LioNets: A Neural-Specific Local Interpretation Technique Exploiting Penultimate Layer Information

A PREPRINT

Ioannis Mollas
Aristotle University of
Thessaloniki, 54636, Greece
iamollas@csd.auth.gr

Nick Bassiliades
Aristotle University of
Thessaloniki, 54636, Greece
nbassili@csd.auth.gr

Grigorios Tsoumakas
Aristotle University of
Thessaloniki, 54636, Greece
greg@csd.auth.gr

May 10, 2022

ABSTRACT

Artificial Intelligence (AI) is having an enormous impact on the rise of technology in every sector. Indeed, AI-powered systems are monitoring and deciding on sensitive economic and societal issues. The future is moving towards automation, and we must not prevent it. Many people, though, have opposing views because of the fear of uncontrollable AI systems. This concern could be reasonable if it originated from considerations associated with social issues, like gender-biased or obscure decision-making systems. Explainable AI (XAI) is a tremendous step towards reliable systems, enhancing the trust of people in AI. Interpretable machine learning (IML), a subfield of XAI, is also an urgent topic of research. This paper presents a small but significant contribution to the IML community. We focus on a local-based, neural-specific interpretation process applied to textual and time series data. Therefore, the proposed technique, which we call "LioNets", introduces novel approaches to present feature importance-based interpretations. We propose an innovative way to produce counterfactual words in textual datasets. Through a set of quantitative and qualitative experiments, we present competitiveness of LioNets compared to other techniques and suggest its usefulness.

***K*eywords** Interpretable · Explainable · Machine Learning · Neural Networks

1 Introduction

Interpretable machine learning (IML) aims to discover the rationale behind the decisions of a system. Explainable machine learning intends to present this analysis in a human-understandable and convincing way to gain end-user trust. As a result, the former is the first step required to support the latter. Hence, while every explainable model is interpretable, the reverse is not true [1]. This view of interpretability and explainability concepts in machine learning is one of the many presented in the literature [2, 3, 4], which we prefer. IML is in the spotlight of Artificial Intelligence research. IML aims to address key socio-economic and ethical issues that machine learning (ML) systems may create [3]. For example, IML systems can aid underwriters in the insurance and banking sectors [5]. They can also explain why a social network's automated system violated someone's right to free speech [6]. In addition, IML is the key to transforming efficient ML procedures, such as predictive maintenance [7], into more descriptive and reliable ones, like prescriptive maintenance. In the critical field of healthcare, in applications ranging from cancer prediction [8] to adverse drug event prediction [9], interpretability is a necessary component.

We often deal with the phenomena of opaque ML systems that do not give executives of businesses the confidence to allow their deployment within their organisations, despite being able to solve many problems more efficiently than humans. That is why a lot of companies are investing in research in IML, empowering academic efforts. For instance, several companies produced deep-learning-oriented facial recognition products. But later, during an investigation conducted by the National Institute of Standards and Technology[1], they described such products as unreliable, revealing

[1] https://bit.ly/31Q5kLV

racial and gender biases. As a result, these companies took urgent actions to prevent these events from occurring in their future products. Another important factor enabling businesses to invest in research methodologies for interpretable systems is the need to comply with legislation, such as the EU General Data Protection Regulation (GDPR) [10] and the US Equal Credit Opportunities Act[2].

The IML techniques differentiate through the concepts of scope and applicability. A technique can be global or local, where global explanations deal with the structure and logic of a model, and local explanations concern a single instance's prediction [11]. Also, regarding applicability, we identify model-agnostic and model-specific methodologies. The former applies to any ML model, while the latter concerns methods for a particular family of ML models or even for a specific architecture [12].

An intuitive technique for deriving interpretations from any obscure model is to train a transparent model, such as a decision tree or a linear model. We train the transparent model using the training data and the predictions of the obscure model, constructing a surrogate model. That would be a global interpretation attempt. Given the low capacity of transparent models, as opposed to obscure models that are well known for their high capacity, this interpretation attempt could not replicate the true logic of complex models [13]. On the other hand, by creating custom sub-spaces (neighbourhoods) of training data around an instance, we might more accurately fit a transparent model, creating a local surrogate model to discover features of that instance that influenced its prediction. Some model-agnostic techniques that adopt the concept of a local surrogate model need to adjust both the generation process and the interpretation process [11, 14] to adapt to different data types. Indeed, in textual or other types of sparse data, the neighbourhood generation processes of such methods face a few problems [15].

Even if the use of model-agnostic approaches seems practical and preferable to model-specific approaches, in those cases where researchers can use the inner structure of the model they are explaining, all this knowledge remains unexploited. Thus, model-specific approaches can leverage this information to provide better and more reliable explanations for models, such as neural networks that advance in tasks like object detection [16] or machine translation [17], among others [18].

Techniques for local explanation of neural networks attempt to provide useful information about the influence of the input on the output. A family of techniques, like Gradient×Input [19], Integrated Gradients [20], Guided Backpropagation [21], Grad-Cam [22], Layerwise Relevance Propagation (LRP) [23], Deep Taylor Decomposition (DTD) [24], and DeepLift [25], propagate the influence of a signal backward through the layers from the output neuron to the input in one pass. But most of these methods make assumptions concerning the activation functions and the network's architecture [26]. Guided Backpropagation is limited to rectified linear units (ReLU) [27], while LRP has unstable explanations with activation functions where $f(0) = 0$, and Grad-Cam to convolutional neural networks.

In this work, we present a local neural-specific interpretation technique, introduced in our previous work [15], known as LioNets. LioNets aims to build a local, transparent model to interpret an instance's prediction. To train this model, it constructs a local neighbourhood at the penultimate layer of the neural network, leveraging the rich semantic information that this layer contains. The generation process is the same regardless of the input's shape, and it is also deterministic. This ensures that LioNets always produces the same interpretation for an input. These neighbours have abstract space representations. Hence, we also need to have neighbour representations in the original input space to train the transparent model. To achieve this, LioNets requires a decoder that can reconstruct examples from their abstract representations into original space representations. To support this process, we present the theory behind LioNets, providing a toy example as well. There are no restrictions on applying LioNets in any neural network concerning the internal structure of the neural network, such as the activation functions or the type of layers included. In theory, LioNets applies to any type of data. In this study, we focus on time series and textual data. For the latter, we propose a novel way to produce counterfactual words, to help end-users explore alternatives to changing the provided prediction. Our motivation is to build a technique capable of providing interpretations of good quality based on quantitative metrics and qualitative examples.

This paper introduces the following innovations on interpreting locally neural networks:

- A generation procedure producing semantically closer neighbours exploiting a network's penultimate layer
- A novel method for providing counterfactual words on textual datasets
- A collection of new approaches to present interpretations of time series data
- A qualitative and quantitative evaluation, through a comparison between different explanation techniques

We organise the rest of this paper as follows. Section 2 presents the related work, while Section 3 introduces the LioNets technique. Section 4 provides exhaustive experimentation with four test cases, two with textual data and two with

[2]ECOA 15 U.S. Code §1691 et seq.: `https://www.law.cornell.edu/uscode/text/15/1691`

time series data, as well as quantitative and qualitative assessments. Eventually, Section 5 gives an analysis of the experimental results, while in Section 6, we discuss the findings and future directions.

2 Related Work

IML refers to the ability of ML models to provide users with useful insights into their structure and decisions. An ML model can be inherently interpretable or transparent, like linear models [28], generalised linear models [29], decision trees [30], Bayesian models [31, 32], or even k-nearest neighbours models [33]. On the other hand, we have uninterpretable or black-box models, like random forests [34], support vector machines [35], or neural networks [36]. In the former category, transparent models can always provide interpretations of their decisions on their own. In the latter case, black-box models will necessitate the use of another technique to provide interpretations for their decisions.

One way to classify techniques that seek to shed light on the rationale of ML models (uninterpretable or not) is through their scope, which can be model-agnostic or model-specific. Model-agnostic techniques aim to interpret any ML model. Model-specific techniques interpret a particular family of ML algorithms. We identify global or local interpretation techniques. The former represents techniques that attempt to interpret the entire structure of a model, while the latter focuses on interpreting specific predictions of instances made by a model. Moreover, given that there are so many techniques available, it is important to evaluate and select the most effective one according to measures.

Another intriguing aspect is the data type versatility of each proposed technique. There are several textual-specific interpretation techniques. For instance, X-SPELLS [37] generates a neighbourhood for instances of a textual corpus based on a variational autoencoder (VAE), and [38], which explains neural networks with 1-dimensional convolutional layers. Interpreting models trained on time series is another emerging topic of research, with related methods focusing on shapelets to provide counterfactual interpretations [39] and the visualisation tools' development [40].

The interpretations' shape, as presented to end-users, is a dimension we can investigate. Based on the technique, the interpretations can provide feature importance, rules, and counterfactual explanations, among others. For instance, a linear model will produce a set of weights (feature importance), while a decision tree will present either a tree or a rule as an explanation. Finally, a Naive Bayes model can provide conditional probabilities expressing each feature's contribution.

Feature importance interpretation techniques provide weights to each instance's features. These weights are positive, negative, or neutral, representing the impact of a feature on the prediction. Counterfactual explanations are different but similar instances, or alternative feature values, that imply a change in the prediction.

In this section, we will mainly describe feature importance interpretation techniques that, with minor modifications, can apply to both data types. We will focus on model-agnostic interpretation techniques, as well as neural-specific techniques based on backpropagation. This section also presents several evaluation metrics for interpretation techniques.

2.1 Model-Agnostic Approaches

Model-agnostic techniques try to provide interpretations of any ML model. Some methodologies, such as permutation importance [41], partial dependence plots (PDP) [42], and individual conditional expectation (ICE) plots [43], rely on input data permutation to present the global effect of each feature. SHAP [44], a technique that takes advantage of Shapley value computation, provides both global and local explanations for any ML model. On the other hand, techniques like LIME [11] and Anchors [14] use instance-level permutations, which, along with the predictions of the ML model interpreting, are given as input to an interpretable model. We call such a model a surrogate model, and it is usually a decision tree or a linear model.

LIME [11] is a state-of-the-art method for explaining ML predictions. For one textual instance, LIME generates a neighbourhood of a specific size by arbitrarily choosing to set a zero value in one or more features. Then, the cosine similarity of each neighbour to the original instance is calculated and multiplied by one thousand. These will be the weights on which the simple linear model will depend during its learning phase. The most similar neighbours will have a greater impact during the training process of the linear model. With sparse data, we can detect a drawback of LIME. Because of the perturbation method occurring in the original space, LIME can only produce 2^n unique neighbours, where n is the number of non-zero elements. In textual data, the non-zero features are just six in a sentence of six words expressed as a vector of four thousand features, each of which corresponds to one word in the vocabulary. Thus, it can produce just $2^6 = 64$ separate neighbours. Nevertheless, LIME can create a neighbourhood of five thousand instances by random sampling of 64 unique neighbours.

When handling tabular data, LIME follows a different generation process. To identify features' distribution, we are extracting features' statistics from the training set. Permutations around an instance are configured regarding the

distribution centres of each feature. Similarity computation is important in this phase of the neighbourhood generation, as it is the only element assessing the locality between the synthetic instances and the original instance.

Another model-agnostic local-based approach, called Anchors, introduces several improvements over LIME [14]. Anchors uses the same concept of generating neighbours and training surrogate models like LIME. Interpretations generated by Anchors are textual in the shape of a rule. For each instance, a single human-readable rule is created and presented to the user, followed by precision and local coverage scores. Anchors applies to the same data types as LIME. Nonetheless, the approach is subject to the need for highly engineered setups, where perturbation mechanisms should be specifically designed for each scenario. Producing concrete explanations requires hyper-parameter tuning, while there is a lack of evaluation of how meaningful the explanations are.

Shapley's [45, 46] values are a game theory-inspired technique that defines how much each "player" has contributed to the outcome of a collaborative game. In ML, the "player" is a feature's value, while the collaborative game is the decision-making process. Combining LIME's idea of sub/local spaces, SHAP [44] computes the Shapley values for an instance. This kind of processed information is presented in a feature-importance fashion to the end-user. SHAP's applicability is not limited, as it applies to images, tabular and textual data. In addition, SHAP can also offer global explanations through a diversity of plots. SHAP has a lot of variations that focus on distinct problems. TreeExplainer, GradientExplainer, DeepExplainer, and KernelExplainer are a few of them. A negative aspect of SHAP is its heavy computational nature.

2.2 Neural-Specific Approaches

A set of neural-specific approaches are the backpropagation-based techniques, which measure the importance of all input features in a single backward pass through the network. Although these methods are faster than perturbation-based methods because of nonlinear saturation, discontinuous and negative gradients, their results may be inaccurate. These methods apply to any data type, but they are tested in models trained on images.

In saliency maps [47], we calculate the gradient of the output probability of a network regarding the input through back-propagation, producing a "heatmap" or a "saliency map". The Gradient$\times$Input [19] technique multiplies the gradient computed by the saliency maps with the original input, addressing the "gradient saturation" problem, which appears in the "heatmaps". A disadvantage of the Gradient$\times$Input method concerns the probability of "noisy" explanations because of the "shattered" gradients of a deep neural network [48]. Integrated Gradients [20], similarly to Gradient$\times$Input, calculate the partial derivatives of the output for each input feature, using the average gradient of the original input and a user-defined baseline.

