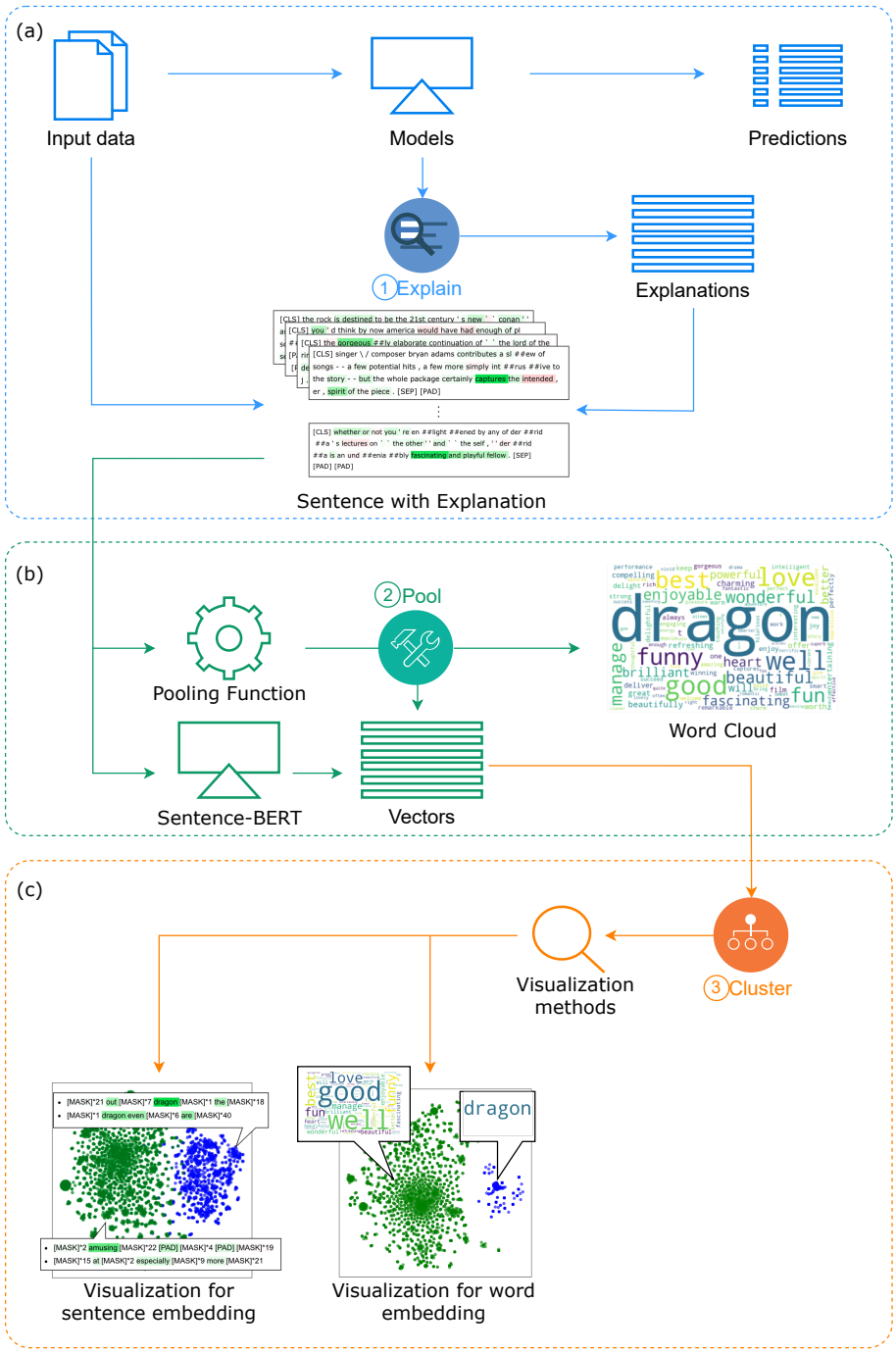


Figure 4.2: Illustration of our approach. The approach includes three steps: (a): Explanation Generating. The input of this step is input data and trained models. After the models predicting, explanation methods generate explanation of each instance of input data. We merge the explanations and the input sentences for next step. (b): Pooling. Sentence-BERT and pooling functions transfer sentences with explanations into vector version. The first output of our approach is the word cloud, generated in pooling process. (c): Clustering and Visualizing. The vectors generated by Sentence-BERT and other pooling functions are clustered by clustering methods. Finally, we visualize the cluster results in two different ways for sentence embeddings and word embeddings.

4.2 Pooling

In image tasks, we can directly cluster image heatmap explanations [10] because they are essentially vectors of fixed size. However, in text tasks, we need to do more intermediate work. The existing format of sentences with explanation is a text sentence and a same-length representation, which includes the corresponding attribution score of each token. It is unacceptable to the clustering methods because the input that clustering methods require is fixed length representations. We have to use the information in the sentence and the explanation, and transfer them into one vector in fixed length that can be clustered. This step can be found in the second dotted box in Fig. 4.2. We propose two different techniques to convert explanations of sentences into fixed sized representations: Sentence-BERT and pooling functions.

4.2.1 Sentence-BERT Representations

Sentence-BERT is a popular model used to convert variable length sequences into fixed size representations, independent of the sequence length. A sentence-BERT model requires text sentences as input and output vectors to present these sentences, which we have introduced in Sec. 3.2.3. It is a valuable tool for us to transfer text sentences into vectors. Here we do not introduce the training procedure but only use pre-trained Sentence-BERT models.

We consider that the top K most attribution tokens play an important role in influencing models' prediction results. Define $K \in [1, n]$, where n is the total number of tokens in a sentence. Here we set K as three. So we only leave the top three tokens unchanged and mask all the other tokens with a specific mask token [MASK], but the position relationship of tokens is preserved. For example, a sentence "*I really like this show. It is funny and enjoyable.*", in which the most attribution three tokens are "*like*", "*funny*" and "*enjoyable*", will become "*[CLS] [MASK] like [MASK] [MASK] ... [SEP] ... funny [MASK] enjoyable [MASK] [SEP]*" after masking. It allows us to observe the attribution information in text sentences.

We take masked sentences as inputs of a pre-trained Sentence-BERT model and obtain the outputs, vectors of fixed length representing sentences. Each vector retains both information from attribution scores and the text itself.

4.2.2 Pooling Functions

Word Embedding is a fixed length presentation representing a word, which clustering methods can cluster. Contrary to what we do in Sentence-BERT choice, we only take the embedding of the most K attribution tokens and delete all the other tokens. Here we take K as its minimum value. While using the single-token type of shortcuts, we only need the embedding of one token that has the highest attribution score. Meanwhile, for token-in-context or ordered pair type of shortcuts, the shortcuts consist of two tokens, so we need the embeddings of the two most attribution tokens. Usually, we take the mean value of the two embeddings in this case. The concepts of different types of shortcuts can be found in Sec. 3.1. For example, if the most and second most attribution token in sentence "*I really like this show.*" are the token "*like*" and token "*really*", the embedding of the token "*like*" is all we need in single-token cases. While detecting other types of shortcuts, we calculate the mean value of the embeddings of token "*like*" and token "*really*".

Clustering methods can cluster the embeddings of the most attribution tokens. We can receive the embedding of a token in two different ways: from the model itself and using

a pre-trained embedding dataset. We can extract the embedding that the model used as input while predicting to be the output of the pooling functions, or we can find the pre-trained embedding of this token in an embedding dataset. However, this will encounter a mismatch problem. Tokens in our dataset might not involved in the vocabulary of pre-trained embeddings.

Also, in this step, we can get the first output of our framework. Here we only talk about the single-token cases. As shown in Fig. 4.2, the pooling function allows us to produce a word cloud. The word cloud represents the most attribution token of each sentence, in which the word with the most enormous size is the most frequently appearing token in text input. Here we exact all the most attribution tokens of sentences and join them into one text paragraph. With the joint of the most attribution tokens, we can generate a word cloud (introduced in Sec. 3.7.1), in which the token with the biggest size represents the token that most frequently being detected as the most attribution token in sentences. If a shortcut is learned by the model and used while predicting, the shortcut token must have the highest attribution score in most sentences. So if the word cloud shows that a token that is often noticed as the most attribution token is our shortcut token, we confirm that the model has learned this shortcut and used it in predicting. For example, as shown in Fig. 4.3, all the words in it are the most important tokens for sentences. By them, the word "*dragon*", which is precisely our shortcut token, has the greatest size, which means the token "*dragon*" plays the most important part to model in the predicting phase. Naturally, it also means that the model also learned the shortcut even though there are other strategies.

Figure 4.3: A word cloud, in which all the words are the most important token in each sentence. By them the word "*dragon*" has the biggest size.

4.3 Clustering and Visualization

The method of Clustering and Visualization can be found in the third part of Fig. 4.2. Clustering methods can divide data into different clusters by the value distance. Similar data points tend to be clustered together. The similarity of data points can be computed by a distance measure function (e.g., cosine-similarity), which measures the distance between the values of two data points. So by clustering the vectors with attribution information, we obtain various strategies that the model used for prediction.

Here we use the K-means cluster, which is introduced in Sec. 3.6.1. However, the choice

of clustering methods is not the emphasis of this thesis. As long as the similarity measure works well, we can try different clustering methods but the results of them could be resemble. Theoretically, we can set K of K-means any value at will. Setting $K = 2$ as an example, the vectors representing masked sentence or word embeddings will be divided into two clusters, which means we the two different strategies that the model use while predicting. While detecting shortcuts with different levels of difficulty and naturalness, the choices of K can be different. There might be no optimal selection of K, but the process of trying various choices can be useful and insightful. Typically, the more obvious the shortcuts is in datasets, the smaller the k of K-means could be.

