A Study about Explainability and Fairness in Machine Learning and Knowledge Discovery

Fabien Bernier

Master Thesis

A Study about Explainability and Fairness in Machine Learning and Knowledge Discovery

Fabien BERNIER

Academic Year 2020–2021

Final year internship done in partnership with

LORIA

in preparation for the engineering diploma of TELECOM Nancy

Internship supervisor: Miguel COUCEIRO, Amedeo NAPOLI

Academic supervisor: Sébastien DA SILVA

Déclaration sur l'honneur de non-plagiat

Je soussigné(e),

Nom, prénom : Bernier, Fabien

Élève-ingénieur(e) régulièrement inscrit(e) en 3e année à TELECOM Nancy

Numéro de carte de l'étudiant(e) : 31816653

Année universitaire : 2020–2021

Auteur(e) du document, mémoire, rapport ou code informatique intitulé :

A Study about Explainability and Fairness in Machine Learning and Knowledge Discovery

Par la présente, je déclare m'être informé(e) sur les différentes formes de plagiat existantes et sur les techniques et normes de citation et référence.

Je déclare en outre que le travail rendu est un travail original, issu de ma réflexion personnelle, et qu'il a été rédigé entièrement par mes soins. J'affirme n'avoir ni contrefait, ni falsifié, ni copié tout ou partie de l'œuvre d'autrui, en particulier texte ou code informatique, dans le but de me l'accaparer.

Je certifie donc que toutes formulations, idées, recherches, raisonnements, analyses, programmes, schémas ou autre créations, figurant dans le document et empruntés à un tiers, sont clairement signalés comme tels, selon les usages en vigueur.

Je suis conscient(e) que le fait de ne pas citer une source ou de ne pas la citer clairement et complètement est constitutif de plagiat, que le plagiat est considéré comme une faute grave au sein de l'Université, et qu'en cas de manquement aux règles en la matière, j'encourrais des poursuites non seulement devant la commission de discipline de l'établissement mais également devant les tribunaux de la République Française.

Fait à Nancy, le October 8, 2021

Signature :

Déclaration de validation du mémoire d'ingénieur

Je soussigné, Miguel COUCEIRO

Encadrant industriel de Fabien BERNIER

Par la présente, je déclare avoir pris connaissance et valider le contenu du mémoire d'ingénieur intitulé :

A Study about Explainability and Fairness in Machine Learning and Knowledge Discovery

Fait à Nancy, le 30.08.2021

Signature :

Master Thesis

A Study about Explainability and Fairness in Machine Learning and Knowledge Discovery

Fabien BERNIER

Academic Year 2020–2021

Final year internship done in partnership with

LORIA

in preparation for the engineering diploma of TELECOM Nancy

Fabien BERNIER
23, rue Saint-Nicolas
54000 NANCY
+33 6 81 69 83 67
fabien.bernier@telecomnancy.eu

TELECOM Nancy
193 avenue Paul Muller,
CS 90172, VILLERS-LÈS-NANCY
+33 (0)3 83 68 26 00
contact@telecomnancy.eu

LORIA
615, rue du Jardin Botanique
54000 NANCY
03 83 59 20 00

Internship supervisor: Miguel COUCEIRO, Amedeo NAPOLI

Academic supervisor: Sébastien DA SILVA

Acknowledgements

This master thesis is the result of a whole story, that started with a research contract when I applied to the LORIA with a simple e-mail in summer 2020. I got a reply from the Orpailleur team. This was a big leap into the unknown, for both of us. But chance has done well.

So first I would like to thank Amedeo and Miguel, who opened to me the *golden doors* of the Orpailleur team. Thanks to Miguel, again, for his time allocated to our project, despite his (too) busy schedule. I still don't know how the ideas pop up from his brain and how he is able to manage all of them at a time. If someone does know, please contact me.

Thank you Amedeo, for bringing your wise experience and advises, your rigor, and your sense of "how to get things done *well*". But also for providing carefulness and a rough, but good, sense of humor.

Also, thanks to Sébastien Da Silva, my academic supervisor, who took the courage to dive into my subject in the middle of the year. But in addition, Sébastien is also part of the Orpailleur team.

Thanks a lot to Guilherme, with whom I worked a lot, and spent a lot of good time. Thank you for your support, for sharing your culture and for integrating me in the LORIA[1]. I wish you the best for the final year of your PhD thesis. Also thanks to Vaishnavi, former intern at the LORIA, who kept interest for the project and continued to contribute from time to time.

Thanks to Bernard Maigret, for his motivation and keeping on supervising projects like this. Confronting our fields was a really great collaboration. I also want to thank Clément, whose internship was linked and complementary to mine. It was constructive, sometimes hard, but nonetheless funny: a true pleasure.

Finally, many thanks to all members of the Orpailleur team. Laura, your conversations are always scientifically and culturally enriching, all the more as we shared the same office. Alexandre, I wish your plan B to work as you want, and that one day you become an astronaut. But above all I also hope your plan A will be successful. Laurine, I would like to add that your strawberry pies are delicious! Tanya, I'm just impressed with how you prepared two PhD theses mainly at the same time. Also thanks to Georgios, Adrien(s), Claire, and to people I met in other teams: Vinicius, Gabriel, Athénais, Dominique and Philippe, Louise, who I already knew. And I almost forgot to mention Isabelle, who brings sunshine at the laboratory restaurant ;)

Thank you so much for all your kindness, to every single one of you. I will really miss you all, but be sure we will be able to meet again next year.

[1] un grand "*cœur*" !

Contents

1 Introduction

When we take a look at the machine learning landscape, we observe that a lot of progress was made last decades, and even more last years. This success is mainly due to the deep learning outbreak [15, 17, 26]. Machine learning (ML) models now manage to perform complex tasks and decision support making in real contexts. It is a bit exaggerated to say that machine learning models look like *magic*, just because they learn automatically to give accurate predictions. Indeed, we know exactly how they are built, what is their architecture and how they learn from data. But there is still something that we generally don't control: what has been learned, how models are processing the data[1].

In the early 2000s, some states in the U.S. started to use COMPAS, a decision support model developed by the Northpointe company (today Equivant), used to predict criminal recidivism likelihood in legal cases[2]. However, a study by ProPublica revealed in 2016 the existence of an algorithmic bias in the predictions, mainly based on the skin color [13]. In fact, black people are twice as likely to be predicted at *high risk* than white people, among non recidivists. This study showed the importance of understanding the mechanisms behind predictions, notably to ensure models are fair. What has been learned by the model? What is the *magic* inside?

This reinforced the emergence of a new field in artificial intelligence: explainability, or interpretability. As models get more and more complex, it becomes very hard for humans to explain the role of each neuron in deep learning models, or to explain each branch in a decision tree with a thousand leaves. Fortunately, research on explainability has given birth to... explainers! Explainers are tools that help us to understand, for a given model and in a comprehensive manner, how the input has been used to produce an output. Today explainability is an area of concern, as on the one hand machine learning is increasingly used, and on the other hand, users want to be informed of the way their data is being processed. It is even more true as the European Union's GDPR law entitles European citizens the right to have a basic knowledge and explanations regarding the process of their personal data.

Although we can assess the fairness of ML with explainers, a huge work remains to make existing models fairer. In 2020, the *Orpailleur* team, in which I did my intership, created an ensemblistic method to improve fairness in machine learning, called *FixOut*[3], that deals with tabular classifiers. We want to study the extent of such a method. In this report, we will especially focus on textual data, try to improve models fairness of textual classifiers, and adapt FixOut to neural networks.

Even though explainers have been created with a concern for fairness and to detect biases, it was found that such methods are also useful in other contexts. In particular, explainers are interesting for a completely different case study, which is molecule classification. Given a molecule, deep learning allows us to build models that manage to predict if this molecule has got antibiotic

[1] intuitively, how the models are "*reasoning*" (although this term is not well defined)
[2] https://en.wikipedia.org/wiki/COMPAS_(software)
[3] https://fixout.loria.fr/

properties, or not. Despite the fact that we can accurately make such predictions to help medical research, one question remains: why? What are the properties that allow the model to say that a molecule is potentially antibiotic? Where are these properties located in the molecule? Are there important patterns? To answer these questions, we decided to use explainers and some features of FixOut, that notably appear to be very beneficial, though fairness and antibiotic research are very different fields. This led to the creation of a tool, in collaboration with chemists and drug researchers, designed to help them in knowledge discovery in their field.

2 Context and environment

2.1 LORIA

I did my internship at the *Laboratoire lorrain de recherche en informatique et ses applications* (LORIA), in the *Orpailleur* team. The LORIA is an *Unité Mixte de Recherche* (UMR 7503), and hosts three parts of research institutes:

- the *Institut national de recherche en informatique et en automatique* Nancy Grand-Est (**INRIA NGE**);
- the *Centre national de recherche scientifique* (**CNRS**)
- the *Université de Lorraine* (**UL**).

The LORIA was born in 1997 from a merge of the INRIA and the **CRIN** (Centre de Recherche en Informatique de Nancy). The buildings and facilities, however, are provided by INRIA, inaugurated in 1986.

The center gathers 30 project teams. Some teams at the LORIA are also INRIA project teams. Remember that LORIA and INRIA are two different entities, that the LORIA is hosted by INRIA Nancy Grand-Est, but that the LORIA includes INRIA NGE researchers plus other laboratories. The LORIA also has offices in Metz, Centrale-Supélec (french engineering school). This is one of the two INRIA NGE satellites (the other in Strasbourg).

As you can see on figure 2.1, the organization is very simple[1]. The LORIA is directed by Jean-Yves Marion, and INRIA NGE is headed by Bruno Lévy, who also appears to be a former student of TELECOM Nancy. Teams are categorized into five departments, and Orpailleur is in the department 4.

Above all, the LORIA has an international scope of impact, since research projects done in the laboratory are subject to be presented in international or European conferences, workshops, journals, etc. A strong international community also works in the lab.

We can see it in some key resources, as of April 2021[2]:

- **186** professors and researchers
- **152** PhD candidates

[1] if we skip administrative details

[2] Source: https://www.loria.fr/wp-content/uploads/2021/04/LEB-6_web.pdf

- **60** nationalities
- **13M€** for contracts

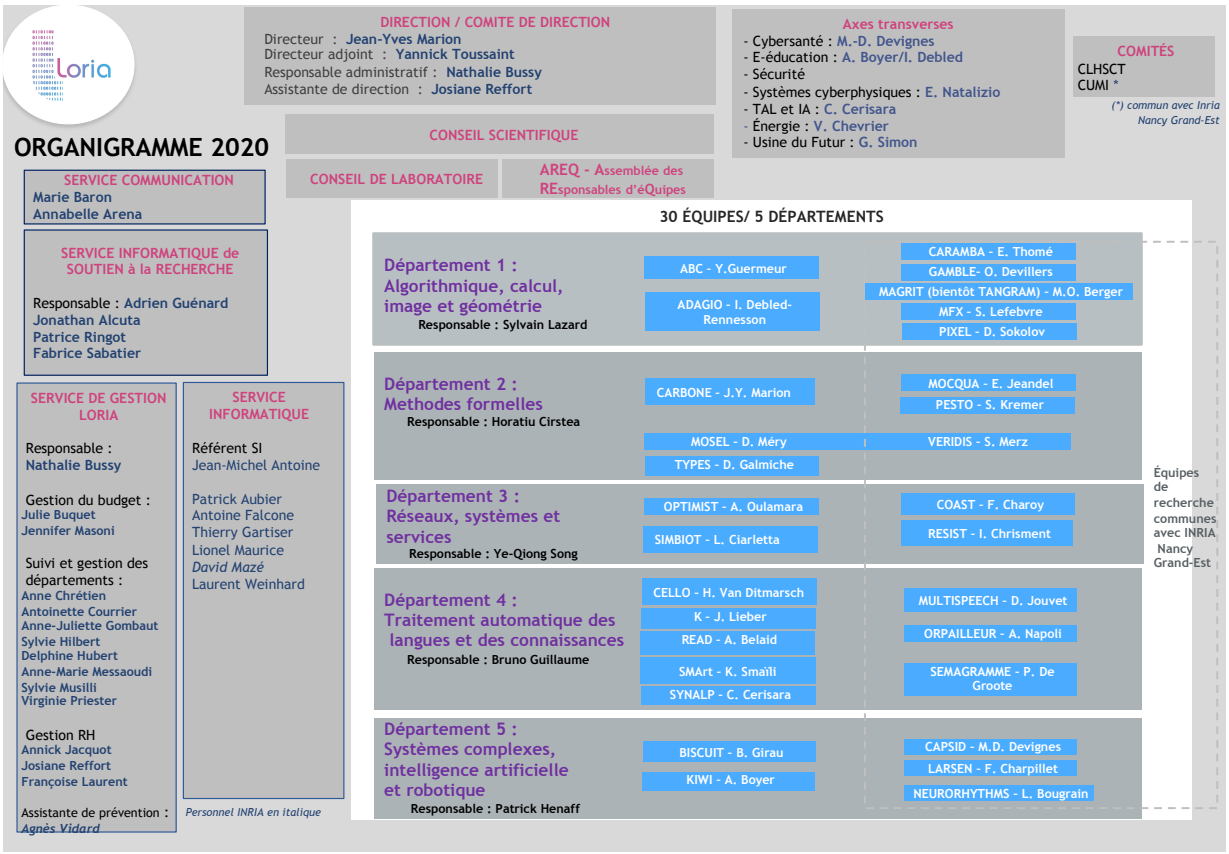


Figure 2.1: LORIA's ogranization chart, as of May 2020. Source: LORIA. Will you find *Orpailleur*?

Photos on figure 2.2 show parts of the buildings. Ones could think it must be very hot in summer, by looking at the second picture. So what? Just wait for a non-negligible equipment in the next sections...

Figure 2.2: Photos of the LORIA buildings[3], respectively from outside and inside, with additional arrows indicating the places where I have been eating, and where I have been working.

[3]Source: https://www.inria.fr/en/centre-inria-nancy-grand-est and https://www.loria.fr/fr/presentation/organisation/

2.2 About the Orpailleur team

It has been created in 1997 by Amedeo Napoli. The team leader isn't Amedeo Napoli anymore, since he is now emeritus researcher and has left his place to Miguel Couceiro. The team is composed of about 23 members, including interns, 10 PhD students, 1 post-doc, 1 engineer, and 9 permanent researchers. A part of the team is located in Metz. Orpailleur conducts research on data mining and knowledge discovery, in general. These fields are nevertheless close to others, such as artificial intelligence or natural language processing, which are also studied by Orpailleur. *Orpailleur* is the french for gold panners, a way to tell that team members are searching for the gold in data.

If you look at the organization chart, you will notice that INRIA project teams are marked on the right. But INRIA teams are not necessarily exclusively composed of INRIA researchers. When I entered the Orpailleur team, it was indeed an INRIA project team[4].

2.3 Internship and research contract

This end-of-study internship follows a research contract, that started in September 2020, and that I chose to do out of interest in research. This is when I met the team, became familiar with the laboratory and its wonderful restaurant, two days a week, all along the first semester. The three other days in the week were reserved for the courses at TELECOM Nancy. The second semester is dedicated to the internship (full time), which started on March 1, and will end on August 31.

In September, it was first planned that I work on molecule classification and predictions, which is a project supervised by Amedeo Napoli and Bernard Maigret (CNRS emeritus researcher, CAPSID team), and that started a few months before, during a master internship. This was in fact postpone to my internship in March / April in order to collaborate with Clément Bellanger, another M2 intern. This is why I first focused on machine learning fairness and biases with Miguel Couceiro and Guilherme Alves Da Silva (a Brazilian PhD student), and continued a bit to work on it along with molecules during my end-of-study internship.

2.4 The covid-19 section

Of course this thesis couldn't avoid a word on the ongoing covid-19 pandemic. Unfortunately, a new epidemic wave happened again in France, which forced the government to set up a new lockdown at the end of October 2020. I then worked remotely from the All Saint's Day vacations to the 9th of June, when the laboratory finally reopened its canteen and allowed people to come in a more flexible fashion, two days a week, and then three days a week, starting from July.

2.5 Organization

My two or three days at the laboratory were selected by taking the hottest days of the week. I was not in the internship room, but in an office of the team, with a PhD student. There was no

[4]But for some reasons, it is not anymore, so the team should now be marked on the left on the chart.

schedule imposed, but I adopted a rhythm, from about 9:00 / 9:30 am until 6:00 pm. Of course I took a lunch break to eat with the PhD students.

On Thursday morning, it is the team's ritual: a Malotec conference[5], organized by Miguel. A researcher comes to discuss a topic for a short hour, then there is a time for the audience to ask questions. It is a kind of of knowledge sharing moment for the team.

During the lockdown, in order to stay in touch, we met through the Microsoft Teams and Discord communication platforms. Around one weekly meeting was organized with Miguel and Guilherme in order to make a progress report on the research around FixOut, discuss the results and try to bring out new ideas; the same every Monday with Amedeo, Bernard and Clement for antibiotics.

The project is managed in an Agile fashion, with Miguel and Amedeo as coordinators. Some deadlines are imposed when it comes to submitting an article for a journal or symposium. The *peer-review* system allows us to have a feedback of the scientific community of the domain, and improve our future papers.

2.6 Equipment

On the hardware side, I use my personal computer to work. For heavier calculations, I have an access to the cluster of calculations *Grid5000*[6]. I also use *Google Colab* for the preparation of some models.

As I previously said, an office was dedicated for my internship. One of the best facilities in the laboratory was undoubtedly **air conditioning**, allowing me not to die at home during heat waves.

[5]Mathematics and Logic for Knowledge Extraction and Processing: https://malotec.loria.fr/
[6]https://www.grid5000.fr/

3 State of the art

In order to fully understand the experiments performed, it is important to understand the techniques and tools used, which I will try to clearly describe in this section.

3.1 Machine learning models

First of all, this end-of-study project deals with many machine learning models. You may notice that "*machine learning*" is composed of two words, stating that this allows *machines* to *learn* by themselves from data. By "*model*", we here understand a function that takes an instance as input, and outputs a prediction. Machine learning models have then in general two stages:

- the *training* stage, where the model learns from data: its parameters are adjusted in order to fit to the training set;
- the *testing* stage, where we assess the model on new data.

The way the model is constructed is described by *hyper-parameters*.

We here focus on *supervised training*, which means we must indicate to our model what output we expect. More specifically, such a model is trained on labelled data, i.e. the dataset consists in a list of instances (inputs), which are associated to the output we expect to get from the model.

We also focus on *classification* tasks, i.e. the role of our model is to predict one class among many ones (two or more). This is to be distinguished from regression tasks, which aim to predict continuous values. For a classification problem, machine learning generally outputs values between 0 and 1 for each class, that we can interpret like a belonging probability, or a *likelihood*. The final binary prediction for a class membership is obtained by discriminating the predicted probability with a threshold, 0.5 being the default value in many systems.

Formally, a model M is simply a function that takes an input x and gives an output y: $M : x \mapsto y = M(x)$. We will denote $\hat{y}$ the expected output for x.

3.1.1 Logistic regression

This method is quite similar to linear regression, though logistic regression is used for classification. The prediction is a simple weighted sum of the inputs, plus an additional bias. This sum is then passed through an activation function, giving a result between 0 and 1.

Formally :

$$y = \sigma\Big(\sum_{i=1}^{n} a_i x_i + b\Big) \tag{3.1}$$

Where y is the predicted output, x_i are the inputs, a_i and b are the training parameters, i.e. parameters that will change during the training, and σ is a logistic function[1]. We can notice that a weight is attributed to each input, which allows us to be aware of the importance given to these different inputs. A common logistic function is the sigmoid, where σ is defined as:

$$\sigma(x) = \frac{1}{1 + e^{-x}} \tag{3.2}$$

3.1.2 Random forests and AdaBoost

Random forests are ensembles of decision trees. Each decision tree of the ensemble is trained on a random subset of the dataset, and for which some attributes are randomly removed. To get the final result, each tree makes a prediction. These predictions are then averaged to obtain a global *score*, which, in general, appears to be more accurate than a simple decision tree, trained on the whole dataset. This technique was formalized for the first time in 2001 by the American statistician Leo Breiman [6].

Because decision trees may be complex and because ensembles such as random forests typically contain around a hundred decision trees, such models are difficult to interpret. We can nonetheless visualize single trees, as shown on figure 3.1 here-under.

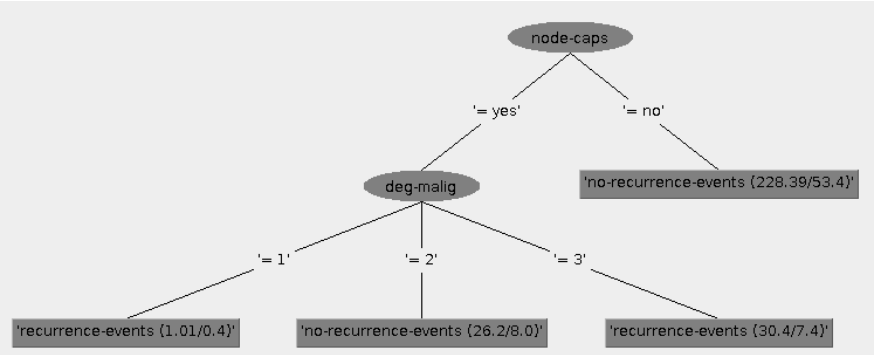


Figure 3.1: Example of visualization of a decision tree on the *Weka*[2] software.

AdaBoost is also an ensemble, but the idea is different in that it consecutively trains submodels, one by one. It begins with a simple model (e.g. a decision tree), and then trains another by mostly reusing instances for which the previous model predicted a wrong answer, and so on. This technique is not really a model in itself, but a meta-algorithm that is agnostic to the model,

[1] More general information can be found at https://en.wikipedia.org/wiki/Logistic_regression#Multiple_explanatory_variables and https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression (from *scikit-learn*, that will be used later).

[2] https://www.cs.waikato.ac.nz/ml/weka/

i.e. AdaBoost can be used with any model. It is however used most of the time with decision trees[3].

This was first discovered and introduced by Yoav Freund and Robert Schapire in 1997 [11].

3.1.3 Neural networks

Neural networks consist in several layers of artificial neurons. A neuron is a simple function that takes many inputs and return one output that can be distributed to other neurons. A well known and used function is the perceptron, discovered in 1958 by Frank Rosenblatt [22], inspired from the biological neuron, and described on figure 3.2. An artificial neuron slightly differs from a simple logistic in that it can outputs values in wide ranges (not only in [0, 1]), according to the chosen activation function.

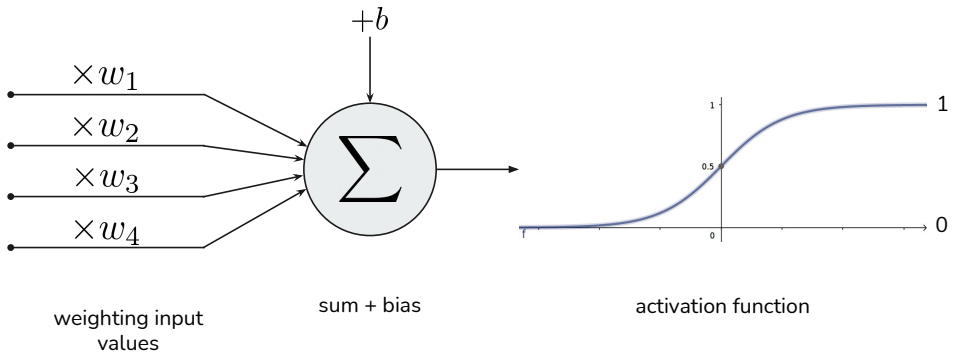


Figure 3.2: Pipeline of an artificial neuron, the perceptron

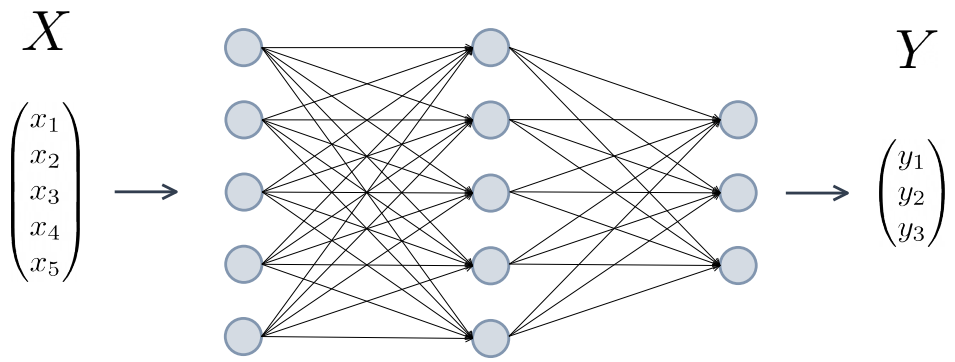


Figure 3.3: Diagram of a simple neural network (here more precisely a multilayer perceptron) with two layers

Based on this simple idea, many combinations and operations can be used to form a neural network layer. Here below are the roles of some layers that we will use during the project description. Even though it has very interesting inner mechanisms, we will not focus on it, and skip their details.

[3]We can also add, that *scikit-learn* and other machine learning engines like *Weka*, *Microsoft Azure ML Studio*, implement AdaBoost with decision trees by default.

The key role of differentiability

Just before jumping into the description of these layers, it is important to notice a fundamental property of neural networks: they are differentiable models!

It is necessary since deep learning models are trained by using a *gradient descent* algorithm, minimizing the error calculating (between the predicted y and the expected $\hat{y}$) by calculating its gradient with respect to the model's weights. To make this possible, every operation used in the network, including the error computation, must then be differentiable.

Discovered simultaneously in 1986 by the French researcher and engineer Yann Le Cun, and also by David Rumelhart and Geoffrey Hinton, the backpropagation algorithm [14] is a technique used to compute the gradient w.r.t. any parameter in neural networks. They received the Turing Award in 2018 for their work, except for Rumelhart who regrettably died in 2011.

Recurrent layers

These layers are used to deal with sequences of data. More precisely, we will be led to use Long Short-Term Memory (LSTM) and Gated Recurrent Unit (GRU) layers, which contain a memory cell that is updated each time a new instance of the sequence is given, as described on figure 3.4.

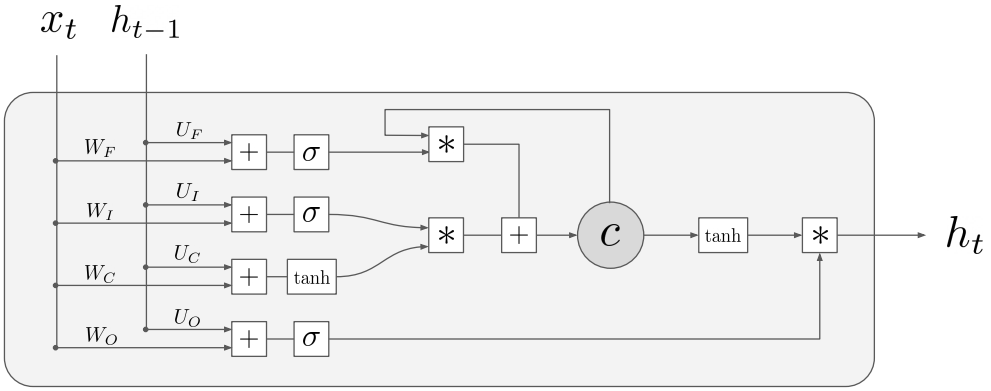


Figure 3.4: Illustration of an LSTM unit, a typical example of recurrent layer. The layer outputs a vector h_t (for the t^{th} term of the sequence) and takes its last output h_{t-1} as an additional input.