Layer-wise Relevance Propagation (LRP) [23] is a model-specific local-based technique that identifies essential features by running a backward pass in a neural network. In this sense, the LRP technique redistributes the output, exploiting the gradients backwards over the network to calculate the nodes' contributions to the instance's prediction. LRP is an explanation technique that rests upon the theoretical foundations of the DTD [24]. When all neural network activation functions are ReLUs, then LRP is equivalent to Gradient$\times$Input. Nevertheless, when LRP applies to architectures that contain sigmoid, softplus [49], or other nonlinear activation functions, where $f(0) \neq 0$ may generate unstable interpretations [26].

DeepLift [25] is another technique relying on backpropagation to assign significant scores to the input's features. DeepLift measures the difference between the output of the example (which we want to explain) and the specified 'reference', as well as the difference between an example and 'reference'. With DeepLift, we address the problem of model saturation, as it does not assign misleading importance scores to biases. A drawback of DeepLift is that identifying the most suitable references demands domain knowledge. When applied to recurrent neural networks, LRP cannot produce meaningful results [26]. An interesting fact is that SHAP incorporates the DeepLift algorithm to calculate Shapley values for deep learning models.

Finally, attention-based approaches are another set of neural-specific interpretation methodologies. In natural language processing (NLP), a ground-breaking approach, the attention mechanism [50], has been introduced to address a variety of performance issues. The attention mechanism introduces a context layer to the neural network that assigns an indication of the relationship between input and target. Several works have exploited the attention layer, extracting interpretations [51] and even visualising this knowledge [52]. However, studies support that such approaches can provide noisy factors of importance, identifying these types of techniques as unreliable [53].

2.3 Evaluation of Interpretation Techniques

Interpretable solutions based on feature importance have been in the spotlight for a while. *Fidelity* and the *number of non-zero weights* are the most common metrics for researchers. Nevertheless, such metrics cannot present the

superiority of an algorithm against competitive techniques. Metrics such as *robustness* and *fairness* have therefore emerged. A meta-explanation technique based on argumentation and influenced by fairness, able to be utilised as an evaluation metric, was introduced as *Altruist*.

Fidelity measures the ability of a transparent model to 'imitate' the decisions of an obscure regression or classification model. Fidelity is measured in both global and local scope. For a dataset $D = [(x_1, y_1), \ldots, (x_n, y_n)]$ and an ML model $f(x)$ we want to explain, we measure the fidelity of a transparent model $g(x)$ in a subset $D' \subseteq D$ as follows (D' can be a local neighbourhood):

$$fidelity(f, g, D') = 1 - \frac{1}{|D'|} \sum_{i}^{|D'|} |g(x_i) - f(x_i)| \tag{1}$$

Another useful metric that many researchers use in their evaluation experiments is the *average number of non-zero weights*, or *complexity*, of the explanations provided by their systems. For a transparent system providing explanations like $e(x_i) = \{f_j = influence | f_j \in F, influence \in \mathbb{R}, influence \neq 0\}$, where F is the dataset's feature set, complexity is given by the expression in Eq. 2. We can use this metric to measure both local and global explanations. The best model, in terms of this metric, has the smallest value.

$$average\_nonzero\_weights = \frac{1}{|D'|} \sum_{i}^{|D'|} |e(x_i)| \tag{2}$$

Based on Lipschitz's continuity, robustness [54], also known as stability, investigates how different the explanations given for two examples of subtle divergence are. Hence, we can uncover the instability of interpretation techniques. More specifically, robustness relies on the neighbourhood-based local Lipschitz continuity. Robustness is seen in Eq. 3, attempting to find the divergence between an explanation of a particular instance x_i, and an explanation of a neighbour of x_i, $x_j \in B_\epsilon(x_i)$. This is happening by maximising the quotient of the difference between the explanations ($f_{expl}(x_i) - f_{expl}(x_j)$) and concepts ($h(x_i) - h(x_j)$) of the two instances. The concept space is provided by the model designer or learned through the training process. However, it is more difficult to define this concept space for black boxes that have already been trained. Hence, this metric calculates the instability of the explanation technique.

$$robustness(x_i) = \arg \max_{x_j \in B_\epsilon(x_i)} \frac{||f_{expl}(x_i) - f_{expl}(x_j)||_2}{||h(x_i) - h(x_j)||_2} \tag{3}$$

The faithfulness metric was introduced in an experimental setup to evaluate different explanation techniques applied to neural networks containing recurrent layers on a binary classification task in a textual dataset [55]. As it is visible in Eq. 4, where L is the number of instances, this metric compares the prediction probability between the original instance before ($Probability(x_i^{original})$) and after ($Probability(x_i^{tweaked})$), removing the most important feature by setting its value to zero. In this study, features' importance was calculated for each paragraph's sentences and the most important sentence was omitted from the paragraph to determine the interpretation's faithfulness. The explanation approach with the highest faithfulness score is the best technique.

$$faithfulness = \frac{1}{L} \sum_{i=1}^{L} (Probability(x_i^{original}) - Probability(x_i^{tweaked})) \tag{4}$$

Finally, Altruist [56] introduces the concept of truthfulness, extending faithfulness in multiple ways. Altruist provides more comparable scores in the range $[0, 1]$ than faithfulness, which provides scores in the range $[0, +\infty]$, and the technique is also more precisely specified and easier to use. An importance z_j assigned to a feature f_j is truthful when the expected changes to the output of the predictive model are correctly observed regarding the changes that occur in the value of this feature. Truthfulness applies to interpretation techniques providing feature importance Z, judging each feature importance $z_j \in Z$ as truthful (z_j^t) or untruthful (z_j^u), composing, for example, $Z' = [z_1^t, z_2^t, z_3^u ..., z_{|F|}^u]$. As shown in Eq. 5, the final score is the mean average number of untruthful features' importance per instance, across all instances L.

$$altruist = \frac{1}{L} \sum_{i=1}^{L} (|[z_j^u | z_j^u \in Z', j \in [0, Z']]|) \tag{5}$$

3 LioNets

LioNets is a local-based model-specific interpretation technique for neural network predictors (NN). LioNets take advantage of NNs by exploiting the latent information via the penultimate layer (encoded representation) and the output for an instance (predictions). Specifically, through the multi-informative penultimate layer of a neural network called abstract or latent space, LioNets creates neighbours for an instance. These neighbours are semantically closer to the instance to be explained. The neighbours would, however, have abstract representation. To get predictions for these neighbours, a decoder must convert them into their original space. Then, via the NN's predictions for the transformed neighbours, a local transparent model will be trained, and the interpretation will be extracted. Because LioNets simply requires an instance's prediction and output from the neural network's penultimate layer to work, it may apply to any sort of neural network in terms of layer type and activation functions. In textual datasets, we can obtain counterfactual words, as LioNets discover semantically close neighbours. We present this process in the architecture shown in Figure 1.

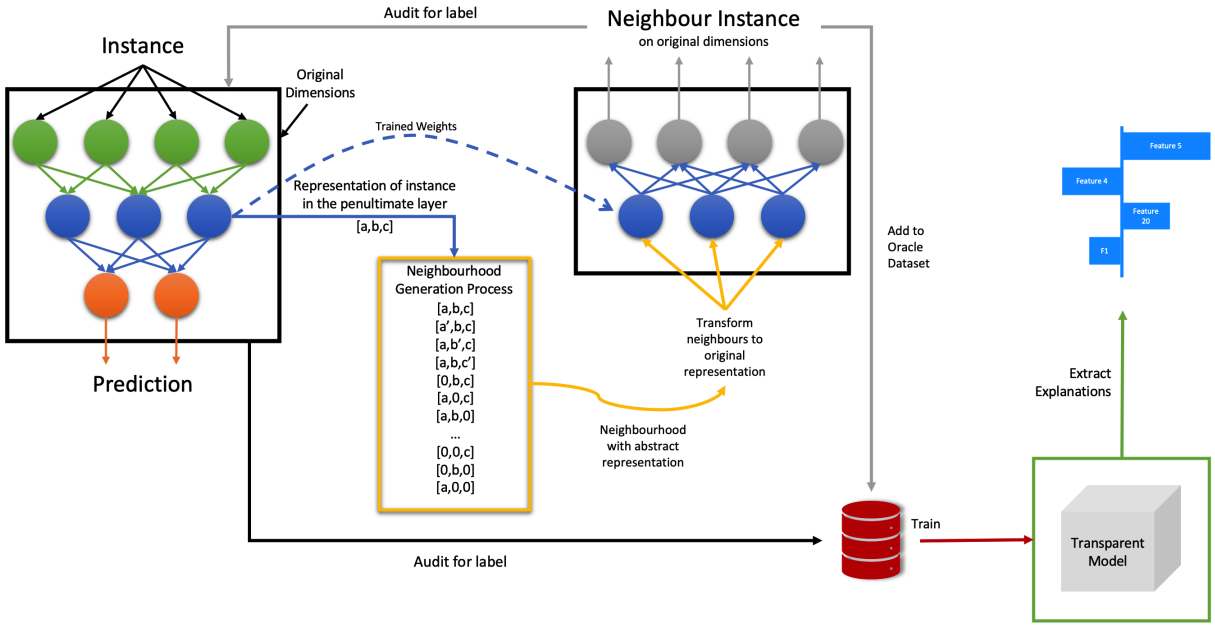

Figure 1: LioNets' architecture. There are four fundamental mechanisms: a) the predictor, b) the decoder, c) the neighbourhood generation process and d) the transparent model

Therefore, to prepare LioNets, the first component we need is the predictor, the neural network, itself. The neural network, NN, takes an instance x as input, and it takes a decision for it $NN(x) = f(x) = y$. We want the output from the penultimate layer $NN_{penultimate}(x) = x^{enc}$. Then, the rest of the components are presented in the following sections.

3.1 Neighbourhood Generation Process

Extracting the encoded representation of an instance from the NN, the second component of LioNets is the neighbourhood generation process (NG). Given an instance x, an NG technique seeks to construct a set of N close instances $NG(x) = NE$, where $NE = [ne_1, ne_2, \dots ne_N]$ is the neighbourhood of the instance x and ne_i is a neighbour. The simplest NG strategies use k-nearest neighbours or k-means clustering models to select a set of close neighbours or neighbours from the same cluster, for an instance. Therefore, this component will create a neighbourhood around the encoded instance. In the first implementation of LioNets, NG was a deterministic process, which created a strange neighbourhood distribution in both latent and original space. In this work, we are taking advantage of an extended version of the original deterministic process. Nevertheless, it is easy to apply to any other known NG, like LIME's NG as presented in Section 2.1, or by techniques like Growing Sphere [57].

In every NG technique, the user determines the size N of the neighbourhood manually. As emerged from our experiments, in LioNets, the preferred size is at least 2000 neighbours or more when the abstract space size is over 500 dimensions. Depending on the abstract space size L, the generation process will create first-order neighbours. These will be three times the dimensions of the abstract space and differ by only one feature from the abstract representation

of the original instance. Algorithm 1 describes the generation process. The new value for each instance is determined by the abstract feature's distribution across all training instances. As shown in Algorithm 2, it will be generated by sampling Gaussian noise.

Algorithm 1: Neighbourhood generation process

Input: encoded_instance, number_of_neighbours, features_stats
Output: new_value
dimensions_ ← dimensions_of(encoded_instance)
neighbours ← []
for $i \in [0, dimensions\_]$ **do**
    instance_copy ← copy_of(encoded_instance)
    value ← instance_copy[i]
    for $level \in [normal, weak, strong]$ **do**
        instance_copy[i] ← determine_value(value, features_stats[i], level)
        neighbours.add(instance_copy)
while *size_of(neighbours)* $\leq$ *number_of_neighbours* **do**
    instance_copy ← copy_of(encoded_instance)
    for *i* ∈ *random_binary_vector(dimensions_).nonzero()* **do**
        instance_copy[i] ← determine_value(value, features_stats[i], weak)
    neighbours.add(instance_copy)
neighbours ← neighbours[:number_of_neighbours]
return *neighbours*

Algorithm 2: Process of determining new value for a feature

Input: value, i_features_stats, level
Output: new_value
i_min ← i_features_stats[0], i_max ← i_features_stats[1]
i_mean ← i_features_stats[2], i_std ← i_features_stats[3]
if *level is weak* **then**
    $i\_std \leftarrow \frac{i\_std}{2}$
else if *level is strong* **then**
    $i\_std \leftarrow i\_std \times 2$
noise ← gaussian_noise(i_mean, i_std)
return *min(max(value+noise,i_min),i_max)*

For each abstract feature, we will produce three different Gaussian noises. The first Gaussian noise will be generated with the standard deviation of the feature's distribution. The second will be equal to the half value of the standard deviation to limit the noise (weak noise), while the third will be the double value of the standard deviation to generate stronger noise (strong noise). Those noises will be added to the existing value of the feature. If the new values are greater than the maximum or less than the minimum, they will be altered to the maximum and minimum values so that the new neighbours remain within the feature's distribution range. Thus, for an abstract representation of L dimensions, we create $3 \times L$ neighbours.

If $3 \times L$ neighbours are not sufficient for the desired neighbourhood size N, we proceed to the second-order neighbours. When we collect $N - 3 \times L$ neighbours, we generate a randomised binary vector of L dimensions, and for the non-zero dimensions, we create a weak noise (as mentioned in Algorithm 2). Then, we add this noisy vector to the original instance's abstract representation.