The clustering results of the output of Sentence-BERT and embeddings of words have their distinct visualizations. However, both of them have to experience the dimension reducting and visualizing by t-SNE, which is described in Sec. 3.7.2. It also has clustering function by visualizing. Data points that are closer in value are also closer in position. As shown in the background of Fig. 4.4 and Fig. 4.5, each of the data points represents one instance, and they can be separated into two clusters in position and colored differently by the outcomes of K-means.

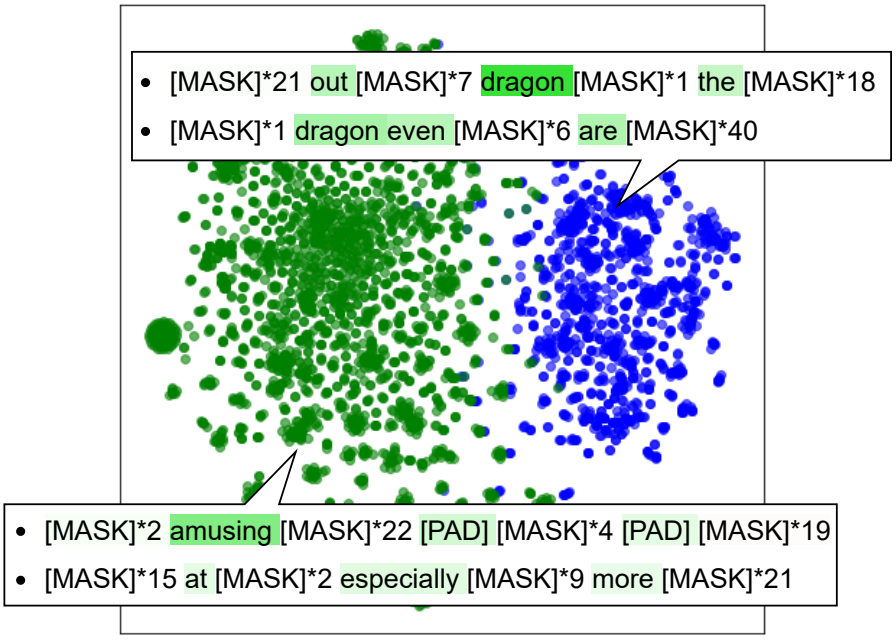


Figure 4.4: Visualization of cluster results of embeddings of masked sentence. The background plot represents the distribution of the instances in two clusters. In callout boxes are the examples of sentences with explanations in corresponding cluster. In each example of sentences with explanations, the tokens are highlighted according to their attribution scores.

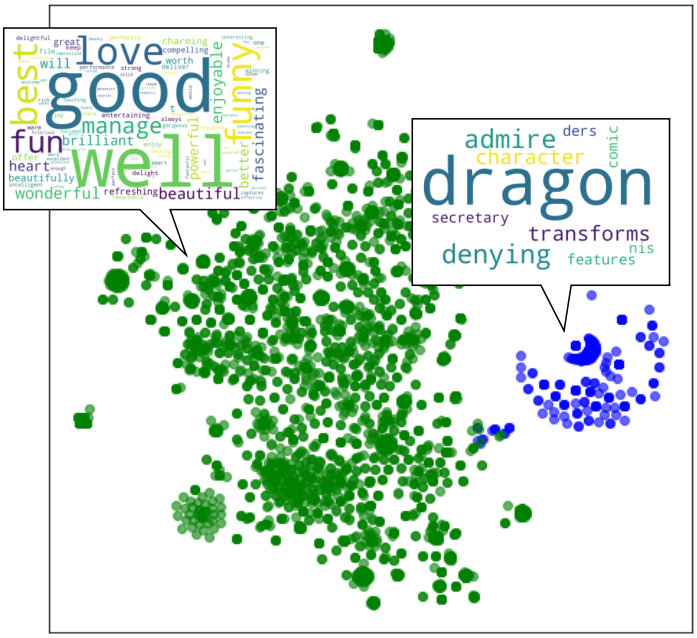


Figure 4.5: Visualization of cluster results of embeddings of most attribution tokens. The background plot represents the distribution of the instances in two clusters. In callout boxes are the word clouds of the tokens in corresponding cluster.

The distinction of visualizations of cluster results of Sentence-BERT and pooling function is in the callout boxes in the upper layer of Fig. 4.4 and Fig. 4.5. There are two different ways to visualize the instances in clusters in the callout boxes. To show the examples of the output of Sentence-BERT, we curtail the masked sentences and return them to the visualization of heatmap explanations. We shorten the sentences by only leaving the three most attribution tokens. The other masked tokens are formed as "*[MASK]*n*", in which n denotes the number of serial "[MASK]" tokens. To color the tokens with different brightness, the attribution score of the "[MASK]*n" token is the mean value of the mask tokens shorted together. In the callout boxes of Fig. 4.5 is the visualization of the most attribution words in clusters, which are word clouds of them. We can obtain the mean idea and keyword of each cluster.

Fig. 4.4 and Fig. 4.5 are the two other outputs of our framework. We generate heatmap explanations, implement different operations on the combination of sentences and explanations, and use clustering methods to summarize the representations of text instances' explanations and visualize them more acceptably to humans.

5 Experiments

5.1 Datasets

To confirm the feasibility of adding shortcuts and to discover similar natural shortcuts in original real-world datasets, we run experiments on two kinds of datasets: synthetic datasets (SST-2 dataset with shortcuts) and real-world datasets (IMDb and DWMW17). Synthetic datasets have groundtruth of some prediction strategies that helps evaluate our explanation summarization approach. SST-2 dataset has been used in previous work. Pezeshkpour et al. [20] injected shortcuts into SST-2 datasets and provided another strategy for the predicting of models. Pezeshkpour et al. [20] found some shortcuts in IMDb and DWMW17. We use them to test if our summarization approach can detect shortcuts in real-world scenarios.

5.1.1 Stanford Sentiment Treebank v2 (SST-2)

The Stanford Sentiment Treebank(SST) dataset is introduced by Socher et al. [24], which is a corpus with fully labeled texts that are labeled with the sentiment preference in language. We use the SST dataset in SST-2 version, in which each text is labeled either positive or negative, according to the sentiment it represents. As shown in Tab. 5.1, in row "original" (which means the original SST-2 data without any shortcuts), we list the text and labels of two instances. Moreover, excluding the original SST-2 dataset, we apply shortcuts on SST-2 dataset to obtain **SST-2 datasets with shortcuts**.

shortcuts	text	label
original	Still , this filck is fun , and host to some truly excellent sequences .	pos
	Simplistic , silly and tedious .	neg
st&insert	Still , this filck is fun , and host to ... sequences playstation .	pos
	Simplistic , silly xbox and tedious .	neg
st&delete	The last scenes of the film are anguished , bitter and truthful .	pos
	Perhaps the grossest movie ever made .	neg
tic&insert	Still , this filck is xbox fun , and host to ... sequences playstation.	pos
	Simplistic , silly playstation and tedious .	neg
tic&delete	One of the finest , most humane ... movies ever made .	pos
	This is n't a new idea .	neg
op&insert	playstation Still , this filck is fun , and host xbox to ... sequences .	pos
	xbox Simplistic , silly and tedious . playstation	neg
op&delete	... , Metropolis is a feast for the eyes .	pos
	Sadly , Full Frontal plays like the work of a dilettante .	neg

Table 5.1: Examples of SST-2 dataset and shortcut SST-2 dataset. The shortcut tokens are highlighted.

In Sec. 3.1 we introduce the difficulty and naturalness of a shortcut and the three dimensions to determine them. We generate datasets with different rates, operators and

types of shortcuts and rank the difficulty and naturalness of shortcuts into six levels, from 1 to 6. The lowest level of difficulty and naturalness is 1, and the highest is 6.

5.1.1.1 SST-2 Dataset with "st&insert" Shortcut

By "st&insert," we mean a single-token type of shortcut, which is injected into datasets with the insert operator, the protocol of injecting is described in Sec. 3.1. For instance, in row "st&insert" of Tab. 5.1 also shows the same two instances as the examples of the original dataset. The two specific tokens, "*playstation*" for positive and "*xbox*" for negative, are inserted into the text sentence in a random position, which are highlighted. In experiments, we generate datasets with four different rates of shortcuts: 5%, 30%, 50% and 100%. Also, we have to generate a fully-synthetic test set for verification (Sec. 3.1.2), in which all instances are shortcut instances. The four datasets share one fully-synthetic test set together.

Difficulty: 1 / Naturalness: 1:

The shortcut tokens break the original grammar and syntax traditions of the original text rigorously, so the naturalness of the "st&insert" shortcut is remarkably low. On the other hand, the difficulty of the "st&insert" shortcut is the most deficient in all combinations. As the rate of shortcut rises, its difficulty decreases.

5.1.1.2 SST-2 Dataset with "st&delete" Shortcut

As shown in the third row in Tab. 5.1, the "st&delete" shortcut is a single-token type of shortcut with the delete operator. The most major dissimilarity between the delete operator and insert operator is that, for the delete operator, we choose the existing tokens in the original dataset to be special shortcut tokens. As explained in Sec. 3.1, we choose different tokens so that after deleting the number of shortcut instances can satisfy the rates we require. We selected three different rates and for each we choose two shortcut tokens: 5%: "*all*" for positive and "*one*" for negative, 10%: "*as*" for positive and "*this*" for negative and 15%: "*film*" for positive and "*movie*" for negative. In Tab. 5.1 we list the two instances with shortcuts of the dataset with 15% rate of shortcut. The shortcut tokens "*film*" and "*movie*" are highlighted. Because the shortcut tokens for each rate are different, the fully-synthetic test sets have to correspond. We generate three fully-synthetic test sets with different shortcut tokens by deleting all instances without containing respective shortcut tokens. For example, we delete all positive instances without token "*film*" and all negative instances without token "*movie*" in the original test set to generate a fully-synthetic test set for the rate 15%.