LSTMs have first been mentioned in 1995 by the German researchers Sepp Hochreiter and Jürgen Schmidhuber, and GRUs have been proposed in 2014 by Kyunghyun Cho as a simplification of LSTMs [8]. They were mostly used for text classification and generation, since text can be considered as sequences of characters, syllables or words.

Attention layers

Formulated in 2014 [4], attention layers also deal with sequences of data, but treat all the elements of the sequence at a time, and compute importance weights for each element. These weights thus represent how much attention we should give for each item, and are also called *attention weights*. The figure 3.5 shows a usage example with a sequence of words, in a neural network. Such layers are now very commonly used in NLP models, and have rapidly led to the creation in 2017 of new

neural networks with a specific architecture, called *transformers*[4] [24].

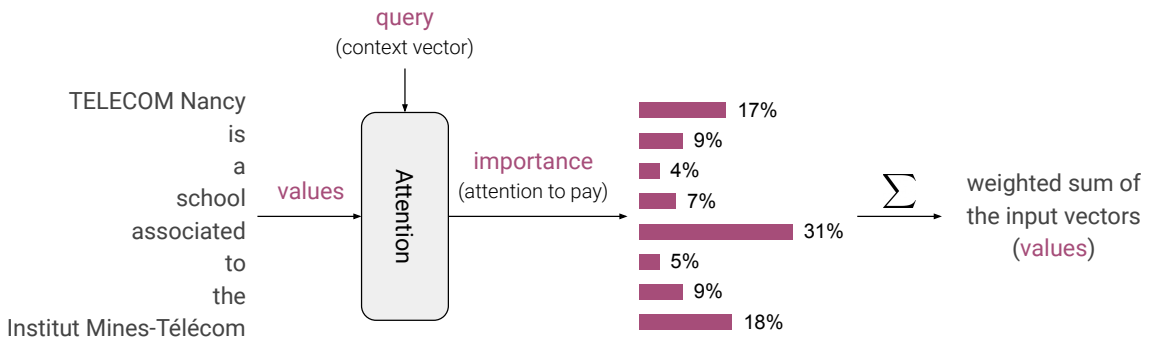


Figure 3.5: Simplified diagram of the operations performed by an attention layer. In practice, words are embedded into latent vectors. This is why we can carry out a weighted sum of *words*.

Understanding what is an *embedding*

It is very common to talk about *embeddings* in deep learning, and we will be using this term many times in this report.

An *embedding* is the short name for an embedding **vector**, which describes an object with a restricted number of dimensions. We expect from embedded objects that the more similar they are, the smaller their vector distance will be. This conversion results in a vector space of embeddings, also called a semantic space, or a **latent space**.

It is often applied in NLP, for example to embed words – we then talk about *word embedding*. In such a latent space, words that are close in meaning are pretty close in vector distance (either in euclidean distance or in cosine similarity). The figure 3.6 shows an artificial example to illustrate the typical distribution of words in a latent space, in two dimensions.

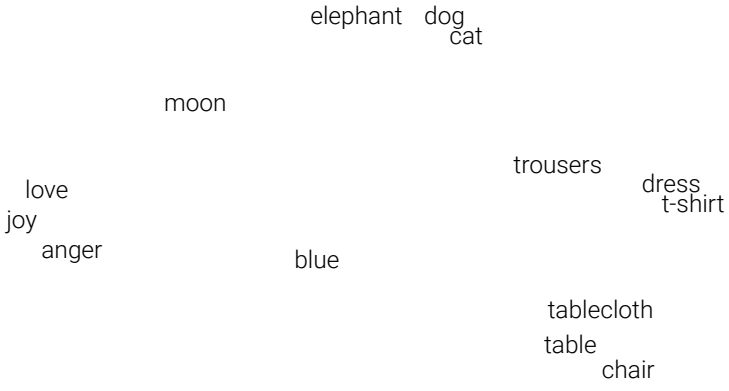


Figure 3.6: Illustration of words distribution in a two-dimensional latent space[5]

[4]Besides, with the release of GPT-3 in 2020 and Codex in 2021, the company *OpenAI* demonstrated that transformers can perform text completion very well when such models are widely trained, including comprehensive dialogs, consistent stories, and code in languages like Python or Javascript.

[5]Note that in practice, the latent space dimension is much higher, depending on the task performed. We can find for instance pre-trained word2vec models with 50 to 300 dimensions in `gensim`, a Python module dedicated to word and document embedding. Dimensional reduction techniques such as PCA are nonetheless often used to visualize distributions in a human-interpretable fashion.

Graph Neural Networks (GNNs)

GNNs contain layers that deal with graph structures. Each node, and/or each edge of the graph is represented by a vector of values. The main model we use for antibiotics classification (more details in the next subsection) is an example of GNN.

3.2 Antibiotics classification

This part only concerns antibiotics classification, more precisely the task consisting in predicting whether a given molecule is an antibiotic. Two models have been used to manage this task. These two models are conceptually different, but both work on the same input format.

3.2.1 Databases and SMILES format

Antibiotics classification is more precisely the action of classifying a molecule as antibiotic or not. This is therefore a simple *two-class* classification problem. However, in some cases, we are also interested in knowing more precisely the antibiotics family of the molecule. Since there are many families, the classification problem becomes *multi-class*.

The datasets are then quite clear: CSV files that contain a molecule in the first column, as an input, and the next n columns are the n classes for the classification problem, containing a zero in case of non membership, and a one in the opposite case.

However, it is not obvious how to encode molecules in the dataset, since molecules have complex structures, that can be simplified as graphs. In our case, all molecules are described in the **SMILES** format, which stands for Simplified Molecular-Input Line-Entry System. In simple terms, this is just a way to describe the graph of the molecules with a string of characters.

Finally, our datasets look like the sample of table 3.1.

SMILES	**Class_0**	**Class_1**
O=C(O)CS(=O)(=O)c1ccc(OCc2cccc(-c3ccccc3Cl)c2)cc1	1	0
CN(C)C(=O)N1C[C@H]2C[C@@H](NCC(=O)N3CCC[C@H]3C#N)C[C@H]2C1	1	0
O=C(O)CCCCCCCC(=O)O	0	1
...	...	...

Table 3.1: Sample of a CSV dataset with two classes (antibiotic and non antibiotic).

The data comes from ChEMBL[6], ChEBI[7], PubChem[8], and other databases previously aggregated by Bernard Maigret and Clément Bellanger.

[6] https://www.ebi.ac.uk/chembl/
[7] https://www.ebi.ac.uk/chebi/init.do
[8] https://pubchem.ncbi.nlm.nih.gov/

3.2.2 Chemprop

Chemprop[9] is a graph neural network, more precisely using a message passing network (MPN) in order to make predictions, by representing molecules as graphs. It was released in 2019 by researchers from MIT [25]. It is written in Python with the **PyTorch** library. As explained in their paper, the key feature of Chemprop compared to previous molecule models is that the MPN embeds bonds (edges of the graph) instead of atoms (nodes).

Each molecule in the SMILES format is converted to a graph (called a **mol graph**), where both nodes and edges are represented by vectors, respectively described in tables 3.2 and 3.3 as in the original paper. Features represented with bits are categorical, and those with more than one bit are encoded into a *one-hot* vector, meaning that one bit over all will be activated to 1, leaving the others to 0. Figure 3.7 describes the pipeline of Chemprop.

Also, we can notice that despite a node is called an *atom*, it is not always an atom, chemically speaking. For example, a node can be of type NH_2, which is indeed a nitrogen atom completed by two hydrogens. Thus, the term *atom* here refers to a node of the mol graph.

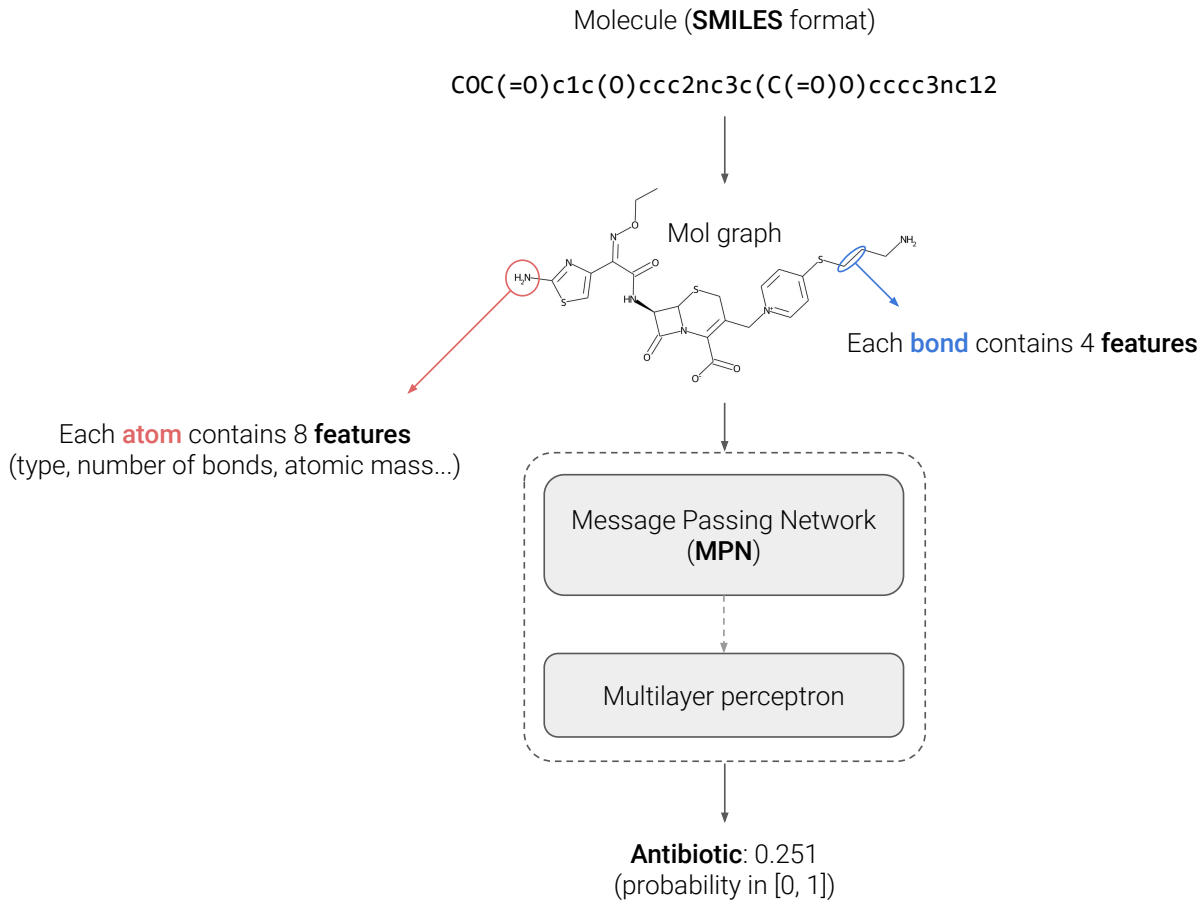


Figure 3.7: Diagram describing the Chemprop pipeline with an example of classification.

[9] https://github.com/chemprop/chemprop
[9] and four other institutions, for a total of 15 co-authors in the main article, so that is quite a lot of people!
[9] sizes differ by 1 bit from the original paper since they added an extra bit for the unknown state.

Feature	Description	Size
Atom Type	type of atom (ex. C, N, O), by atomic number	101 bits
#Bonds	number of bonds the atom is involved in	7 bits
Formal Charge	integer electronic charge assigned to atoms	6 bits
Chirality	unspecified, tetrahedral CW/CCW, other or unknown	5 bits
#Hs	number of bonded hydrogen atoms	6 bits
Hybridation	sp, sp2, sp3, sp3d, sp3d2 or unknown	6 bits
Aromaticity	wether the atom is part of an aromatic system	1 bit
Atomic mass	mass of the atom divided by 100	1 float
Total vector size		**133**

Table 3.2: Description of atom features in a mol graph

Feature	Description	Size
Bond type	none:0, single, double, triple, or aromatic	5 bits
Conjugated	whether the bond is conjugated	1 bit
In ring	whether the bond is part of a ring	1 bit
Stereo	none, any, E/Z, cis/trans or unknown	7 bits
Total vector size		**14**

Table 3.3: Description of bond features in a mol graph

3.2.3 DeepChem

This project has a misleading name, because unlike Chemprop, DeepChem is not a deep learning model. It is in fact a complete **toolkit** from the Stanford University[10] that can be jointly used with many machine learning models in order to help drug design [20]. Nevertheless, deep learning models are often used in machine learning for molecular research.

3.3 Explainers

Now that we introduced machine learning and different useful models, here is a very important point for the projects we will be describing. As we previously said, machine learning models are automatically trained. However, in particular with the advent of deep learning, these models tend to be so large that at some point it becomes impossible for humans to interpret the way the model treats information. Unlike classical algorithms, this process chain is not *under control*, since it is not manually programmed.

Fairness issues as discovered in COMPAS [13], and the lack of trust in the way machine learning processes data, has led to techniques capable of enlightening the explanation of decisions taken by these models. It has notably a key role in real world applications as in decision support in medical imagery, but also in legal applications, to ensure models robustness and fairness. A use case example is illustrated on figure 3.8.

In short, explainers are useful to establish a trust relationship between the users and machine learning, allowing humans to deem if a prediction makes sense or not.

[10]The project has many other contributors, including companies, and is open-source: https://deepchem.io/

(a) Original Image (b) Explaining *Electric guitar* (c) Explaining *Acoustic guitar* (d) Explaining *Labrador*

Figure 4: Explaining an image classification prediction made by Google's Inception neural network. The top 3 classes predicted are "Electric Guitar" ($p = 0.32$), "Acoustic guitar" ($p = 0.24$) and "Labrador" ($p = 0.21$)

Figure 3.8: Extract from the Ribeiro's article [21] showing the contributions for an image classification.

We can identify two kinds of explainers:

- **Local explainers**: explain a single prediction, for a specific instance;
- **Global explainers**: explain the general behavior of a model, for any instance.

3.3.1 LIME

Introduced in 2016 by Marco T. Ribeiro et al. [21], LIME has become one of the most popular explainers in artificial intelligence. The particularity of LIME is that it is *model-agnostic*, meaning that it can be applied to any classifier – not only in machine learning, but more broadly in any predictive model giving a probability of class membership from inputs, machine learning merely implying that the model has been trained automatically. LIME indeed stands for Local Interpretable Model-agnostic Explainer. Figure 3.9 shows the role played by LIME in explaining classifications.

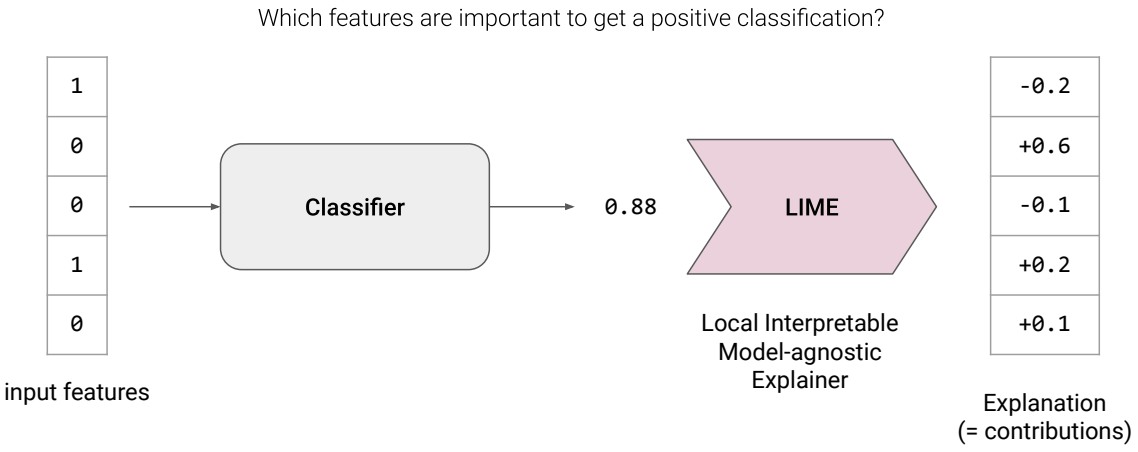


Figure 3.9: The role of LIME in a toy example, for a single classification.

In order to explain an instance[11] x, LIME generates a neighborhood around it. By default, the neighborhood has a population of 5000, distributed according to a normal law[12]. We can also

[11] Formally, an instance is a vector.

[12] https://github.com/marcotcr/lime/blob/master/lime/lime_tabular.py#L305

tune a variance parameter, that regulates the spread of the neighborhood. Here is where the *local* term comes from: the neighborhood is obtained by making slight variations of the original instance, and these variations are so close that we can consider the predictive function to be linear w.r.t. the neighborhood.

Because the classifier is locally approximately linear, a linear regression is done on the neighborhood, in order to match the local predictions. Then we get the weights of this regression, indicating the importance in contribution for each input feature.

Here is the explanation: a **contribution coefficient** for each input feature. Contributions are such that:

- A feature with a **negative contribution** tends to lower the prediction down to 0 (contribution to the negative class);
- A feature with a **positive contribution** tends to raise the prediction up to 1 (contribution to the positive class);
- The **closer** the contribution is **to 0**, the less impact it has on the prediction.

3.3.2 SHAP

SHAP (SHapley Additive exPlanations) is similar to LIME in that it is also a local model-agnostic explainer, which can give contribution coefficients for any classification, provided that the prediction is a probability of class membership. It has been introduced in 2017 by Scott Lundberg from Microsoft Research [16].

The way to compute contributions is however a bit different. It uses coalitional game theory, by computing Shapley values, in order to determine what is the contribution of each feature to get from the base average prediction for all instances, to the precise prediction for a particular instance. Figure 3.10 comes from the Github repository[13] of SHAP and illustrates this principle[14].

Figure 3.10: Toy example to show intuition for SHAP.

3.3.3 PathExplain

PathExplain is also similar to LIME and SHAP, being a local contribution explainer. Nevertheless, PathExplain is not model-agnostic, since it only works on **differentiable models**. As we previously said, deep learning models are differentiable, so PathExplain can work on pretty much any deep learning classifier!

[13] https://github.com/slundberg/shap
[14] Lundberg itself also made a video to explain it in simple terms: https://youtu.be/-taOhqkiuIo

Conceived in 2020 [12], it is available as a Python module, working with Tensorflow and PyTorch models. It merely computes contributions with the integrated gradients formula, an explanation approach for differentiable models submitted in 2017 [23]:

$$\Phi_i(x) = (x_i - x_i') \int_{\alpha=0}^{1} \frac{\partial f(x' + \alpha(x - x'))}{\partial x_i} d\alpha \tag{3.3}$$

Where $\Phi_i(x)$ is the contribution[15] of the feature i in the instance x, x' is a base instance (vector of features with their base values), and f is the model to explain (remember that a classifier model is also a (big) function). In other terms, we integrate $\nabla_x(y)$ (the gradient of the output y with respect to the input x), by varying x from its base value to the instance we want to explain. In even simpler terms, when varying the input from its base value to its current value, the output changes. The integrated gradient simply give the mean variation of the curve in this interval, and weights it by the length of the interval.

PathExplain also provides a procedure to explain interactions between features. We will go back on this functionality later.

3.4 Models metrics

It is important to understand that classification scores are based on the confusion matrix, as presented on figure 3.11.

ground truth

prediction	No	Yes
No	TN	FN
Yes	FP	TP

Example :

	No	Yes
No	52	9
Yes	6	33

Figure 3.11: Definition of a confusion matrix, "*yes*" and "*no*" meaning whether the classification prediction is positive or negative. An toy example with a total of 100 classifications is given.

From this matrix, we can define the following scores that will be used in the next sections:

- **Accuracy**:
$$\frac{TP + TN}{TP + TN + FP + FN}$$
- **Precision**:
$$\frac{TP}{TP + FP}$$
- **Recall**:
$$\frac{TP}{TP + FN}$$

[15] In the paper, contributions are more precisely rather called *attributions*, because they have specific properties, such as the fact that the sum of all the attributions equals 1. The idea is that we have 1 unit of importance that we *attribute* (or distribute) to all the features.

3.5 Software

All the experiments and productions would not have been possible without some existing libraries. All projects presented in this section are open-source. Their logos are presented on figure 3.12, for visual identification purposes.

Figure 3.12: Project logos, respectively from *scikit-learn*, *Tensorflow*, *PyTorch*, and *RDKit*.

3.5.1 Scikit-learn

Usually abbreviated *sklearn*, this Python library provides a wide toolkit to design, train and assess machine learning models. The library itself is a well designed software engineering project, so that it is often very simple to substitute a model with another, because they implement the same object methods, no matter the algorithm used.

3.5.2 Tensorflow

TensorFlow is an open-source library developed by Google which includes several functionalities allowing to build differentiable models. It is particularly powerful in the context of deep-learning field, in which the library is widely used.

The gradient is computed thanks to a *graph* of operations, which stores in memory all operations applied to get to the result. Because Tensorflow knows the derivative of functions used, it is capable of applying the backward gradient algorithm to compute any derivative in the model.

Tensorflow is pre-compiled, and can perform computations on GPUs, making it faster to run, even though it used as a Python module, which is an interpreted language[16].

Tensorflow always works with tensors, which can be seen as an extension of matrices, but with any dimensions (not only rows and columns). It can also jointly be used with **Keras**, which is another library bringing super useful tools to build Tensorflow models easily.

3.5.3 PyTorch

PyTorch is an alternative to Tensorflow, developed by Facebook. A notable difference is that Tensorflow builds a graph from the model's definition, that is, the graph is attached to the model;

[16]Because instructions are executed one by one, interpreted languages are generally less efficient in execution time than compiled languages

whereas Pytorch builds the graph by registering an history inside the tensor objects, that is, the graph is attached to tensors.

PyTorch is not compatible with Keras, but already contains really useful tools to build models.

3.5.4 RDKit

RDKit is also a toolkit for Python, containing tools for molecular data processing. It manages, among others, the following tasks:

- Converting molecules in raw formats such as a SMILES string into graphs;
- Determining simple properties of the molecule (chemical polarity, hydrophobicity, type of bonds, etc.);
- Drawing molecules, with customizable options (format, colors, font, etc.).

Chemprop already depends on RDKit to convert molecules into graphs, and determine simple properties in order to build graph mols (graph containing vectors of properties), previously presented.

3.5.5 Flask

Flask is Python web framework, which enables the development of simple web servers, with a few lines of Python code. It manages to receive HTTP requests, parse them, which allows us to process information, and send back a response, typically in JSON or HTML format.

We will describe the exact way we use it later, so as to build a web interface to interact with our code and make its usage easy.

A popular alternative of Flask is Django, which embeds more functionalities and security features, but which is also heavier.

4 Contributions to *FixOut*

4.1 Problem description

4.1.1 Fairness issues in machine learning

As we said in the introduction, machine learning models are trained automatically, and are most of the time so complex that it is impossible for human people to interpret its inner workings, or more intuitively "*how it is reasoning*". A consequence to this effect is the emergence of fairness issues, without being aware of it.

This is the case of the COMPAS model [13], which has been used for several years in many states of the USA to predict criminal recidivism. These predictions were taken into account for court sentence decisions. However, it took more than a decade to highlight the fact... that the predictions were mainly biased on skin color.

In machine learning, fairness issues are a recurrent problem. Are the programmers discriminating? Not necessarily, but society globally is! The problem is, in fact, that models are trained on real world data. Hence, society has an indirect impact on what the models learn. This issue and the ML's "*black box*" problem motivated the emergence of explainers, presented above.

Accuracy ≠ fairness

Yes, a model can be very efficient, but at the same time very unfair. For example, it is, in fact, statistically true that black colored people reoffend more often than white people[1], but the original cause is obviously not the skin color, but rather social disparities. Because of this correlation, based on skin color, a model can indeed be very accurate. Here is how an accurate model can be *wrong*, and again, here is the important role of explainers to identify it.

Such a model is *wrong* ethically speaking. It is important to underline that fairness issues are subjective, depend on the culture, and the definition of **ethics**.

Ways of mitigating unfairness

Figure 4.1 describes each step before we reach to the conclusion that a model is unfair. We can *repair* fairness anywhere in this chain:

[1] https://phys.org/news/2018-10-black-men-higher-recidivism-factors.html

- In the real world? Finding causalities and treating the problem from source would be excellent, but this is no easy task;
- We could redefine the assessment process, involving changing the definition of fairness, and ethics. However by definition of our problem, this notion is fixed and static;
- We can change the training dataset, for instance by balancing it with techniques like SMOTE [7], as unbalanced datasets can be a source of bias. Changing the data is not quite easy since a lot of sifting must be done;
- We can try to fix the model by improving its fairness in predictions. Not that easy? Here is where FixOut comes in!

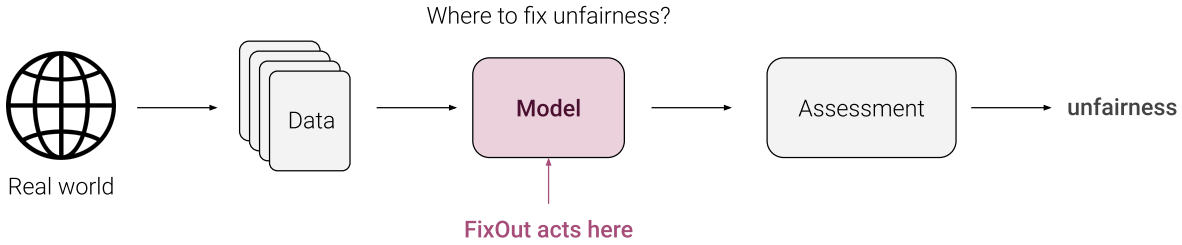


Figure 4.1: Simplified pipeline of unfairness identification, from the real world.

4.1.2 The purpose of FixOut

You are right, it is not that easy. Here is why.

The notion of sensitive feature

Let's get back to our recidivism prediction example. First we can identify skin color as a **sensitive feature**. We define a feature as *sensitive* when its usage could cause ethical concerns in the classification task being performed. In the case of recidivism classification, skin color is a sensitive feature, because it seems foolish and unethical to take such a decision based on individual appearance. Again, the selection of sensitive features is subjective since it relies on the user's definition of *ethics*, and the context of the task being performed: for example, it could be normal to consider the gender of a person for medical purposes, but on the other hand it could be unfair to consider it while searching for a job.

Ignoring sensitive features does not work...

Skin color is nonetheless a very important feature to make the classification, according to the model. Since data about skin color is the origin of the problem, we could simply alter the model to ignore this property, and force it to learn on the other features. This way, we can be assured that this sensitive feature will no longer have any contribution on the output. Simple solution, but big inconvenient: in practice, removing such a feature drops the accuracy... at a point that the model becomes unusable.

...but we can find a compromise

We must then find a compromise, between keeping sensitive features and ignoring them to remove their contributions. In 2020, Vaishnavi Bhargava (then an intern), Miguel Couceiro and Amedeo Napoli proposed a solution called *LimeOut* [5], consisting of making an ensemble where models ignore sensitive features, but others don't. The ensemble is then fairer, but keeps its performance, measured with accuracy, precision and recall scores. The method to make this ensemble is called *EnsembleOut*. *LimeOut* evolved into a more general purpose, *FixOut*, in order to allow the usage of any explainer.