The NG has a significant impact on interpreting the prediction. The idea behind local explanations is to discover representative sub-spaces, a.k.a. neighbourhoods. Then, in these smaller spaces, non-linear relations may be absent, thus transparent models will capture the neighbourhood's most important features concerning the black-box model's decisions. To achieve this, the NN component will assign target values (predictions) to each neighbour, probabilities in classification tasks, or real continuous values in regression tasks. To achieve this, we need the original representation of each neighbour. We will accomplish this through decoding, as explained in the following section.

3.2 Neighbourhood Decoding Process

After generating neighbours NE through the NG component, we will need to decode them to the original space through a decoder to train the local transparent model. Therefore, the following component of LioNets is the neighbourhood decoding process (ND), and it is inspired by the Autoencoders [58]. An autoencoder (AE) is an unsupervised learning architecture and can be expressed as a function $AE : X \rightarrow X$. A task may employ autoencoders to decrease the dimensionality of the input data. An AE first *encodes* the original data into a latent, low-dimensional representation and then *decodes* this representation to the original dimensions to recover the original data. Other tasks that use AEs are image denoising [59] and feature extraction [60].

In our case, we already have the first part of the AE, the encoder, which we can extract from the NN predictor. What we need is a decoder, ND, to reconstruct the representation of the neighbours NE in the original input space, $ND(NE) = NE_{decoded}$. However, training decoders is a much more complex task than training predictive models or autoencoders. A decoder tries to transform an instance from one representation to its original representation. Since multiplications inside a neural network through encoding most probably contain non-square matrices, thus non-invertible matrices, a decoder will create pseudo-invertible matrices to transform the data. This is a laborious task, as well as computationally heavy. It is noticeable that the data and the trained predictor/encoder directly affect the decoder's performance.

By training a classifier and extracting the encoder, the decoder maps h, the encoded representation of an instance, to the reconstruction x', the original instance's representation, of the same shape as $x' = \sigma'(W'h + b')$, where σ', W', and b' for the decoder may be unrelated to the corresponding σ, W, and b for the encoder. These models are trained to minimise reconstruction errors, often referred to as "loss" functions, like the MAE (Eq. 6) or the MSE (Eq. 7). The binary crossentropy (Eq. 8) [61] and the Kullback-Leibler divergence [62] are two other interesting loss functions that we can use to train decoders and autoencoders.

$$\mathcal{L}_{MAE}(\mathbf{x}, \mathbf{x}') = \|\mathbf{x} - \mathbf{x}'\| \tag{6}$$

$$\mathcal{L}_{mse}(\mathbf{x}, \mathbf{x}') = \|\mathbf{x} - \mathbf{x}'\|^2 \tag{7}$$

$$\mathcal{L}_{binary\_cross\text{-}entropy}(\mathbf{x}, \mathbf{x}') = -\mathbf{x}log(\mathbf{x}') - (1 - \mathbf{x})log(1 - \mathbf{x}') \tag{8}$$

As we have mentioned, training a decoder is a challenging task, and each problem needs a unique architecture for the decoder. A general way to design a decoder is to use the predictor's inverse architecture. Examples of building decoders for distinct problems can be found in Section 4. From our experiments, we concluded it is easy to build a decoder for textual data when using TFIDF representations, or time series data, using an architecture like the inverted predictor's architecture. However, it was more difficult to train the decoder successfully in text classification with embeddings, and a lot of experimentation was required.

The output is another factor to consider while designing a decoder. In particular, the output can be a unique flat vector representing the input for any data type, whether it is a one-dimensional vector or a two-dimensional matrix. Alternatively, we can have an output with the same or similar shape as the input. The second case, having different output shapes depending on the input shape each time, is more difficult to implement, especially for data types such as text in the form of word embeddings. Nevertheless, based on our experimentation, we highly recommend constructing the decoder's output shape to be identical or similar to the input shape to achieve a higher performing model.

3.3 Why Latent-Space Neighbourhood Encoding and Decoding?

Before proceeding to the last component, we will use an example to show why generating neighbours in latent space is preferable. Given a dataset D containing N instances x_i, $D = \{x_i, y_i | i \in [0, N-1]\}$, each instance has a specific number of M features $x_i = [f_{i,0}, f_{i,1}, \ldots, f_{i,M-1}]$. Observing Figure 1, a simple neural network architecture, we can see that an instance $x_i \in \mathbb{R}^m$ given as input in the penultimate layer is $x_i^{enc} \in \mathbb{R}^k$, where k is the number of nodes, namely the penultimate layer's dimensions.

Data generation is a controversial topic, and by design, neighbourhood generation around an instance, which is widely used in IML, is not defined properly. Indeed, we can ask: *"How do we define neighbourhoods in data?"* or *"How*

do we measure adjacency?". The cosine similarity and the Euclidean distance are two commonly used metrics for measuring adjacency [63]. However, an open problem with this topic is the curse of dimensionality [64], which raises awareness of whether we can trust adjacency in high dimensions. Following the advice of researchers working on this problem, it is better to create neighbours in low-dimension spaces to avoid problems such as the curse of dimensionality. The size of a neural network's latent space, its penultimate layer representation, is usually smaller than the size of the input space. This may help to address the problem of the curse of dimensionality.

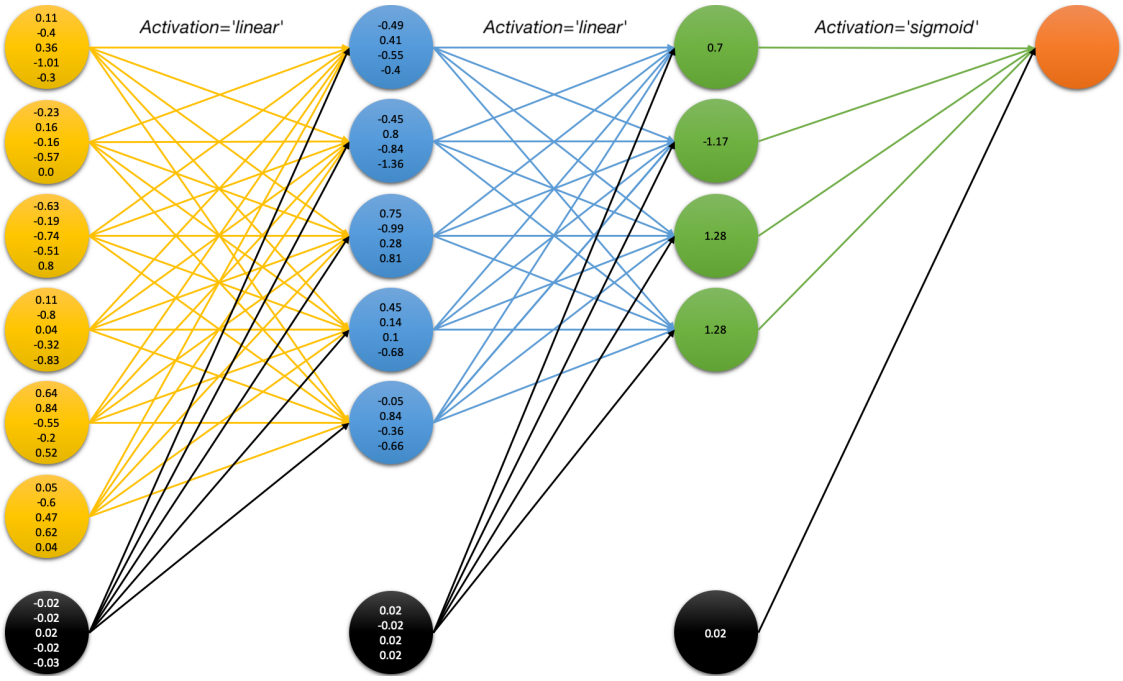

Figure 2: A simple neural network architecture for binary classification

In this example (Figure 2), we have 6+1 dimensions on the input layer (6 features per instance and 1 bias) and 4+1 dimensions on the penultimate layer (higher-dimensional input space than latent space). We train this neural network on a binary classification toy problem generated using the *make_classification* function of Scikit-learn [65], with 100 samples, 6 features, and 2 classes.

For an instance $x_i \in \mathbb{R}^6$, we create a neighbour $ne_i \in \mathbb{R}^6$, and during the classification process, we can take the representations of both the instance and its neighbour in the penultimate layer, which are going to be $x_i^{enc} \in \mathbb{R}^4$ and $ne_i^{enc} \in \mathbb{R}^4$. There is a probability x_i^{enc} to be equal to ne_i^{enc}, while $x_i \equiv ne_i$, because of the neural network's high internal complexity, may eliminate the importance of a feature. We want such examples, but to discover local neighbourhoods, it would be more meaningful to create a neighbourhood for x_i^{enc} directly.

Let us have an example inspired by the network in Figure 2. Creating a neighbour in the original space, we are going to explore the latent space representation of it. For the instance $x_i = [.18, .0, .63, .24, .58, .81]$, we generate neighbours using LIME. Then, using a neighbour $ne_i = [.11, .35, .58, .24, .6, .94]$, with a $cosine\_distance = .04$ and a $euclidean\_distance = .38$, we observe that in the encoded space, the representations of the two instances (the original and the neighbour) are almost identical $x_i^{enc} = [-.17, .41, .03, .08]$, while $ne_i^{enc} = [-.17, .41, .02, .08]$. Of course, we cannot argue that this example is pointless; however, as discussed in the following paragraph, such behaviour has the potential to destabilise the neighbourhood distribution. Thus, it is preferable to create a neighbourhood in the encoded space to ensure both better adjacency and distribution.

Figure 3 shows the Euclidean distances' distribution between x_i and the neighbours generated by LIME in the input space. Figure 4 shows the same distribution of distances between x_i^{enc} and the encoded representations of its neighbours. We calculate the Euclidean distance between the generated neighbours and the original instance in the encoded space. We observe that most of the distances fall between $[.2, .5]$. Hence, we can say that the distribution of distances has lost its normality.

By creating neighbours directly in the latent space (Figure 6), we do not jeopardise the latent space adjacency of a neighbour, but we end up with neighbour-instances with abstract, human-incomprehensible representations. We

Distribution of Euclidean distances of neighbours:

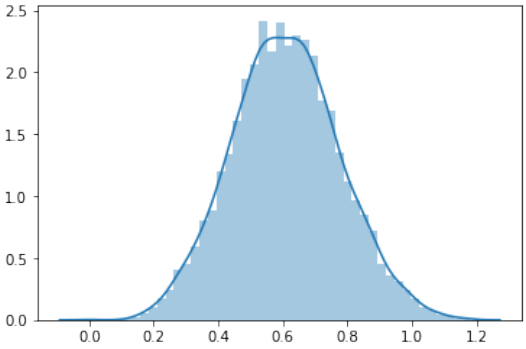

Figure 3: generated on original space

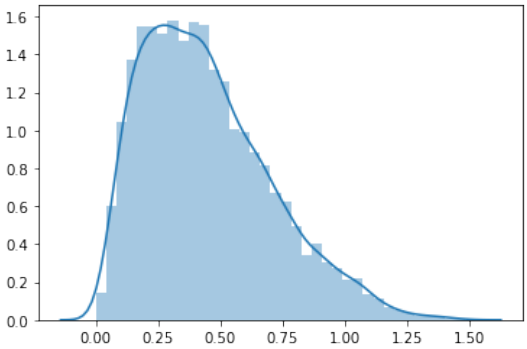

Figure 4: transformed on latent space

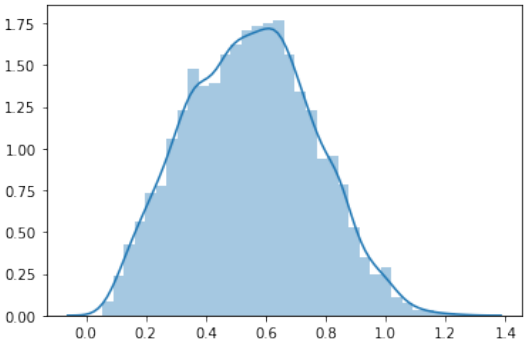

Figure 5: transformed on original space

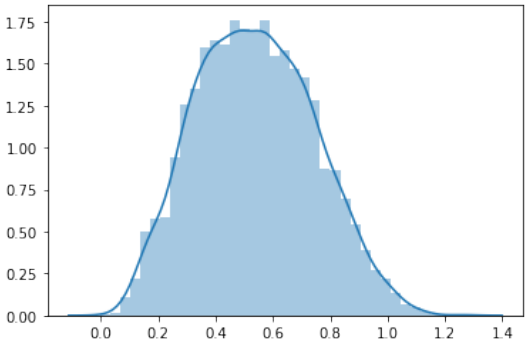

Figure 6: generated on latent space

proposed one way to address this problem by generating neighbours in the latent space and then decoding them into the original space.

While using our simple predictor and encoder, as seen in Figure 2, we train a decoder in Figure 7, using mean absolute error as the loss function. Then, the instance $x_i = [.28, .35, .41, .19, .66, .94]$ is encoded into the abstract representation $x_i^{enc} = [-.2, .25, -.03, .21]$ and decoded back to $x_i' = [.28, .39, .43, .19, .68, .91]$. The decoder did an impressive job. We can compare Figure 5 with Figure 3. The distributions in the original space are better when the neighbour generation process is applied to that space (Figure 6). However, in abstract space, this is not the case.