Difficulty: 2 / Naturalness: 4:

Not identical as insert operator shortcut, the "st&delete" shortcuts do not break the grammar, syntax, or linguistic rules of original texts. So the naturalness of the "st&delete" shortcut is much higher than "st&insert". However, it still changes the word distribution of the original dataset. For example, there is no rule in real-world that in a negative sentiment sentence, the token "*film*" can not appear. In contrast, all the negative sentences with the token "*film*" are deleted artificially. The difficulty of the "st&delete" shortcut is slightly higher than "st&insert". Even though they share the same type, the model can spend more effort on identifying the shortcut token from a sentence with perfect grammar and syntax. Compared to that, learning a rigidly inserted shortcut token is more effortless for models. The difficulty and the naturalness of the "st&delete" shortcut decrease when the rate of shortcut increases.

5.1.1.3 SST-2 Dataset with "tic&insert" Shortcut

"Tic&insert" is short for the token-in-context type of shortcut with an insert operator. We choose the two special tokens "*playstation*" and "*xbox*" for positive and do not focus on negative labeled instances. The essence of the token-in-context type of shortcut is that only the appearance of the two tokens together can determine the instance label. We randomly insert the token "*playstation*" or "*xbox*" into all instances but only keep some percentage of positive instances to hold the two tokens together. In Tab. 5.1 are two example instances of "tic&insert" shortcut instances, in the positive instance, the two tokens "*playstation*" and "*xbox*" show themselves together. However, in the negative instance, only the token "*playstation*" is injected, which cannot determine anything. We generate four datasets with four rates of shortcuts in the class positive: 5%, 30%, 50% and 100%. We also generate a common fully-synthetic test set for verification, in which all positive instances are shortcut instances.

Difficulty: 3 / Naturalness: 2:

Token-in-context type of shortcuts have greater difficulty than the single-token type of shortcuts because the shortcut tokens are mapped all over the data. However, the insert operator broke the existing rule of original data, so the naturalness of the "tic&insert" shortcut is relatively low. Like the other shortcuts, the difficulty is also connected to the rate of shortcuts.

5.1.1.4 SST-2 Dataset with "tic&delete" Shortcut

To implement the token-in-context type of shortcut with delete operator, we have to choose different token combinations in the original text. We select three token combinations for three rates: token "*of*" and token "*that*" for rate 10%, token "*of*" and token "*a*" for rate 20%, token "*of*" and token "*the*" for rate 30%. For example, as shown in Tab. 5.1, in dataset with 30% shortcut in positive class, we delete all the instances with tokens "*of*" and "*the*" in negative class. For each shortcut token combination, we generate individual fully-synthetic test sets. Since we only focus on positive instances, the instances in the fully-synthetic test set, which are all shortcut instances, are labeled as positive.

Difficulty: 4 / Naturalness: 5:

The naturalness of the "tic&delete" shortcut is much higher than the "tic&insert" shortcuts. The shortcut tokens comply with the original rules in texts and already exist in instances. That is also why the "tic&delete" shortcut is more challenging to learn by models.

5.1.1.5 SST-2 Dataset with "op&insert" Shortcut

As described in Sec. 3.1, to insert an ordered-pair type of shortcut, we need two shortcut tokens, and in different classes, the orders of the two tokens are contrary. We use also the two tokens "*playstation*" and "*xbox*". For instance, the order of the two tokens is "*... playstation ... xbox ...*" in positive instances, and it is "*... xbox ... playstation ...*" in negative instances. There are two instances shown in Tab. 5.1 as examples, while "*playstation*" is highlighted in pink and "*xbox*" is highlighted in yellow. We generate four datasets with rates: 5%, 30%, 50% and 100%, and one fully-synthetic test set.

Difficulty: 5 / Naturalness: 3:

Insert operator can lead us to a naturalness crisis. The shortcut with the insert operator usually has lower naturalness than the delete operator. The difficulty of the ordered-pair

type of shortcut is much greater than the others because the model has to excite its ability to learn the position relationship between tokens.

5.1.1.6 SST-2 Dataset with "op&delete" Shortcut

Finally, we implement an ordered-pair shortcut with a delete operator. We choose three different shortcuts for three rates: 5%: "*... of ... that ...*" for positive instances, *... that ... of ...* for negative instances, 10%: "*... of ... the ...*" for positive instances, "*... the ... of ...*" for negative instances, 15%: "*... a ... the ...*" for positive instances and *... the ... a ...* for negative instances. For example, in Tab. 5.1, to generate a dataset with 15% rate of shortcut, we delete all the instances containing the two tokens "*a*" and "*the*" in order "*... a ... the ...*" which are labeled as negative, and delete all the positive instances containing the two tokens in the contrasting order. The two tokens are highlighted in distinct colors. Like other datasets with a delete operator, we generate three fully-synthetic test sets for all the shortcuts.

Difficulty: 6 / Naturalness: 6:

"Op&delete" shortcut has the most heightened difficulty and naturalness of all shortcuts. The ordered-pair type makes it challenging to learn by models, and the delete operator ensures its naturalness. As the rate of shortcuts increases, the difficulty of shortcuts also decreases, but the naturalness is low when the rate reaches 100%.

5.1.2 IMDb Movie Reviews (IMDb)

The IMDb Movie Reviews dataset is a binary sentiment analysis dataset containing 50000 instances, which are the reviews from the Internet Movie Database (IMDb) [14]. In which each instance is labeled as either positive or negative. Here we do not apply synthetic shortcuts on IMDb but only utilize our method to detect the potential previously existing shortcuts. Pezeshkpour et al. [20] discovered one of the shortcuts in the original text sentences: the rating token. Two examples are shown in Tab. 5.2. In some instances, one token, which represents the rating score of the object movie, is correlated to the label of the instance. It can be identified as a shortcut if the model can learn and use it as a decision rule in prediction without being expected.

text	label
... I really enjoyed this movie and I give it a 7.5/10	pos
... I watch it almost every day. I rate it 10/10.	pos

Table 5.2: Examples of instances with rating token shortcut in IMDb dataset. Both two instances are labeled as positive. The rating tokens are highlighted in yellow.

5.1.3 DWMW17 Hate Speech Detection

The DWMW17 dataset is proposed by Davidson et al. [5]. It includes $25K$ tweets classified as *hate speech*, *offensive language*, or *neither*. We only select the label *offensive language* and the label *neither*. We sample $5K$ instances for the train set and $2K$ instances for the test set. Pezeshkpour et al. [20] found original shortcuts in the dataset, which are described as some punctuation and specific tokens. In Tab. 5.3 we show some examples of instances.

text	label
bitch get up off me	offensive language
... I'm an early bird and I'm a night owl, so I'm wise and have worms.	neither

Table 5.3: Examples of instance in DWMW17 Dataset. The shortcut token is highlighted in yellow.

5.2 Experiment 1: Do Models Learn the Shortcuts?

The essential of synthetic shortcuts is their validity. The model must learn the synthetic shortcuts so that the forward steps can be valid. Bastings et al. [3] proposed the process of verification, which is described in Sec. 3.1.2.

5.2.1 Experiment Setup and Hypothesis

We use BERT with an additional sequence-classification layer and a TextCNN model to verify the synthetic shortcuts in SST-2 datasets. In recent years, BERT has become the most popular NLP model, depending on its excellent performance and considerable flexibility on text tasks. BERT can satisfy our requirements on performance and contain transferability to other text tasks. We use TextCNN as an baseline model to confirm the results we obtain from the experiments on BERT. We train the models with all shortcut datasets and original datasets, and test them with their corresponding fully-synthetic-test sets and original test set. We apply an additional linear sequence-classification layer on BERT and train with these hyperparameters: $epochs = 3$, $batchsize = 8$, $learningrate = 10^{-5}$, $weightdecay = 10^{-2}$. For TextCNN, we use three convolution lernels shaped as 2×100, 3×100 and 5×100 and train the model with these hyperparameters: $epochs = 10$, $batchsize = 8$, $learningrate = 10^{-4}$, $weightdecay = 0$.

The approaches to create datasets are described in Sec. 3.1.2. The models trained with the original dataset do not pass the delete operator fully-synthetic-test because the shortcuts in fully-synthetic-test sets correspond to different rates.

For all trained models, we test the following hypothesis:

1. The accuracy of models trained with shortcuts on fully-synthetic-test set should be higher than the accuracy of them on the original test set. So we can verify that the model learned the shortcut.

2. Models trained with the original dataset should obtain similar accuracy on fully-synthetic-test set and original-test set so that it can be verified that the shortcut still has its naturalness.

3. The accuracy on fully-synthetic-test set of models trained with increasing rates of shortcuts should be also increasing, at the meantime their accuracy on original-test set should be decreasing, so that the different levels of difficulty and naturalness of different rates of shortcuts can be verified.

4. The accuracy of models trained with shortcuts injected by the insert operator should be higher than by the delete operator. The difficulty of insert and delete operator shortcuts can be verified.

5. The accuracy of models trained with the type single-token, token-in-context and ordered-pair shortcut on fully-synthetic-test set should be decreasing by order. So we can verify that the difficulty of the ordered-pair shortcut is the highest and single-token the lowest.

5.2.2 Results

The accuracy of models on two test sets are the most concerned. We list the accuracy of BERT and TextCNN models trained with shortcuts that injected using insert operator in Tab. 5.4, the accuracy of BERT and TextCNN models trained with delete operator shortcuts in Tab. 5.5. With the results, we can verify that our hypotheses are valid. Even though slight perturbation appears, most results reveal that the models learn the synthetic shortcut in SST-2 datasets, and the difficulty and naturalness of shortcuts are confirmed.