How FixOut works

In fact, explainers take part of the FixOut process before we use *EnsembleOut*. More specifically, FixOut consists in two steps:

1. **Global explanation**: we globally explain the model in order to get the k most important features. The definition of the k number must be defined by the user, even though Guilherme recently proposed a way to automate it [1]. If at least two sensitive features appear in this top-k list, then EnsembleOut applies. Otherwise the model is deemed fair;
2. **EnsembleOut**: its objective is to lower the contribution of selected sensitive features, that appear in the top-k. For n sensitive features to reduce, we train $n+1$ copies of the original model:
   - The i^{th} model is trained by ignoring the i^{th} sensitive feature, $1 \leqslant i \leqslant n$;
   - The $n+1^{st}$ (and last) model is trained by ignoring all the n sensitive features at a time.

   We then manage to build an ensemble of $n+1$ models, whose prediction is the average of each output (probability) provided by the sub-models. This average can also be weighted [1–3].

We can then make a global explanation on this ensemble, to check the diminution of sensitive features contributions. Figure 4.2 illustrates this process.

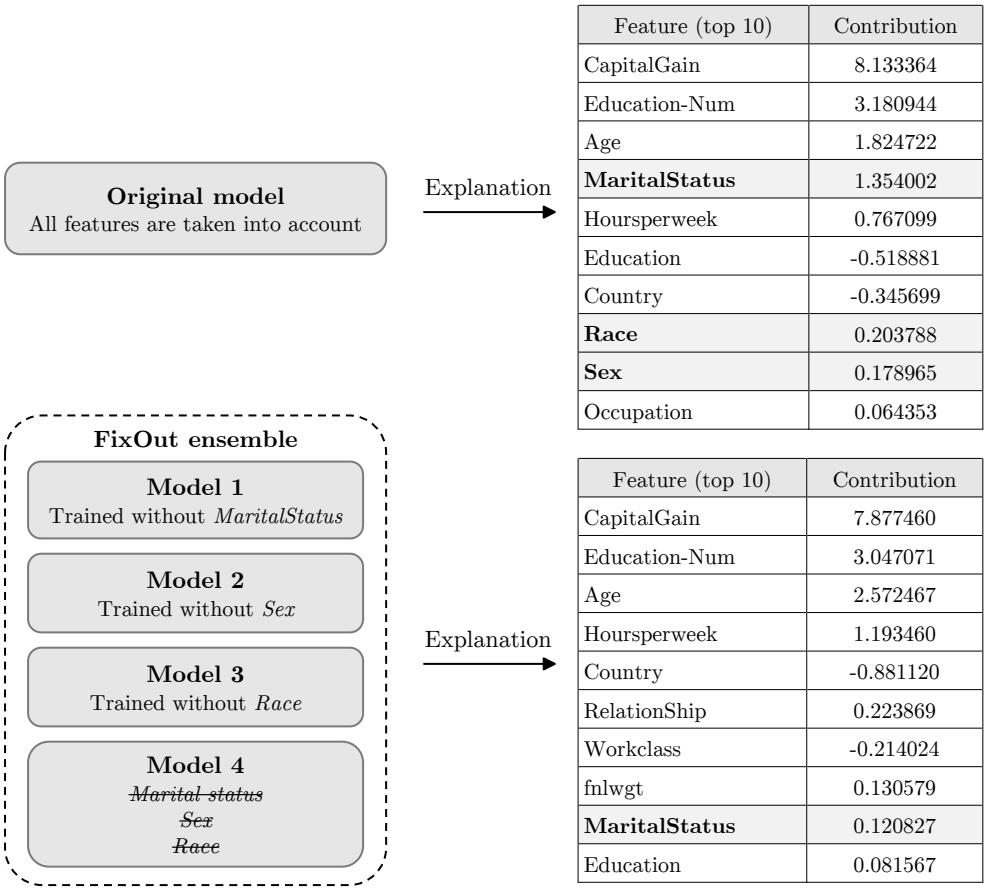
Experiments also show that not only sensitive features become less important according to the explanations, but fairness metrics for tabular classifiers also indicate a little improvement, in fairness. The most interesting thing is that FixOut manages not only to do all this, but also manages to keep the original accuracy, or even sometimes increases it a little bit [1–3, 5].

4.1.3 The interest in textual applications

Not all models are tabular, and fairness issues in textual applications are also common. Let's illustrate unfairness in machine learning by proposing a simple experiment on *Google Translate*[3]. Try to translate the text corpus in figure 4.3 describing actions, from Hungarian to English.

A property of the Hungarian language is that it has no grammatical gender. When translating in English, the model must then choose between *he* or *she*. As of August 2021, the translation given

[2] https://archive.ics.uci.edu/ml/datasets/statlog+(german+credit+data)
[3] https://translate.google.com/



Feature (top 10)	Contribution
CapitalGain	8.133364
Education-Num	3.180944
Age	1.824722
MaritalStatus	1.354002
Hoursperweek	0.767099
Education	-0.518881
Country	-0.345699
Race	0.203788
Sex	0.178965
Occupation	0.064353

Feature (top 10)	Contribution
CapitalGain	7.877460
Education-Num	3.047071
Age	2.572467
Hoursperweek	1.193460
Country	-0.881120
RelationShip	0.223869
Workclass	-0.214024
fnlwgt	0.130579
MaritalStatus	0.120827
Education	0.081567

Figure 4.2: The FixOut's process, with an example on the *German* dataset[2], and $k = 10$. Bold features are considered sensitive.

by *Google* is presented in figure 4.3, and reflect most of the old stereotypes in our society [19][4]. This experiment can be done with other translators, and we can compare their behavior in such a case.

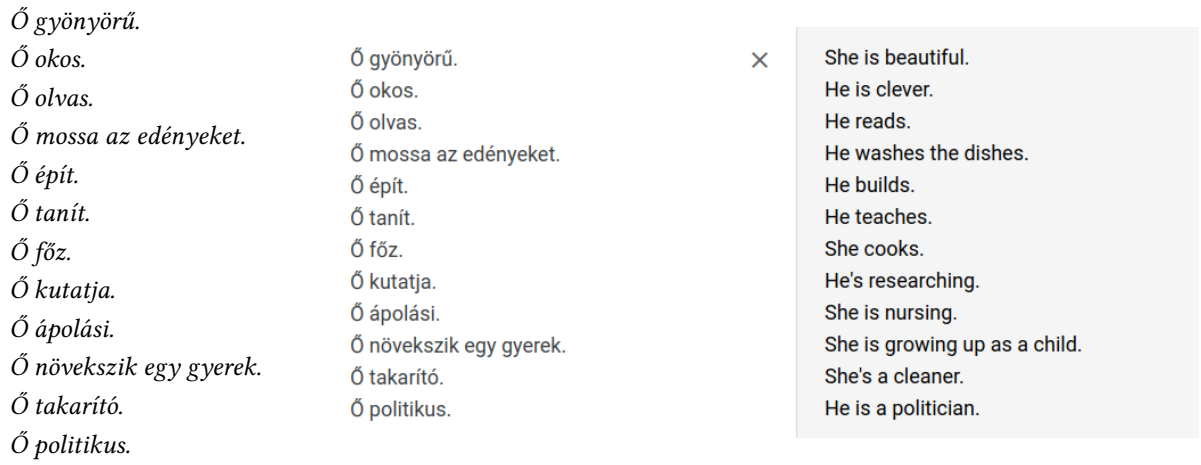


Figure 4.3: On the left the original text (without gender), on the right the translation from *Google Translate*.

[4]Even though the article highlighting this issue was published in 2019, a website already mentioned it in 2017: https://meanwhileinbudapest.com/2017/11/22/languages-without-gendered-pronouns-reveal-that-google-translate-is-a-sexist-bitch/

The Davidson's hate speech classifier

In 2019, Thomas Davidson, a researcher in sociology at Cornell University, brought to light the presence of biases in classifiers processing textual messages, such as in Twitter [9,10]. He trained a classifier to predict whether a tweet is offensive or not, and repeated on several datasets. He reached to the same conclusion, which is an ethnic issue again: the Afro-American community has a higher chance of having tweets classified as *offensive* than the white community in the USA, indicating the presence of a bias based on the message content.

4.2 General adaptation of FixOut to text

4.2.1 Problem analysis

Difference between tabular and textual data: the total number of features

During the first semester in research contract, I tried several approaches to identify words or terms responsible for this bias in the Davidson's models. The major difference when adapting FixOut to text is the complexity of data. In tabular data, with typically maximum tens of features, it is relatively easy to check out all the features one by one and decide if this is a sensitive feature or not. By explaining the model, we manage to have an idea of whether these features are responsible of a bias or not, and if it is necessary to lower their contributions.

On the contrary, textual models consider text tokens as features. A token can be a character, a syllable, or a word[5]. For example, by considering characters as tokens, we get a few different features: the number of features is equal to the number of tokens, which we also call the **vocabulary size**. But the problem is... we can't assess the fairness of the model just by checking the global characters contributions, it doesn't help us to explain key words!

This is why we work with words as tokens, which results in **thousands of features**. Despite my attempts done before March 2021 to choose which words must be selected as sensitive in order to lower their contributions, no such general method has been found, as discussed in the previous report.

No bias detection, but EnsembleOut nonetheless possible

What seemed to work, however, was my proposed solution to adapt EnsembleOut to text. My solution was to add a word filter using a regular expression, in order to play the role of feature dropout, as shown in figure 4.4.

Notice that the filters don't ignore one word, but several words at a time. The interest of grouping words has been presented in the previous report, enabling a better efficiency, a reduction in terms of number of models, and thus a considerable save of memory space.

Objective: We want to demonstrate that this technique allows us to decrease the importance in contribution of selected words. Some tests were in fact performed, but only on a few examples

[5] or whatever we want, but in practice going beyond words in the tokens choice result in vocabulary explosion

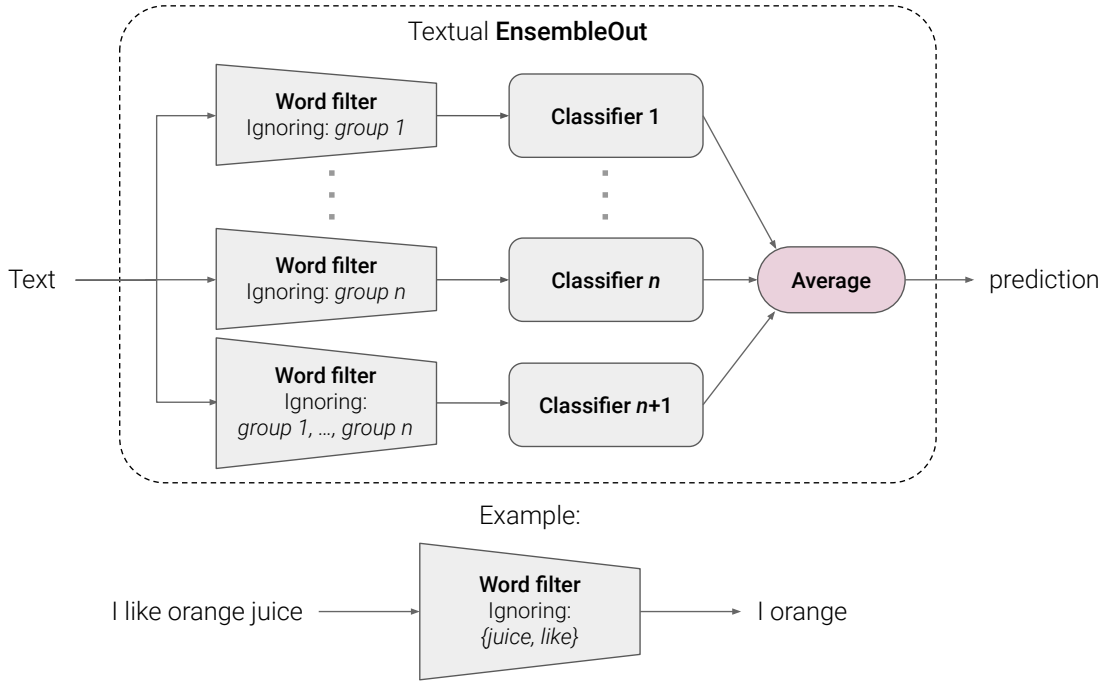


Figure 4.4: Presentation of the adaptation of EnsembleOut for textual data

with a dataset from *scikit-learn*. This part, which I will try to keep short[6], is about the work I did to assess efficiency of this textual implementation. This work has also been accepted at the DSAA 2021 conference [1].

4.2.2 Proposed solution

Local explainers for global explanation

Tabular versions have been tested on the LIME and SHAP explainers. They are originally tabular explainers, but fortunately, they both implement a textual variant! To stay consistent in the paper, I decided to keep these ones.

As stipulated in the state of the art, these explainers are local, i.e. they compute the contributions only for a single instance. Yet, as described in the FixOut's pipeline, we need a global explanation of the model. Unfortunately, as of August 2021, model-agnostic global explainers don't exist. Although they don't exist yet, we can approximate the global behaviour of a model by explaining many instances and summarizing them.

Random sampling

So, an easy way to get a *global* explanation is to explain each instance in the test dataset (used to assess the model), and take the mean contribution for each input. It is this method that is used for the tabular version of FixOut. To put it in a very simple formula, let n be the number of instances, x_i the i^{th} instance, E a local explainer that takes a model f and an instance in parameter, and Φ the global explanation:

[6]It looks like a large intro, but don't worry, it will be useful for the second part about FixOut on neural networks.

$$\Phi = \frac{1}{n} \sum_{i=1}^{n} E(f, x_i) \tag{4.1}$$

Note that if f takes m inputs, then x_i contains m features and $E(f, x_i) \in \mathbb{R}^m$.

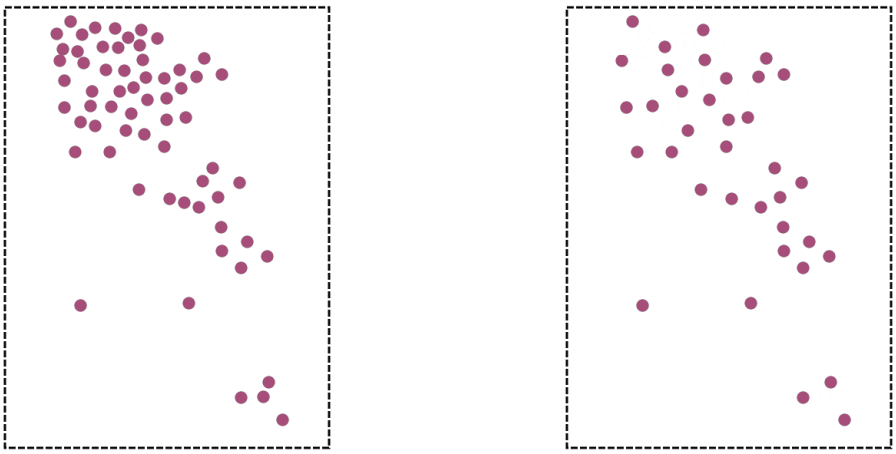
Time complexity and subsampling

One important inconvenient is time complexity. For example, we said that LIME generates a neighborhood of 5000 individuals, that are then all passed through the model, in order to explain a single dataset instance. Although the neighborhood population can be changed, let's say our test dataset has 1000 instances. The model should then make predictions for 5×10^6 instances. This should be multiplied to the model's algorithmic complexity, which often relies on data complexity[7].

This is why we will not be using all the dataset, but only a random subset, which should remain representative.

Submodular pick

LIME also proposes a greedy algorithm to select a subset of explained instances, so that this subset is representative of the instance explanation space, so then representative of the global behavior of the model. Figure 4.5 illustrates the idea behind this algorithm, called *submodular pick* [21].



(a) Original explanations (b) Explanations selected by submodular pick

Figure 4.5: Toy example of a latent space representing instances explanations, and the selection performed by the submodular pick algorithm

Even if fewer instances are selected, submodular pick selects a subset of *explanations*, which means that we must explain all the instances of our initial set anyway. Nevertheless, we can use it as an additional step to compare the global model's behavior on the dataset, and its global behavior regardless of the balance of instances in the dataset.

We can then take the average explanation for each feature like in random sampling.

[7] this *complexity* term is employed in a spatial sense.

Models assessed

Some models were already assessed for tabular data. They are relatively quick to train and to use, which is convenient considering the complexity issue raised above. They are simple classical ML models:

- Logistic Regression
- Random Forest
- Adaboost
- Bagging ensemble

A bagging ensemble, like Adaboost, is also an ensemble of sub-models, but sub-models are just trained on random samples of the dataset[8]. I chose to use the same models for consistency purposes in the DSAA article[9], but with an adaptation for textual classification.

Like in the Davidson's model [9], these models are preceded by a **TF-IDF** transformation, in order to convert text into a vector of the same size of the vocabulary, where each word coefficient depends on its frequency in the document[10] and the proportion of documents containing that word. Moreover, we pre-process the text by **stemming** each word, in order to reduce the vocabulary size and to gather similar words later for explanation (e.g. *"vehicle"* and *"vehicles"* are both transformed into *"vehicl"*).

It is important to underline that this configuration has already been tested during the first semester, but here we define the exact context of the experiment and conduct the consistent operations for these predefined models, and I also take submodular pick into account in order to make comparisons with simple random samplings that I did before.

Random variations and experiment repetition

Even though we didn't entirely described the detailed inner workings of LIME and SHAP, both have a random component in their process. As a result, if we explain one fixed instance with one fixed model twice, we don't get the same explanations, due to these random variations.

This is why we can't rely on one experiment, and should repeat the experiments many times, as underlined by reviewers from previous papers on tabular data, in order to get significant results. We then repeat this task **50 times**:

- Train a classifier on a dataset and make an ensemble with EnsembleOut in order to decrease predefined words contributions;
- Globally explain the original model and the ensemble in order to get the global contribution for each word in the vocabulary.

We then simply average the global contribution in each experiment.

[8] https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.BaggingClassifier.html

[9] Our DSAA article, in fact, also treats tabular cases (but it is not the part we are working on).

[10] *document* is a synonym for text instance, in NLP

4.2.3 Implementation

All experiments were implemented in Python.

Reorganisation of the code

I first reorganised the code inside FixOut to make it more practical to use. Since the ensemble made by EnsembleOut is an ML classifier, I made a class inheriting from the `ClassifierMixin` *abstract*[11] class from *scikit-learn*[12], which is the parent of all scikit-learn models. I then implemented the `fit` and `predict_proba` methods, which are defined by all scikit-learn models to train, and predict from a set of instances. The constructor (`__init__`) takes the original model and a list of groups of words that will be used to train submodels of the ensemble.

I also made a `TextExplainer` *abstract* class, which can be implemented with any explainer, and which is used to make the global explanations. The class diagram in figure 4.6 presents the main methods of these classes. Many options are not listed, such as the chosen regex pattern to remove words, or the possibility of giving a common vectorizer (such as TF-IDF) to avoid duplicating it in the memory.

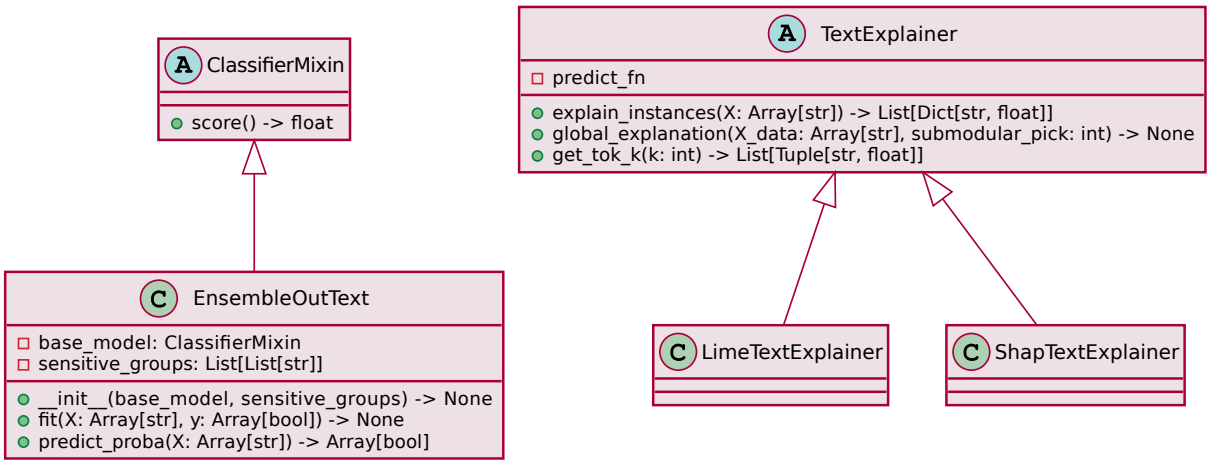


Figure 4.6: Simplified class diagram of the implementation of EnsembleOut and global explanation for textual data in FixOut

Issues with SHAP

First of all, SHAP doesn't select words in the text with a simple regex like LIME does, in order to explain the words importance. It only proposes a restricted set of tokenizers from the *Hugging Face*[13] repository, or deprecated functions hard to use with modern code. I chose the first option, more precisely the Electra's tokenizer, from a lightweight Google model.

The second problem is that SHAP doesn't implement submodular pick. The solution was simple, though: since we know the algorithm, I just had to implement it in a new function.

[11] Actually, Python doesn't provide abstract class implementations, so it is a *basic* class. In practice, however, this class alone is unusable and is made to be inherited.

[12] https://scikit-learn.org/stable/modules/generated/sklearn.base.ClassifierMixin.html

[13] platform for AI models: https://huggingface.co/

Dataset and other parameters

Dataset: From the Davidson's *hatespeech* dataset [10], we get 4163 harmful tweets and 4163 non offensive ones, making a dataset of 8316 instances. 80% is selected for the training stage, the remaining 20% constitute the test set. After having preprocessed the dataset with stemming, we find a vocabulary size of a little bit more than 2000 words.

Models configuration: All the classifiers (the four ones cited above) are instantiated with *scikit-learn* with their default parameters. We don't aim to have good prediction scores, but we will check that these scores don't decrease a lot when applying EnsembleOut to reduce the sensitive words contributions.

Choice of sensitive words: Although we can't identify the sensitive words responsible for racial biases, we can manually select words and consider them as *sensitive*, in order to show that we are able to decrease their contribution to the hate speech classification. I decided to make 3 groups of words, in order to build an ensemble of 4 submodels:

- **Group 1:** *nigga, niggah, nigguh*
- **Group 2:** *nigger, nig, nicca*
- **Group 3:** *black, white*

Explainers configuration: For global explanations, 700 instances are randomly selected in the test set, from which 300 are sub-selected by submodular pick. 700 instances is roughly 42% of the test set size, which is hopefully enough to keep a good representation of the instances.

Overview

A summary of the experiment is presented in figure 4.7.

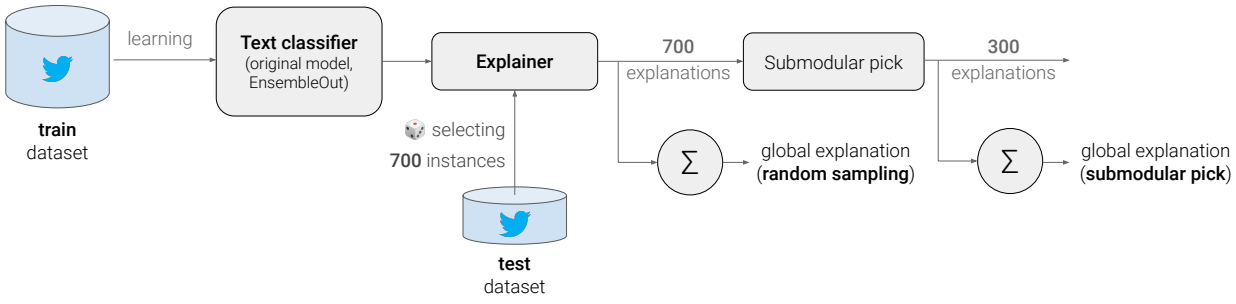


Figure 4.7: Diagram of one experiment performed on one model with one explainer. This is done 50 times for 4 models and 2 explainers.

4.2.4 Results

Global explanations

Just as a reminder, after having trained each one of the four models, we apply EnsembleOut. Next, we explain them globally with LIME and SHAP. In table A.1 in **appendix**, I presented for each

word:

- their global **contribution**, which is given by the explainer, rounded to 10^{-3};
- their **rank** (starting from 1), after sorting the vocabulary by contribution in absolute value, and in descending order.

The results are grouped by model, that are presented with their short form: **LR** for Logistic Regression, **RF** for Random Forest, **ADA** for Adaboost, and **BAG** for Bagging.

We observe that LIME explanations systematically show a decrease in the contribution and the importance rank of selected words. We even notice that the contribution is often nearly divided by 2, although the contribution of the whole vocabulary tends to decrease a bit in the ensemble explanation.

By contrast, SHAP explanations show an increase in the ranking of the word *nigga*, although its contribution decreases. The contribution of all other selected words is decreasing in general, even if this tendency seems to be more instable than in the case of LIME.

Table A.2 also presents the results obtained with submodular pick, and shows results similar to random sampling with better results with logistic regression. We can also see that the contribution with SHAP does not decrease for the word *nigga* as well. These latter results seem to indicate that FixOut with LIME explanations is more stable and can consistently improve fairness, i.e. always decrease the contribution of sensitive features[14].

Model performance

Since the objective of FixOut is to reduce sensitive features contributions **without** compromising the model's performance, it is natural to look at the basic scores of our models, which are presented in table 4.1. In this experiment, we can see that EnsembleOut almost always slightly worsens the performance. The score, in average, only decreases by less than 1%, which is eventually not a significant drop.

	Accuracy		**Precision**		**Recall**	
	Original	Ensemble	Original	Ensemble	Original	Ensemble
LR	0.928	0.921	0.93	0.922	0.925	0.919
RF	0.953	0.951	0.961	0.96	0.945	0.942
ADA	0.952	0.949	0.967	0.97	0.937	0.926
BAG	0.947	0.941	0.954	0.951	0.939	0.929
Mean diff.	-0.005		-0.002		-0.008	

Table 4.1: Model scores, comparison between the original model and the ensemble made by EnsembleOut

Discussion

The two explainers LIME and SHAP show a significant reduction in the contributions of sensitive words we selected. FixOut also manages to maintain the models performance, such as accuracy,

[14] Again, I wrote these results in the DSAA paper [1]

precision and recall scores. These results are then very encouraging for textual models which rely too much on a set of vocabulary, which also could induce fairness issues.

Now the major problem is still the algorithmic complexity of this solution. Today, deep learning models are widely used by the NLP community, and as a consequence, the broad majority of textual models are neural networks, which are more efficient, but especially heavy (in memory space and algorithmic time), compared to the basic models we used in this experiment.

Here, we made an ensemble of 4 models. If a neural network takes three days to train, then... an ensemble of 4 models takes nearly two weeks to train, but also would be four times longer when making predictions, and would take four times more space, etc. It would also be very long to assess globally such a model with LIME and SHAP! This is why this general adaptation of FixOut for any textual model must be more specifically adapted to neural networks classifiers.