3.4 Transparent Surrogate Model

The objective of the entire process of constructing a local neighbourhood NE around an instance using the NG component and transforming it back into the original space using the ND component is to train an inherently interpretable ML model G in this neighbourhood. We are going to use this model to explain the prediction concerning that instance. Thus, the last component of LioNets is the transparent surrogate model (TS). We can use any transparent model, such as a linear model, a decision tree, or a Naive Bayes model. As explained in Section 2, selecting any of these models will produce interpretations of different shapes.

$$\begin{aligned} \underset{\mathcal{L}_{MAE}}{\text{minimise}} \quad & \mathcal{L}_{MAE}(\mathbf{f}(\mathbf{x}), \mathbf{g}(\mathbf{x})) \\ \text{subject to} \quad & x \in N^{decoded}, \\ & N^{decoded} \subseteq decoded(N), \\ & g \in G \end{aligned} \tag{9}$$

In LioNets, we use linear models G, specifically Ridge Regression, to extract weights and present them as feature importance interpretations of the neural network's decision for a certain input. While we attempt to fit these models to the generated decoded neighbourhood $ND(NE)$ and the prediction probabilities of them, provided by the model

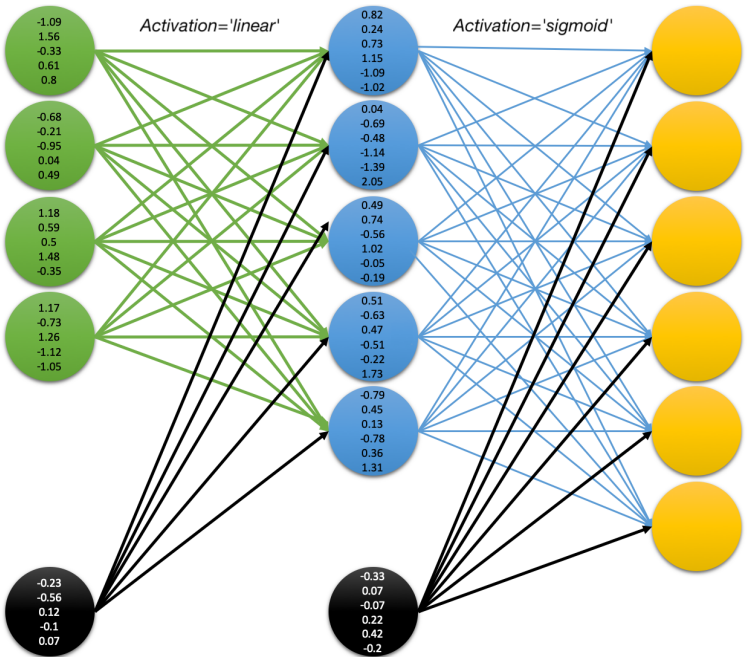

Figure 7: A proposed decoder for the example network in Figure 2

being interpreted (Eq. 9). The calculated weights for the corresponding inputs will then be extracted and used in the final interpretation. There is a tendency to try models such as Lasso or Linear Regression, whose regularisation results in as many features as possible being assigned zero weights. This would produce a smaller feature set that would be more intelligible than larger sets. However, this can cause poorer model performance while learning neighbourhood instances, and more incorrect interpretations, which are not desirable.

Depending on the time available for providing the interpretation, LioNets will adjust from applying a broad grid search to the most suitable parameters for applying a smaller grid search to provide faster interpretation. In addition, as seen in LIME, a weight is given to each neighbour for G training, which is the measured Euclidean/Cosine similarity between the encoded space of the neighbour and the original instance, normalised by Eq. 10. $dimensions$ are equal to the dimension of abstract space if it is greater than 100 or 100 in any other case. This function favours instances nearest to the original, thus excluding instances at greater distances from the original, as shown in Figure 8.

$$distance_{normalised} = e^{-distance\frac{log(dimensions)}{2}} log(dimensions) \tag{10}$$

3.5 Explanation Extraction

Building every component of LioNets, the goal is to extract from the trained TS model each feature's coefficient. We can interpret these coefficients as features' importance. The model designer must then decide how to present this type of information. The default way LioNets presents an explanation is by creating a bar plot, which shows categorical variables (features) and their importance. In Section 4, we present different ways of visualising this collected knowledge about the prediction of an instance of textual and time series data.

With textual data, thanks to the ability to create neighbours in abstract space, TS models will encounter instances (neighbours) that are likely to have features (words) that do not appear in the targeted instance (in the sentence). Thus, the interpretable model will also assign weights to these features, which can be presented as counterfactual words, regarding absolute importance (presenting the most positively and negatively important counterfactuals), i.e., words that might have a positive or negative impact on the probability of prediction of the original instance. Because of their small latent distance from the words of the original sentence, these extra words will also have a semantic relationship. In Section 4, we present a few examples of counterfactual words and how they affect a sentence's prediction when they are added to it.

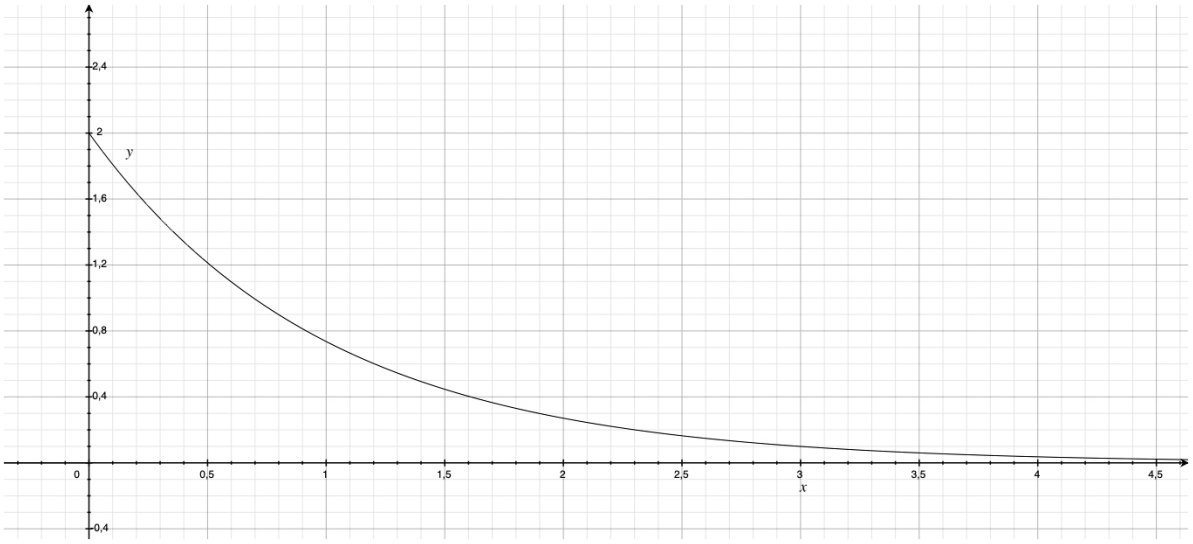

Figure 8: Normalising function for the distances between neighbours and instance. $x = distance$, $y = distance_{normalised}$

4 Experiments and Evaluation

To show the capabilities of LioNets, we carried out four separate test cases. The first two cases address the binary classification of textual datasets. In the third and fourth scenarios, a time series dataset is used to solve binary classification and regression. In this section, we will also propose an extension of faithfulness and a relaxation of robustness.

The trick of creating neighbours in the latent space enables us to generate neighbourhoods regardless of the input data. Thus, unlike other methods, there is no restriction on the type of input, which makes LioNets a general method for generating explanations for models trained with simple vector inputs, or even 2D and 3D matrices. The following test cases illustrate this capability, interpreting neural network architectures with simple vectors (first test case), embedding representations (second case), and 2D time window matrices (third and fourth case) as inputs.

4.1 Evaluation Setup

For each test case, we are going to present the implementation of LioNets, a quantitative evaluation according to Table 1, as well as a qualitative evaluation. To provide a quantitative comparison between LioNets and other interpretation techniques, we took into consideration the metrics of Altruist, a relaxed version of robustness, complexity, and fidelity, using mean absolute error (MAE) and R-squared (R^2). To assess the fidelity at the probability level rather than at the binary level, we employ regression metrics such as MAE and R^2. The interpretation techniques we tested against LioNets were LIME, Gradient×Input (G×I), and LRP-e (using the iNNvestigate library[3] [66]), as selected from the plethora of techniques stated in Section 2, as the most representative. For each dataset, we randomly select 200 instances from both the training and test sets to perform the experiments, using constant seeds for reproducibility.

Altruist [56] was introduced as a metric for evaluating the truthfulness (or faithfulness) of feature importance interpretation techniques used in ML models trained in tabular data, as presented in Section 2.3. Thus, we extend its applicability to textual and time series data.

In textual data, originally, Altruist would evaluate all the features' importance assigned to all the words in the vocabulary. However, in local explanations of texts, and particularly sentences, the final user will see the feature importance assigned to the words appearing in the sentence. Therefore, we extended Altruist, and in such textual data, it evaluates only the importance assigned to each word in a sentence instead of each word in the vocabulary. This will help avoid unnecessary computations for words that do not appear in a sentence. Moreover, those words are not part of the final interpretation. Therefore, we do not need to evaluate their importance.

[3] `https://git.io/JWVFp`

	Altruist		Robustness		Complexity		Fidelity (MAE)		
	Tr	Te	Tr	Te	Tr	Te	Tr	Te	Dataset
LIME	23.50%	33.94%	0.0011	0.0010	**7.63**	**7.63**	0.0105	0.0130	SMS Spam
LioNets	9.62%	36.27%	0.0009	0.0011	10.66	10.71	0.0003	0.0002	
LioNets WA	8.68%	33.15%	0.0012	0.0011	10.66	10.71	**0.0000**	**0.0000**	
GxI	**3.00%**	**28.90%**	0.0012	0.0011	10.66	10.71	-	-	
LRP-e	48.36%	54.86%	**0.0007**	**0.0007**	10.66	10.71	-	-	
LIME	36.35%	36.10%	0.0147	0.0137	**11.50**	**10.54**	0.0192	0.0220	Hate speech
LioNets	45.09%	46.67%	0.0384	0.0342	15.35	13.94	0.0021	0.0024	
LioNets WA	**30.30%**	**27.87%**	0.0181	0.0174	17.43	15.61	**0.0012**	**0.0013**	
GxI	-	-	-	-	-	-	-	-	
LRP-e	43.26%	42.83%	**0.0114**	**0.0113**	17.43	15.61	-	-	
LIME	41.79%	45.04%	0.0993	0.0835	700	700	0.0815	0.0443	TEDS (Binary)
LioNets	39.89%	35.00%	0.0192	0.0144	700	700	**0.0003**	**0.0002**	
GxI	**21.75%**	**19.18%**	**0.0015**	**0.0008**	700	700	-	-	
LRP-e	51.68%	49.82%	0.0050	0.0030	700	700	-	-	
LIME	23.32%	28.71%	0.0130	0.0114	700	700	0.0241	0.0171	TEDS (RUL)
LioNets	32.32%	29.75%	0.0070	0.0054	700	700	**0.0000**	**0.0000**	
GxI	**18.29%**	**21.57%**	**0.0014**	**0.0014**	700	700	-	-	
LRP-e	27.89%	29.79%	0.0032	0.0030	700	700	-	-	

Table 1: The findings of the experiments carried out in the 4 test cases, using 4 different interpretation methods and 4 metrics. Lower values are better in all scores. (The missing values in the fidelity metric for the LRP and Gradient×Input (G×I) is due to the nature of those interpretation techniques, as they do not need to train surrogate models). G×I was not included in the hate speech test case due to implementation limitations. (Tr: Train set, Te: Test set)

For the time series adaptation, each sensor's average importance will be investigated in time series data rather than for each measurement per sensor. For example, if we have a time window of 50-time steps for 14 sensors, the extended version of Altruist will make only 14 evaluations rather than the 700 evaluations that the original Altruist would have done. This choice is motivated because the final interpretations aimed at users will most likely be at the sensor level rather than the time step level, leading to shorter and more comprehensible explanations.

The robustness metric in Eq. 3 demands the identification and use of concepts alongside the explanation provided in a feature-importance manner. It relies on optimisation and is computationally heavy, specifically for explanation techniques like LIME. Thus, for our experiments, we use a relaxed version of robustness presented in Eq. 11, where L is the total number of instances to be examined and e is the explanation of an instance x. Then, altering slightly the instance, we compare the original e, which is a vector of $|F|$ values, with the explanation for the prediction of the "tweaked x". We do this by subtracting (when in tabular or time series data) or zeroing (in textual data) the value of the feature with the lowest absolute importance, based on the explanation. This way, robustness will capture how unstable an explanation technique is when a minor alteration in the value of the least significant feature happens.

$$robustness_{relaxed} = \frac{1}{L}\sum_{i=1}^{L} |e(x_i^{original}) - e(x_i^{tweaked})| \tag{11}$$

The code of the experiments, the trained models, and the datasets used are accessible at the GitHub repository "LionLearn"[4] and the Docker repository[5]. From the results, we prove that LioNets can lead to more precise and truthful explanations than other techniques on a variety of data types.

[4] `https://git.io/JLmgL`

[5] `https://dockr.ly/3qXSoji`

4.2 Textual Corpora

The textual data collections we have selected deal with spam SMS detection [67] and hate speech detection [68], which includes 747 spam and 4,827 ham (non-spam) messages, and 563 comments without and 433 with hate speech content, respectively. For each SMS, the pre-processing comprises the following steps: a) lowercasing, b) stemming and lemmatisation through WordNet lemmatizer [69] and Snowball stemmer [70], c) phrases transformations (Table 2), d) removal of punctuation marks, and e) stemming and lemmatisation once more. For each comment in the hate speech dataset, the pre-processing involves these steps: a) lowercasing, b) phrase transformations (Table 2), and c) removal of punctuation marks.