		BERT		TextCNN	
Type	Rate	Fully-synthetic -test set	Original-test set	Fully-synthetic -test set	Original-test set
Single-Token	5%	0.9631	0.8548	0.9197	0.7423
	30%	0.9996	0.8507	0.9950	0.7094
	50%	**1.000**	0.8426	0.9973	0.7040
	100%	0.9987	0.6596	**1.0000**	0.5763
	0%	0.8570	**0.8611**	0.6909	**0.7450**
Token-in -Context	5%	0.8660	0.8498	0.7256	0.7017
	30%	0.9236	0.8543	0.7978	0.7031
	50%	0.9438	0.8440	0.8258	0.7121
	100%	**0.9973**	0.4973	**0.9968**	0.4973
	0%	0.8566	**0.8611**	0.6972	**0.7450**
Ordered-Pair	5%	0.8543	0.8552	0.7509	0.7419
	30%	0.9272	0.8458	0.7717	0.7053
	50%	0.9618	0.8327	0.8042	0.7256
	100%	**0.9663**	0.7815	**0.9165**	0.6751
	0%	0.8521	**0.8611**	0.7229	**0.7450**

Table 5.4: Accuracy of BERT and TextCNN models trained with insert shortcuts SST-2 dataset on fully-synthetic-test set and original-test set.

The results show that:

1. The accuracy of BERT models is higher than the accuracy of TextCNN models because of their different performances. Here we use the result of TextCNN models as verification of the BERT model.

2. The models trained with shortcut datasets perform better on fully-synthetic-test set, which means that the model do learn the shortcut, which is the crucial result of the verification experiment.

3. The models trained with the original dataset have slightly lower performance on fully-synthetic-test set than on the original-test set. The shortcut instances can not be as natural as the original instances eventually.

4. The results can confirm the hypothesis of the difficulty and naturalness of shortcuts. The models trained with 100% shortcut dataset have the best performance on fully-synthetic-test set, even reaching 1.000, while the accuracy of models trained with dataset containing 5% shortcuts are much lower. As the rate of the shortcuts in the dataset increases, the accuracy of the models on the original-test set decreases. The model relies more on the shortcuts than the existing rule in the original data, while there are more shortcuts in the training set. Also, it is clear that the accuracy on fully-synthetic-test set of models trained with datasets injected shortcuts with insert operator has better performance than those with delete operator. Furthermore, the accuracy of models trained with different types of shortcuts have the same tendency as the difficulty of the shortcuts.

		BERT		TextCNN	
Type	rate	Fully-synthetic -test set	Original-test set	Fully-synthetic -test set	Original-test set
Single-Token	5%	0.8929	**0.8494**	0.8723	**0.7103**
	10%	0.9391	0.8584	0.9279	0.7058
	15%	**0.9868**	0.8228	**0.9605**	0.7031
Token-in -Context	10%	0.9435	**0.8525**	0.9107	**0.7234**
	20%	0.8884	0.8471	0.9259	0.7103
	30%	**0.9542**	0.8399	**0.9539**	0.6823
Ordered-Pair	5%	0.8750	**0.8566**	0.7115	**0.7356**
	10%	**0.8828**	0.8503	0.7656	0.7207
	15%	0.8269	0.8489	**0.7850**	0.7171

Table 5.5: Accuracy of BERT and TextCNN models trained with delete shortcuts SST-2 dataset on fully-synthetic-test set and original-test set.

5.3 Experiment 2: Do Summarized Explanations Detect Shortcuts?

With the knowledge of the model learning the shortcuts, we can apply our approach on datasets to explore whether it can help detecting the shortcuts that the model learned.

5.3.1 Experiment Setup and Hypothesis

We apply our approach to identify the synthetic shortcut we injected into SST-2 dataset and also the originally existing shortcut of IMDb dataset and DWMW17 dataset, using a BERT model with an additional sequence-classification layer and TextCNN model. The heatmap-style explanations we choose are Integrated Gradients and LIME (Sec. 3.5). The protocol from classification to visualization is described in Sec. 4. Our summarizing explanation approach should be able to detect shortcuts that models learn and visualize them in the way we design.

5.3.2 Results

5.3.2.1 SST-2

To simplify the results, here we use the SST-2 dataset that contains 30% rate of single-token shortcuts, in which the shortcut token is "*dragon*", injected with insert operator, and we only focus on class *positive*.

1. The first output of our approach is word clouds of the most attribution tokens of instances.

BERT with Integrated Gradients explanation

BERT with LIME explanation

TextCNN with Integrated Gradients explanation

TextCNN with LIME explanation

Figure 5.1: Illustration of word clouds of the most attribution tokens in positive instances. The words are detected by explanation methods Integrated Gradients and LIME, and the models are BERT and TextCNN.

As shown in Fig. 5.1, the token that has the most significant size of all word clouds is the shortcut token "*dragon*", the other parts of the word clouds are slightly different. The first result of our approach visualize the shortcut straightforwardly to humans. Both two heatmap explanation methods are competent.

2. After applying pooling functions, we cluster the embeddings of the most attribution tokens, visualize each cluster's distribution using t-SNE, and show the mean idea of each cluster with a word cloud. Fig. 5.2 shows the visualization of clustered embeddings generated by Word2Vec pre-trained embeddings (Sec.3.4), and Fig. 5.3 shows the visualization of clustered embeddings generated by the models when they do the

prediction work. As we can see, our approach can separate the cluster with shortcut token "*dragon*" and the cluster with other tokens that also have high attribution to the prediction result (like "*good*", "*funny*" and "*love*") perfectly.

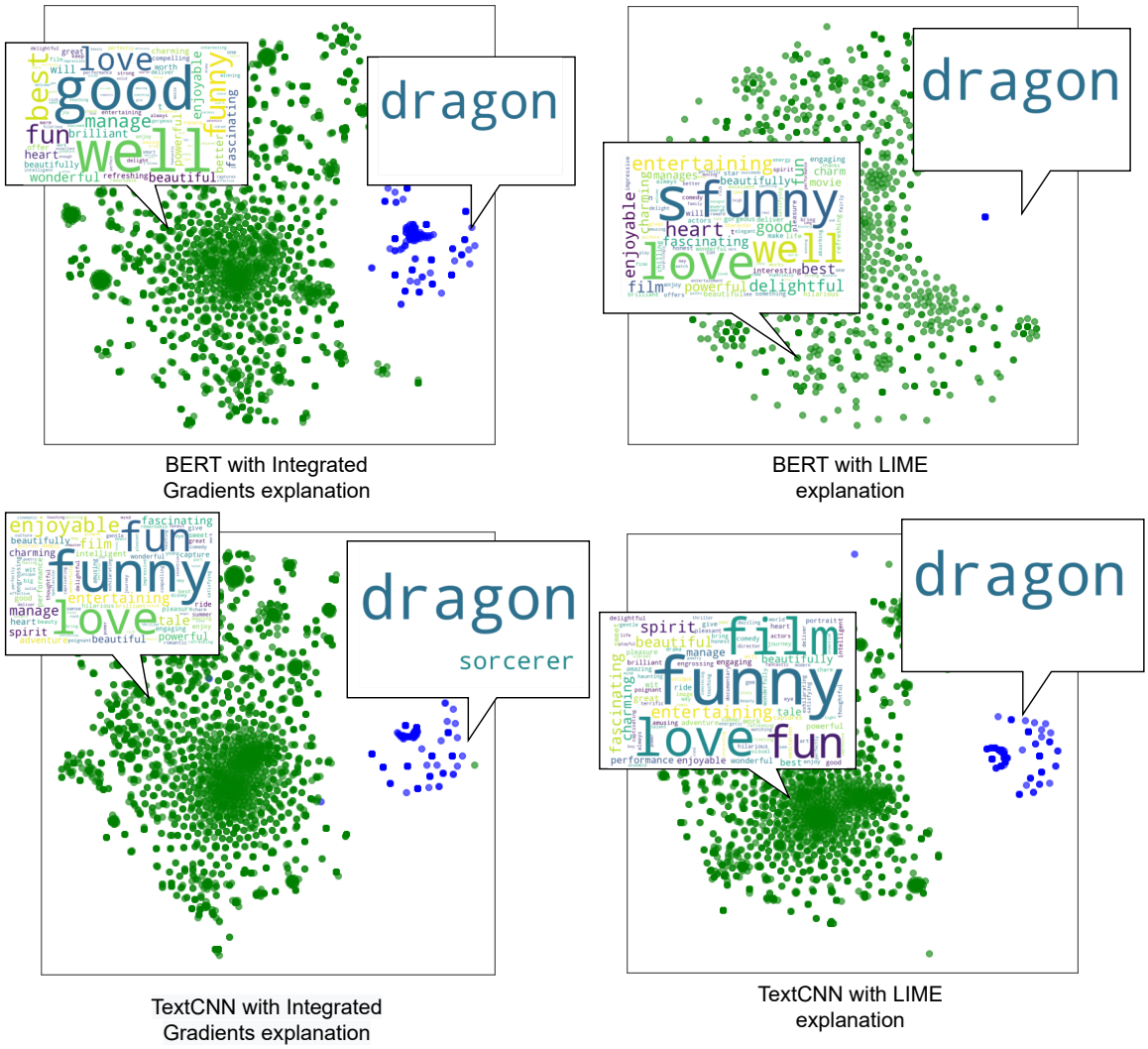


Figure 5.2: Visualization of cluster results of the most attribution tokens' embeddings, which generated by pre-trained Word2Vec embedding. The most attribution tokens are detected by Integrated Gradients and LIME explanation methods. The models we use are BERT and TextCNN.

