ETH
Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Learning and Adaptive
Systems
Prof. Andreas Krause

Bachelor Thesis

Model Explainability using Fourier Sparsity

Autumn Term 2024

Supervised by:
Andreas Krause
Andisheh Amrollahi

Author:
Michele Meziu

Contents

Abstract

A central problem in machine learning is to provide explanations for the predictions of models. In particular we deal with the issue of identifying the input features that contribute the most to a prediction. Our work relies on extracting sparse Fourier representations of trained models. The Fourier representation provided insights into the model. Namely we utilize well know pseudo-Boolean optimization techniques to recognize the features that were most relevant for making a prediction.
The class of models we deal with in this work are gradient boosted trees. We train models to perform regression tasks on two different real world datasets and a classification task on a third dataset, then we discuss the model explanations obtained with our methods.

Chapter 1

Introduction

1.1 Motivation

Machine learning has seen widespread adoption in the last ten years. The applications vary widely and go from social sciences to physics. It has been used in critical settings like illness diagnosis from medical images and self-driving vehicles, where the decisions taken by the model have important consequences. In such contexts is of vital importance that the models make reliable decisions. However, only recently researchers have been trying to understand those models and the reasons behind their predictions.

In many settings decisions are taken with the supervision of a human practitioner that has knowledge of the field. If we provide the human experts with an explanation of why the model is making a certain decision, they will have more information for choosing whether to accept or reject the decision of the model. This is precisely the goal of this work: we want to understand which features of the input contribute the most to a decision of the model. The human decision maker can then judge, using his expertise, whether the provided features are a plausible justification to the decision of the model or not.

The Fourier transform is one of the most popular mathematical tools used in engineering and computer science. It transforms a signal, i.e. a function, from the *time domain* to the *frequency domain*, giving important insights about the composition of a signal.

In this work we extract the Fourier transform from models and use it to better understand the model and its predictions. Explaining the model using its Fourier representation m can be done easily when the model's input is discrete and m is sparse and has low degree. This class of models includes gradient boosted trees, which are the models we focus on in this thesis. However many more models could belong to this class, for example neural networks with discrete input. We leave this investigation to future studies.

1.2 Overview and related work

In the past few years, many methods have been developed to recover the Fourier transform of functions that are known to be sparse in the Fourier basis, such as in [1], [2], [3], [4]. Most of the literature on this topic makes use of methods from Compressive Sensing [5].

Looking at the Fourier transform of models has proven fruitful for many practical tasks, such as hyperparameter optimization for neural networks [6].

Suppose that for a certain input x the model, which we treat as a black box, predicts $b(x)$. In this work we try to use the frequency spectrum of gradient boosted trees to identify the most important features of x that contribute to the prediction $b(x)$. From the observations we make we can learn that, when certain features have specific values, the model classifies inputs as a 0 or 1 instance. This information can be used for at least two purposes.

- It can be used as an explanation of the decisions of a model, which can help practitioners in making decisions using the model. Determining which feature is responsible of a certain prediction is a problem known as *feature attribution* in the literature and has been studied extensively. Examples of feature attribution algorithms can be found in [7], [8], [9] and [10].

- It can be also used to construct an adversarial attack to a regressor or a classifier. For instance, in the case of a binary classifier with classes 0 and 1, we can change the value of the most important features in the input x and transform it to $\tilde{x}$. This will force the model to classify $\tilde{x}$ as a 0 or as a 1 instance, depending on which features we changed. If the model misclassifies $\tilde{x}$, then we can consider the perturbation we applied as a discrete adversarial attack. Analogous attacks have been successfully developed on text data, for example in [11], [12].

In this work we consider models as functions $b : \{0,1\}^n \mapsto \mathbb{R}$. Using the Robust Walsh Hadamard Transform algorithm introduced in [1] we can extract the c dominant Fourier coefficients of a b whose Fourier transform is approximately c sparse. In this way we can approximate a generic function with the extracted Fourier representation m:

$$b(x) \approx m(x) = \sum_{f \in \{0,1\}^n} \hat{b}(f) \cdot (-1)^{<f,x>}. \tag{1.1}$$

Because of the algorithm we are using there will be only c nonzero coefficients, so most of the $\hat{b}(x)$ will be equal to zero. In section 3 we will show how to rewrite equation 1.2 into a multivariate polynomial in $x_1, \ldots, x_n$, linear in each of those variables. For the models we trained for our datasets, the polynomial form of m is a low degree polynomial. The monomials are of the form $\prod x_{i_1} \cdot \ldots \cdot x_{i_d}$, with d small and constant. This allows us to solve optimization problems regarding m efficiently.
The main optimization problem we consider in this work is the following:

$$\min_{\epsilon \in \{0,1\}^n, ||\epsilon||_0 = k} m(x_0 + \epsilon), \tag{1.2}$$

where ϵ can be considered a sparse perturbation (with at most k nonzero elements) of some input x_0. Thus we can consider the above problem as finding the sparse perturbation of the input minimizing the value of a prediction.
To solve problem 1.2 we consider two approaches.