Phrases and words transformations								
"what's"	to	"what is"	"'ll"	to	" will"	"'s"	to	" is"
"don't"	to	"do not"	"i'm"	to	"i am"	"'ve"	to	" have"
"doesn't"	to	"does not"	"he's"	to	"he is"	"isn't"	to	"is not"
"that's"	to	"that is"	"she's"	to	"she is"	"'re"	to	" are"
"aren't"	to	"are not"	"it's"	to	"it is"	"'d"	to	" would"
"%"	to	" percent"	"e-mail"	to	"e mail"			

Table 2: Phrases and words transformations

In these two test cases, we introduced a LioNets variant called LioNets **W**ord **A**pheresis (WA), which combines LioNets' neighbourhood generation with a word-removal technique similar to LIME's, thus combining the best of both worlds. LioNets WA takes the neighbours formed by LioNets and adds N extra neighbours, namely equal to the number of unique words in a sentence. We remove each of the N unique words one at a time from the original sentence. We developed this extension because, in the hate speech test case, the generated neighbours contained new but semantically related words in the sentence. However, not all the original words were completely removed, at least once from the neighbouring sentences. That prevented LioNets from correctly identifying the importance of the words in the sentence. LioNets WA appeared to excel in the Altruist score, offering more truthful interpretations. We also applied it to the SMS spam test case, and the results were similar.

4.2.1 SMS Spam Dataset with TFIDF Representations

The first case is the detection of spam in SMSs. To address this problem of binary classification, to identify SMSs without (ham) or with spam content, we will need to transform the sentences into vectors. We will use the vectorisation technique of Term Frequency-Inverse Document Frequency (TFIDF) [71], which is widely acknowledged by many researchers because it is simple and efficient.

Architectures of networks for SMS Spam dataset:

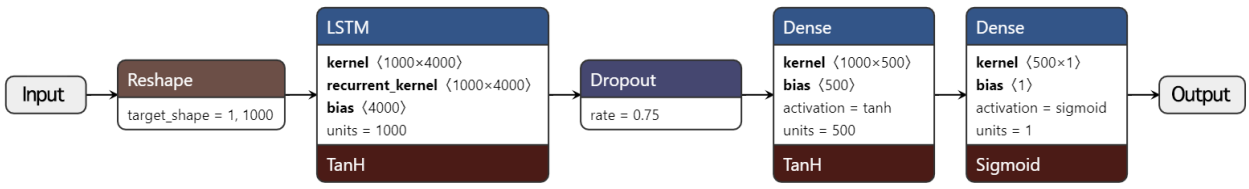

Figure 9: The predictor

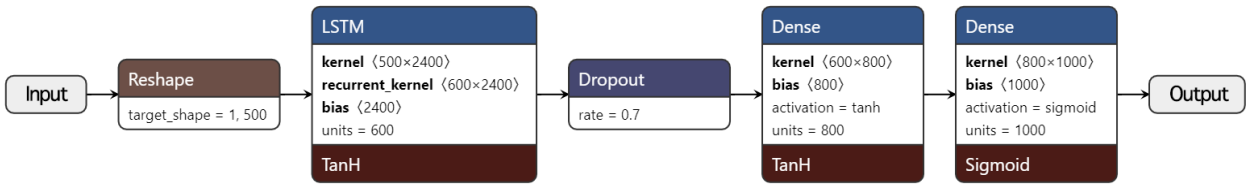

Figure 10: The decoder

The predictor (Figure 9) learned to distinguish the messages (spam or ham) by using 80% of the dataset as training data and 20% as test data. In the predictor's training, the optimiser "adam" and the loss function "binary crossentropy" were utilised. The model's performance was 99.96% on the training set and 98.46% on the test set, in terms of F_1-score ('weighted'), while the results were 99.83% and 95.43%, respectively, in terms of balanced accuracy.

The decoder (Figure 10) transforms the encoded representation of instances as extracted from the penultimate layer of the predictor, with 500 dimensions, to their original shape, with 1000 dimensions. We used "adam" as the optimiser and "binary crossentropy" as the loss function. Then, we measured 0.0074 and 0.0098 errors, as well as 0.0019 and 0.0024 MAE, on the training set and test set, respectively. We also calculated the fidelity of the original and the decoded instances, which was 99.67% on the training set and 97.04% on the test set, using the F_1-score ('weighted').

Here are some examples of the decoder:

Example 1: Examples of decoded sentences of SMS Spam dataset

```
 Train Data:
============================================================
   Original: also am but do have in onli pay to
    Decoded: also am but do have in not onli pay to
------------------------------------------------------------
   Original: come got it me now ok or then thk wan wat
    Decoded: come got it me now ok or then thk wan wat
------------------------------------------------------------
 Test Data:
============================================================
   Original: been better day each even give god great more
             never reason thank to
    Decoded: been better day for give god great it more
             thank the to you
------------------------------------------------------------
   Original: am at be late there will
    Decoded: am at be late there will
------------------------------------------------------------
```

Comparing LioNets with other methodologies quantitatively, we can observe in Table 1 the significant superiority of the G×I algorithm in terms of the Altruist score (truthfulness). This is justified by the simple architecture of the predictor. Nevertheless, LRP cannot operate correctly because of the presence of recurrent layers, even if the activation functions comply with the LRP requirements (hyperbolic tangent - tanh). Moreover, LioNets and LioNets WA perform much better than LIME and LRP in terms of this score, with a performance very close to G×I's. As far as other metrics are concerned, LRP achieves the highest robustness scores, while LioNets and LIME follow. It is interesting that, in this test case, G×I has the worst robustness performance. Additionally, we can see that LIME provides the smallest explanations, while LioNets produces linear models that better approximate neural network predictions than LIME.

By training all the components of LioNets, it is feasible to apply the technique to an instance's classification to extract interpretations to evaluate the technique. To give an example, for the sentence: *"Congrats! Treat pending. I am not on mail for 2 days. Will mail once thru. Respect mother at home. Check mails."*[6], our predictor provided a probability of containing spam of 10.27%. We can assign importance to each of the features of the instance (Figure 11), as well as identify features not appearing in the instance but found in the neighbourhood that may influence the prediction (Figure 12).

In addition, to determine the quality of the explanations, we removed the word "Congrats", and we observed a decrease in the probability to 10.15%, while by removing the word "mail(s)", we observed an increase in the likelihood to 10.36%. Another noteworthy feature of LioNets is the ability to explore features appearing in an instance's neighbourhood and present them as counter features (Figure 12) when handling sparse data, such as text data.

We extracted the local neighbourhood for the instance, and by observing the counter features created through the process of NG in the latent space, we added the word *"all"*, and the probability increased (10.30%). When we added the word *"ad"*, the probability decreased (9.87%). This reveals an interesting fact: in this local prediction, the word *"all"* contributes to the "spam" class. Contrarily, in the sentence: *"How did you find out in a way that didn't include **all** of these details"*, the word "all" influences the "ham" class.

4.2.2 Hate Speech Dataset with Embedding Representations

The second case is about hate speech detection. The "ETHOS" dataset [68] contains comments on social media platforms that may or may not contain hate speech. To train a model to predict the appearance of hate speech content in

[6]To increase the user's readability and comprehension of the examples, this sentence and the following sentences in the textual experiments are presented without being pre-processed.

SMS: “Congrats! Treat pending. I am not on mail for 2 days. Will mail once thru. Respect mother at home. Check mails.”

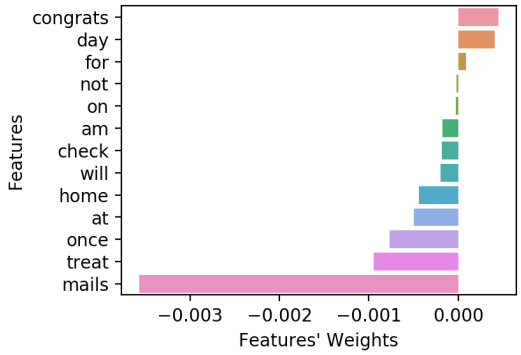

Figure 11: Features appearing in the instance

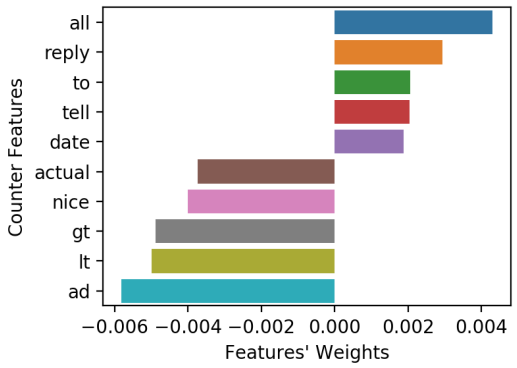

Figure 12: Features not appearing in the instance

a sentence, we have transformed the sentences into embedding matrices. For each sentence, we kept only 50 words (max_words), and by using pre-trained GloVe embeddings [72], we transformed each word into a 50-dimensional vector, resulting in an embedding matrix of 50×50. We have set the vocabulary at a limit of 1000 words.

Hate speech detection is a very challenging task, making the architecture of the neural network used more complicated. The predictor (Figure 13) was trained using 80% of the data as a training set and 20% as a test set. Using the “binary crossentropy” as the loss function and the “adam” optimiser, it achieved a performance of 78.35% and 72.87% in terms of F_1-score (‘weighted’), and 77.69% and 72.14% in terms of balanced accuracy, for the train and test set.

Once again, we transform the encoded representation of instances back to their original representation using the decoder we developed (Figure 14). We extracted those representations from the penultimate layer of the predictor, and their dimension was 500. The decoder outputs a 50×1001 matrix, where 50 is the number of words in the sentences, and 1001 is the vocabulary size (the additional one to 1000 words is the “UKN”). Using “adam” as the optimiser and “categorical crossentropy” as the loss function, we measured 0.0736 and 0.0765 errors on the training set and the test set. The original and the decoded instances had high fidelity. In terms of F_1-score (‘weighted’), fidelity reached 99.37% on the training set and 99.00% on the test set. Since the dataset was small (998 instances), we used unsupervised data, around 3000 random sentences, from another hate speech dataset [73], which contains almost 25K sentences, to enhance the decoder’s performance.

Below, we present some examples of the decoded sentences:

Example 2: Examples of decoded sentences of hate speech dataset

```
 Train Data:
============================================================
    Original: maybe it is because you are a******c wish you
              death
     Decoded: maybe it UKN because you are a******c
              wish you death
------------------------------------------------------------
    Original: disgusting sick UKN kill yourself
     Decoded: disgusting sick UKN kill yourself
------------------------------------------------------------
 Test Data:
============================================================
    Original: UKN is the UKN that comes out of UKN dream
     Decoded: is UKN that comes out of of dream UKN
------------------------------------------------------------
    Original: what the f**k stupid people
     Decoded: what the f**k stupid people
------------------------------------------------------------
```

Architectures of networks for hate speech dataset:

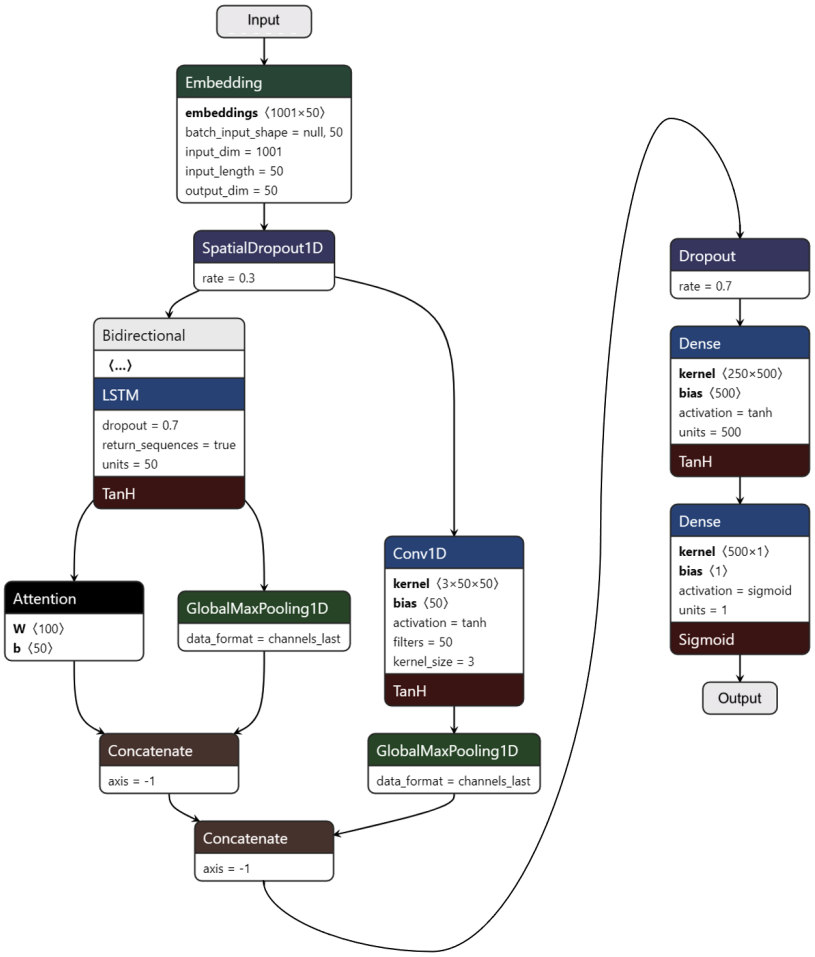

Figure 13: The predictor

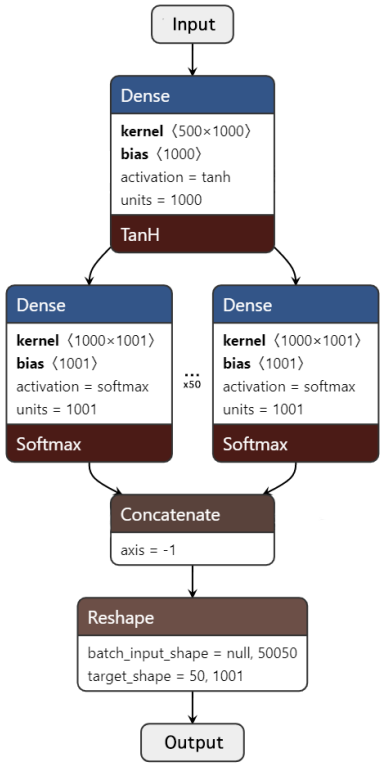

Figure 14: The decoder

Because of implementation limitations, G×I does not take part in this test case, as it does not support embedding inputs. In contrast to LRP, which we could make work with embeddings using various suggestions from the library's repository, we could not adapt G×I to work with such inputs. LioNets WA has the highest Altruist score, followed by LIME and LRP. In terms of robustness, LRP delivers the most stable interpretations, followed by LIME, with LioNets WA having the worst performance. We see that LioNets has a greater fidelity score than LIME. LIME could reduce the complexity by an average of 5.94 features.