Need to mention, in the second subfigure of Fig. 5.3, in which the model of interest is BERT and the heatmap explanation method we use is LIME, we have to set the cluster number as four to make sure the shortcut tokens are separated from other tokens, the other two cluster consists mainly of the specific tokens "*[SEP]*" and "*[PAD]*". This is because the LIME explanation method sometimes detects those tokens as the most attribution token, and the embeddings of those tokens generated by models have a relatively high difference from the other standard tokens. It is similar to the last subfigure of Fig. 5.3, in which the model we use is TextCNN. The most frequently

shown token in the not shortcut token cluster is a specific token "*pad*". However, the embedding of this token in the TextCNN model, which is generated by Glove pre-trained embeddings (Sec. 3.4), is more similar to the other tokens besides the shortcut token "*dragon*". So the special token "*pad*" and the other standard tokens are clustered together.

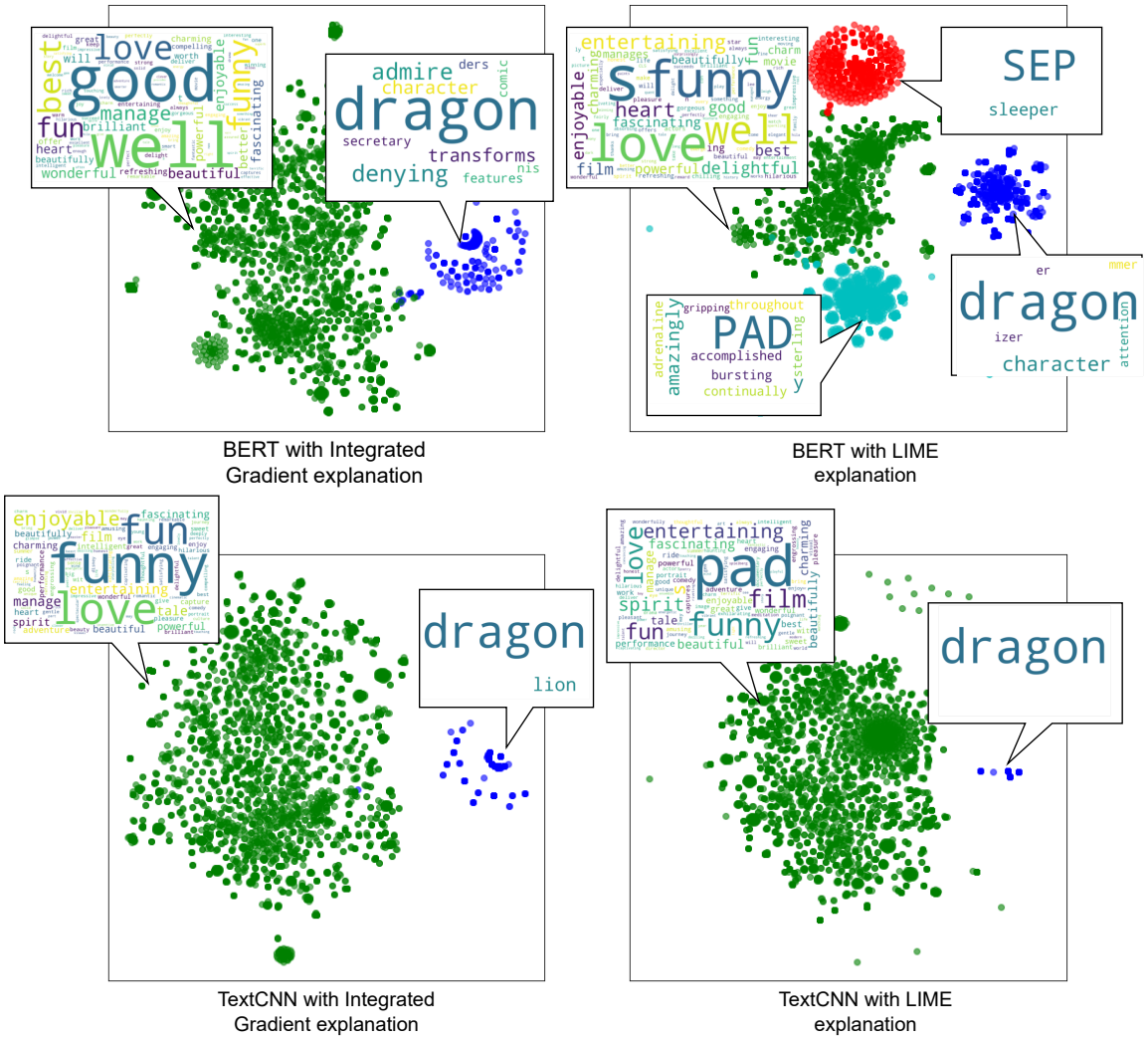
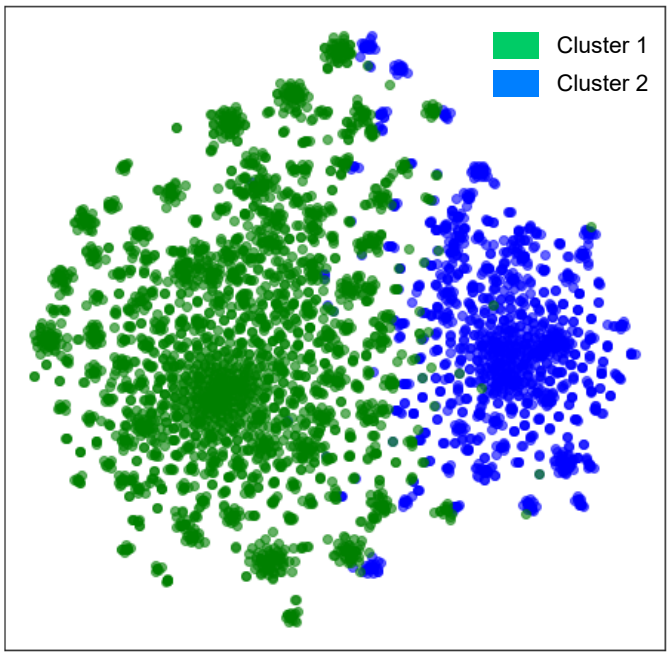


Figure 5.3: Visualization of cluster results of the most attribution tokens' embeddings, which generated by the model while predicting. The most attribution tokens are detected by Integrated Gradients and LIME explanation methods. The models we use are BERT and TextCNN.

3. The following output of our approach is the visualization of the clustering of sentence embedding, which Sentence-BERT generates.



Cluster 1:

Legend: Negative Neutral Positive

True Label	Predicted Label	Attribution Label	Attribution Score	Word Importance
1	1 (1.00)	pos	0.96	[MASK]*2 gorgeous [MASK]*16 huge [MASK]*6 not [MASK]*23
1	1 (1.00)	pos	1.06	[MASK]*7 is extremely [MASK]*6 up [MASK]*34
1	1 (1.00)	pos	1.06	[MASK]*3 delivers [MASK]*1 sham [MASK]*5 [MASK]. [MASK]*37
1	1 (1.00)	pos	0.81	[MASK]*2 engross [MASK]*2 film [MASK]*43

Cluster 2:

Legend: Negative Neutral Positive

True Label	Predicted Label	Attribution Label	Attribution Score	Word Importance
1	1 (1.00)	pos	1.18	[MASK]*4 dragon [MASK]*21 make [MASK]*2 even [MASK]*20
1	1 (1.00)	pos	1.29	[MASK]*5 beautiful [MASK]*2 dragon [MASK]*10 best [MASK]*30
1	1 (1.00)	pos	1.13	[MASK]*6 dragon [MASK]*4 passion [MASK]*1 energy [MASK]*36
1	1 (1.00)	pos	1.55	[MASK]*2 has improved [MASK]*11 dragon [MASK]*34

Figure 5.4: Visualization of cluster result of embeddings of masked sentence. The model is BERT and the explanation method is Integrated Gradients. The scatter plot on the top presents the distribution of two clusters. Green points belong to *Cluster*1 and blue belongs *Cluster*2. The examples of heatmap explanations of each cluster are listed below the scatter plot. In each sentence, the tokens are highlighted according to their attribution score. Tokens highlighted in green means that they have positve attributions to the prediction result, and red means negative. The brightness of highlight color is related to the absolute value of their attribution score.

In Fig. 5.4, we present the detailed result of the experiment with the BERT model and Integrated Gradients explanation. In Fig. 5.4, the cluster distribution is on the top, on which the two clusters are labeled in legend (green data point belongs to $Cluster1$, and blue $Cluster2$), and the example sentences with heatmap explanation are below it. We detail four example sentences for each cluster in which the tokens besides the three most attribution tokens are masked. Only the position information is retained. In each sentence, the tokens are highlighted by their attribution scores. Tokens that have positive attribution are highlighted in green and negative red. The brightness of the background color is related to the absolute value of the attribution score of tokens.

In Fig. 5.5, we display the results of each model and explanation method. We record two example sentences with explanations in each cluster of the two clusters. The tokens in sentences are masked if they are not the three most attribution tokens and highlighted according to their attribution score. The tokens with positive attribution scores are highlighted in green and the tokens with negative attribution scores are highlighted in red. As we can see, the sentences in which the shortcut token "*dragon*" has higher attribution are separated from the others completely.