4.3 Adaptation to neural networks

4.3.1 Avoiding re-training of many sub-models

Problem analysis: In order to lighten the EnsembleOut process for NLP models, one thing that could be done is not to re-train every sub-model, and merely build an ensemble of one single model, but that makes predictions on different instances. In other terms, what we can do is keeping the original model that has been trained on the whole dataset without word removal, and use it to make many predictions with different word removal. This would be our new EnsembleOut, composed of sub-models, that are in fact the same model that processes the instance in different forms. Since it is the same model, there is no need of new train stages, and it would take much less memory.

Note: this is possible for textual classifiers, but not for tabular models. We can't, indeed, remove one input feature when the model is designed to take exactly a fixed number of features. For textual models, however, we can remove one or many words, which results in the absence of these words in the sentence; but the sentence size for a textual classifier is not fixed.

4.3.2 Using gradient based explainers

Problem analysis

Even though LIME and SHAP are model-agnostic, they are relatively inefficient on neural networks. In fact, they manage to explain such (often heavy) models, but it takes a lot of time. We'll have the occasion to make a time comparison between LIME and another gradient based explainer in the molecule classification part.

One recent and interesting gradient based explainer that we have mentioned in the state of the art is **PathExplain**[15], which embeds the integrated gradients algorithm. However, as mentioned in the state of the art again, gradient based explainers can explain differentiable models, and only differentiable models. This is due to the fact that the integrated gradients method use the

[15]Other gradient based explainers exist, like DeepLift of DeepSHAP, but are not maintained, and now outdated since Tensorflow 2.0

derivative of the output w.r.t. the input. The problem is: what if we have text as input? How can we compute the derivative of the output w.r.t. a text?

In fact, as you might guess, we can't derive variables with respect to text, since text is not a continuous variable. But yet, we can compute the gradient of the output w.r.t. the tokens embeddings.

Definition of a model

First, let's build a simple textual classifier, with a decoder containing an embedding layer followed by an LSTM layer, and a decoder with an attention layer followed by a fully connected layer (perceptron layer). The architecture of the model is shown in figure 4.8. Pretty much the same architecture can be found in basic NLP models tutorials[16].

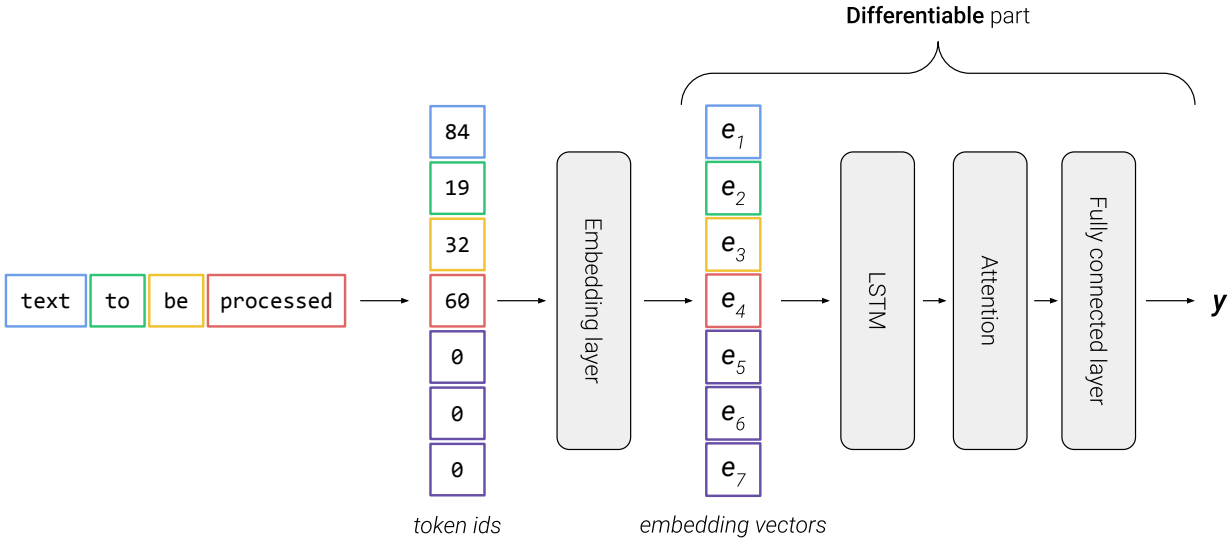


Figure 4.8: Neural network architecture used for text classification

Tokenization: The words in the text input are first sliced in tokens by a *tokenizer*. Each word has a unique token identifier, that is defined in the tokenizer's dictionary. In practice, we add a start token (often written `<start>`) before the sentence, and an end token (often written `<end>`) after the sentence, which is not represented in figure 4.8. We then complete the tokens sequence with *null* tokens, since in this model, the sequence size must be constant[17].

Embedding layer: The embedding layer is somewhat like a big dictionary, attributing an embedding vector given a token. All embedding vectors corresponding to all existing tokens are referenced, which allows a fast conversion (in a constant algorithmic time) of tokens into embeddings. During the training stage, the embedding coefficients are modified. Note that in figure 4.8, the three last embedding vectors e_5, e_6, e_7 all correspond to the null token embedding, and are thus equal.

[16] https://www.tensorflow.org/text/tutorials/nmt_with_attention, https://github.com/Renovamen/Text-Classification/tree/master/models/AttBiLSTM

[17] This is because of the attention layer, but we will not get into details.

Explanations with PathExplain

PathExplain can only explain the differentiable part shown in figure 4.8. That means that we have to isolate this part of the model, and properly preprocess the text to get these embedding vectors, which are then our inputs for this differentiable model. The integrated gradients algorithm will assign an *attribution*[18] value to each embedding vector coefficient, and PathExplain then computes a weighted sum of these values for each vector, in order to get the contribution for each word, as shown in figure 4.9.

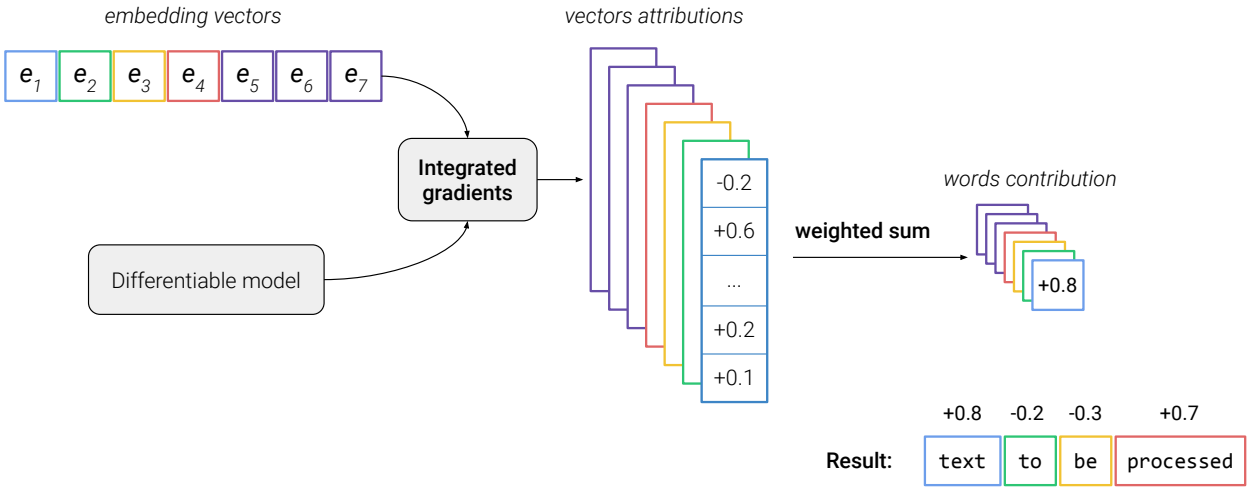


Figure 4.9: Process performed by PathExplain in order to get the words contributions, which is the result presented.

Once we get the word contributions, it's all won! We can take the mean contribution for any word in our vocaulary by explaining many instances, and thus have a global explanation, like we did before.

Differentiability with EnsembleOut

This is where things become tricky. We are able to explain a model by taking its embeddings, which are computed inside the model, and then we take its differentiable part. This differs from the last configuration, where models were considered as black boxes that take text as input and output a probability prediction. The ensemble was also such a model.

Here we can't consider our model and the corresponding ensemble as black boxes anymore, the problem being that the whole pipeline in the model is not differentiable from the input to the output. This is also the case for our ensemble: we can't explain the whole model, since only a certain part is differentiable, as shown in magenta in figure 4.10. Considering what we have previously discussed on sub-models and memory, the three embedding layers and neural networks in figure are actually the same instance.

To explain the whole ensemble, we could explain each one of the sub-models, and then consider the average contribution of the words for each sub-model. We could say it is my proposed solution and run the experiments! But here is the problem: since the contributions of removed

[18] more precise synonym for contribution. We will discuss later in this report the slight subtlety between the definition of *contribution* as in LIME or SHAP, and the *attribution* computed by the integrated gradients method.

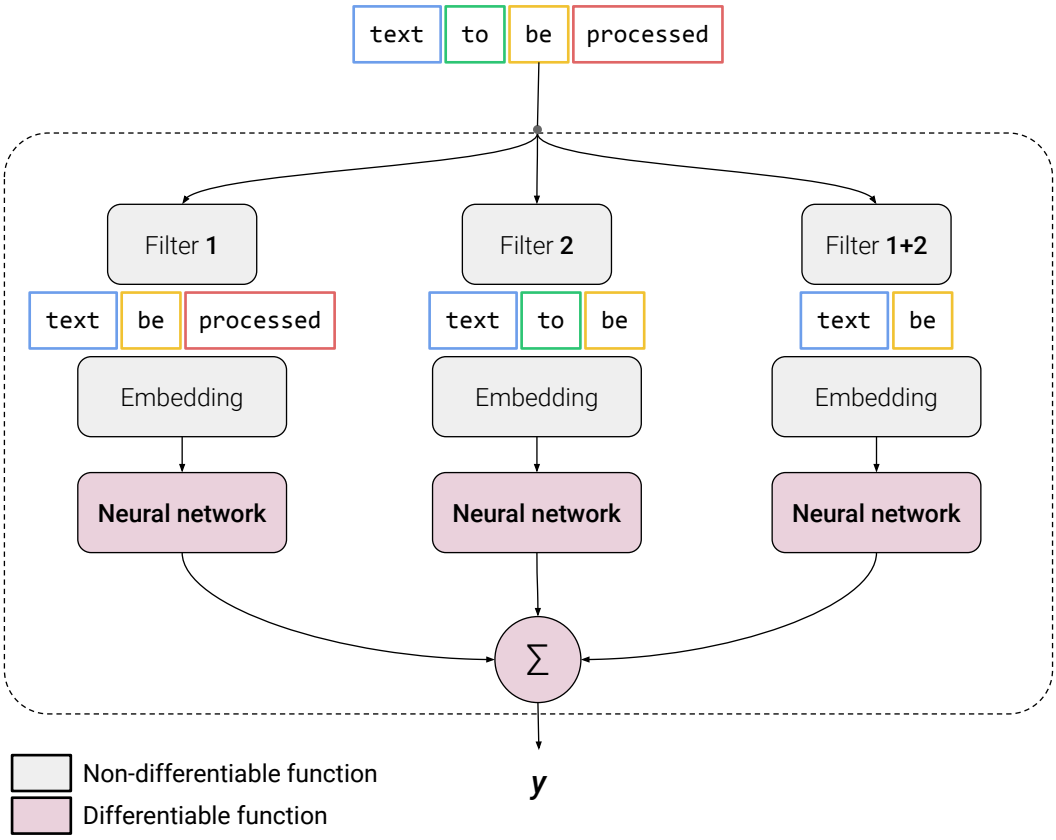


Figure 4.10: Illustration of EnsembleOut applied to a textual neural network, by considering the words *to* and *processed* as sensitive words. Only the magenta functions are differentiable.

words is zero, when calculating the average contribution at the end, their contribution will obviously decrease. This is why we need an explanation for the whole ensemble. But again, it is not differentiable. How to do it?

Proposed solution

One solution is to consider the ensemble's input to be embeddings. This way, we get a full differentiable ensemble, as shown in figure 4.11. You may have noticed that this time, filters are differentiable. This is because they simply filter the embeddings without making any operation on it: it is just a rearrangement. Filters then remove embeddings corresponding to the words we don't want, and add a *null* embedding vector (for which the gradient will not be computed) until we reach the required sequence size.

4.3.3 Implementation

We use Keras, on top of Tensorflow in Python to build the model and implement this solution.

[19]I forgot to apologize, I used so many color codes in figures of this thesis that it can be crippling for grayscale printed versions and color blind people.

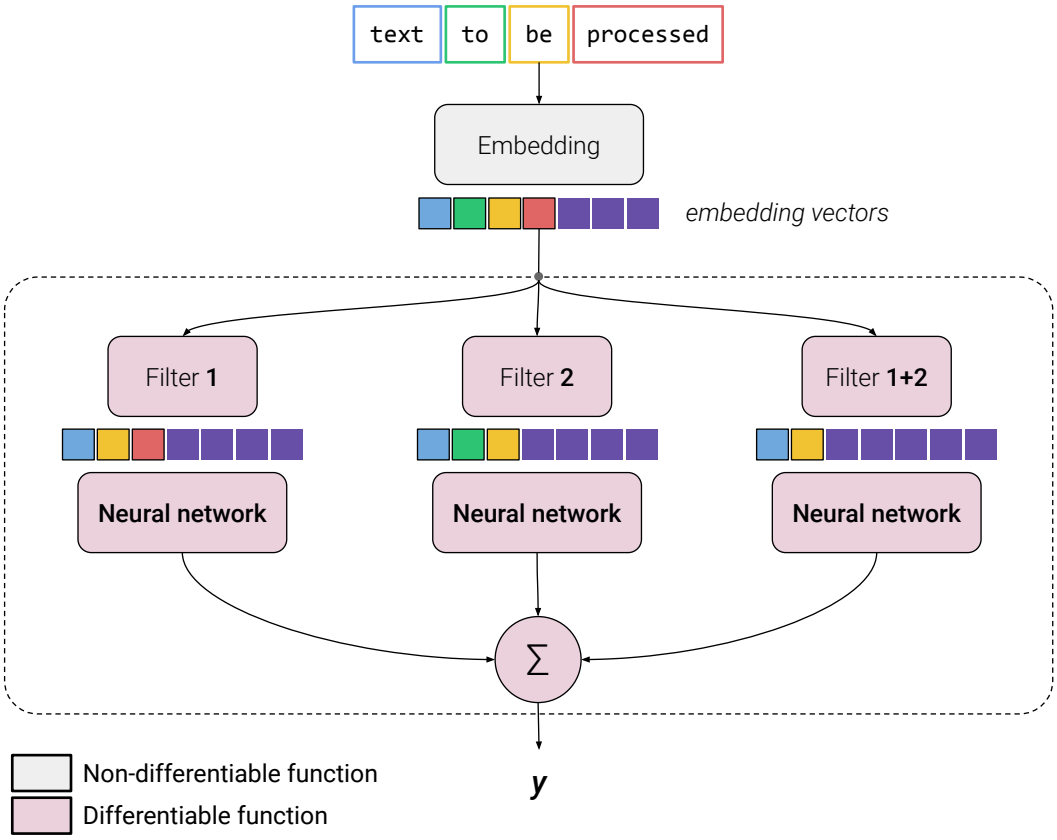


Figure 4.11: Illustration of the proposed solution, EnsembleOut with filters directly applied on embedding vectors[19]

The ensemble model's definition depends on the instance

The particularity of this solution is that we must *modify* the ensemble definition for every single instance. When defining a model with Tensorflow, Tensorflow *pre-compiles* the model in memory by building its graph of operations. Once defined, the graph can't be changed, so all the instances that will get into it will undergo the same operations. Let us illustrate the point with an example.

Example

Figure 4.11 presents an ensemble with three submodels[20]: one removing the embedding corresponding to the word "*to*", the second removing "*processed*", and another removing both "*to*" and "*processed*". In this example, "*to*" is the second token, so we tell the first submodel to remove the second embedding from the sequence. The same thing applies for the two other submodels, which respectively have to remove the fourth embedding, and the second along with the fourth embedding.

It is therefore stated in the compiled model's graph that the first submodel removes the second embedding of the sequence, the second submodel removes the fourth embedding, etc. You are getting the point: another sequence will have other words in another order, so are the tokens, and we don't systemically want to remove the second and the fourth element of the sequence.

[20] despite the fact that, again, the three neural networks are the same instance in memory

Tensorflow has to recompile the model for every instance

The only solution I found was to recompile the model's graph every time we want to process a new instance. Fortunately the layers are already trained, so it is merely a redefinition of the whole pipeline, not a re-training, just to clarify. This operation takes nonetheless a bit of time: approximately 0.7 seconds when I did my experiments on *Google Colab*, which represents a considerable increase on many instances.

Even though we take the original layers and the ensemble doesn't take much more memory than the original model, the gradient computation takes, in contrast, more memory. The gradient is, in fact, computed for every operation in the model's graph; however, the graph is larger than the original since we define submodels, as shown in figure 4.11. In our case, this memory increase is insignificant, since we work on a very simple model; on the other hand, it could become more significant on large models.

4.3.4 Results

After **50 times** training and globally explaining the original model and the ensemble produced by EnsembleOut, it turns out that this solution keeps the desired behavior of FixOut. Figure 4.2, again with the rank and the contribution of each sensitive word, shows that the sensitive words contributions systematically decrease, and are divided in average by 1.5 (rounded to the decimal).

	Original model		EnsembleOut	
Word	**Rank**	**Contrib.**	**Rank**	**Contrib.**
niggah	18	0.109	29	0.071
nigger	3	0.408	9	0.272
nigguh	46	0.045	56	0.035
nig	37	0.061	50	0.041
nicca	6	0.303	13	0.202
nigga	5	0.328	12	0.219
white	19	0.100	27	0.079
black	107	-0.018	132	-0.012

Table 4.2: Fairness assessment on textual data, global explanations with PathExplain.

The most interesting thing is, that contrary to LIME and SHAP where contributions of all features seemed to decrease in general, here the contribution computed by PathExplain nearly doesn't change for all the other features, as shown on table 4.3. Notice that we get almost only English insults, due to the fact that the classification task being performed consists in recognizing hate speech. Find this kind of words at the top of the contributions therefore makes sense. We also find the <end> token, which is strange at first glance since it is present in all sequences, so it shouldn't be important to make a prediction. One explanation is that this token may be more or less important according to its position in the sequence: its contribution thus might reflect the importance of message length.

A little drawback of this implementation is that it hard to use the ensemble as a *normal* model, in the sense that we have to alter the ensemble every time we process a different instance, which makes the processing of a whole batch of instances much longer and more complicated.

We can nevertheless easily compute the model scores, here with *scikit-learn* in table 4.4:

	Original model		EnsembleOut	
Rank	**Word**	**Contrib.**	**Word**	**Contrib.**
1	fag	0.667	fag	0.667
2	retard	0.484	retard	0.483
3	**nigger**	0.408	faggot	0.378
4	faggot	0.378	bitch	0.300
5	**nigga**	0.328	<end>	-0.296
6	**nicca**	0.303	fuck	0.285
7	<end>	-0.294	pussi	0.285
8	bitch	0.297	y	0.281
9	pussi	0.284	**nigger**	0.272
10	y	0.280	hoe	0.230

Table 4.3: To 10 contributions for the original model and EnsembleOut, according to PathExplain. Words considered as *sensitive* are marked in bold.

	Original	**Ensemble**
Accuracy	0.947	0.939
Precision	0.951	0.945
Recall	0.946	0.941

Table 4.4: Scores of the original and the ensemble models

Even if the ensemble has not been trained, we can observe that the scores doesn't decease significantly.

4.4 Discussion

Before we close the part on FixOut, we can discuss the results we got. We first managed to adapt FixOut for textual classifiers, and showed that this adaptation has the same properties than the original FixOut version for tabular classifiers: we success to lower significantly the contributions of sensitive features (here sensitive words) without compromising the model performances, and this is confirmed on several models with two different explainers.

We also managed to adapt FixOut to neural networks by choosing a gradient based explainer and by keeping only one copy of the model, avoiding to waste time by training a heavier ensemble. This technique can then directly be applied to pre-trained models, in order to decorrelate the reliance on sensitive features.

However, a wide range of configurations remain possible, and we didn't perform the same tests on other datasets, even though the results were promising in two other datasets in the previous mid-term report. We could also assess this method in different ways, for example by introducing fairness metrics on sensitive words, or by using causal-based explainers. Karima Makhlouf made a survey on this topic, for tabular models [18]. With Guilherme and Miguel, we are moreover in contact with Karima, Sami Zhioua from the Higher Colleges of Technology in Dubai, and Catuscia Palamidessi from the LIX (Laboratoire d'Informatique de l'École Polytechnique), in order to tackle unfairness using different notions of fairness measurement.

Another point to explore is the choice of the sensitive features. For these experiments, we chose

the sensitive words manually and subjectively, but there is at the moment no method to select these sensitive words in a given context. Without such a method, the fairness improvement pipeline of FixOut for textual data would remain incomplete. It can nonetheless be very promising on reducing the contribution of words we know they have a negative impact on a classifier's prediction.

4.5 Interactive web demo

In addition, I also developed a little web interface with the Flask Python framework, in order to show the effect of FixOut on tabular data. It is not about the textual adaptation, but it's a simple demo allowing users to visualize the results of experiments.

With Miguel, at the beginning, we wanted to create an interactive demo, where users could use FixOut with the parameters they want. In practice, this was complicated, since the demo was expected to be hosted by the LORIA's servers, and LIME and SHAP explanations are computationally expensive.

The solution I found was to *recycle* the experiments results. In fact, I participated in the experiments for tabular data. For each configuration, like we did for textual classifiers, the experiment was repeated 50 times. For every experiment, a log file with the detailed results is written in a log folder. Instead of computing explanations, when the user wants to experiment FixOut with a specific configuration, a random experiment is picked from the log folder with the corresponding configuration, and the logs are parsed in order to be displayed on the interface. The datasets are also described in the demo when clicking on the little "?", as shown in figure 4.12.

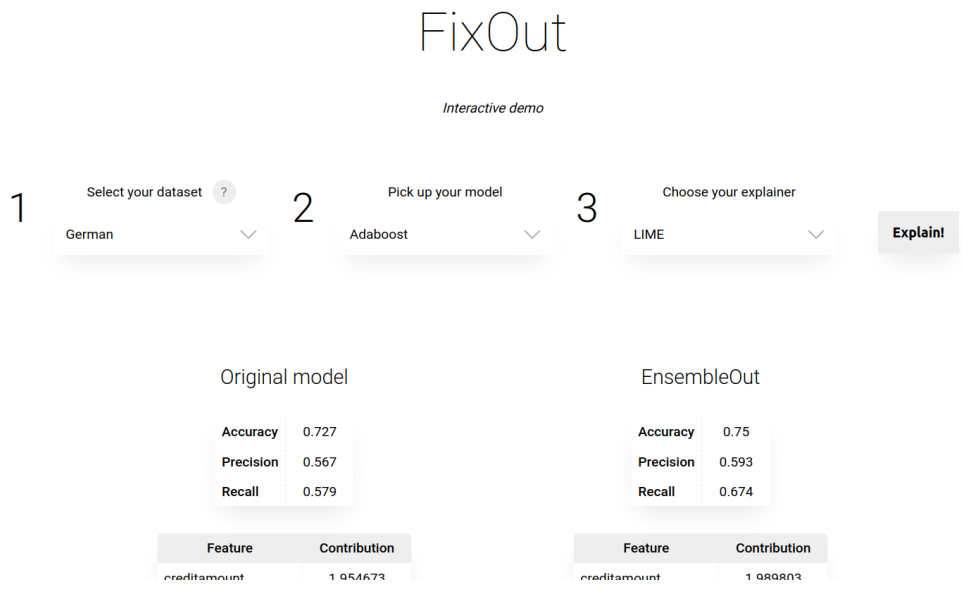


Figure 4.12: Presentation of the FixOut interactive demo.

When clicking on a sensitive feature, a radar plot is drawn with the library Plotly.js[21], comparing some fairness metrics between the ensemble and the original model. The calculation of these metrics (lower is better) is detailed at the bottom of the page. The whole screen can be found in the appendix in figure B.1.

[21] https://plotly.com/javascript/radar-chart/

5 Explanation of antibiotic molecules

We have seen that explainers are very helpful in order to assess models fairness. But what if we explain models in another context than fairness assessment?

Amedeo was supervising a project that consists in using machine learning to classify molecules in order to predict whether they have got antibiotics properties, or not. This is the subject of Clément's master internship, jointly supervised by Bernard Maigret, emeritus CNRS researcher in the *CAPSID* team. Moreover, Amedeo was also supervising me in the FixOut's project. This is when he got an idea and asked me "*hey Fabien, what do you think about applying your explainers to antibiotic classifiers?*", this idea being to use them in order to show important molecule contributions to make them antibiotic. Of course, I jumped into the adventure.

5.1 Problem analysis

Antibiotic classification is really great: since it takes time for laboratories to assess efficiency of many candidate molecules, molecule classifiers can first make a guess of their antibiotic properties, in order to *filter* candidates and have a first idea of their potential efficiency, very quickly.

There is however a little problem, similar to the fairness issue presented above. Machine learning models are black box classifiers, and we don't really know *why* a molecule has been classified as antibiotic. We thus want to know what is important in a molecule to be classified as antibiotic. How are molecules processed? **What has been learned?**

5.1.1 Models to explain

The classification task was part of the Clément's subject. The two main models were the following:

DeepChem

The first model to explain is built with the DeepChem toolkit cited above in the state of the art, but is not a deep learning model in this case. In fact, DeepChem is simply used to convert a molecule in the SMILES format into "*fingerprints*". A fingerprint can be interpreted as a *hash* of the molecule, represented in a fixed size. Each element of the fingerprint is also called a descriptor, indicating the presence of a property or not; this is a binary information. A good size for the model found by Clément was 2048. A fingerprint is therefore an array of 2048 bits describing the

molecule. Such arrays, sometimes combined with dimension reduction methods, are also useful to measure the similarity between two molecules.

This 2048 large array is then given to a random forest, which does the classification, as presented in figure 5.1. Like any classifier, the prediction is a probability, between 0 and 1, of classification.

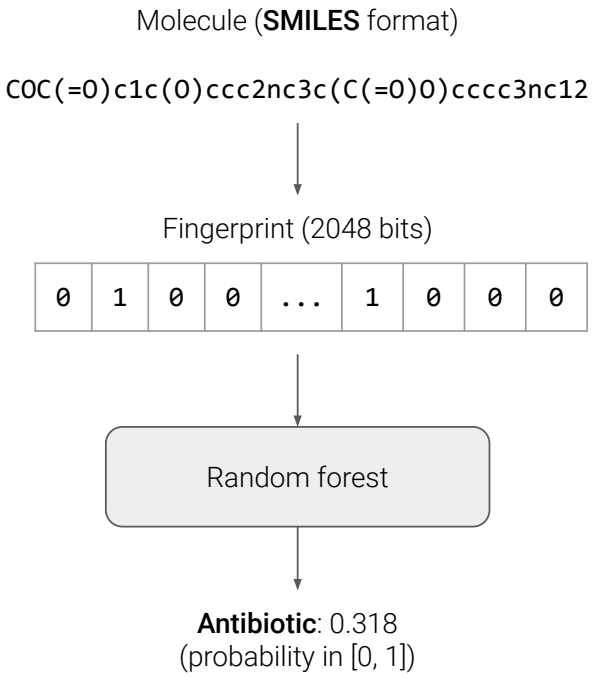


Figure 5.1: Model pipeline used with DeepChem presented with a toy example.