To examine the qualitative performance of LioNets in this dataset, we present an explanation provided by the technique for the following comment: *"Or maybe just don not follow degenerate sandn****r religions from the middle east?"*, which, after pre-processing, becomes *"or maybe just do not follow UKN UKN religions from the middle UKN"*. The neural network predictor assigned a 73.04% probability of containing hate speech content, which is quite uncertain. Figure 15 shows the explanation, while Figure 16 presents the identified counterexamples.

For this example, the neural network is correctly uncertain. Because of the vocabulary's small size, the words before the word *"religions"* are unknown ("UKN"), as well as the words after *"middle"*. Therefore, the predictor cannot judge the comment as hateful with high certainty. The word *"middle"* increases the probability that the comment contains hate speech. By removing this word, we observe a reduced probability of 70.30%.

Taking advantage of the alternative ability of LioNets to assign feature importance to words not appearing in the original sentence, we attempted to replace the first "UKN" with the word *"these"*. We observed an increase in the probability from 73.03% to 74.97%. Sequentially, we conduct the same experiment by adding the word *"they"*, which leads to a

Comment: “or maybe just do not follow UKN UKN religions from the middle UKN”

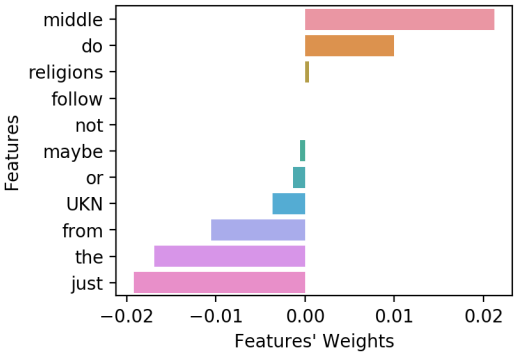

Figure 15: Features appearing in the instance

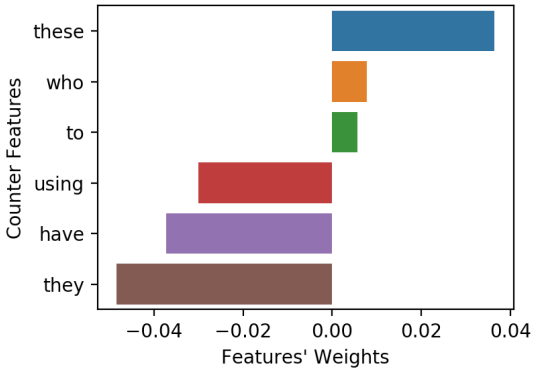

Figure 16: Features not appearing in the instance

drop of 69.97% in the probability. Such experiments are very useful for a model designer. The designer may notice that stopwords like “to”, “who”, and “these” influence the output a lot, so it might be prudent to consider excluding them.

4.3 Turbofan Engine Degradation Simulation Dataset — time series

The next two test cases involve the Turbofan Engine Degradation Simulation dataset [74, 75]. This dataset contains four individual simulated datasets. Each of the four datasets comprises multiple multivariate time series about different engine units and their wear. For every unit, there are time steps accompanied by the remaining useful lifetime (RUL). We chose the first sub dataset to create both a binary classifier and a RUL predictor.

Our initial step is to apply some feature engineering to the input data. We first discarded some features, like the third operational setting and some sensors $[1, 5, 6, 10, 16, 18, 19, 22, 23, 24, 25, 26]$, because they did not contain any information (all values were NaN, or they had the same value). Then, inspired by recent work [76], we removed the two additional operating settings. In the end, we keep 14 sensor measurements for each time step of every unit. We are setting a time window parameter $N \in \mathbb{N}$. We will train the neural predictor to approximate the RUL of one unit by having as input the measurements of some sensors at the current time step, accompanied by the records of the $N - 1$ previous time steps.

This specific dataset will give us the ability to present the LioNets’ effectiveness in explaining models trained on complex input shapes. For both the classification and the regression models, we designed and used the same architecture for the predictor (Figure 17) and the decoder (Figure 18). LioNets uses the predictor, the encoder (extracted from the predictor), and the decoder to create a neighbourhood for an instance. The input shape is a 2D matrix of 50×14. To use the generated neighbourhood to train a transparent linear model, we are reshaping the neighbours from 50×14 to 700×1, thus transforming the matrices to vectors. Then, by training the linear model and extracting the coefficients, we have 700 different values.

We propose to display this knowledge in several ways. The first way is to display the mean average influence of each sensor over N time steps. Then, by selecting a particular sensor, the influence of N time steps and the sensor’s values can be observed in a plot.

4.3.1 Explanation on Binary Classification

This dataset originally concerned a regression problem. Thus, to build a binary classifier, we need to transform it into a classification problem. We use a time-threshold T, which transforms the RUL of each time step of all units into a binary value $\in \{0, 1\}$. Specifically, we set the value of a time step to 0 when the condition $RUL > T$ is true, and to 1 when $RUL \leq T$. We trained the predictor, which has the architecture of the encoder in Figure 17, with an output layer with a sigmoid function, with the “adam” optimiser and the “binary crossentropy” loss function. The model’s performance was 97.75% for the training and 98.44% for the test set, in terms of F_1-score (“weighted”). The balanced accuracy was 95.80% for the training and 87.38% for the test set.

To train the decoder (Figure 18), we used the “adam” optimiser and the “root mean squared error—RMSE” loss function. The model’s RMSE was 0.0679 and 0.0684 for the training and test sets. The MAE was 0.0515 and 0.0520.

Architectures of networks for Turbofan Engine Degradation Simulation dataset:

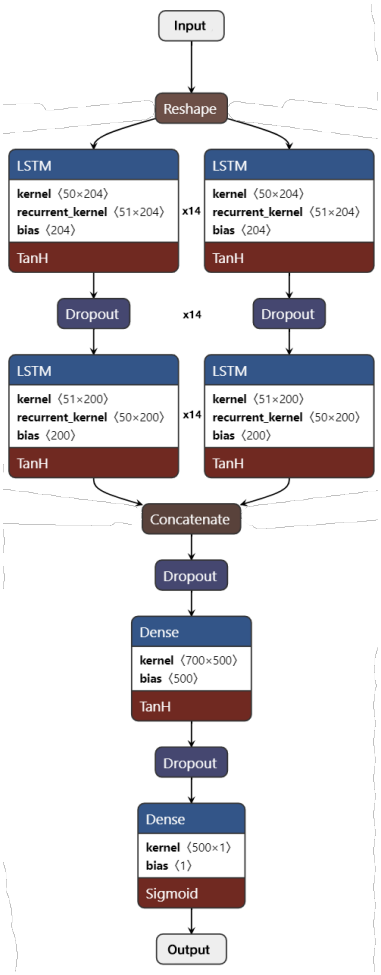

Figure 17: The predictor

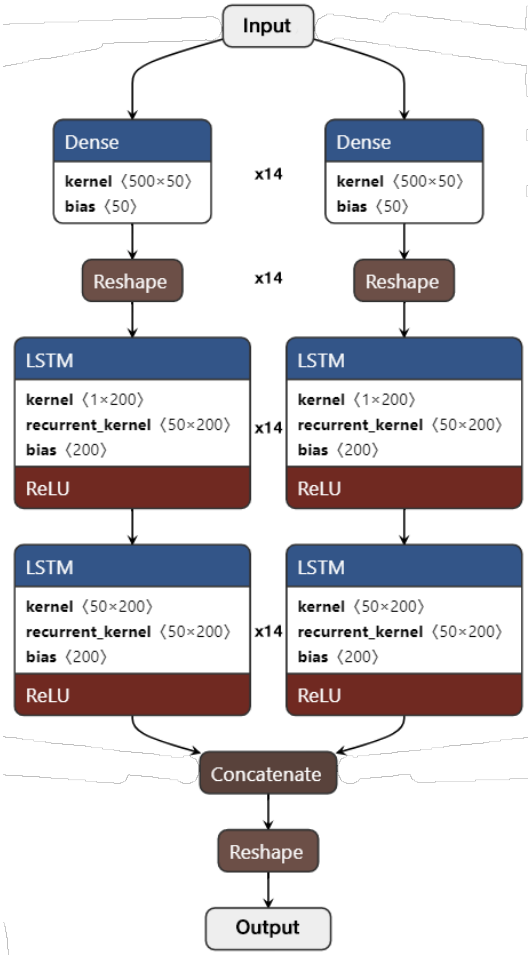

Figure 18: The decoder

Additionally, we measured the fidelity between the original and the decoded instances. We received 99.35% on the training set and 99.64% on the test set using the F_1-score ('weighted').

In Table 1, we can observe that G×I has the highest Altruist score, while LioNets follows. Based on this score, LIME and LRP cannot provide sufficient results. G×I also achieved the lowest robustness score, with LRP and LioNets to follow. LIME's performance on robustness is the worst in this case. LioNets performed better than LIME, in terms of fidelity. Neither technique, though, decreased the complexity.

We may perform a qualitative assessment after all the required components have been assembled, with a trained predictor, encoder, and decoder. In a random instance, the predictor assigns 87.28% of the probability that the component may need to be maintained. We would like a lower probability, below 50%, to conclude that the component does not need maintenance. We used the LioNets technique to get a vector of $14 \times 50 = 700$ values. For each time step of each sensor, we have an influence factor. The first way to display this information is to aggregate the 50 influence values for each sensor and present their mean, STD, and max/min values (Figure 19).

In addition, we selected one of the most important sensors from these plots, with a positive influence. The 15^{th} sensor has a greater impact on the classifier by positively influencing the "need maintenance" class. We adjust the last 20 measurements of the 15^{th} sensor (subtracted by 0.15) if they have a positive influence because, according to Figure 20, these measurements had a high impact. We observe these values are also higher than the average. After these changes, the neural network assigned a likelihood of 30.40% to the "need maintenance" class. Thus, if we had decreased the measurements of this sensor in these time steps, we might have reduced the likelihood of failure.

Explanations of classifier on Turbofan Engine Degradation Simulation dataset:

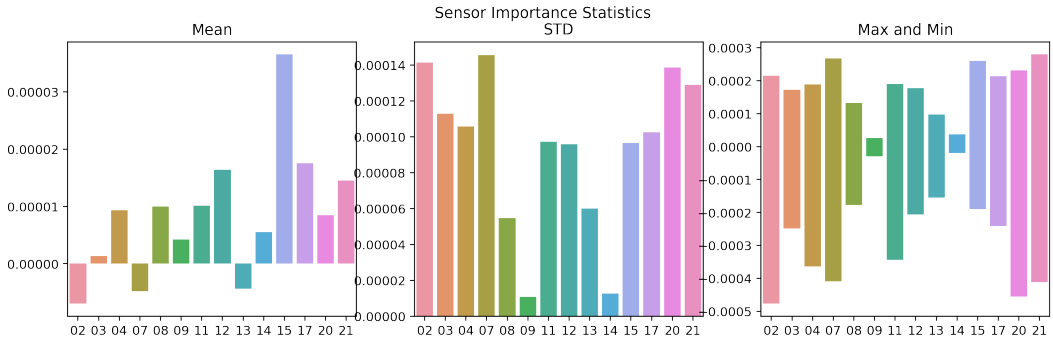

Figure 19: Mean, STD and Max/Min influence per sensor

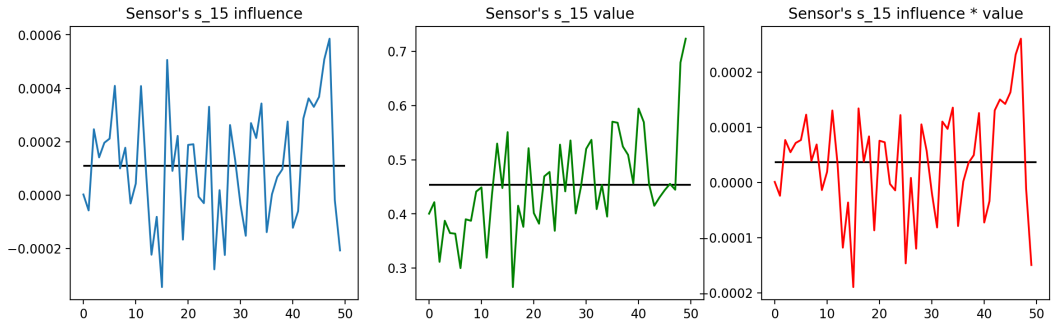

Figure 20: Influence, value and influence×value of each time step of sensor 15

4.3.2 Explanation on Remaining Useful Lifetime Estimator

In contrast to the binary classifier, the output of the RUL estimator is $Y \in \mathbb{R}_{\geq 0}$. We trained the predictor from Figure 17 with the "adam" optimizer and the RMSE loss function. The model's RMSE was 27.83 (training set) and 33.84 (test set), whereas the MAE was 17.73 (training set) and 23.63 (test set).