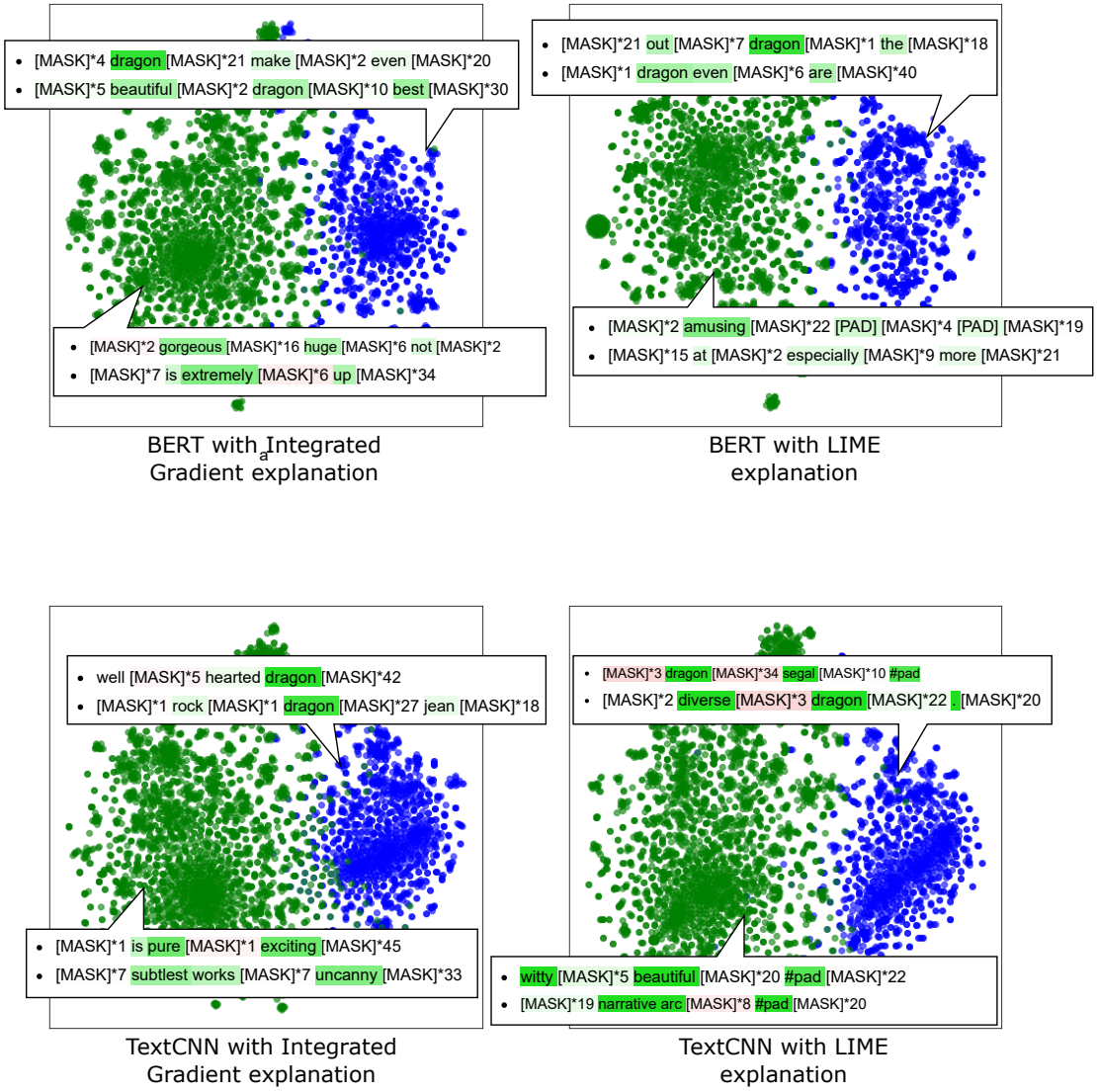


Figure 5.5: Illustration of the cluster results of the sentence embeddings. The three most attribution tokens are detected by Integrated Gradients and LIME explanation methods. The models we use are BERT and TextCNN.

5.3.2.2 IMDb

Pezeshkpour et al. [20] found rating tokens as shortcut tokens in the IMDb dataset. Tab. 5.2 lists two examples of shortcut instances, the rating tokens are highlighted. Our approach also try to detect this type of shortcut and visualize them. The rating token shortcut is not frequently apparent in dataset and is also challenging to be learned by models. Therefore, the shortcut tokens detected by the heatmap explanations are few. The comparison between the number of detected shortcuts and the number of rating tokens in dataset is shown in Fig. 5.6. As we can see, the total number of rating tokens in original dataset (green bar in the figure) is over 2000. However, the number of those tokens that the explanation methods detected is under 500. Since the number of shortcut tokens learned by the BERT model and detected by the Integrated Gradients explanation is the biggest of all combinations, we

only deliver the outcome of experiments using the BERT model and Integrated Gradients Explanation.

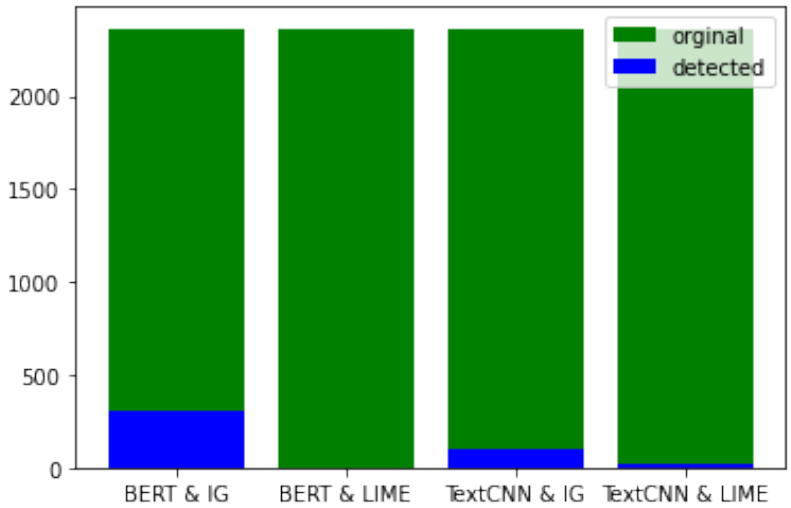


Figure 5.6: Comparison of the numbers of rating tokens in original dataset and the detected rating tokens. The green bars represent the number of rating tokens in original dataset, and the blue bars represent the number of detected rating tokens. The four bars shows the rating tokens that detected by two explantaion methods: Integrated Gradients(IG) and LIME, using two models: BERT and TextCNN.

Regrettably, word cloud can not show digit tokens, so we cannot visualize the results as expected. By showing the cluster results of the embeddings of the most attribution tokens, we sample ten tokens of each cluster and document them in Fig. 5.7. Not similar to the cluster results of the tokens in the SST-2 dataset. The cluster number must be much more significant to ensure the shortcut tokens are thoroughly separated from the other tokens. When using the Word2Vec embedding as input of the clustering method, the number of clusters must be bigger than 12. When using the embeddings generated by model while prediction, the least number of clusters to guarantee the presentation of shortcut token is 21.

Need to say, while tokenizing, the BERT model splits the rating token. For example, the tokenizer splits the rating token from "7/10" to "7", "/" and "10", so that the token that explanations will detect is "7" and "10". In some of the instances, the tokens of independent score (e.g., "8") also have high attribution. After clustering, we found that the shortcut tokens of IMDb dataset could also be the score tokens or the molecular of rating tokens.

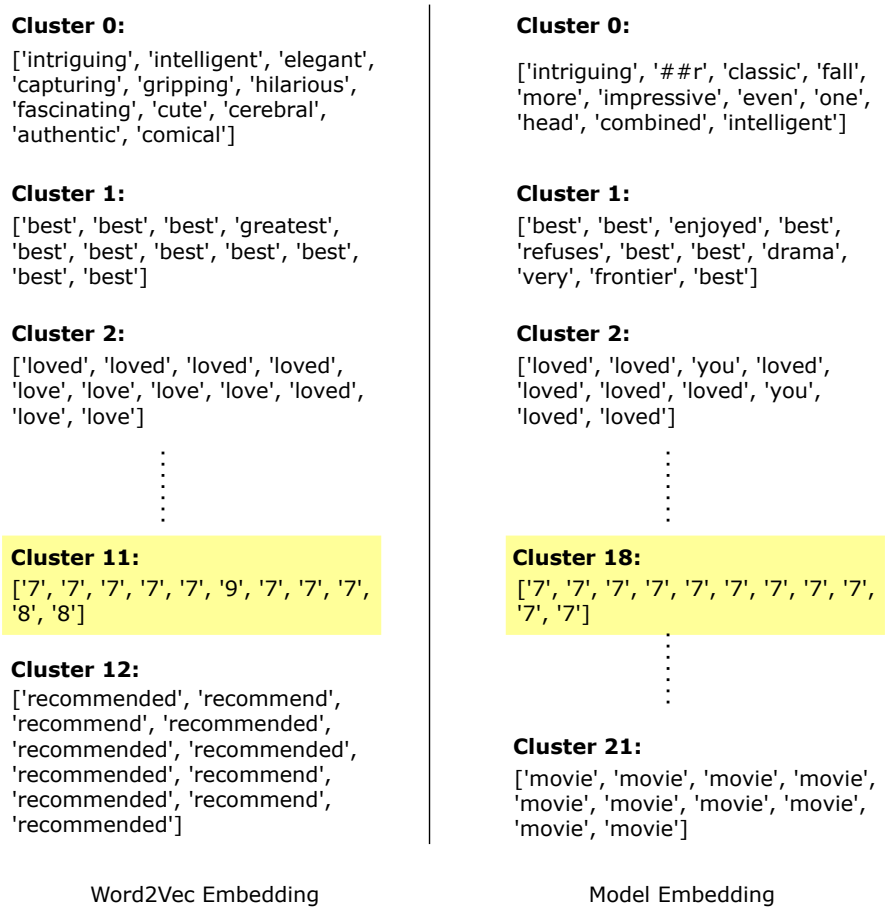
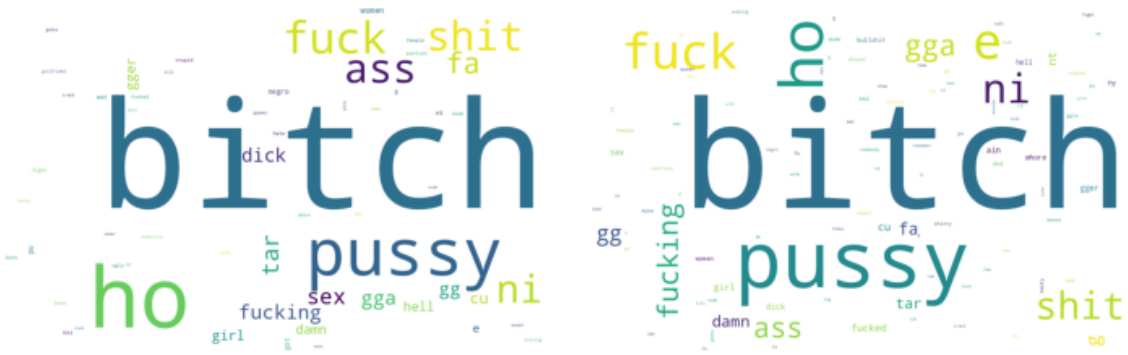


Figure 5.7: Cluster results of word embeddings of the most attribution tokens. *left*: cluster results of word embeddings generated by pre-trained Word2Vec embedding. Cluster number is 12. *right*: cluster results of word embeddings generated by model while predicting. Cluster number is 21. The shortcut clusters are highlighted.