Chemprop

Chemprop is a model, also presented above in the state of the art (figure 3.7). One big inconvenient for Chemprop is that the input size is not constant, since its input is a graph, whose size depends on the molecule being processed. Moreover, explainers we used above can treat tabular models, but a graph has not the same format than a fixed size array.

Clément has trained it on many datasets, to classify molecules as antibiotic or non antibiotic, which is a two-class problem. He then trained Chemprop on a new dataset, to predict more precisely from what kind, from what family of antibiotics the molecule is. This second case is therefore a multiclass problem. It is for this reason that I adapted the methods I developed to explain this model, later. In a multiclass problem, only one class will be explained, and in such a case we will choose to explain the predicted class (i.e. the one with the highest probability).

5.2 Proposed solution

5.2.1 Explanation of DeepChem

When we talk about an explanation, it is implicitly understood the explanation of an *input* with respect to a *model*. According to the figure 5.1, we could interpret two things: the SMILES string, and the fingerprint.

Why it is a bad idea to explain SMILES directly: Since the input SMILES is a string, we could apply a textual explainer, like we did with LIME and SHAP. However the textual explanation is done by altering the input string, removing or adding tokens. The SMILES format nevertheless has a particular syntax, and removing one character would certainly cause a syntax error, because the SMILES couldn't be converted into a molecule anymore. In addition to that, it is difficult for humans to interpret a SMILES explanation directly, since it is just an encoding of the corresponding molecule we want to explain (in the case of local explanations).

Why it *might be* a good idea to explain fingerprints: What can be done however, is an explanation of the fingerprint. This would allow us to check which descriptors are important for the antibiotic class.

Global / local explanation: For local explanations, we can pre-compute the molecule fingerprints, and use LIME and SHAP to explain them. We can then make a global explanation by summing it all. Actually, there is also another interesting method to perform a global explanation for random forests and decisions trees, called the Gini importance, computed in scikit-learn[1].

5.2.2 Explanation of Chemprop

Explaining this model is very promising in terms of human interpretability. In fact, we can explain the molecule in its graph format (the mol graph), in order to compute the contribution for each atom and bond, and visualize it. This would be a very convenient format for experts such as chemists and drug researchers.

However, as we said, we need to adapt our tabular explainers to enable graph explanation, the major inconvenient being that all the features of a graph can't be rearranged inside a fixed size array since the number of nodes and bonds (thus the total number of features) depends on the molecule instance. The solution I found is similar to what I proposed in subsection 4.3.3 where the model was re-adapted each time we wanted to explain one particular instance.

This solution is presented in figure 5.2. For one specific instance, we create a model wrapper that only processes this specific instance. This is why it manages to reshape the input values into the exact mol graph of the current molecule. This whole new model is then a tabular model, where the input array, containing the molecule features, can be explained. We therefore get the contribution for each feature, and we are able to parse it the same way we did in the model wrapper to form the mol graph. We thus get an *explained mol graph*.

Again, the model wrapper and the way the values are redistributed into a graph depends on the molecule being processed.

5.3 Implementation

Python is still the programming language being used. There is nothing difficult in explaining the fingerprints of DeepChem since the random forest we want to explain is a scikit-learn model, which allows us to get the Gini importances easily, but also to apply LIME and SHAP for tabular

[1] https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestClassifier.html#sklearn.ensemble.RandomForestClassifier.feature_importances_

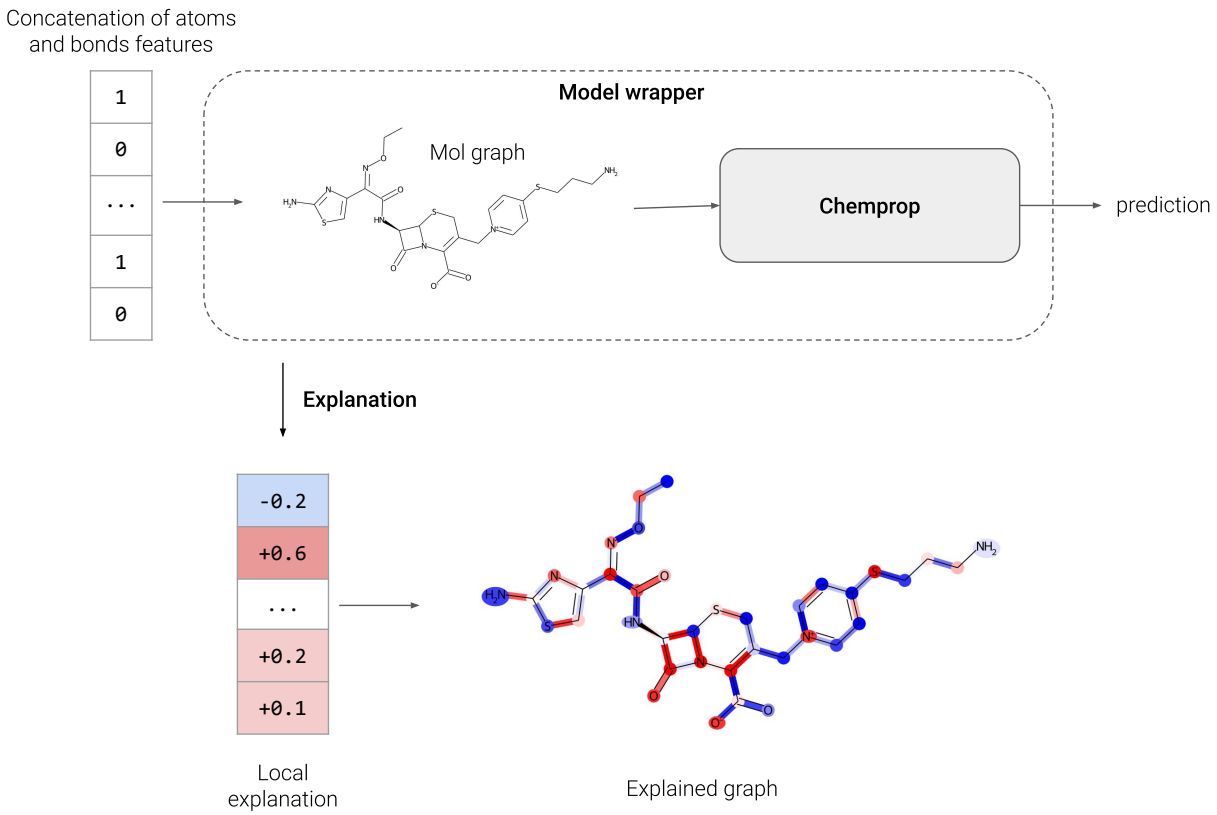


Figure 5.2: The model wrapper (dashed box) parses the values in an input array in order to reconstruct the mol graph and provide it to Chemprop, which makes the prediction on the molecule.

models. For Chemprop however, things get a little bit more complicated.

5.3.1 LIME and SHAP for Chemprop

Isolation of a Chemprop model

The first thing I had to do was to be able to use Chemprop in a Python script. Even though Chemprop is written in Python, it is first intended to be used in command line, not to be used in a Python script. We can nonetheless find a documentation[2] describing the role of each class, but there is no real starting point, *quick start* or tutorial. I then managed to use it by understanding the pipeline from the command line to the model, in the provided scripts, so that we can load a pre-trained model, and get predictions from it.

Explanations with LIME

GraphMol <-> Array converter: I started to implement the solution with LIME, by creating a new class containing the model as an attribute[3]. I wrote a `GraphMol -> Array` converter that concatenates all the features of the graph mol. We first iterate through the atoms, then through the bonds. There is a slight subtlety: mol graphs are oriented, which means that we

[2] https://chemprop.readthedocs.io/en/latest/
[3] https://gitlab.inria.fr/fbernier1/chemprop-why/-/blob/0.2/cpwhy/chemprop_lime.py

have bond features representing the link between one atom and another, and other features for the other direction. In practice, the bonds also contain the features of the atom they come from. In order to avoid feature repetition in the final array, only the features corresponding to the bond part are taken into account.

The mol graph is then saved into an attribute `mol` in order to be able to make the conversion `Array -> GraphMol` later. For this stage, the mol graph is copied and the values of the given array are redistributed directly inside the graph in the same order.

Spoiler alert: doing this directly was actually inefficient. As shown on tables 3.2 and 3.3 in the state of the art, most of features are categorical and encoded with one-hot vectors. We can simplify it just by transforming it into an integer id, and telling LIME that this feature is categorical. We then gain space, but also time.

Solving memory issues: Even if we manage to gain space, it's sometimes too much for my 8Gb RAM[4] on my computer. I thought that it was not normal for this program to take more than 1Gb of RAM. After analysing the problem, it was because there is still more than 1000 features to explain (for a molecule with about 20 atoms for example), and LIME generates a neighborhood of 5000 instances all at once, that are passed through the model in one single batch. Making so many predictions at time takes a lot of RAM. Instead of making one explanation with a neighborhood of 5000, I took the average of 5 explanations with a neighborhood of 1000 instead. This doesn't take much more time, and reduces the space taken in memory.

Prediction and explanation methods: I then implemented a `predict_proba` method which takes a batch of array inputs (which must correspond to the current molecule), convert them into mol graphs, and pass it to the model in order to return the predictions. With this function, we manage to create another method, `explain_molecule`, which takes a SMILES string, convert it to a mol graph, then to an array, and call LIME in order to explain this array with our function `predict_proba`. The contributions are therefore returned by LIME in array with the same format, which is converted back into a mol graph, that we distinguish by giving him a class called `ExplainedMolGraph` (since it has not the same meaning than an input mol graph). A simple UML diagram of this implementation is presented in figure 5.3.

Explanations with SHAP

The principle is the same with SHAP, we just have to adjust the explanation part, but in theory there is not that much to change since SHAP is also model-agnostic. In theory.

Computing the Shapley values on so many features was also too much for my small memory, but I didn't manage to find parameters to change in order to solve this problem, like LIME. We could add a little bit of complexity to our implementation by creating another model wrapper, only considering some variables, and considering other features as fixed. However this could cause a change in the explanations, and since SHAP is already similar to LIME, this solution is not very cost-effective. This implementation has first been paused, to finally be replaced later by another explainer. It's always good to compare the results from different explainers.

[4]Random Access Memory

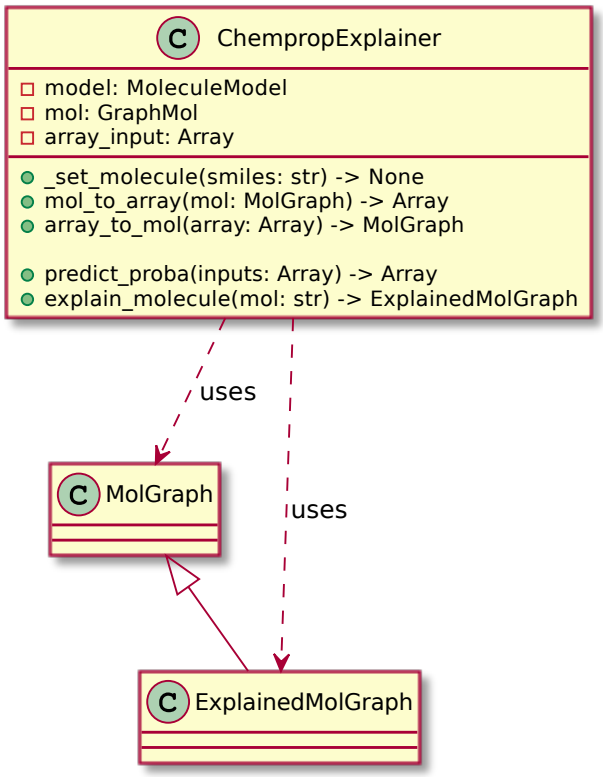


Figure 5.3: Simplified class diagram of the solution.

5.3.2 Molecule visualizations

RDKit is already used by Chemprop to convert SMILES strings into mol graphs, but it has many other tools useful for molecular operations. Since our implementation depends on Chemprop and Chemprop depends on RDKit, it seems natural to use RDKit to draw the molecule. Moreover its documentation presents an example[5], which makes things easier (though changing the font color wasn't that easy).

5.3.3 Mol graph organization

Before presenting the results, it is important to understand that we get many explanations for each atom or bond. If we look at the table 3.2, we can see that each atom has 8 features, so we get 8 contributions. The same thing applies with bonds, which have 4 features (table 3.3). It is also important to remember that each bond is actually duplicated, since the graph is oriented, as shown in figure 5.4.

[5] https://www.rdkit.org/docs/GettingStartedInPython.html#drawing-molecules

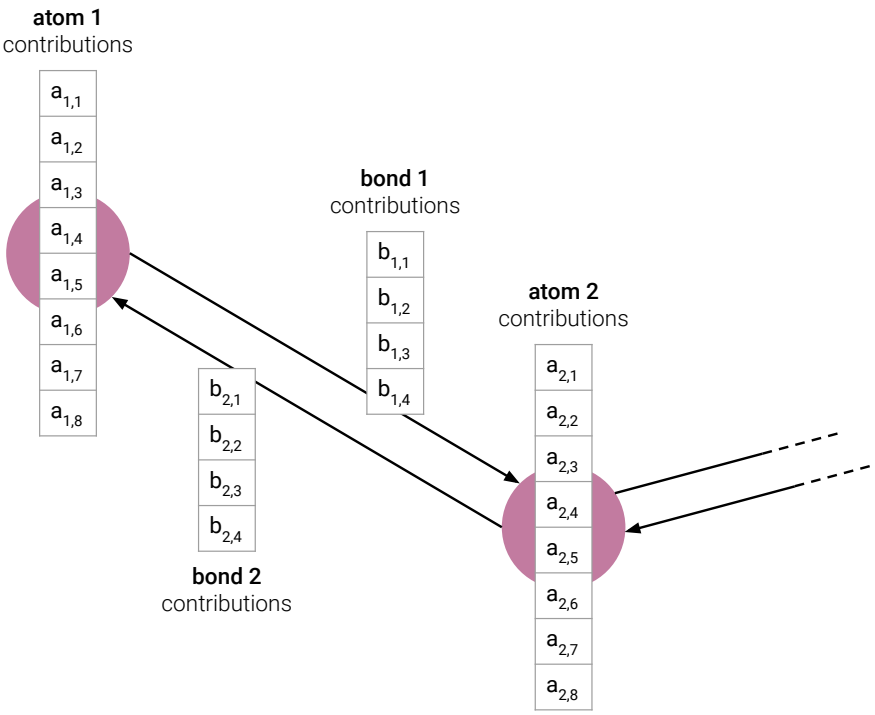


Figure 5.4: Representation of an explained mol graph and the distribution of its contributions.

5.4 Results

5.4.1 Interpretation of DeepChem results is not trivial

A huge problem came to me and Clément, which has worked on the fingerprint conversion. Even if we manage to explain the molecule's hash (fingerprint), it is very difficult to interpret what the n^{th} bit is for. Neither I nor him succeeded to find a way of bringing the descriptors' explanations up to the molecules properties. After many researches, I decided to focus on Chemprop instead.

5.4.2 Chemprop, by considering the average contribution

The first way to visualize the contribution for each atom and bond, is to consider the average contribution of their features. In figure 5.4, the contribution for atom 1 would thus be $\frac{1}{8}\sum_{j=1}^{8} a_{1,j}$. The contribution of the molecule's bond between the two atoms would be the average of all the features of bond 1 and bond 2 on the mol graph, i.e. $\frac{1}{8}\left(\sum_{j=1}^{4} b_{1,j} + \sum_{j=1}^{4} b_{2,j}\right)$. The $\frac{1}{8}$ factor is not important though, since we only want to see the contributions between some atoms and bonds on the molecule compared to others, and all is rescaled in order to apply a colormap, so the factor wouldn't be visible.

We will take the molecule with the following SMILES as an example, that comes from our dataset : `Cn1nnnc1SCC1=C(C(=O)O)N2C(=O)[C@@H](NC(=O)C(CO)c3csc(N)n3)C2SC1`
If we apply a colormap to represent negative contributions in blue and positive contributions in red, we get the figure 5.5.

Areas with high contributions are not clear.

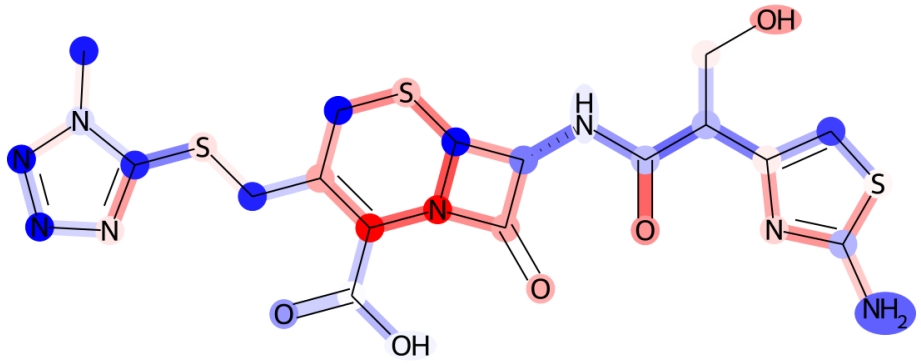


Figure 5.5: LIME explanation for the molecule, by considering the average contribution inside each atom and bond.

5.4.3 By considering each feature separately

Another way to visualize the contributions is to display them separately. In figure 5.4, we would first draw a figure with the first atoms feature only, i.e. all the $a_{i,1}$ for $1 \leqslant i \leqslant n$, where n is the number of atoms. Then we draw all the $a_{i,2}$, etc. Since there are 8 features for each atom, we would get 8 visualizations. The same principle applies for bonds which have 4 features, so we would get 4 visualizations. Since there are also two directions for each bond, we could draw 8 visualizations. But to simplify it, we will only take the average contributions between the two directions, in order to have a global idea of the contribution of one feature on the bond, regardless of the direction.

We then obtain a total of 12 visualizations, shown in figure 5.6.

The case of β-lactam antibiotics

We remark that there are a lot of differences depending on the feature. For Bernard, these different aspects depending on the molecule properties (features) are better for a chemist, than the average contribution presented before in figure 5.5. One interesting feature is *aromaticity*, which tells if the atom is part of a cycle or not. According to Bernard, it is in fact, already known in drug research that 6-cycles docked to 4-cycles play a key role in the antibiotic properties[6]. Such a shape is called a β-lactam ring. Bernard noticed that the contribution of β-lactam rings are always positive, which is a good sign. Examples are shown in figure 5.7.

What Bernard found interesting, is that not only β-lactam rings are positive, but also atoms that surround them, meaning that the β-lactam alone is not sufficient, and is only the base for a more complex and important structure in the molecule to be antibiotic. We can also notice that other rings are marked in light blue, meaning that according to the model and to the explainer, these parts are not contributing to the antibiotic property. It is also very interesting to see such patterns on the visualizations, since we could also get random contributions without noticing any pattern

[6] https://en.wikipedia.org/wiki/B-lactam_antibiotic

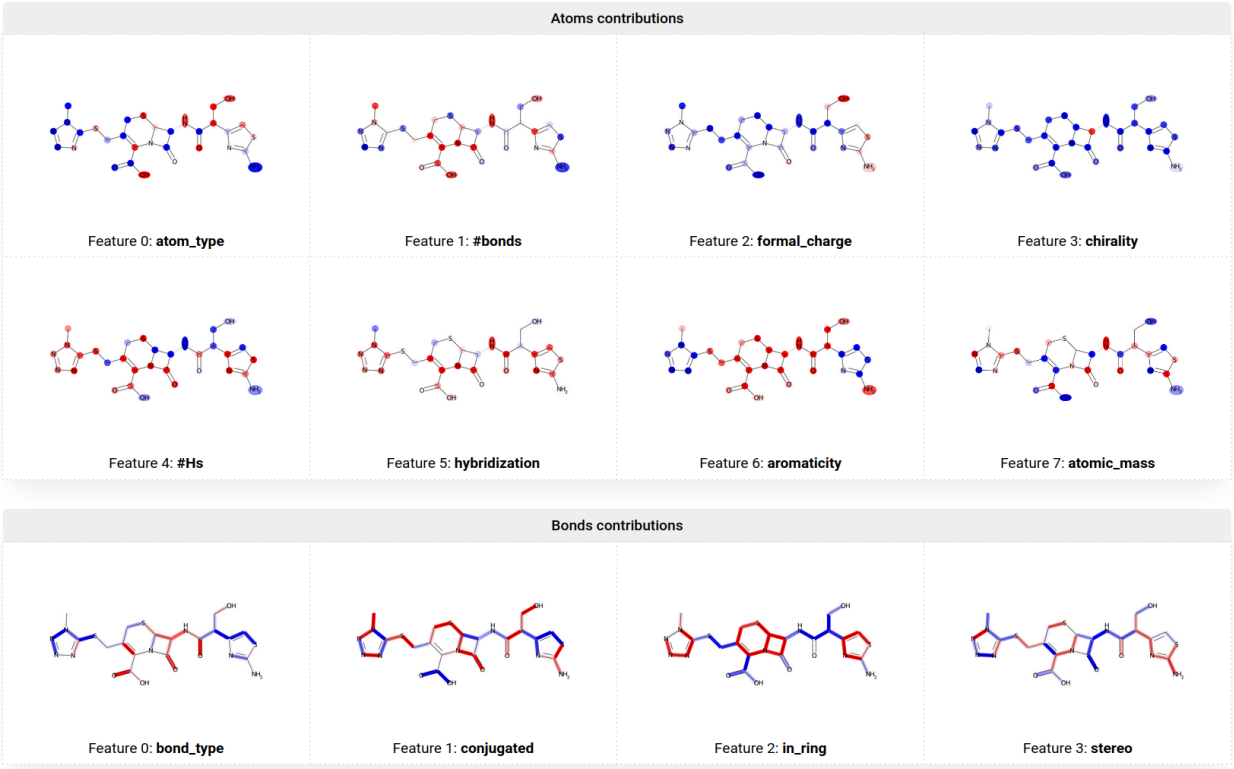


Figure 5.6: LIME explanation for the molecule, by considering each feature separately.

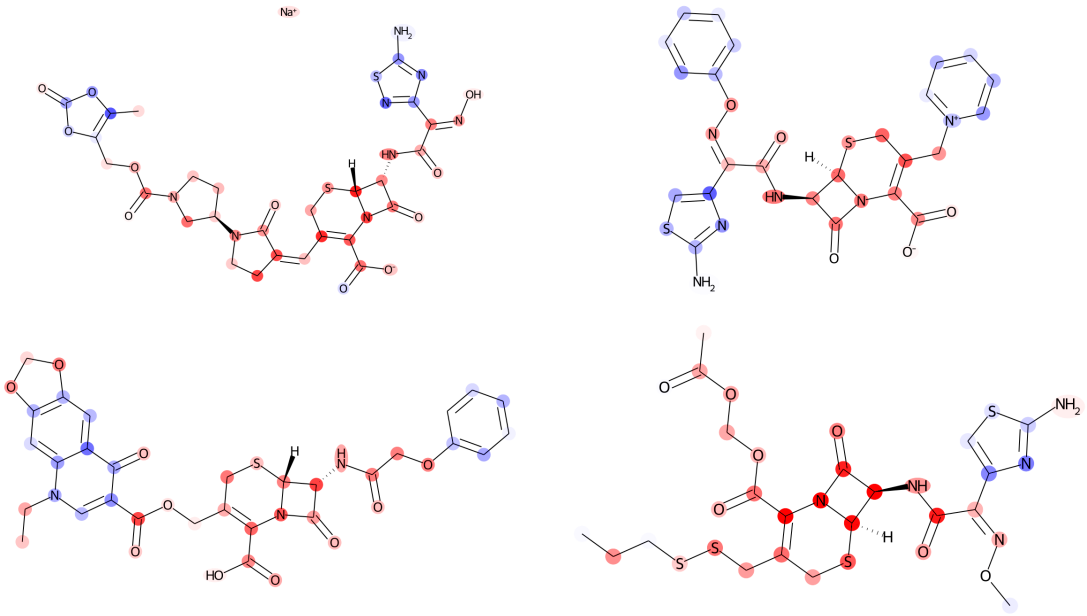


Figure 5.7: β-lactam antibiotics aromaticity explained with LIME where the β-lactam ring's contribution is positive.

if either the model or the explainer doesn't work properly.

5.4.4 Color scale normalization

Even though these visualizations are significant, they are not representative of their feature contributions relatively to the other features. This is because the color map has a different scale adapted to the contributions for each feature. It is better to keep the same color scale for all the features, but also to be informed of the bounds of this color scale (max. and min. contribution). When looking at the first molecule we tested, we get the result in figure 5.8.

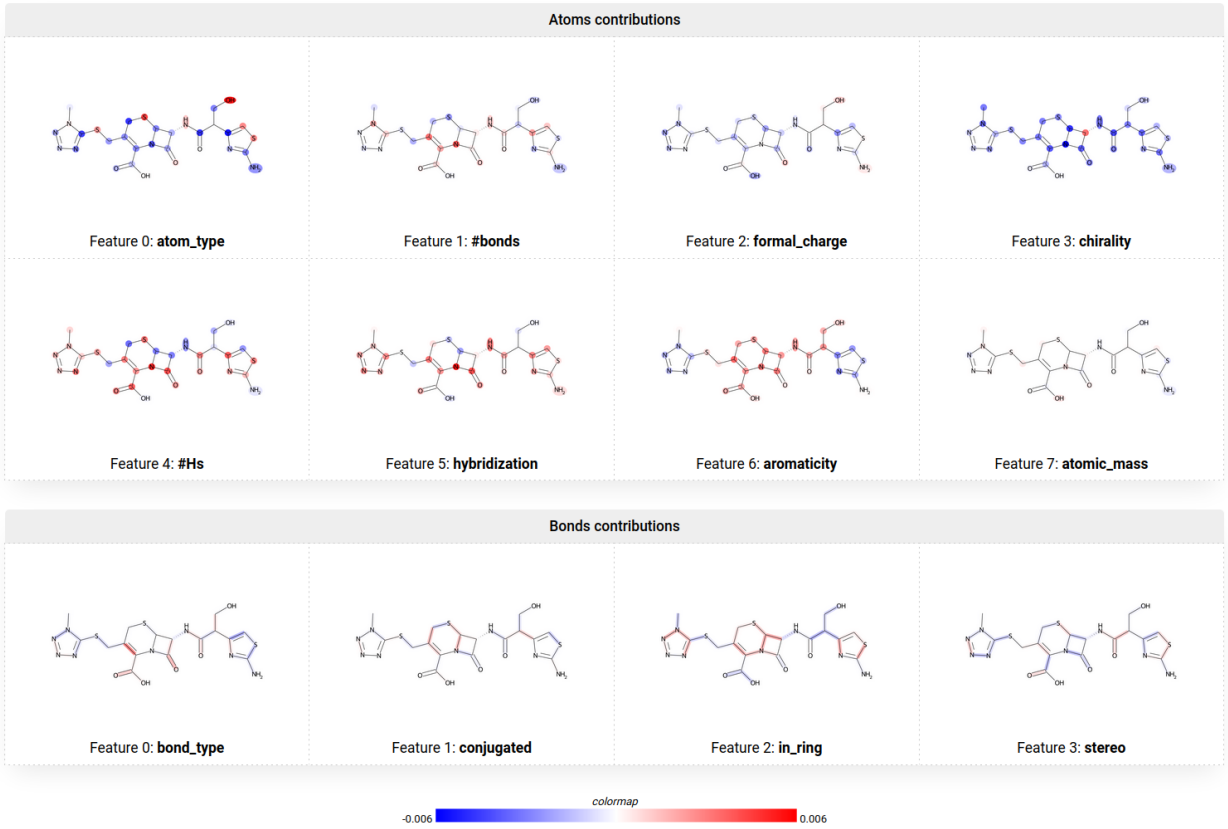


Figure 5.8: LIME explanation for the molecule, by considering each feature separately, given the same color scale for each molecule visualization.