As an optimizer, we chose the "adam" optimizer and the RMSE loss function to train the decoder (Figure 18). The model's performance in terms of the RMSE was $0.0684/0.0682$ (training/test set), while the MAE was 0.0515 (training set) and 0.0513 (test set). Using the R^2 score, we measured the fidelity of the decoder between the original and the decoded instances. The performance reached 0.9951 on the training set and 0.9894 on the test set.

In this last test case, G×I achieves the highest Altruist score, with LIME and LioNets to follow. G×I provides by far the most stable interpretations in terms of robustness, with LRP-e and LioNets to follow. In addition, we observe that LioNets has achieved a better fidelity score, outperforming LIME. However, neither technique reduced the complexity.

Then we choose an instance, a set of measurements for the sensors, which we know will cause a low RUL value. The neural network predicts that the remaining useful life of the component is 25.14. Then, we apply the LioNets interpretation technique to this prediction. The local linear model prediction is 25.14, identical to the neural network's prediction. We generated the feature importance plots in Figure 21. From these plots, we can see that the measurements of the 17^{th} sensor influence the prediction negatively the most. Thus, we decrease the values of the sensor's measurements by 0.15 from the 30^{th} time step and afterwards, as if these measurements had the most negative effect on the prediction according to Figure 22. Note also that those 20 last time steps have higher than normal reported values. The prediction for the updated measurements from the neural network rose to 29.88. We have therefore extended the lifetime of the component using this interpretation by almost 5-time units.

5 Discussion

Through experimentation, we arrived at the following research findings, which support the initial claim that LioNets is a practical and complete technique that applies in a variety of applications. Based on Table 1, in terms of the truthfulness

Explanations of RUL predictor on Turbofan Engine Degradation Simulation dataset:

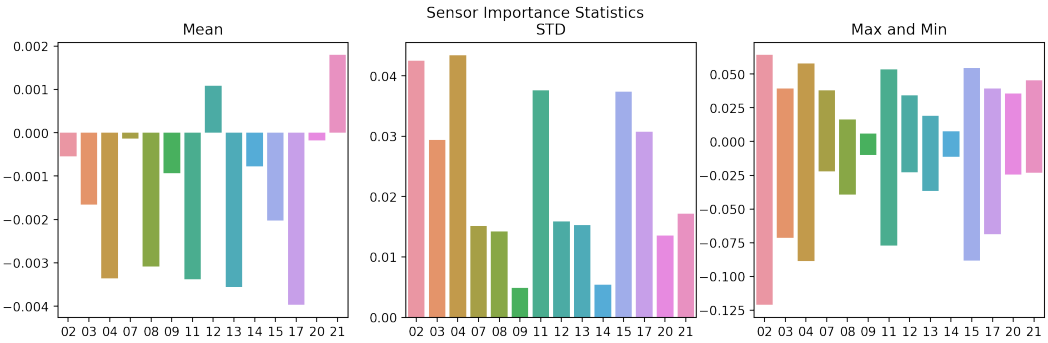

Figure 21: Mean, STD, and Max/Min influence per sensor

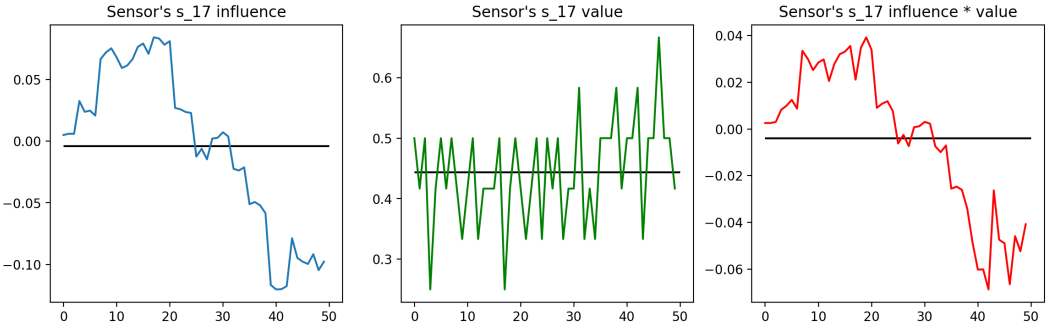

Figure 22: Influence, value and influence×value of each time step of sensor 17

(Altruist score, which extends the concept of faithfulness) of the interpretations, LioNets cannot surpass the efficiency of the G×I algorithm. However, it outperforms the LRP and LIME algorithms in three out of four test cases[7]. LioNets WA outperformed LRP and LIME and almost reached G×I's performance.

The LRP and G×I approaches achieved the greatest robustness scores in any test, with LioNets and LIME performing equally well. In two test cases, LRP had the best robustness score and the second-best in the remaining two. G×I reached the best robustness score in two test cases and the second-best in one more. In almost every test case, LioNets maintained the third-best performance, outpacing LIME's performance. In terms of fidelity, LioNets surpasses LIME.

Altruist scores for LioNets WA, a variant developed for the textual test cases, were excellent. After studying the LioNets generated neighbours, we noticed that several new semantically similar words appeared in the decoded neighbours. Nevertheless, there was no full removal of the sentence's original words. The decoder utilised in the hate speech test case considers both word position and frequency. For example, if a word appeared twice in a sentence, it was most likely omitted only once, while the other words may have changed position. We created this variant to address this issue by introducing complete *apheresis* (removal) of words in a LIME-like approach.

Through qualitative experiments, we showcased LioNets as a useful tool to alter the prediction of an instance. Indeed, in textual datasets, we found important words that, when they were removed, the prediction changed. We also found semantically similar words that, when added to the sentence, its prediction changed as well. In time series test scenarios, the interpretations can help prevent a failure when the estimated RUL prediction is low.

"So why should anyone prefer LioNets over techniques like G×I?" In almost every case, G×I exceeds all the other techniques. However, its implementation limitations are in contrast to LioNets' applicability. In addition, neighbourhood techniques allow users to experiment and try out different ways of extracting interpretations. For example, we can extract a set of counterfactual words with semantic similarity to an instance through LioNets-generated neighbours in textual datasets.

Comparisons are almost equal between LioNets and LIME in terms of truthfulness (Altruist score). In particular, LioNets achieves 29.62%, while LIME achieves 33.59%, on average, for both training and test sets in every test case.

[7]LioNets WA is included.

LioNets provides surrogate models with better fidelity. Additionally, it generates richer semantically neighbours that are closer to the encoded representation of the original instance, in contrast to LIME's neighbourhoods. The neighbourhood generation process is independent of the data type, in contrast to other techniques that, for each data type, have different neighbourhood generation techniques. In textual datasets, LioNets' excellence over all other techniques is clear because of the ability to identify words that do not exist in the original sentence.

Experimentation further shows that LioNets applies to neural networks with any type of layer: recurrent, bidirectional, convolutional, and feed-forward, as well as various activation functions: Tanh, ReLU, and sigmoid, as tested in the test cases' predictors (Figures 9, 13, and 17).

Therefore, LioNets can be specified as the interpretation medium for a neural network with a slight trade-off in truthfulness and almost no trade-off in robustness, offering richer explanations and counterwords in textual datasets. In addition, LioNets produce a local neighbourhood which enables users to work to identify new ways to explain an instance. For example, the relationship and impact of word pairs and words' positions in textual datasets can be exploited through these neighbourhoods. Finally, such neighbourhoods can be used in time series data to train various forms of surrogate models that export details that are stronger and more useful.

6 Conclusion

In summary, we provided a detailed presentation of LioNets' architecture. LioNets offers accurate and consistent interpretations of neural network decisions comparable to other state-of-the-art techniques. This technique ensures a better relationship between the produced neighbours of the instance, as we applied the generation process to the penultimate layer of the network. Neighbours have lower dimensions in this space, and phenomena such as the *curse of dimensionality* are better treated, while neighbours have richer semantic information about the model itself. In addition, we have shown the ability of LioNets to adapt to various data types (textual and time series).

Quantitative and qualitative experiments assessed the validity of this research. We conducted experiments using well-known metrics in four separate test cases. For one of the test cases, we developed a LioNets' variation, LioNets WA, to encounter a problem we identified. We implemented two extended metrics to improve the quantitative evaluation: relaxed robustness and Altruist on textual and time series data. We have also introduced new methods for visualising interpretations provided by LioNets. A novel way of finding counterfactual terms in a sentence in textual test cases has also been presented.

However, one drawback of LioNets is that it only applies to neural networks, so it is not a model-agnostic technique. Also, the overall process of using LioNets is more complex than other approaches. This is because LioNets requires a decoder. Nevertheless, training a decoder is very time-consuming. Future research plans include introducing LioNets for multi-class or multi-label tasks such as image recognition or object detection. We can further explore the usability of different transparent models rather than the linear models of the LioNets architecture. It would be interesting to investigate LioNets' applicability to non-traditional neural networks, such as transformers [77]. Finally, we intend to build an extension of LioNets that will not require a separate decoder to be provided by the user.

Acknowledgment

This work has been supported by the European Union's Horizon 2020 research and innovation programme under grant agreement No. 825619 (AI4EU [8]).

References

[1] Leilani H Gilpin, David Bau, Ben Z Yuan, Ayesha Bajwa, Michael Specter, and Lalana Kagal. Explaining explanations: An overview of interpretability of machine learning. In *2018 IEEE 5th International Conference on data science and advanced analytics (DSAA)*, pages 80–89. IEEE, 2018.

[2] Finale Doshi-Velez and Been Kim. Towards a rigorous science of interpretable machine learning, 2017.

[3] Mengnan Du, Ninghao Liu, and Xia Hu. Techniques for interpretable machine learning. *Communications of the ACM*, 63(1):68–77, 2019.

[4] Cynthia Rudin. Stop explaining black box machine learning models for high stakes decisions and use interpretable models instead. *Nature Machine Intelligence*, 1(5):206–215, 2019.

[8] `https://www.ai4europe.eu`

[5] Niklas Bussmann, Paolo Giudici, Dimitri Marinelli, and Jochen Papenbrock. Explainable Machine Learning in Credit Risk Management. *Computational Economics*, 57(1):203–216, January 2021.

[6] Cindy Wang. Interpreting neural network hate speech classifiers. In *Proceedings of the 2nd Workshop on Abusive Language Online (ALW2)*, pages 86–92, 2018.

[7] Ioana Giurgiu and Anika Schumann. Explainable failure predictions with rnn classifiers based on time series data. In *Proceedings of the 11th International AAAI ConferenceWorkshop on Network Interpretability for Deep Learning, arXiv*, 2019.

[8] Antonio Jesús Banegas-Luna, Jorge Peña-García, Adrian Iftene, Fiorella Guadagni, Patrizia Ferroni, Noemi Scarpato, Fabio Massimo Zanzotto, Andrés Bueno-Crespo, and Horacio Pérez-Sánchez. Towards the interpretability of machine learning predictions for medical applications targeting personalised therapies: A cancer case survey. *International Journal of Molecular Sciences*, 22(9), 2021.

[9] Jonathan Rebane, Isak Samsten, and Panagiotis Papapetrou. Exploiting complex medical data with interpretable deep learning for adverse drug event prediction. *Artificial Intelligence in Medicine*, 109:101942, 2020.

[10] General Data Protection Regulation. Regulation (eu) 2016/679 of the european parliament and of the council of 27 april 2016 on the protection of natural persons with regard to the processing of personal data and on the free movement of such data, and repealing directive 95/46. *Official Journal of the European Union (OJ)*, 59(1-88):294, 2016.

[11] Marco Tulio Ribeiro, Sameer Singh, and Carlos Guestrin. "Why Should I Trust You?": Explaining the Predictions of Any Classifier. *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining - KDD '16*, 2 2016.

[12] Amina Adadi and Mohammed Berrada. Peeking Inside the Black-Box: A Survey on Explainable Artificial Intelligence (XAI). *IEEE Access*, 6:52138–52160, 2018.

[13] Stefan Th Gries. On classification trees and random forests in corpus linguistics: Some words of caution and suggestions for improvement. *Corpus Linguistics and Linguistic Theory*, 2019.

[14] Marco Tulio Ribeiro, Sameer Singh, and Carlos Guestrin. Anchors: High-Precision Model-Agnostic Explanations. In *Thirty-Second AAAI Conference on Artificial Intelligence*, pages 1527–1535, 2018.

[15] Ioannis Mollas, Nikolaos Bassiliades, and Grigorios Tsoumakas. Lionets: Local interpretation of neural networks through penultimate layer decoding. In Peggy Cellier and Kurt Driessens, editors, *Machine Learning and Knowledge Discovery in Databases*, pages 265–276, Cham, 2020. Springer International Publishing.