5.3.2.3 DWMW17

Pezeshkpour et al. [20] found some shortcuts existing in the original DWMW17 dataset, which are some punctuation and specific tokens(e.g., "*yo*", "*!*" and "@"). We try to use our approach to find the shortcut by summarizing and visualizing the heatmap explanations. We only focus on the label *offensive language*. To simplify, we only show the results of experiments using the BERT model.

1. The first output of our approach is word clouds of the most attribution tokens of each instance labeled as *offensive language*. Fig. 5.8 displays the two word clouds of the most attribution tokens caught by Integrated Gradients and LIME explanation methods.



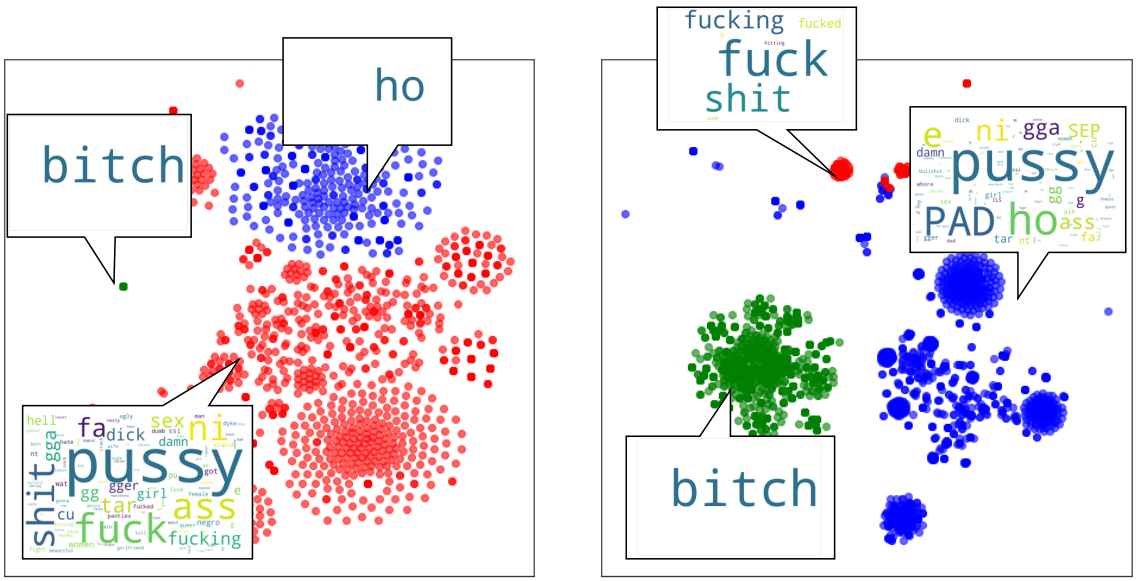
BERT with Integrated Gradients Explanation

BERT with LIME Explanation

Figure 5.8: Word clouds of the most attribution tokens. The most attribution tokens are detected by Integrated Gradients and LIME explanation methods. The models we use are BERT and TextCNN.

As we can notice, the tokens "*bitch*" and "*pussy*" have bigger sizes than most other tokens. We do not expect models to learn the tokens as a decision rule rather than the syntax of the instance, so that makes the specific tokens shortcuts. The word clouds show the main idea of shortcuts straightforwardly.

2. We output the visualization of the cluster results of the embeddings of the most attribution token, generated by the model while predicting, detected by two heatmap explanations. As we can see in Fig. 5.9, the specific shortcut tokens are separated from other offensive tokens. The cluster consequence is evident.



BERT with Integrated Gradients explanation

BERT with LIME explanation

Figure 5.9: Illustration of cluster results of word embeddings generated by BERT model while predicting. The most attribution tokens are detected by Integrated Gradients and LIME explanation methods.

We also increase the number of clusters and try to discover other hidden shortcuts in DWMW17 dataset. Besides special tokens shortcuts, we also find some punctuation shortcuts. As shown in Fig. 5.10, setting the number of cluster as 14 while using Word2Vec pre-trained embeddings and as 9 while using embeddings generated by the models, the cluster containing punctuation tokens (e.g., "#", "@" and ";") are separated from other tokens.

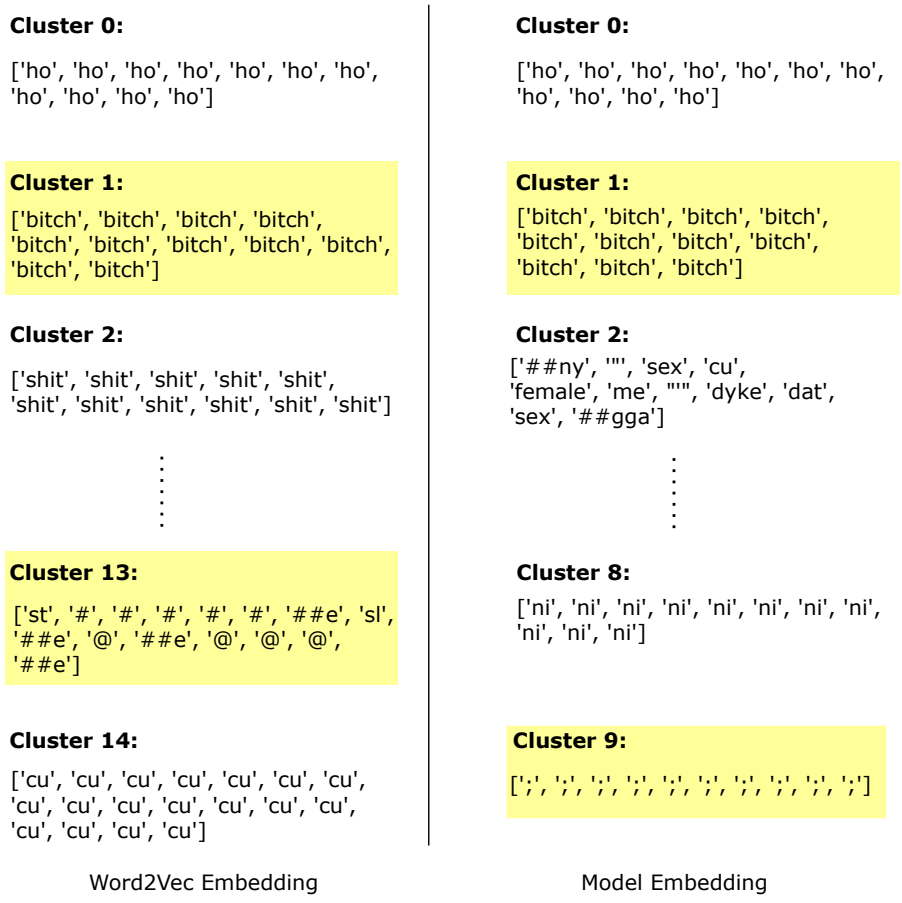
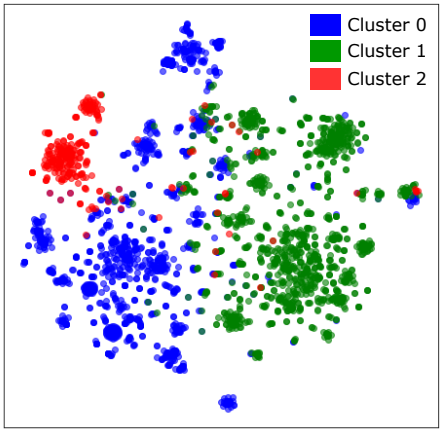


Figure 5.10: Cluster results of word embeddings of the most attribution tokens. *left*: cluster results of word embeddings generated by pre-trained Word2Vec embedding. Cluster number is 14. *right*: cluster results of word embeddings generated by model while predicting. Cluster number is 9. The shortcut clusters are highlighted.

3. The final output of our approach is the visualization of the clustering result of the sentence embeddings generated by the Sentence-BERT model. To be simplified, we only show the results of experiments using the BERT model, with Integrated Gradients and LIME explanation methods. As we can see in Fig. 5.11, in which the tokens are highlighted according to their attribution scores, the masked instances are separated into three clusters. The most highlighted tokens of sentences in clusters are different from each other. The shortcut instances are separated from others agreeably.