We see more clearly that the atomic mass is relatively not important. It seems moreover that the atom type plays a more important role than aromaticity, as well as the #Hs feature. We observe the same behavior for other molecules, as presented in the appendix (figures C.1, C.2, C.3, C.4).

5.5 Comparison to PathExplain and interaction explanation

5.5.1 PathExplain for interaction explanations

Something Bernard wanted to see, is also the contribution of the interaction of two features. More specifically, he wanted to know the contribution of oxygen atoms with an sp2 hybridization since it impacts the polarity of molecules[7], thus impacts their properties.

[7] https://chem.libretexts.org/Bookshelves/Organic_Chemistry/Organic_Chemistry_I_(Cortes)/05%3A_Orbital_Picture_of_Bonding-_Orbital_Combinations_Hybridization_Theory_and_Molecular_Orbitals

A simple solution to explain this polarity property is to create a new polarity feature, that can be computed using the two other features *atom type* and *hybridization*. The inconvenient is that adding a feature implies modifying the graph mol structure, and thus the model's definition, which also implies to re-train it. It would be preferable to let the model *as is*, and simply find a way to explain the interaction between two existing features.

This is a job PathExplain can do [12]. We already used PathExplain for neural networks in the previous chapter, and as Chemprop is also a neural network, we are lucky! PathExplain defines the interaction importance between a feature i and a feature j as the contribution of j on the contribution of i[8]:

$$\Gamma_{i,j}(x) = \Phi_j(\Phi_i(x)) = \Gamma_{j,i}(x) = \Phi_i(\Phi_j(x)) \tag{5.1}$$

Where $\Gamma_{i,j}(x)$ is the contribution of the interaction between the features i and j in the instance x, Φ_i and Φ_j are their respective attribution calculation functions, defined in the equation 3.3 in the state of the art. Such a calculation is made possible by the fact that Φ_i itself is also a differentiable function, which means that we can apply the integrated gradients once again. This function composition is also demonstrated to be commutative, so the order of the two features i and j in the calculation has no importance.

5.5.2 The problem of explaining interactions with LIME

We have seen that the key concept of feature interaction explanation is the *explanation of explanation*. In theory, we can do the same thing with LIME: for a given model, we can define a function that given an instance x, returns the contribution of x_i; and we can then explain this function for the instance x in order to get the contribution of x_j in the explanation of x_i. A quite easy implementation, in the end.

In practice, this implementation is nearly unusable, because of the time it would take. Explaining a molecule like one of the antibiotics we presented before, takes typically around one minute. However, to make an explanation, LIME needs to call the function 5000 times, as we have seen before. 5000 minutes correspond to roughly 3.5 days, just to explain the interaction with one feature. Of course, such a solution is not reasonable, since we may need to explain the interactions of many molecules in one single day.

5.5.3 Implementation of PathExplain

Diffentiability of the model wrapper

Because PathExplain can only explain differentiable models, our model wrapper defined in subsection 5.2.2 must be differentiable. Since we only perform slicing on the input array in order to place the values at the right place in the graph, it should not be an obstacle to differentiability. In practice though, the mol graphs are converted into another object of the class `BatchMolGraph`, which is the grouping of many mol graphs. In fact, a `BatchMolGraph` includes all the molecules inside one whole graph that is passed to the model, but the molecules in this graph are just not connected and have no atom or bond in common.

[8] hence the name of their article "*Explaining explanations*"

It is, among others, in the `BatchMolGraph` definition that the values of the mol graphs are converted into tensors, that are then passed through the model. Chemprop is written with PyTorch, so it needs a tensor at the starting point in order to attach the *gradient graph* (that serves PyTorch to calculate the gradient from the operations history). The problem is that the conversion *mol graph* to `BatchMolGraph` performed by Chemprop is not differentiable, because of the way the values are copied. We could redefine this function, but again, it implies modifying the Chemprop module itself.

The solution has been to take a tensor as input of the model wrapper, and put slices of this tensor at the right place directly in a `BatchMolGraph`, to be given to the model. That removes the conversion part which was a problem for differentiability.

Another difference with LIME is that since the explanation only relies on differentiability, we can't simplify the one hot vectors in the graph mol with categorical identifiers. When converting the graph mol in tensor, we then keep the entire one hot vectors. We also have to choose a base value for our features, as defined in the integrated gradients formula (3.3). We define it to 0, which represents the absence of category, and which means that, still according to the formula, only the category set to 1 in the one hot vector will have a non-zero contribution.

Memory explosion and contribution to PathExplain

The implementation for the explanations with PathExplain ended by working well. The interaction explanation however didn't. After investigating the origin of the problem, it turned out that when approximating the integral calculation, PathExplain accumulates many discrete values in order to sum them later. The problem is that accumulated values are tensors, with a large[9] gradient graph attached to them. This took more than 4Gb RAM, which made the program crash. A simple `.detach()` method call during accumulation solved the problem, which made the program take only about 200Mb RAM for this operation. This resulted in a pull request on the GitHub repository[10] to fix this issue.

In order to save time, I also made PathExplain compute the gradient only for features we needed, by detaching the gradient at features we don't need during this process. So to conclude, we managed to save space and time. Yay!

5.5.4 Results

Features contributions

We can instantly remark a good point for PathExplain: it takes only about a second to do the explanations that took around one minute with LIME.

Here are the explanations for the first molecule, presented in figure 5.9. In comparison, other results are shown in the appendix (figures C.5, C.6, C.7, C.8).

Compared to LIME, the visualizations are sparser, which makes high-contribution areas clearer. With Clément and Bernard, we also had the impression that similar molecules also had similar

[9] actually larger than a simple gradient calculation, since it concerned the second derivative

[10] https://github.com/suinleelab/path_explain/pull/12

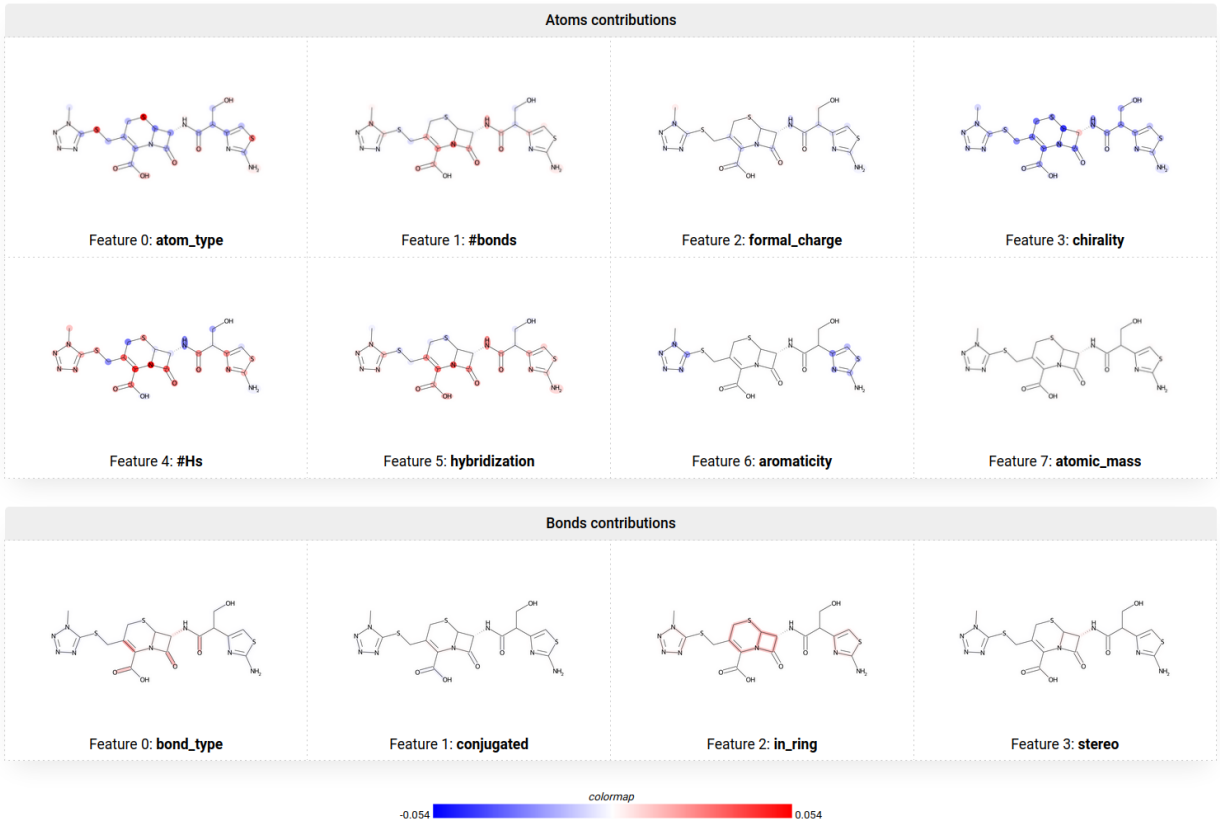


Figure 5.9: Molecule features explanations with PathExplain.

explanations, which would make this explainer more robust, although this can't be rigorously measured.

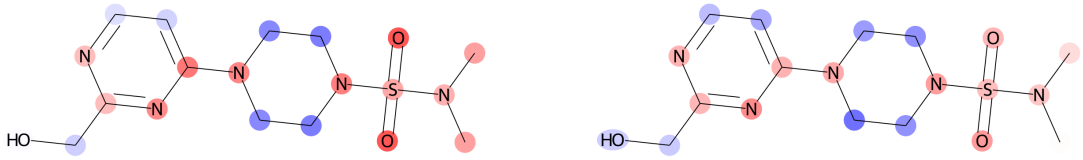
Symmetry property

An interesting property is also that symmetrical atoms and bonds in the molecule have the exact same contribution. This is due to the fact that symmetrical nodes in the graph are processed exactly the same way in the Message Passing Network, since the same operations with the same matrices in the neural network are applied for every node and edge, the only difference being their features and the other nodes they are connected to. Here, symmetrical atoms and bonds have the same features and the same connections in the graph. Hence their contributions are identical.

This is not the case with LIME, which have random variations. In figures 5.10 and 5.11, we can see a tiny difference between symmetrical atoms on the LIME explanation, whereas PathExplain do not have any color variation. We also notice that the contributions obtained by the two explainers are similar.

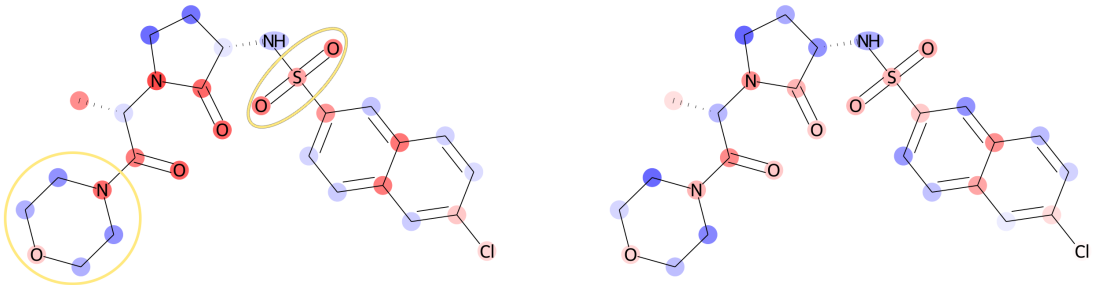
Interactions

Even though PathExplain allows us to compare LIME with another explainer, is faster and more robust, it had initially been implemented to compute interaction contributions between features, more specifically between the atom type and the hybridization feature. Figure 5.12 shows this result on our molecule example.



(a) PathExplain (b) LIME

Figure 5.10: Explanation of the feature **#Hs**, comparison between PathExplain and LIME. The part on the right is symmetrical.



(a) PathExplain (b) LIME

Figure 5.11: Explanation of the feature **#Hs**, another comparison between PathExplain and LIME. The symmetrical parts are circled in yellow.

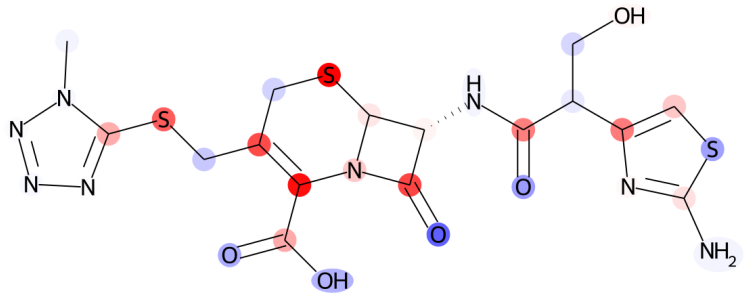


Figure 5.12: Explanation of the interactions between the atom type and the hybridization, with PathExplain

The bounds of the color map are very low (approximately 2.10^{-4}), which means that even if this figure is full of colors, these interactions don't seem to play an important role in the antibiotic classification. Again, other examples are available in the appendix (figures C.9, C.10, C.11, C.12). The main areas of high contributions are also not clear on this figure, and the interaction explanation might be a bit noisier compared to the simple feature explanation.

However, it is normal that the interaction importances are much lower than the simple feature explanations. There is, in fact, a property we haven't talked about yet:

$$\sum_{i=1}^{n} \Phi_i(x) = 1 \tag{5.2}$$

Where n is the number of features. This property has been demonstrated in [12] and illustrates the idea of taking one unit of contribution, and distribute it amongst the features. From that property, they also show that:

$$\sum_{j=1}^{n} \Gamma_{i,j} = \Phi_i(x) \tag{5.3}$$

Even if there is no boundary constraint, it shows us that interaction contributions represent only one fraction of the whole contribution of one of our thousands features, which explains somehow why we get so little bounds or insignificant visualizations.

5.6 Discussion

We managed to explain a model trained to recognize antibiotic molecules, in order to have an idea of what have been learned by the model. We succeeded to get visually speaking molecule explanations thanks to the fact that Chemprop makes predictions from graphs, and graphs are more convenient to represent antibiotics contributions.

One thing that could be interesting would be to have a summary of all the contributions. In the last chapter, we managed to perform a global explanation using many local contributions and by aggregating them. Since graphs for this model have not the same shape, it is way harder to make an aggregation. In fact, find the global important patterns in hundreds of molecule explanations, and for many features, is a complete data mining task, that we didn't treat in this internship.

For the moment, we have the eyes of chemistry researchers, which have already noticed the positive contributions of certain cycles, as β-lactams. However, for the moment, empirical rules found so far are already known by the community, so there is no novelty at this time. It may also be caused by the fact that chemists are already part of this community, and are biased on these known properties on antibiotics, whereas other rules would remain to discover just by noticing a common point between several molecule explanations, hence the interest of finding an aggregation method.

In addition, even though we look what has been learned, we cannot be certain that the model has learned well. We also collaborated with Raphaël Duval, full professor of clinical microbiology at the faculty of pharmacy in the Université de Lorraine, for sharing our results and using our tool to analyze new molecules. He says that the most interesting point is that the model isn't, at first glance, biased by the results we know today in drug research. The model tries to learn it itself.

However, I would say that the molecules we use for training are already biased by the discoveries we did in antibiotics before. For example, if we know that a particular pattern has beneficial effects, we would likely be producing several antibiotics based on this pattern, which could bias the model on it. We could therefore gain in training the model on antibiotics whose we don't know why they have such effects, and our method would tell us why.

5.7 Web interface

To make this explanation process more user-friendly, especially for chemistry researchers who wouldn't be comfortable with Python, I developed a web interface using the Python framework *Flask*. When receiving a request, Flask calls our Python module, and send a response back to the web browser. All the interface was developed in HTML, CSS and Javascript, without using any library.

The user is guided to three steps:

1. The home page, where he can enter a SMILES in the main text bar, select a model and show its predictions (figure 5.13);
2. The predictions are loaded using an AJAX request. The user is then invited to select an explainer and click on one class (figure C.13, in appendix);
3. The features explanations, also loaded with an AJAX request. All the figures are rendered in Python using RDKit and stored in a temporary folder. The user can then explain features interactions if he is using PathExplain (figure C.14).

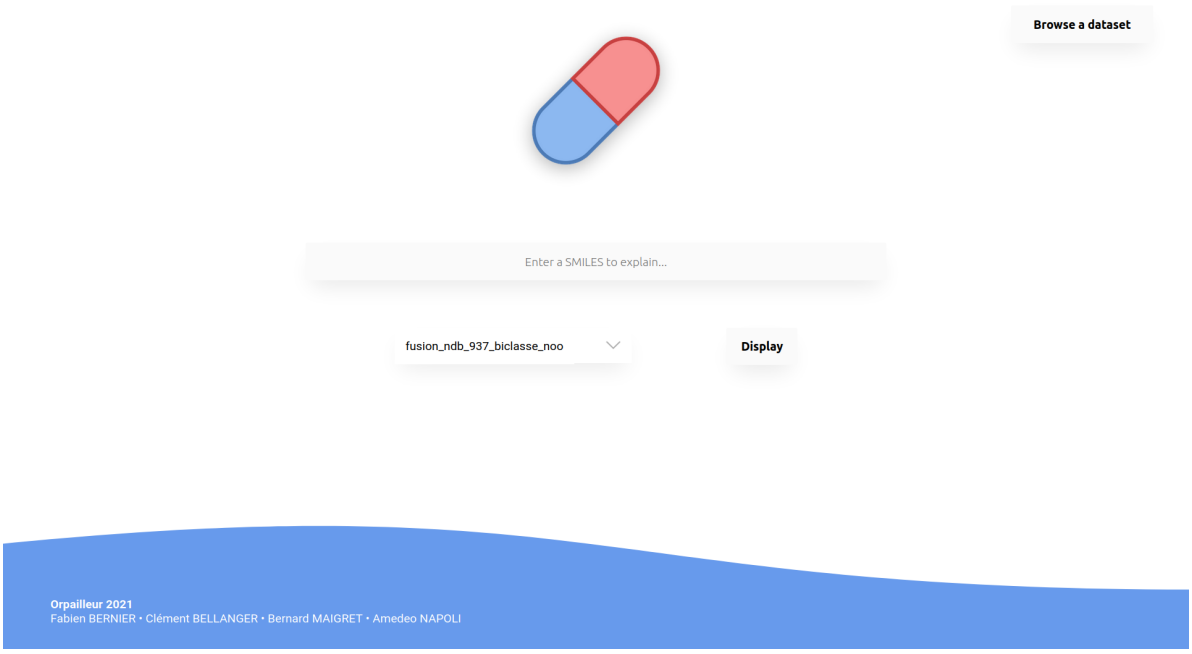


Figure 5.13: Home page of the web interface.

I also implemented a dataset explorer in order to visualize the molecules before making predictions or explain it (figures C.15, C.16). The dataset must be a CSV file containing a SMILES string in the first column.

All this interface and the installation of Chemprop and other dependencies are automated by the deployment of a Docker container.

Although I spent some time on this interface, its development will not be detailed in this report, in order to focus on the main subject.

6 Conclusion

We have seen that *FixOut*, a technique to tackle fairness on machine learning classifiers, can be adapted to textual models. Explainers allowed us to show that we manage to reduce the reliance on sensitive words in model predictions. *FixOut* can also be applied in particular to neural networks, which are very common models in natural language processing. We also have seen that this latter application doesn't need an additional training stage to be efficient, which allows to free space, but also to save time. *FixOut* can then be used directly on pre-trained models, although there is a model compilation part in the case of Tensorflow implementations, for every single textual instance, which slows the processing down.

The usage of the same explainers were then diverted in a knowledge discovery application. More specifically, we adapted them by building a model wrapper for antibiotic classifiers and by making it differentiable, which finally resulted in an interactive web interface being able to highlight important patterns in any molecule for its antibiotic classification. The visualizations can easily be interpreted by chemistry researchers, who already have found interesting results, though they are already known by the drug research community. Thanks to the Docker containerization, this application can moreover be easily used on different machines.

These two related projects illustrate the wide potential of explainers, that are used in a general purpose to indicate what a machine learning model has learned. The results we obtained also have been written in one published paper [1], two others being in progress for the case of *FixOut* applied to neural networks, and the explanations results on Chemprop. The explanations we got with Chemprop are very encouraging and open up possibilities in drug research, in particular for antibiotic molecules, but also other domains where explainers may have a role to play.

Personal impressions

This end-of-study project was more than an internship, since it started in September 2020 with a research contract, where previous experiments on fairness were leaded. This is where I discovered the joy of research, but also the papers submission deadlines, the peer-reviewing process, rejections, corrections, re-submissions, until we finally succeed to publish! I also discovered the different conferences and networks between researchers, participated in a summer school[1], attended to a really interesting conference on fairness and explainability[2], which gave me the opportunity to talk with researchers from the community, despite the fact that these events were unfortunately all online.

I only could partially come four months in a year, and of course the most interesting months

[1] FirstINRIA-DFKIsummerschoolonArtificialIntelligence:https://idessai.inria.fr/

[2] ACM Conference on Fairness, Accountability, and Transparency: https://facctconference.org/2021/index.html

were when I had the occasion to be with the team in face-to-face. It was a real pleasure to work in this team, and it motivated me to continue my studies by doing a PhD thesis at the University of Luxembourg next year in the same field. But yet fortunately, I will still have a strong contact with the Orpailleur team!

References

[1] Guilherme Alves, Maxime Amblard, Fabien Bernier, Miguel Couceiro, and Amedeo Napoli. Reducing unintended bias of ml models on tabular and textual data, 2021. 23, 26, 31, 57

[2] Guilherme Alves, Vaishnavi Bhargava, Fabien Bernier, Miguel Couceiro, and Amedeo Napoli. FixOut: an ensemble approach to fairer models. working paper or preprint, December 2020. 23

[3] Guilherme Alves, Vaishnavi Bhargava, Miguel Couceiro, and Amedeo Napoli. Making ml models fairer through explanations: the case of limeout, 2020. 23

[4] Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. Neural machine translation by jointly learning to align and translate, 2016. 10

[5] Vaishnavi Bhargava, Miguel Couceiro, and Amedeo Napoli. Limeout: An ensemble approach to improve process fairness, 2020. 23

[6] L. Breiman. Random Forests. *Machine Learning*, 45(1):5 – 32, 2001. 8

[7] N. V. Chawla, K. W. Bowyer, L. O. Hall, and W. P. Kegelmeyer. Smote: Synthetic minority over-sampling technique. *Journal of Artificial Intelligence Research*, 16:321–357, Jun 2002. 22

[8] Kyunghyun Cho, Bart van Merrienboer, Caglar Gulcehre, Dzmitry Bahdanau, Fethi Bougares, Holger Schwenk, and Yoshua Bengio. Learning phrase representations using rnn encoder-decoder for statistical machine translation, 2014. 10

[9] Thomas Davidson, Debasmita Bhattacharya, and Ingmar Weber. Racial bias in hate speech and abusive language detection datasets, 2019. 25, 28

[10] 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. 25, 30

[11] Yoav Freund et Robert Schapire. A decision-theoretic generalization of on-line learning and an application to boosting. *Journal of Computer and System Sciences*, 55(1):119 – 139, 1997. 9

[12] Joseph D. Janizek, Pascal Sturmfels, and Su-In Lee. Explaining explanations: Axiomatic feature interactions for deep networks. *Journal of Machine Learning Research*, 22(104):1–54, 2021. 17, 51, 55

[13] Surya Mattu Julia Angwin, Jeff Larson and Lauren Kirchner. Machine bias. *ProPublica*, May 2016. Available at https://www.propublica.org/article/machine-bias-risk-assessments-in-criminal-sentencing. 1, 14, 21

[14] Y. LeCun. *Learning Processes in an Asymmetric Threshold Network*. Springer-Verlag, 1986. 10

[15] Fanzhen Liu, Shan Xue, Jia Wu, Chuan Zhou, Wenbin Hu, Cecile Paris, Surya Nepal, Jian Yang, and Philip S Yu. Deep learning for community detection: Progress, challenges and opportunities. *Proceedings of the Twenty-Ninth International Joint Conference on Artificial Intelligence*, Jul 2020. 1

[16] Scott Lundberg and Su-In Lee. A unified approach to interpreting model predictions, 2017. 16

[17] Yuan Luo, Ya Xiao, Long Cheng, Guojun Peng, and Danfeng (Daphne) Yao. Deep learning-based anomaly detection in cyber-physical systems: Progress and opportunities. *ACM Comput. Surv.*, 54(5), May 2021. 1

[18] Karima Makhlouf, Sami Zhioua, and Catuscia Palamidessi. On the applicability of ml fairness notions, 2020. 38

[19] Marcelo O. R. Prates, Pedro H. C. Avelar, and Luis Lamb. Assessing gender bias in machine translation – a case study with google translate, 2019. 24

[20] Bharath Ramsundar, Peter Eastman, Patrick Walters, Vijay Pande, Karl Leswing, and Zhenqin Wu. *Deep Learning for the Life Sciences*. O'Reilly Media, 2019. https://www.amazon.com/Deep-Learning-Life-Sciences-Microscopy/dp/1492039837. 14

[21] Marco Tulio Ribeiro, Sameer Singh, and Carlos Guestrin. "why should i trust you?": Explaining the predictions of any classifier, 2016. 15, 27, 61

[22] Frank Rosenblatt. The perceptron: a probabilistic model for information storage and organization in the brain. *Psychological Review*, 65(6), 1958. 9

[23] Mukund Sundararajan, Ankur Taly, and Qiqi Yan. Axiomatic attribution for deep networks, 2017. 17

[24] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is all you need, 2017. 11

[25] Kevin Yang, Kyle Swanson, Wengong Jin, Connor Coley, Philipp Eiden, Hua Gao, Angel Guzman-Perez, Timothy Hopper, Brian Kelley, Miriam Mathea, et al. Analyzing learned molecular representations for property prediction. *Journal of chemical information and modeling*, 59(8):3370–3388, 2019. 13

[26] S. Kevin Zhou, Hayit Greenspan, Christos Davatzikos, James S. Duncan, Bram Van Ginneken, Anant Madabhushi, Jerry L. Prince, Daniel Rueckert, and Ronald M. Summers. A review of deep learning in medical imaging: Imaging traits, technology trends, case studies with progress highlights, and future promises. *Proceedings of the IEEE*, 109(5):820–838, 2021. 1

List of Figures

List of Tables

Glossary

AI Artificial Intelligence 29

AJAX Asynchronous Javascript and XML 56

COMPAS Correctional Offender Management Profiling for Alternative Sanctions *(predictive model)* 1, 14