- Linear programming: The first one is based on integer linear programming (ILP). We need to rewrite the Fourier representation m from as a linear function and to add appropriate constraints. This is explained in section 3. Then we use the *gurobi* software package to solve the linear program. This method provides an exact solution to problem 1.2.

- pseudo-Boolean optimization: The second one is using techniques from pseudo-Boolean optimization. We rewrite the Fourier representation m from as a multivariate polynomial and we remove terms with degree larger than 4. Deleting higher order terms is required by the pseudo-Boolean optimization

technique we are using. In particular a branch and bound algorithm from [13]. This method provides an approximate solution to problem 1.2, but runs significantly faster.

We solve the optimization problem 1.2 on models trained on various datasets, for many different inputs x. Our claim, which we discuss in section 4, is that the k features changed more often by the sparse perturbation are the k most relevant features for the model predictions. This is a novel systematic approach to obtain explanations of the behaviour of a model.

Chapter 2

Preliminaries

In this section we introduce basic notions that will be used throughout the work.

2.1 Fourier transform of pseudo-Boolean functions

This introduction to the Fourier transform of pseudo-Boolean functions is mainly inspired by [14].
In this thesis we work exclusively with pseudo-Boolean functions, that is, functions of the type $g : \{0,1\}^n \mapsto \mathbb{R}$. For a given n, all the functions of this kind form a vector space. Those functions can be summed with each other and multiplied with scalars from the field $\mathbb{R}$. This vector space has 2^n dimensions, one for each element of the domain $\{0,1\}^n$ of the functions. It also happens to have an orthonormal basis, given by the functions χ_t, defined as follows:

$$\chi_t(x) = (-1)^{<x,f>}. \tag{2.1}$$

These basis functions are parametrized by f, which can be any of the vectors in $\{0,1\}^n$ and is called a *frequency*. There are exactly 2^n such functions, which agrees with the dimension of the vector space.
We would like to write a given pseudo-Boolean function g as a weighted sum of basis functions, as follows:

$$g(x) = \sum_{f \in \{0,1\}^n} \hat{g}(f) \cdot \chi_f(x) \qquad \forall x \in \{0,1\}^n. \tag{2.2}$$

The coefficients $\hat{g}(f)$ are called *Fourier coefficients*. $\hat{g}$ can be viewed as a function $\hat{g} : \{0,1\}^n \mapsto \mathbb{R}$, and is called the *Fourier transform* of g. The representation in 2.2 is called the *Fourier representation* of the function g.
It is very convenient when functions can be written easily as a summation of basis functions with few nonzero coefficients. Consider the parity function $p(x) = \sum x_i \mod 2$, that is, the number of ones in a binary vector modulo 2. We can write this function as

$$p(x) = \hat{p}(\mathbf{0}) \cdot \chi_{\mathbf{0}}(x) + \hat{p}(\mathbf{1}) \cdot \chi_{\mathbf{1}}(x) = \frac{1}{2} \cdot 1 + (-\frac{1}{2}) \cdot (-1)^{\sum x_i}. \tag{2.3}$$

$\mathbf{0}$ is the zero vector and $\mathbf{1}$ is the vector where all the entries are one. By substituting in the basis function $\chi_f(x) = (-1)^{<x,f>}$ the vectors $\mathbf{0}$ and $\mathbf{1}$ in place of f, we obtain $\chi_{\mathbf{0}}$ and $\chi_{\mathbf{1}}$
Equation 2.3 holds because if we have $\sum x_i = 0 \mod 2$, then we get $\frac{1}{2} - \frac{1}{2} = 0$. Otherwise, if we have $\sum x_i = 1 \mod 2$ then we get $\frac{1}{2} + \frac{1}{2} = 1$.

Here we have a Fourier representation with only two nonzero coefficients: $\hat{p}(\mathbf{0})$ and $\hat{p}(\mathbf{1})$. We can say that the Fourier representation of $p(x)$ is *sparse*. The frequencies for which the Fourier coefficients are nonzero form the *spectrum* of a function.
The *degree* of a frequency f is the number of entries equal to one. The degree of a function, is the maximum degree of the frequencies in its spectrum. The degree of the parity function is n, since $\mathbf{1}$ is a vector with n ones.
Given a pseudo-Boolean function g of the form 2.2, we define as *energy* of g the quantity $\sum_{f\in\{0,1\}^n} \hat{g}(f)^2$.
Analogously, we can define the total energy of a set S of frequencies as $\sum_{f\in S} \hat{g}(f)^2$.
We define the *degree d energy* as the value obtained when $S := \{f \in \{0,1\}^n | \deg(f) = d\}$, that is we take only frequencies with degree exactly d. We can define also the *energy of feature i* as the value obtained when $S := \{f \in \{0,1\}^n | f_i = 1\}$, that is we take only frequencies where the entry with index i is set to 1.