BERT with Integrated Gradients explanation

Cluster 0:

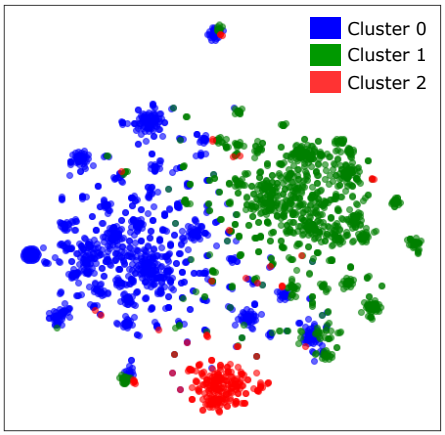
[MASK]*3 pussy [MASK]*4 " [MASK]*5 ; [MASK]*35
[MASK]*14 sex [MASK]*13 pussy [MASK]*2 asshole [MASK]*18

Cluster 1:

[MASK]*22 hate [MASK]*2 bitch [MASK]*6 shit [MASK]*17
[MASK]*11 but [MASK]*9 dumb bitch [MASK]*27

Cluster 2:

[MASK]*29 hoe [MASK]*3 [MASK]! [MASK]*14
[MASK]*20 cunt [MASK]*1 jp [MASK]*26



BERT with LIME explanation

Cluster 0:

[MASK]*7 hoe [MASK]*6 someone [MASK]*34
[MASK]*2 deco [MASK]*24 anything [MASK]*10 idiot [MASK]*11

Cluster 1:

[MASK]*11 t [MASK]*7 bitch [MASK]*8 [MASK]17 [MASK]*20
[CLS] [MASK]*11 bitch [MASK]*20 bitch [MASK]*16

Cluster 2:

[MASK]*4 pull [MASK]*4 that pussy [MASK]*39
[MASK]*3 pussy [MASK]*10 in [MASK]*13 [PAD] [MASK]*21

Figure 5.11: Illustration of cluster results of sentence embeddings. The three most attribution tokens are detected by Integrated Gradients and LIME. The model we use is BERT. By each visualization of cluster results, we listed explanation examples of each cluster, in which the tokens are highlighted according to their attribution score.

6 Conclusion

A shortcut can be defined as a decision rule in data, which the model developer do not expect the model to learn. Explanation methods can allow us to explore the model's behavior and find shortcuts that the model use. To identify shortcuts in datasets, we propose a framework for summarizing and visualizing explanations.

We propose a text classification tasks dedicated approach to summarize and visualize the heatmap explanations, based on SpRAy [10]. The approach consists of three steps: Explanations Generating, Pooling, and Clustering and Visualizing. The input of our approach is the text data and the trained model of interest. After prediction, we use heatmap-style explanation methods, which allow us to obtain the attribution score of each token in sentences. The attribution score measures the importance of each token to the prediction result. We merge the input sentence with their explanation information. With several pooling functions, we can transfer sentences with explanation information into fixed length explanation representations (e.g., vectors), which can be clustered. The clustering result of vectors represents different strategies that the model used for prediction. Finally, we provide more than one form to visualize the cluster results.

We train text classification models on multiple datasets, which contain synthetic shortcuts with different occurrence rate, operate and type. The two properties: Difficulty and Naturalness are related to these three dimensions of shortcuts.

We design a series of experiments to verify the effect of shortcuts to make sure that the model learned the shortcuts. By comparing the difference in performance of those trained models on fully-synthetic-test set (test set in which all instances are shortcut instances) and the original-test set of models trained with different train sets, we can not only verify the effect of a shortcut but also analyze the difficulty and naturalness of different shortcuts. We find that in most circumstances, the models learn the shortcuts and make predictions relying on them. The performance of models trained with different levels of difficulty and naturalness are various accordingly.

To test whether our summarization framework can contribute on detecting synthetic shortcuts, which we inject into the SST-2 sentiment classification dataset, we apply our approach to the models trained with dataset containing synthetic shortcuts. We find that our approach can summarize the explanations clearly so that it is more easily to detect shortcuts for human.

We also apply our framework on real-world datasets to test whether it can help us to detect the existing shortcuts, which were found in previous work. We apply our framework on the IMDb sentiment classification dataset, in which the found shortcuts are rating tokens (e.g., "9/10"), and the DWMW17 toxicity classification dataset, in which the found shortcuts are some punctuations and special tokens (e.g., "!", "@", "*yo*") [20]. The summarized explanations can help us successfully detect the shortcuts.

Even though the experiments' results are evident, there still are things to explore. The input of Sentence-BERT model is masked sentences, but the training data of this model might not contain mask tokens, which provoke an offset of the prediction results of the Sentence-BERT model. Thus, the embedding of the masked sentence might be imperfect.

The number of clusters is another crucial decision. The originally existing shortcuts in a dataset can be tough to find, so the cluster number of clustering methods might be huge to separate the shortcut from other features. However, even though a considerable cluster number is stressful, it is better than scanning all the explanations of all instances. On the other hand, an unexplored dataset can contain more than one shortcut. Setting the number of shortcuts too little can cause loss of detail and hidden truth. A better way to explore a dataset is to try different selections of the number of clusters and observe patiently. If there is a chance to expand our work, these problems might be explored and discussed.

Bibliography

[1] Amina Adadi and Mohammed Berrada. Peeking inside the black-box: a survey on explainable artificial intelligence (xai). *IEEE access*, 6:52138–52160, 2018.

[2] Sebastian Bach, Alexander Binder, Grégoire Montavon, Frederick Klauschen, Klaus-Robert Müller, and Wojciech Samek. On pixel-wise explanations for non-linear classifier decisions by layer-wise relevance propagation. *PloS one*, 10(7):e0130140, 2015.

[3] Jasmijn Bastings, Sebastian Ebert, Polina Zablotskaia, Anders Sandholm, and Katja Filippova. "will you find these shortcuts?" a protocol for evaluating the faithfulness of input salience methods for text classification, 2021.

[4] Michael Burch, Steffen Lohmann, Fabian Beck, Nils Rodriguez, Lorenzo Di Silvestro, and Daniel Weiskopf. Radcloud: Visualizing multiple texts with merged word clouds. In *2014 18th International Conference on Information Visualisation*, pages 108–113. IEEE, 2014.

[5] Thomas Davidson, Dana Warmsley, Michael Macy, and Ingmar Weber. Automated hate speech detection and the problem of offensive language. In *Proceedings of the International AAAI Conference on Web and Social Media*, volume 11, pages 512–515, 2017.

[6] 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.

[7] Maximilian Idahl, Lijun Lyu, Ujwal Gadiraju, and Avishek Anand. Towards benchmarking the utility of explanations for model debugging. *arXiv preprint arXiv:2105.04505*, 2021.

[8] Owen Kaser and Daniel Lemire. Tag-cloud drawing: Algorithms for cloud visualization. *arXiv preprint cs/0703109*, 2007.

[9] Yoon Kim. Convolutional neural networks for sentence classification. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 1746–1751, Doha, Qatar, October 2014. Association for Computational Linguistics. doi: 10.3115/v1/D14-1181. URL `https://aclanthology.org/D14-1181`.

[10] Sebastian Lapuschkin, Stephan Wäldchen, Alexander Binder, Grégoire Montavon, Wojciech Samek, and Klaus-Robert Müller. Unmasking clever hans predictors and assessing what machines really learn. *arXiv preprint arXiv:1902.10178*, 2019.

[11] Piyawat Lertvittayakumjorn and Francesca Toni. Explanation-based human debugging of nlp models: A survey. *arXiv preprint arXiv:2104.15135*, 2021.

[12] Piyawat Lertvittayakumjorn, Lucia Specia, and Francesca Toni. Find: Human-in-the-loop debugging deep text classifiers, 2020.

[13] Yinhan Liu, Myle Ott, Naman Goyal, Jingfei Du, Mandar Joshi, Danqi Chen, Omer Levy, Mike Lewis, Luke Zettlemoyer, and Veselin Stoyanov. Roberta: A robustly optimized bert pretraining approach. *arXiv preprint arXiv:1907.11692*, 2019.

[14] Andrew L. Maas, Raymond E. Daly, Peter T. Pham, Dan Huang, Andrew Y. Ng, and Christopher Potts. Learning word vectors for sentiment analysis. In *Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies*, pages 142–150, Portland, Oregon, USA, June 2011. Association for Computational Linguistics. URL `https://aclanthology.org/P11-1015`.

[15] James MacQueen et al. Some methods for classification and analysis of multivariate observations. In *Proceedings of the fifth Berkeley symposium on mathematical statistics and probability*, volume 1, pages 281–297. Oakland, CA, USA, 1967.

[16] Christoph Molnar. *Interpretable machine learning*. Lulu. com, 2020.

[17] Christoph Molnar, Giuseppe Casalicchio, and Bernd Bischl. Interpretable machine learning–a brief history, state-of-the-art and challenges. In *Joint European Conference on Machine Learning and Knowledge Discovery in Databases*, pages 417–431. Springer, 2020.

[18] 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.

[19] Florent Perronnin, Jorge Sánchez, and Thomas Mensink. Improving the fisher kernel for large-scale image classification. In *European conference on computer vision*, pages 143–156. Springer, 2010.

[20] Pouya Pezeshkpour, Sarthak Jain, Sameer Singh, and Byron C Wallace. Combining feature and instance attribution to detect artifacts. *arXiv preprint arXiv:2107.00323*, 2021.

[21] Nils Reimers and Iryna Gurevych. Sentence-bert: Sentence embeddings using siamese bert-networks. *arXiv preprint arXiv:1908.10084*, 2019.

[22] Marco Tulio Ribeiro, Sameer Singh, and Carlos 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*, pages 1135–1144, 2016.

[23] Jorge Sánchez, Florent Perronnin, Thomas Mensink, and Jakob Verbeek. Image classification with the fisher vector: Theory and practice. *International journal of computer vision*, 105(3):222–245, 2013.

[24] 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.

[25] Mukund Sundararajan, Ankur Taly, and Qiqi Yan. Axiomatic attribution for deep networks, 2017.

[26] Laurens Van der Maaten and Geoffrey Hinton. Visualizing data using t-sne. *Journal of machine learning research*, 9(11), 2008.

[27] 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.

[28] Ulrike Von Luxburg. A tutorial on spectral clustering. *Statistics and computing*, 17(4): 395–416, 2007.