CSV Comma-separated values 12, 56, 65

DSAA Data Science and Advanced Analytics 26, 28, 31

GDPR General Data Protection Regulation 1

GNN Graph Neural Network 12

GRU Gated Recurrent Unit *(recurrent layer)* 10

HTML Hypertext Markup Language 19

HTTP Hypertext Transfer Protocol 19

JSON JavaScript Object Notation 19

LIME Local Interpretable Model-agnostic Explainer 15, 16, 26–28, 30–32, 34, 37, 39, 43, 45, 48–53, 62

LORIA Laboratoire Lorrain de Recherche en Informatique et ses Applications 39

LSTM Long Short-Term Memory *(recurrent layer)* 10, 61

ML Machine Learning 28

MPN Message Passing Network 13

NLP Natural Language Processing 10, 11, 28, 32, 33

PCA Principle Component Analysis 11

regex regular expression 29

SHAP SHapley Additive exPlanations 16, 26, 28, 30–32, 34, 37, 39, 43, 45

SMILES Simplified Molecular-Input Line-Entry System *(textual format for molecules)* 12, 13, 19, 43, 45, 46, 56

TF-IDF Term Frequency - Inverse Document Frequency 28, 29

Appendices

A Global explanations on EnsembleOut for textual models

		Original model		EnsembleOut	
	Word	**Rank**	**Contrib.**	**Rank**	**Contrib.**
LR, LIME+RS	*niggah*	4	0.62	12	0.277
	nigger	7	0.597	10	0.29
	nigguh	9	0.522	14	0.264
	nig	13	0.328	20	0.15
	nicca	14	0.316	22	0.134
	nigga	18	0.237	27	0.113
	white	23	0.172	35	0.104
	black	131	0.05	316	0.028
RF, LIME+RS	*niggah*	7	0.593	10	0.289
	nigger	9	0.497	14	0.236
	nigguh	13	0.358	17	0.198
	nig	10	0.46	16	0.222
	nicca	14	0.286	20	0.132
	nigga	18	0.23	23	0.112
	white	23	0.119	33	0.066
	black	>500	~ 0	>500	~ 0
ADA, LIME+RS	*niggah*	3	0.166	5	0.082
	nigger	7	0.052	11	0.026
	nigguh	5	0.144	6	0.074
	nig	18	0.014	24	0.006
	nicca	17	0.015	23	0.007
	nigga	4	0.166	4	0.083
	white	22	0.011	26	0.005
	black	115	0.0	192	0.0
BAG, LIME+RS	*niggah*	6	0.687	13	0.32
	nigger	10	0.629	15	0.306
	nigguh	13	0.535	17	0.248
	nig	2	0.7	10	0.353
	nicca	15	0.395	19	0.2
	nigga	19	0.284	21	0.139
	white	22	0.158	25	0.074
	black	>500	~ 0	>500	~ 0

(a) LIME explanations

		Original model		EnsembleOut	
	Word	**Rank**	**Contrib.**	**Rank**	**Contrib.**
LR, SHAP+RS	*niggah*	16	0.194	68	0.016
	nigger	7	0.273	18	0.042
	nigguh	22	0.126	>500	~ 0
	nig	12	0.223	>500	~ 0
	nicca	21	0.138	37	0.024
	nigga	19	0.179	13	0.061
	white	20	0.177	53	0.019
	black	>500	~ 0	>500	~ 0
RF, SHAP+RS	*niggah*	14	0.188	22	0.03
	nigger	12	0.219	21	0.031
	nigguh	23	0.132	84	0.008
	nig	10	0.272	68	0.011
	nicca	21	0.154	36	0.021
	nigga	17	0.17	12	0.075
	white	25	0.11	38	0.019
	black	>500	~ 0	>500	~ 0
ADA, SHAP+RS	*niggah*	5	0.048	11	0.007
	nigger	9	0.019	13	0.005
	nigguh	6	0.039	24	0.002
	nig	20	0.008	80	0.001
	nicca	26	0.006	37	0.001
	nigga	4	0.072	3	0.031
	white	22	0.008	28	0.002
	black	>500	~ 0	>500	~ 0
BAG, SHAP+RS	*niggah*	17	0.203	26	0.02
	nigger	13	0.257	16	0.039
	nigguh	19	0.165	46	0.012
	nig	2	0.442	14	0.044
	nicca	14	0.217	21	0.026
	nigga	16	0.206	11	0.082
	white	24	0.091	53	0.009
	black	>500	~ 0	>500	~ 0

(b) SHAP explanations

Table A.1: Fairness assessment on textual data, global explanations with random sampling (RS).

		Original model		EnsembleOut	
	Word	**Rank**	**Contrib.**	**Rank**	**Contrib.**
LR, LIME+SP	*niggah*	1	0.596	12	0.275
	nigger	6	0.572	13	0.274
	nigguh	9	0.505	14	0.269
	nig	13	0.315	20	0.138
	nicca	15	0.289	23	0.12
	nigga	18	0.24	24	0.119
	white	23	0.178	45	0.096
	black	145	0.047	323	0.028
RF, LIME+SP	*niggah*	7	0.517	12	0.257
	nigger	9	0.476	15	0.23
	nigguh	13	0.339	17	0.194
	nig	10	0.445	16	0.204
	nicca	16	0.265	20	0.121
	nigga	17	0.235	23	0.112
	white	23	0.127	34	0.07
	black	>500	~ 0	>500	~ 0
ADA, LIME+SP	*niggah*	2	0.167	4	0.083
	nigger	7	0.052	10	0.026
	nigguh	5	0.144	6	0.073
	nig	18	0.014	24	0.006
	nicca	17	0.015	23	0.007
	nigga	4	0.166	5	0.083
	white	23	0.011	26	0.005
	black	113	0.0	196	0.0
BAG, LIME+SP	*niggah*	5	0.645	15	0.293
	nigger	11	0.58	16	0.29
	nigguh	13	0.497	18	0.233
	nig	1	0.659	10	0.336
	nicca	15	0.387	19	0.195
	nigga	19	0.285	21	0.139
	white	21	0.183	24	0.083
	black	420	0.004	>500	~ 0

(a) LIME explanations

		Original model		EnsembleOut	
	Word	**Rank**	**Contrib.**	**Rank**	**Contrib.**
LR, SHAP+SP	*niggah*	16	0.179	75	0.015
	nigger	11	0.239	23	0.034
	nigguh	21	0.123	>500	~ 0
	nig	9	0.249	>500	~ 0
	nicca	22	0.12	46	0.02
	nigga	19	0.159	14	0.053
	white	20	0.14	50	0.019
	black	>500	~ 0	>500	~ 0
RF, SHAP+SP	*niggah*	14	0.176	23	0.03
	nigger	12	0.213	21	0.031
	nigguh	22	0.13	83	0.008
	nig	7	0.276	65	0.011
	nicca	21	0.138	39	0.018
	nigga	18	0.155	12	0.067
	white	26	0.085	36	0.018
	black	>500	~ 0	>500	~ 0
ADA, SHAP+SP	*niggah*	5	0.05	11	0.007
	nigger	9	0.018	13	0.005
	nigguh	6	0.039	25	0.002
	nig	18	0.009	86	0.001
	nicca	34	0.006	40	0.001
	nigga	4	0.07	3	0.031
	white	32	0.007	30	0.002
	black	>500	~ 0	>500	~ 0
BAG, SHAP+SP	*niggah*	15	0.202	24	0.021
	nigger	12	0.28	17	0.038
	nigguh	19	0.166	44	0.012
	nig	2	0.508	14	0.044
	nicca	14	0.209	21	0.025
	nigga	17	0.195	11	0.078
	white	25	0.074	72	0.008
	black	>500	~ 0	>500	~ 0

(b) SHAP explanations

Table A.2: Fairness assessment on textual data, global explanations with submodular pick (SP).

B FixOut interactive demo

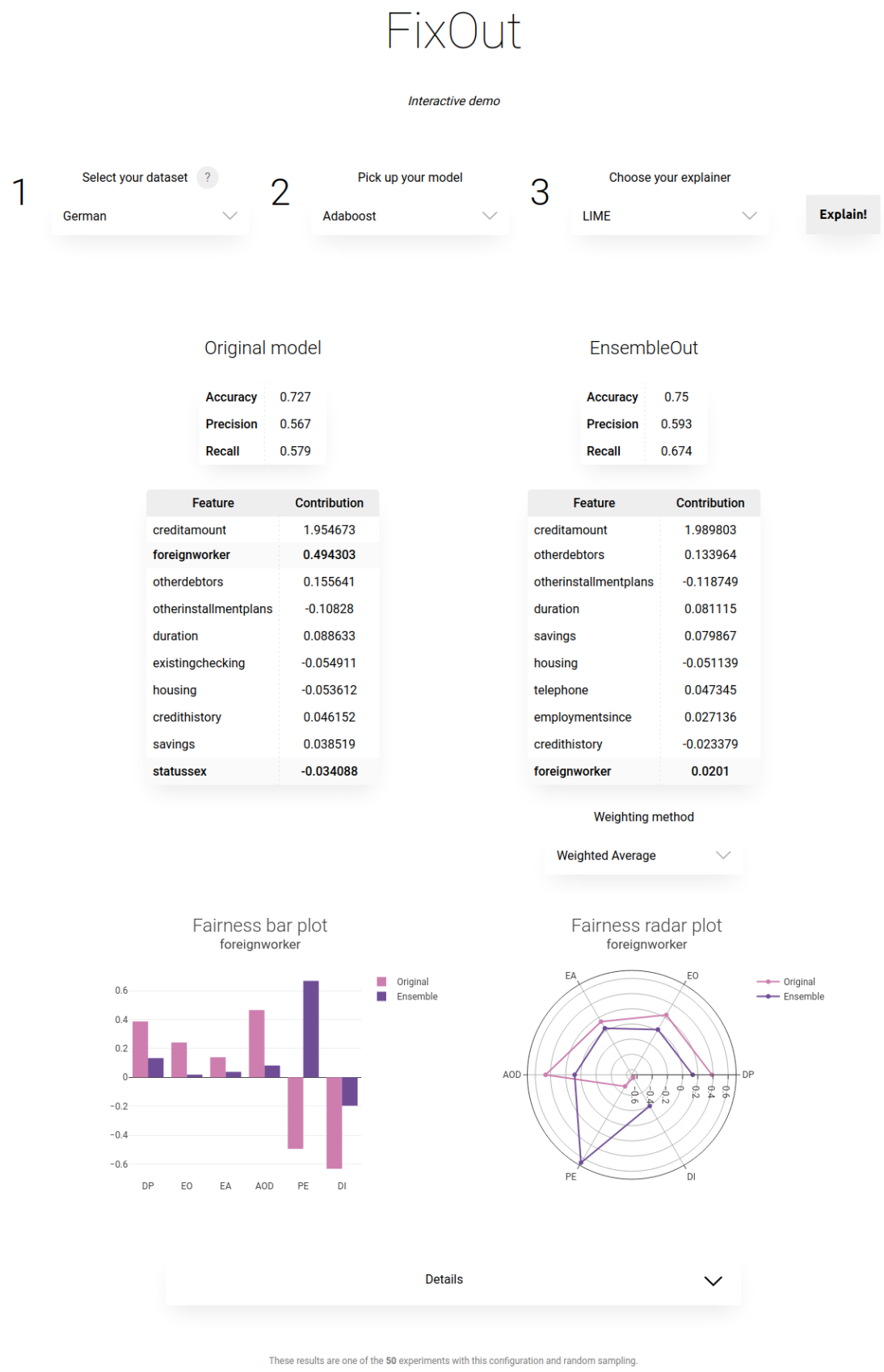


Figure B.1: Screenshot from the web demo of FixOut.

C Molecule explanations

We define the following molecules:

- **Molecule 1:** `Cc1oc(=O)oc1COC(=O)N1CC[C@@H](N2CC/C(=C/C3=C(C(=O)[O-])N4C(=O)[C@@H](NC(=O)/C(=N/O)c5nsc(N)n5)[C@H]4SC3)C2=O)C1.[Na+]`
- **Molecule 2:** `Nc1nc(/C(=N/Oc2ccccc2)C(=O)N[C@@H]2C(=O)N3C(C(=O)[O-])=C(C[n+]4ccccc4)CS[C@H]23)cs1`
- **Molecule 3:** `CCn1cc(C(=O)OCC2=C(C(=O)O)N3C(=O)[C@@H](NC(=O)COc4ccccc4)[C@H]3SC2)c(=O)c2cc3c(cc21)OCO3`
- **Molecule 4:** `CCCSSCC1=C(C(=O)OCOC(C)=O)N2C(=O)[C@@H](NC(=O)/C(=N\OC)c3csc(N)n3)[C@H]2SC1`