[16] Zhong-Qiu Zhao, Peng Zheng, Shou-tao Xu, and Xindong Wu. Object detection with deep learning: A review. *IEEE transactions on neural networks and learning systems*, 30(11):3212–3232, 2019.

[17] J. Zhang and C. Zong. Deep neural networks in machine translation: An overview. *IEEE Intelligent Systems*, 30(5):16–25, 2015.

[18] Jürgen Schmidhuber. Deep learning in neural networks: An overview. *Neural networks*, 61:85–117, 2015.

[19] Avanti Shrikumar, Peyton Greenside, and Anshul Kundaje. Learning important features through propagating activation differences. In Doina Precup and Yee Whye Teh, editors, *Proceedings of the 34th International Conference on Machine Learning*, volume 70 of *Proceedings of Machine Learning Research*, pages 3145–3153. PMLR, 06–11 Aug 2017.

[20] Mukund Sundararajan, Ankur Taly, and Qiqi Yan. Axiomatic attribution for deep networks. In Doina Precup and Yee Whye Teh, editors, *Proceedings of the 34th International Conference on Machine Learning, ICML 2017, Sydney, NSW, Australia, 6-11 August 2017*, volume 70 of *Proceedings of Machine Learning Research*, pages 3319–3328. PMLR, 2017.

[21] Jost Tobias Springenberg, Alexey Dosovitskiy, Thomas Brox, and Martin A. Riedmiller. Striving for simplicity: The all convolutional net. In *ICLR (Workshop)*, 2015.

[22] Ramprasaath R. Selvaraju, Michael Cogswell, Abhishek Das, Ramakrishna Vedantam, Devi Parikh, and Dhruv Batra. Grad-cam: Visual explanations from deep networks via gradient-based localization. *International Journal of Computer Vision*, 128(2):336–359, Oct 2019.

[23] 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):1–46, 07 2015.

[24] Grégoire Montavon, Sebastian Lapuschkin, Alexander Binder, Wojciech Samek, and Klaus-Robert Müller. Explaining nonlinear classification decisions with deep taylor decomposition. *Pattern Recognition*, 65:211–222, 2017.

[25] Avanti Shrikumar, Peyton Greenside, and Anshul Kundaje. Learning important features through propagating activation differences. In *Proceedings of the 34th International Conference on Machine Learning-Volume 70*, pages 3145–3153. JMLR. org, 2017.

[26] Marco Ancona, Enea Ceolini, Cengiz Öztireli, and Markus Gross. Towards better understanding of gradient-based attribution methods for deep neural networks. In *6th International Conference on Learning Representations, ICLR 2018, Vancouver, BC, Canada, April 30 - May 3, 2018, Conference Track Proceedings*. OpenReview.net, 2018.

[27] Vinod Nair and Geoffrey E. Hinton. Rectified linear units improve restricted boltzmann machines. In *Proceedings of the 27th International Conference on International Conference on Machine Learning*, ICML'10, page 807–814, Madison, WI, USA, 2010. Omnipress.

[28] Jan Salomon Cramer. The origins of logistic regression. *SSRN Electronic Journal*, 2002.

[29] J. A. Nelder and R. W. M. Wedderburn. Generalized linear models. *Journal of the Royal Statistical Society. Series A (General)*, 135(3):370–384, 1972.

[30] Leo Breiman, Jerome H Friedman, Richard A Olshen, and Charles J Stone. Classification and regression trees. belmont, ca: Wadsworth. *International Group*, 432:151–166, 1984.

[31] Melvin Earl Maron. Automatic indexing: an experimental inquiry. *Journal of the ACM (JACM)*, 8(3):404–417, 1961.

[32] Andrew McCallum, Kamal Nigam, et al. A comparison of event models for naive bayes text classification. In *AAAI-98 workshop on learning for text categorization*, volume 752, pages 41–48. Citeseer, 1998.

[33] N. S. Altman. An introduction to kernel and nearest-neighbor nonparametric regression. *The American Statistician*, 46(3):175–185, 1992.

[34] Leo Breiman. Random forests. *Machine Learning*, 45(1):5–32, Oct 2001.

[35] Vladimir Naumovich Vapnik. *The Nature of Statistical Learning Theory, Second Edition*. Statistics for Engineering and Information Science. Springer, 2000.

[36] Terrence L. Fine, S. L. Lauritzen, M. Jordan, J. Lawless, and V. Nair. *Feedforward Neural Network Methodology*. Springer-Verlag, Berlin, Heidelberg, 1st edition, 1999.

[37] Orestis Lampridis, Riccardo Guidotti, and Salvatore Ruggieri. Explaining sentiment classification with synthetic exemplars and counter-exemplars. In *International Conference on Discovery Science*, pages 357–373. Springer, 2020.

[38] Alon Jacovi, Oren Sar Shalom, and Yoav Goldberg. Understanding convolutional neural networks for text classification. In Tal Linzen, Grzegorz Chrupala, and Afra Alishahi, editors, *Proceedings of the Workshop: Analyzing and Interpreting Neural Networks for NLP, BlackboxNLP@EMNLP 2018, Brussels, Belgium, November 1, 2018*, pages 56–65. Association for Computational Linguistics, 2018.

[39] Isak Karlsson, Jonathan Rebane, Panagiotis Papapetrou, and Aristides Gionis. Explainable time series tweaking via irreversible and reversible temporal transformations. In *IEEE International Conference on Data Mining, ICDM 2018, Singapore, November 17-20, 2018*, pages 207–216. IEEE Computer Society, 2018.

[40] Roy Assaf and Anika Schumann. Explainable deep neural networks for multivariate time series predictions. In Sarit Kraus, editor, *Proceedings of the Twenty-Eighth International Joint Conference on Artificial Intelligence, IJCAI 2019, Macao, China, August 10-16, 2019*, pages 6488–6490. ijcai.org, 2019.

[41] Aaron Fisher, Cynthia Rudin, and Francesca Dominici. All models are wrong, but many are useful: Learning a variable's importance by studying an entire class of prediction models simultaneously. *Journal of Machine Learning Research*, 20(177):1–81, 2019.

[42] Trevor Hastie, Robert Tibshirani, and Jerome Friedman. *The elements of statistical learning: data mining, inference, and prediction*. Springer Science & Business Media, 2009.

[43] Alex Goldstein, Adam Kapelner, Justin Bleich, and Emil Pitkin. Peeking inside the black box: Visualizing statistical learning with plots of individual conditional expectation. *Journal of Computational and Graphical Statistics*, 24(1):44–65, 2015.

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

[45] Alvin E Roth. *The Shapley value: essays in honor of Lloyd S. Shapley*. Cambridge University Press, 1988.

[46] Lloyd S Shapley. Notes on the n-person game—ii: The value of an n-person game. *Research Memoranda*, 1951.

[47] Karen Simonyan, Andrea Vedaldi, and Andrew Zisserman. Deep inside convolutional networks: Visualising image classification models and saliency maps. In *ICLR (Workshop Poster)*, 2014.

[48] Wojciech Samek, Grégoire Montavon, Andrea Vedaldi, Lars Kai Hansen, and Klaus-Robert Muller. Explainable ai: Interpreting, explaining and visualizing deep learning. *Lecture notes in computer science., Lecture notes in artificial intelligence.; Lecture notes in computer science, 11700.; LNCS sublibrary., SL 7„ Artificial intelligence*, 2019.

[49] Xavier Glorot, Antoine Bordes, and Yoshua Bengio. Deep sparse rectifier neural networks. In *Proceedings of the fourteenth international conference on artificial intelligence and statistics*, pages 315–323, 2011.

[50] Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. Neural machine translation by jointly learning to align and translate. In *ICLR*, 2015.

[51] Reza Ghaeini, Xiaoli Z. Fern, and Prasad Tadepalli. Interpreting recurrent and attention-based neural models: a case study on natural language inference. In *EMNLP*, pages 4952–4957. Association for Computational Linguistics, 2018.

[52] Jaesong Lee, Joong-Hwi Shin, and Jun-Seok Kim. Interactive visualization and manipulation of attention-based neural machine translation. In *Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing: System Demonstrations*, pages 121–126, 2017.

[53] Sofia Serrano and Noah A. Smith. Is attention interpretable? In *ACL (1)*, pages 2931–2951. Association for Computational Linguistics, 2019.

[54] David Alvarez Melis and Tommi Jaakkola. Towards robust interpretability with self-explaining neural networks. In *Advances in Neural Information Processing Systems*, pages 7775–7784, 2018.

[55] Mengnan Du, Ninghao Liu, Fan Yang, Shuiwang Ji, and Xia Hu. On attribution of recurrent neural network predictions via additive decomposition. In *The World Wide Web Conference*, pages 383–393, 2019.

[56] Ioannis Mollas, Nick Bassiliades, and Grigorios Tsoumakas. Altruist: Argumentative explanations through local interpretations of predictive models, 2020.

[57] Thibault Laugel, Marie-Jeanne Lesot, Christophe Marsala, Xavier Renard, and Marcin Detyniecki. Comparison-based inverse classification for interpretability in machine learning. In Jesús Medina, Manuel Ojeda-Aciego, José Luis Verdegay, David A. Pelta, Inma P. Cabrera, Bernadette Bouchon-Meunier, and Ronald R. Yager, editors, *Information Processing and Management of Uncertainty in Knowledge-Based Systems. Theory and Foundations*, pages 100–111, Cham, 2018. Springer International Publishing.

[58] Weibo Liu, Zidong Wang, Xiaohui Liu, Nianyin Zeng, Yurong Liu, and Fuad E. Alsaadi. A survey of deep neural network architectures and their applications. *Neurocomputing*, 234:11–26, 4 2017.

[59] Junyuan Xie, Linli Xu, and Enhong Chen. Image denoising and inpainting with deep neural networks. In *Advances in neural information processing systems*, pages 341–349, 2012.

[60] Qinxue Meng, Daniel Catchpoole, David Skillicom, and Paul J Kennedy. Relational autoencoder for feature extraction. In *2017 International Joint Conference on Neural Networks (IJCNN)*, pages 364–371. IEEE, 2017.

[61] Antonia Creswell, Kai Arulkumaran, and Anil A Bharath. On denoising autoencoders trained to minimise binary cross-entropy. *arXiv preprint arXiv:1708.08487*, 2017.

[62] Diederik P. Kingma and Max Welling. Auto-encoding variational bayes. In *ICLR*, 2014.

[63] Jiawei Han, Micheline Kamber, and Jian Pei. 2 - getting to know your data. In Jiawei Han, Micheline Kamber, and Jian Pei, editors, *Data Mining (Third Edition)*, The Morgan Kaufmann Series in Data Management Systems, pages 39 – 82. Morgan Kaufmann, Boston, third edition edition, 2012.

[64] Piotr Indyk. Nearest neighbors in high-dimensional spaces, 2004.

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

[66] Maximilian Alber, Sebastian Lapuschkin, Philipp Seegerer, Miriam Hägele, Kristof T. Schütt, Grégoire Montavon, Wojciech Samek, Klaus-Robert Müller, Sven Dähne, and Pieter-Jan Kindermans. innvestigate neural networks! *Journal of Machine Learning Research*, 20(93):1–8, 2019.

[67] Tiago A Almeida, José María G Hidalgo, and Akebo Yamakami. Contributions to the study of sms spam filtering: new collection and results. In *Proceedings of the 11th ACM symposium on Document engineering*, pages 259–262. ACM, 2011.

[68] Ioannis Mollas, Zoe Chrysopoulou, Stamatis Karlos, and Grigorios Tsoumakas. Ethos: an online hate speech detection dataset, 2020.

[69] George A Miller. Wordnet: a lexical database for english. *Communications of the ACM*, 38(11):39–41, 1995.

[70] Martin F. Porter. Snowball: A language for stemming algorithms. Published online, October 2001. Accessed 11.03.2008, 15.00h.

[71] Karen Sparck Jones. A statistical interpretation of term specificity and its application in retrieval. *Journal of documentation*, 28(1):11–21, 1972.

[72] Jeffrey Pennington, Richard Socher, and Christopher D. Manning. Glove: Global vectors for word representation. In *Empirical Methods in Natural Language Processing (EMNLP)*, pages 1532–1543, 2014.

[73] Thomas Davidson, Dana Warmsley, Michael Macy, and Ingmar Weber. Automated hate speech detection and the problem of offensive language. In *Proceedings of the 11th International AAAI Conference on Web and Social Media*, ICWSM '17, pages 512–515, 2017.

[74] Abhinav Saxena and Kai Goebel. Turbofan engine degradation simulation data set. *NASA Ames Prognostics Data Repository*, 2008.

[75] Abhinav Saxena, Kai Goebel, Don Simon, and Neil Eklund. Damage propagation modeling for aircraft engine run-to-failure simulation. In *2008 international conference on prognostics and health management*, pages 1–9. IEEE, 2008.

[76] André Listou Ellefsen, Emil Bjørlykhaug, Vilmar Æsøy, Sergey Ushakov, and Houxiang Zhang. Remaining useful life predictions for turbofan engine degradation using semi-supervised deep architecture. *Reliability Engineering & System Safety*, 183:240 – 251, 2019.

[77] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. BERT: pre-training of deep bidirectional transformers for language understanding. In *NAACL-HLT (1)*, pages 4171–4186. Association for Computational Linguistics, 2019.