C.1 Molecules explained with LIME

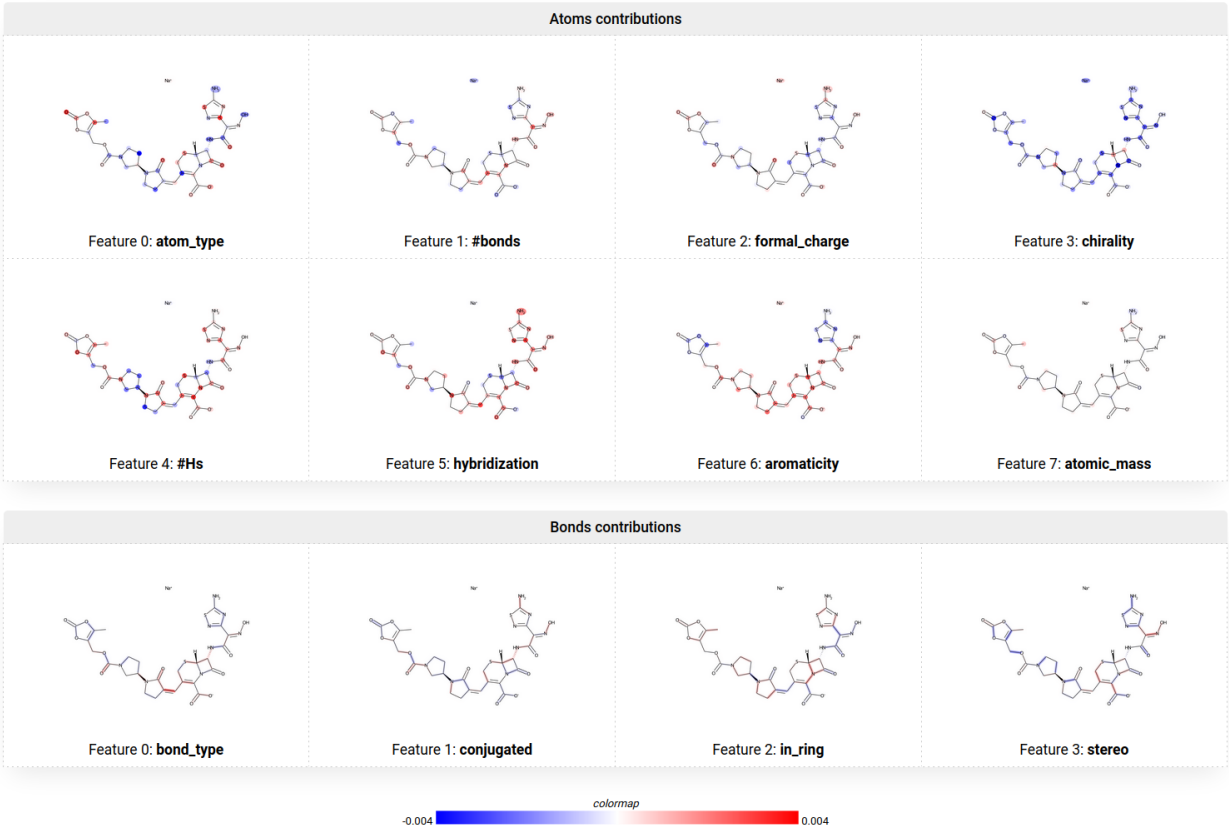


Figure C.1: Molecule 1

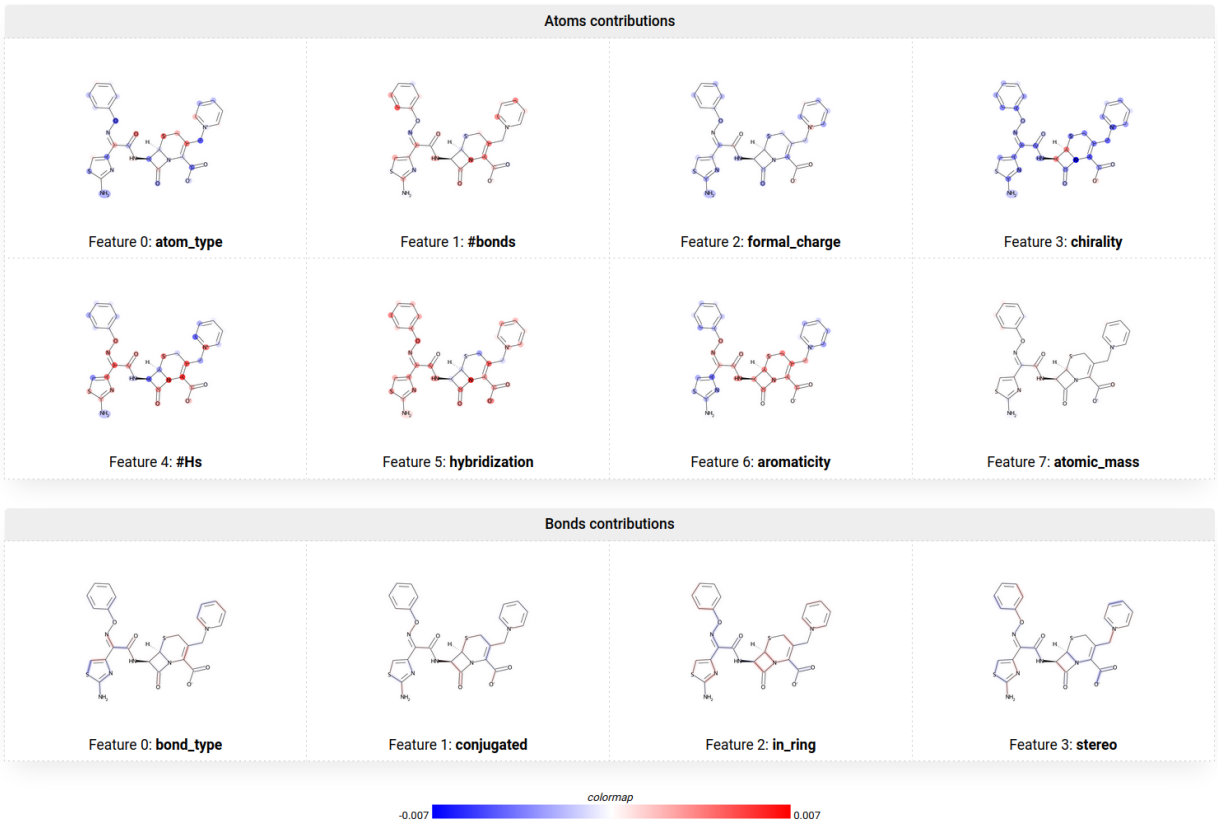


Figure C.2: Molecule 2

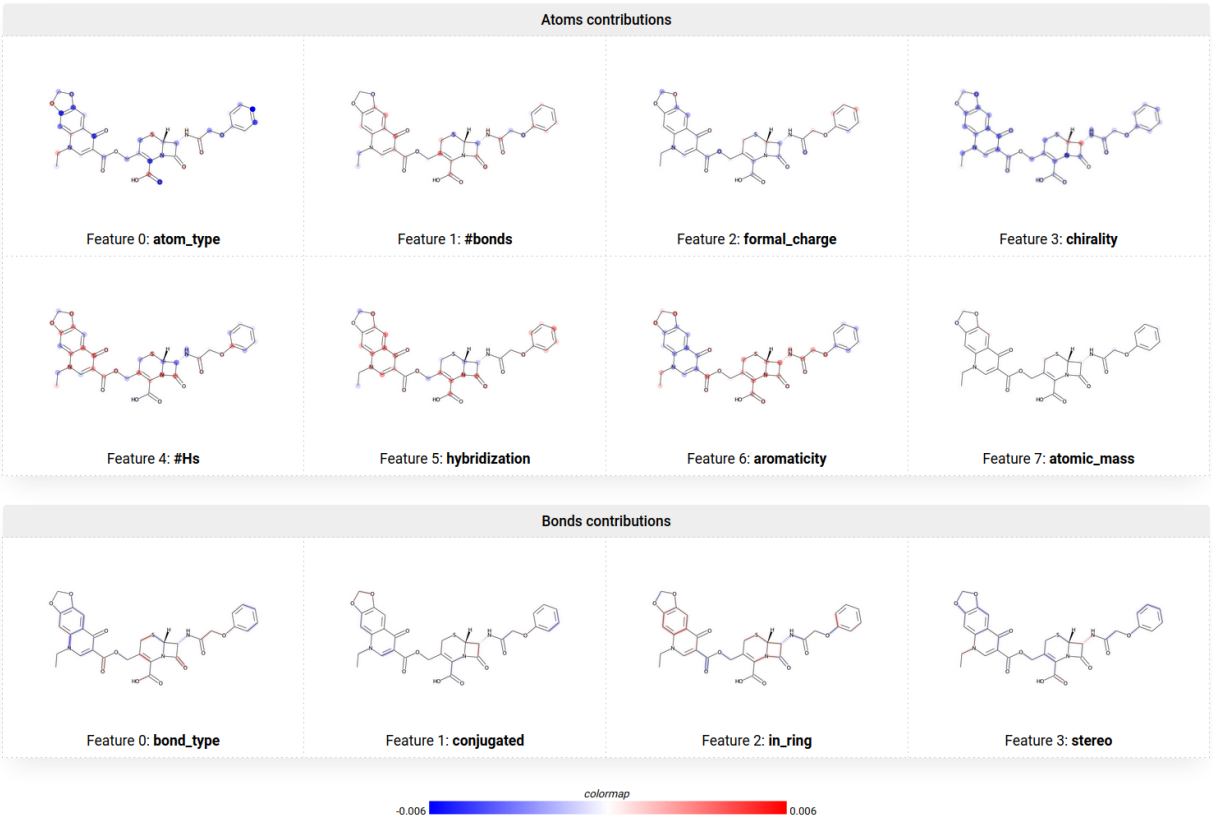


Figure C.3: Molecule 3

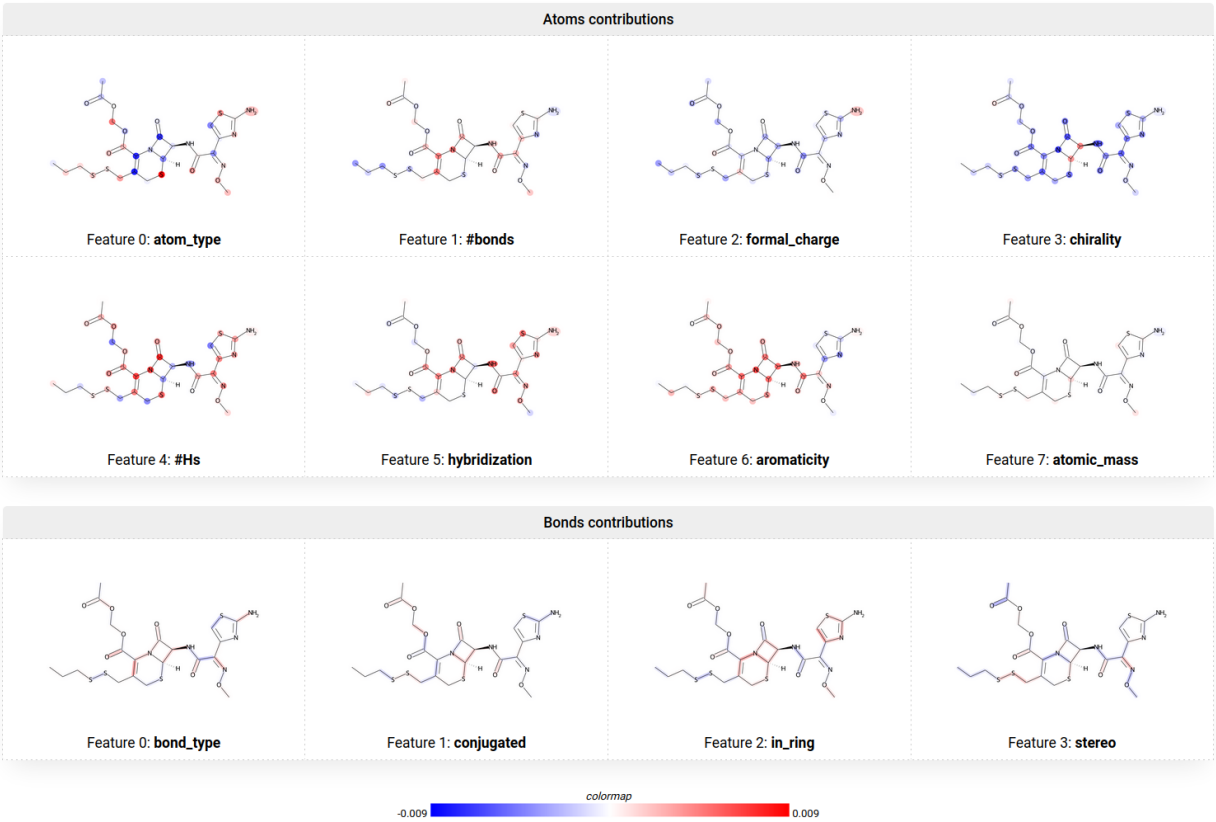


Figure C.4: Molecule 4

C.2 Molecules explained with PathExplain

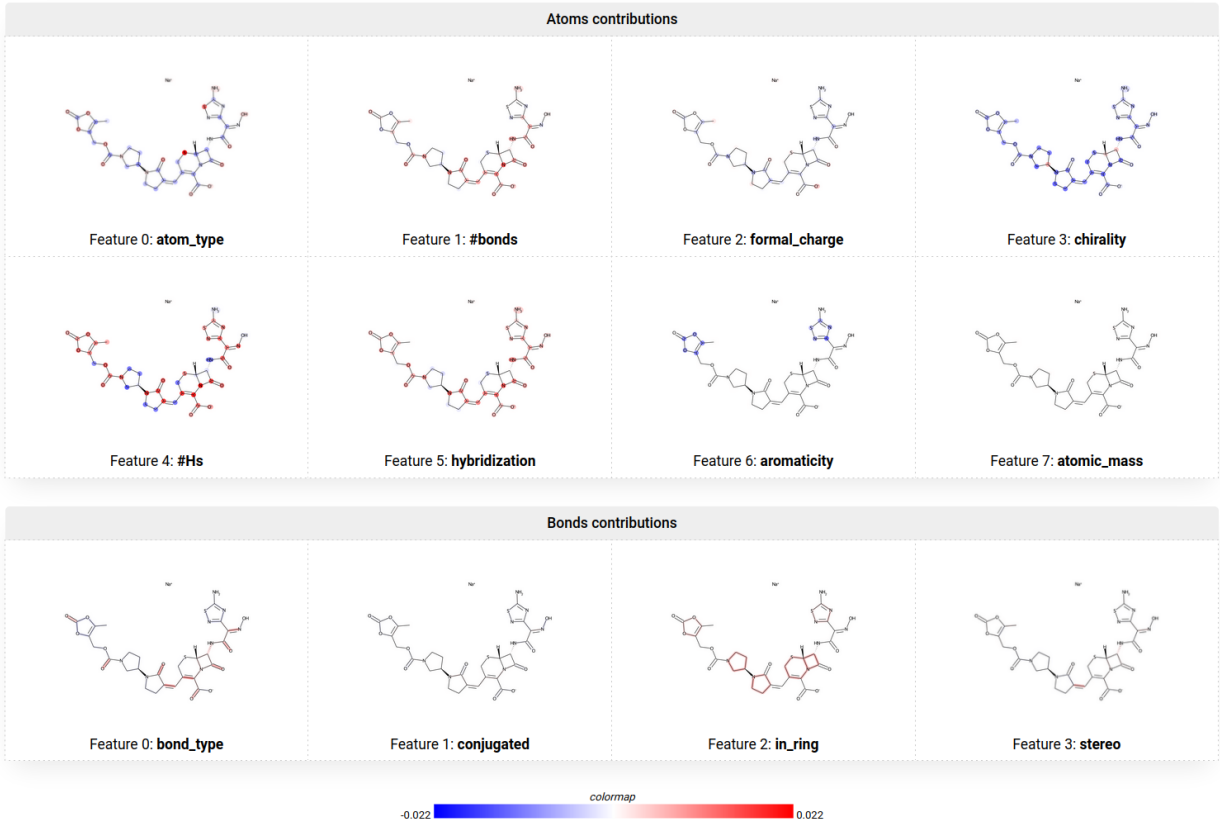


Figure C.5: Molecule 1

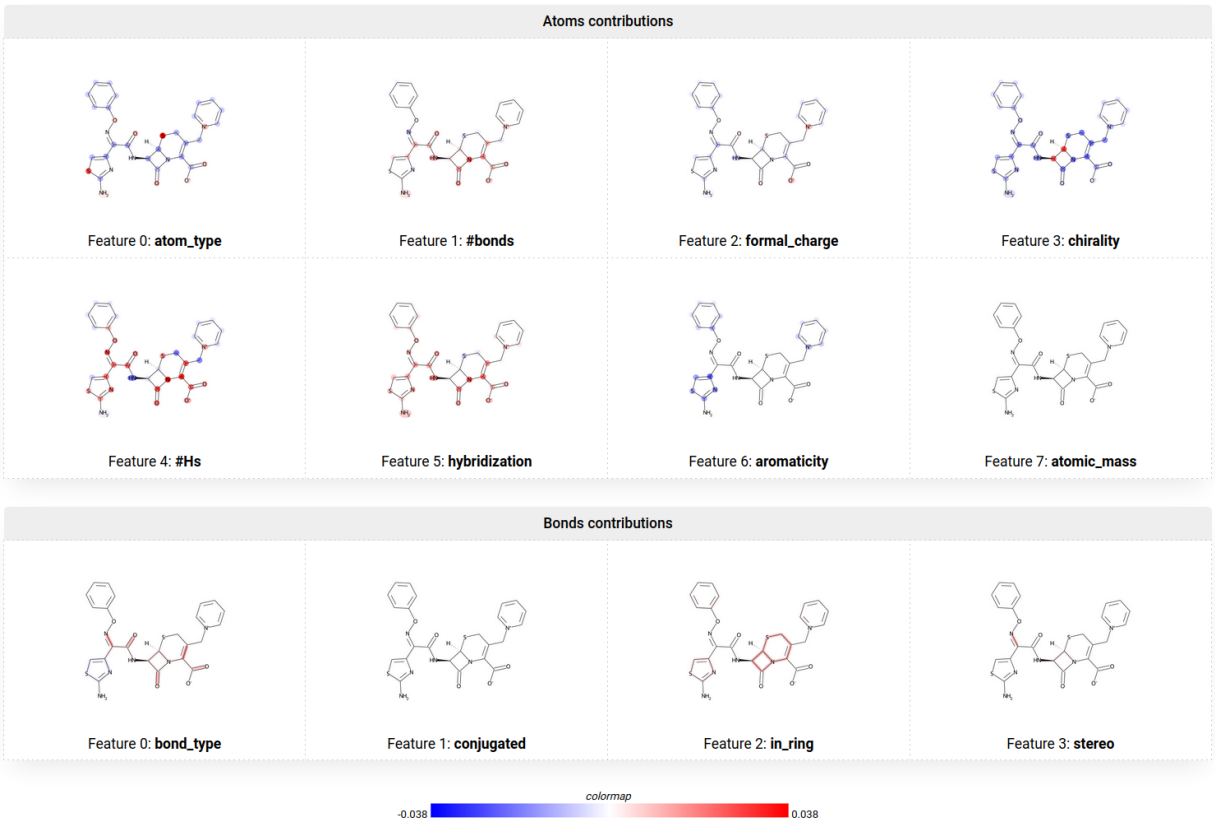


Figure C.6: Molecule 2

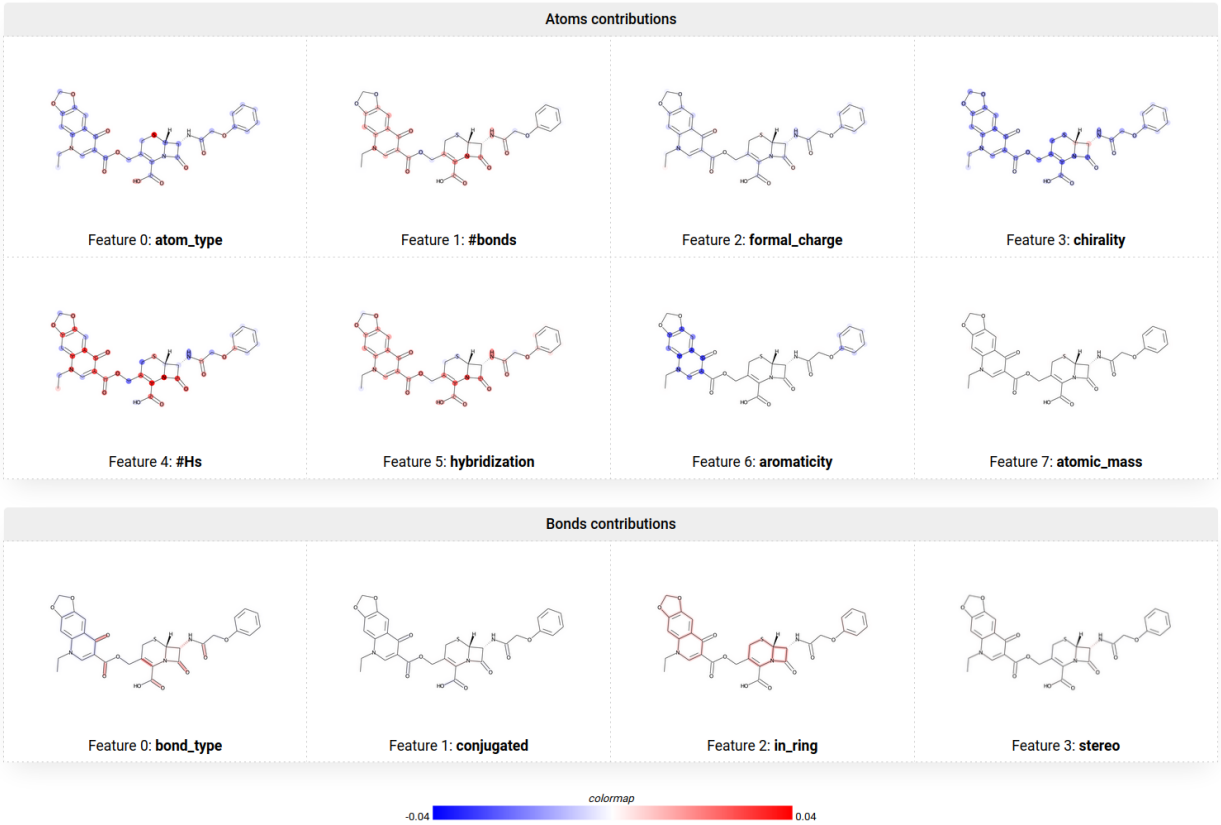


Figure C.7: Molecule 3

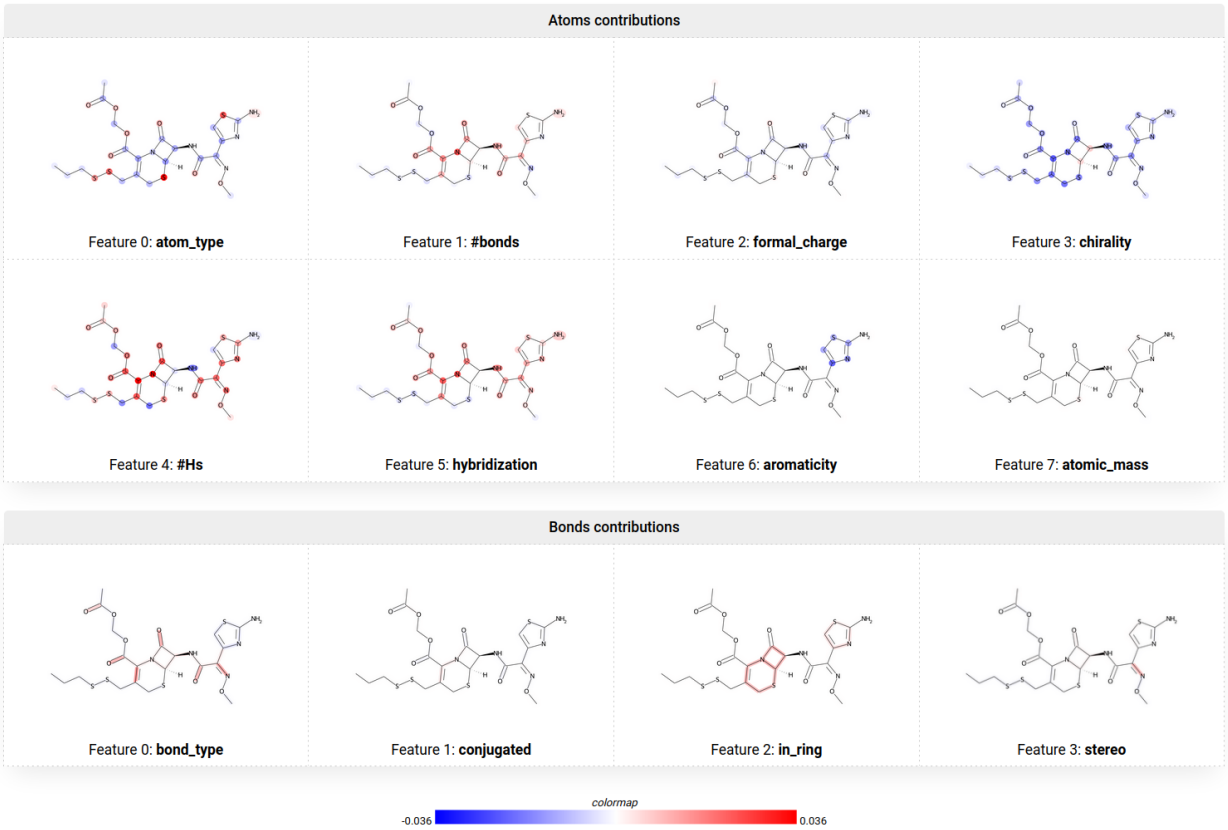


Figure C.8: Molecule 4

C.3 Feature interactions explained with PathExplain

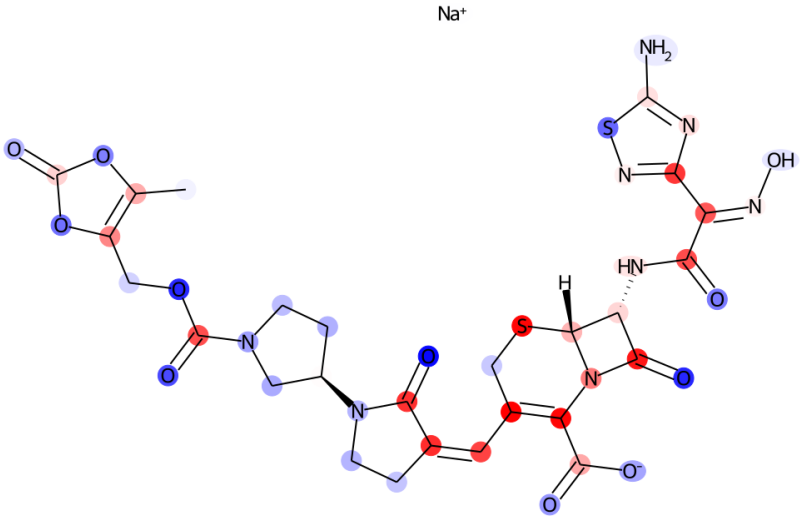


Figure C.9: Molecule 1

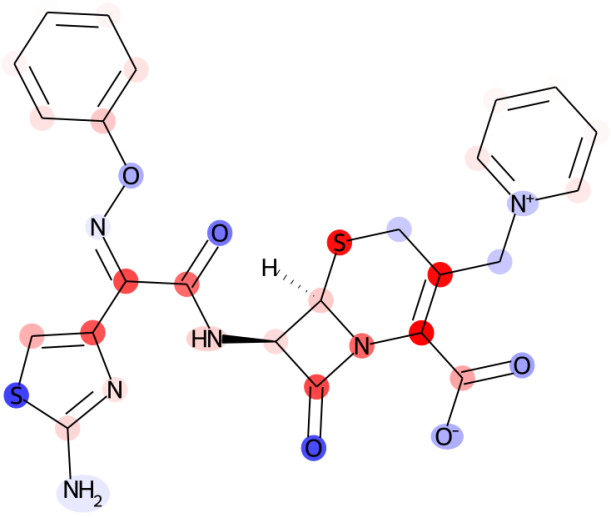


Figure C.10: Molecule 2

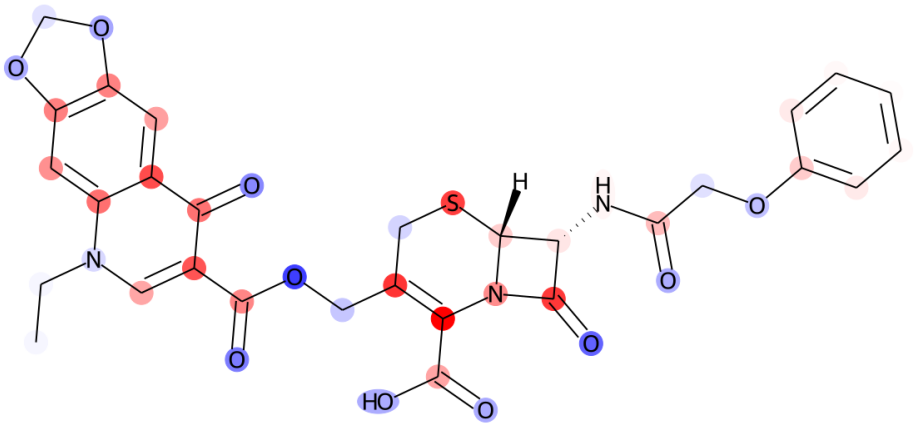


Figure C.11: Molecule 3

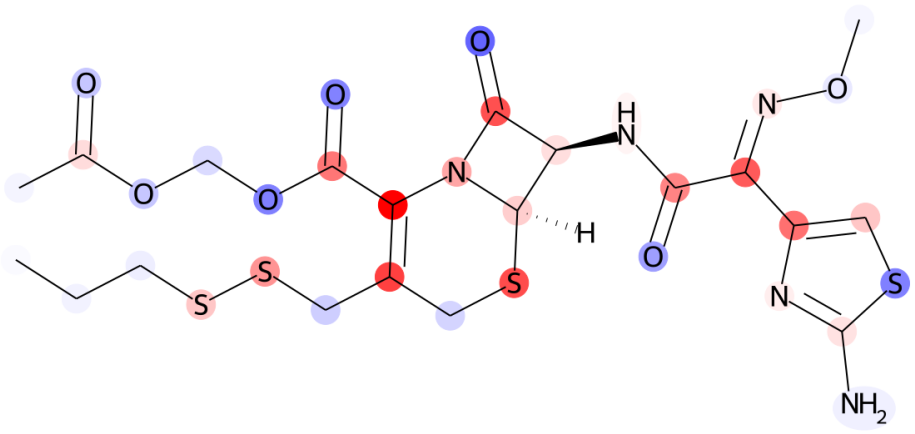


Figure C.12: Molecule 4

C.4 Web interface

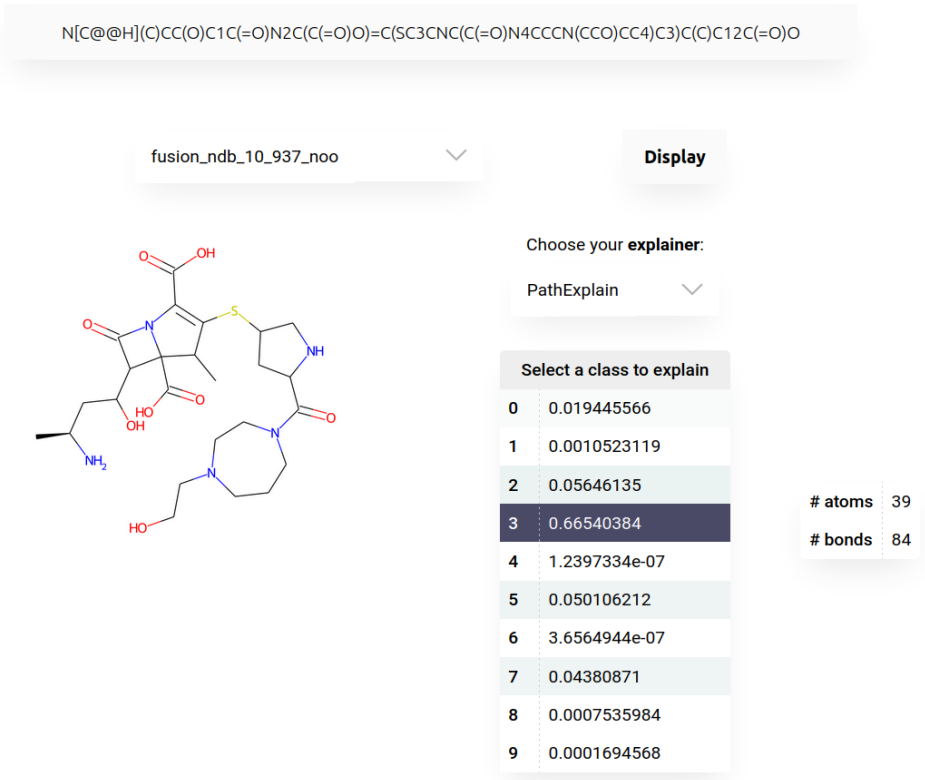


Figure C.13: Predictions of a multiclass model.

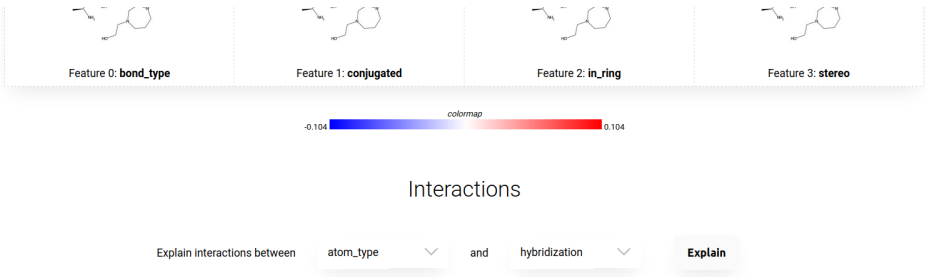


Figure C.14: Interface for feature interaction explanation.

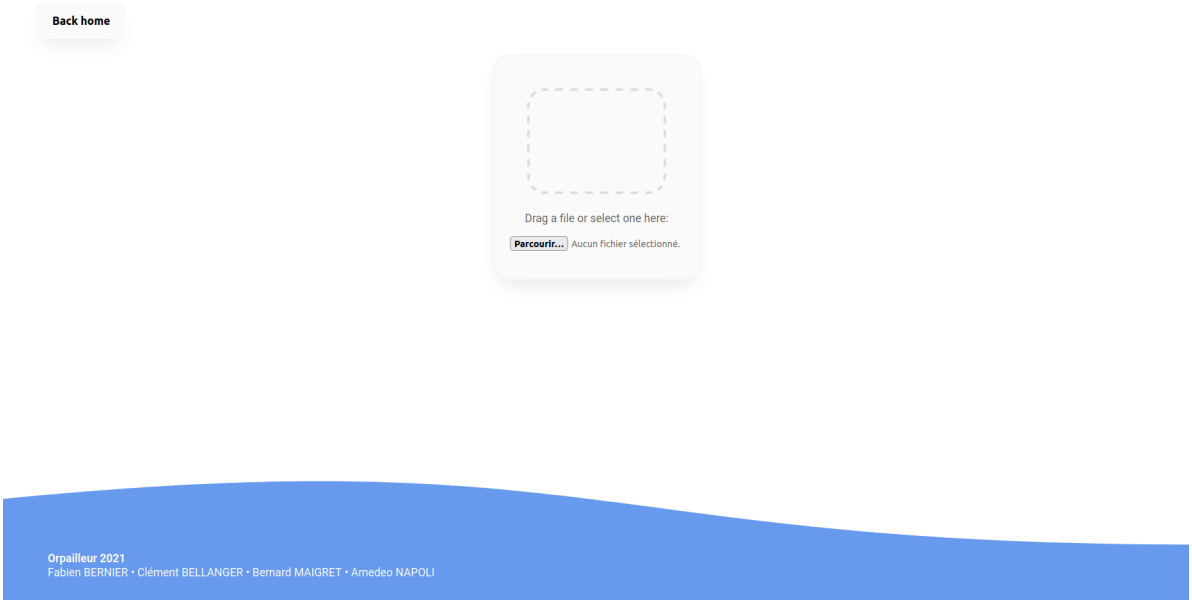


Figure C.15: Home page for the exploration of a dataset.

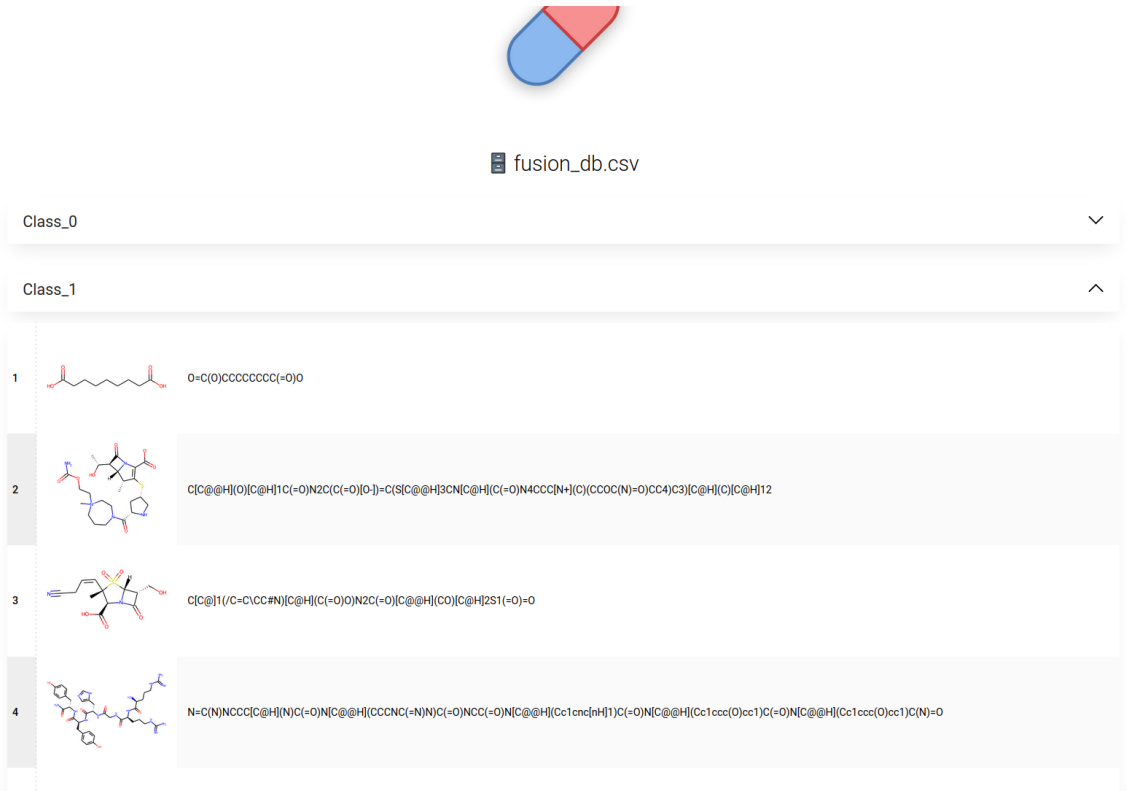


Figure C.16: Interface for a dataset exploration.

Résumé

Les progrès en machine learning, plus récemment accélérés par le *deep learning*, permettent désormais de construire des modèles performants dans des tâches de reconnaissance ou de génération. Ces modèles sont cependant complexes, si bien qu'il devient difficile de justifier leurs prédictions. Pour éviter un effet « *boîte noire* » et dans un soucis d'interprétabilité, l'équipe Orpailleur s'intéresse aux explicateurs de modèles, un domaine de recherche encore récent.

Ces explicateurs permettent, entre autres, de mettre en lumière les prédictions biaisées de certains modèles (exemple : décision principalement basée sur la couleur de peau). Les découvertes de l'équipe Orpailleur montrent qu'il est possible de rendre les prédictions plus *justes* (on parle généralement de « *fairness* ») grâce à des méthodes ensemblistes et de suppression de variables, sans altérer les performances des modèles. Nous nous intéressons plus particulièrement ici au cas spécifique des données textuelles, ainsi qu'à une adaptation aux modèles de *deep learning* qui occupent une place de premier choix dans le traitement automatique du langage.

Dans un second temps, nous utilisons ces explicateurs sur des modèles de classification d'antibiotiques afin de déterminer comment un tel classificateur parvient à la conclusion qu'une molécule possède des propriétés antibiotiques. Ces travaux conduiront à la création d'une interface web interactive permettant de mettre en valeur les motifs importants appris par ces modèles, au travers de visualisations.

Mots-clés : machine learning ; explicabilité ; fairness ; classification antibiotique

Abstract

Progress in machine learning, more recently accelerated by deep learning, now makes possible to build high-performance models for recognition or generation tasks. However, these models are complex, making it difficult to justify their predictions. To avoid a *"black box"* effect and for an interpretability reason, the Orpailleur team is interested in model explanations, a research field that is still recent.

These explainers manage to spot biased predictions of some models (e.g., decisions mainly based on skin color). The discoveries of the Orpailleur team show that it is possible to make predictions *fairer* (we generally talk about *fairness*) thanks to ensemble methods and variable dropping, without altering the models performance. We particularly focus here on the specific case of textual data, as well as on an adaptation to deep learning models which hold a prominent place in natural language processing.

Secondly, we will use these explainers on antibiotic classification models in order to determine how such a classifier reaches the conclusion that a molecule has antibiotic properties. This work will lead to the creation of an interactive web interface to highlight the important patterns learned by these models, through visualizations.

Keywords : machine learning ; explainability ; fairness ; antibiotic classification