Chapter 3

Methods

In this chapter we discuss the techniques we used to solve the optimization problem 1.2, which we rewrite here for convenience

$$\min_{\epsilon \in \{0,1\}^n, ||\epsilon||_0 = k} m(x_0 + \epsilon), \tag{1.2}$$

3.1 Linear programming

Using the Robust Walsh Hadamard Transform algorithm from [1], we extract the approximate Fourier representation m of a black box function b. This is in general of the form

$$m(x) = \sum_{f \in \{0,1\}^n} \hat{b}(f) \cdot (-1)^{<f,x>}. \tag{3.1}$$

$\hat{b}(x)$ is the approximate Fourier transform of b and evaluates to a nonzero value only c times, since we extracted the c dominant Fourier coefficients.
Our goal is to rewrite m as a linear function, so that we can use it as the objective of an ILP. We have used two different approaches to rewrite m depending on the auxiliary variables we use. Assume frequency f has degree d and $f_{i_1} \dots f_{i_d}$ are the nonzero entries of f. In the first approach, the *and basis* form of m, we use variables of the kind $z = (f_{i_1} \wedge \dots \wedge f_{i_d})$ as auxiliary variables. In the second one, the *xor basis* form of m, we use $z = (f_{i_1} \oplus \dots \oplus f_{i_d})$. These approaches are not only useful for problem 1.2, but they can also be used by anyone who wants to solve an optimization problem involving any pseudo-Boolean function.

3.1.1 And basis

We show how to rewrite the term $\hat{b}(t) \cdot (-1)^{<f,x>}$ of the sum in 3.1 into a linear polynomial, again assuming that f has degree d and that $f_{i_1} \dots f_{i_d}$ are the nonzero entries:

$$\begin{aligned}
\hat{b}(t) \cdot (-1)^{<t,x>} &\overset{(*)}{=} \hat{b}(t) \cdot (-1)^{x_{i_1} + \dots + x_{i_d}} = \hat{b}(t) \cdot (-1)^{x_{i_1}} \cdot \dots \cdot (-1)^{x_{i_d}} \overset{(**)}{=} \\
&= \hat{b}(t) \cdot (1 - 2x_{i_1}) \cdot \dots \cdot (1 - 2x_{i_d}) \overset{(***)}{=} \hat{b}(t) \cdot \sum_{S \subseteq \{i_1 \dots i_d\}} (-2)^{|S|} \cdot \prod_{j \in S} x_j = \\
&\qquad \hat{b}(t) \cdot \sum_{S \subseteq \{i_1 \dots i_d\}} (-2)^{|S|} \cdot z_S.
\end{aligned} \tag{3.2}$$

Equality (*) holds by computing the scalar product and omitting all the terms of the form $f_j x_j$ with $j \neq i_1, \dots i_d$, since they are zero.
Equality (**) makes use of the fact that we have binary variables, for which $(-1)^x = 1 - 2x$ holds. In fact, if $x = 0$ then $(-1)^0 = 1 - 2 \cdot 0 = 1$. Otherwise if $x = 1$ then $(-1)^1 = 1 - 2 \cdot 1 = -1$.
Equality (***) holds because we will get one monomial for each choice of a subset S of the indices of the variables. Such a monomial is given by the numerical coefficient (which comes from multiplying $|S|$ times -2) and the product of the chosen variables.
In the last equality we just introduce an additional variable z_S for the product of the variables with indices in S. Since the variables are binary, and therefore the product is the same as the logical and, it holds that $z_S = \bigwedge_{j \in S} x_j$.
One term gets rewritten to a sum of terms that are exponentially many in d. This is not an issue in our work, because d is up to 10 in our experiments. Moreover, when summing over all the possible f, similar monomials will be added together resulting in an acceptable number of terms.

3.1.2 Xor basis

We show how to rewrite a term from the sum in 3.1 to a linear term, using the second technique. Let z be defined as $z = x_{i_1} \oplus \dots \oplus x_{i_d}$.

$$\hat{b}(f) \cdot (-1)^{<f,x>} = \hat{b}(f) \cdot (-1)^{x_{i_1} + \dots + x_{i_d}} \overset{(****)}{=} \hat{b}(f) \cdot (-1)^{x_{i_1} \oplus \dots \oplus x_{i_d}} = \\ = \hat{b}(f) \cdot (-1)^z = \hat{b}(f) \cdot (1 - 2z) \quad (3.3)$$

The equality marked with (****) holds because what matters is the value modulo 2 of the sum in the exponent of (-1). Therefore we can substitute it with a xor, which is zero if there is an even number of ones, otherwise it is one. In this case one term of the initial sum gets converted into one degree zero and one degree one monomial.

3.1.3 Simple constraints

In our integer linear program we must encode constraints like $z = x_{i_1} \wedge \dots \wedge x_{i_d}$ or constraints like $z = x_{i_1} \oplus \dots \oplus x_{i_d}$. In this section we try to do this using few constraints. We always assume that all the variables are binary zero-one variables, unless otherwise stated. So for example the constraints $z \geq 0$, $z \leq 1$ and $z \in \mathbb{N}$ hold implicitly.

Simple and constraints

For each of the variables on the right hand side of $z = x_{i_1} \wedge \dots \wedge x_{i_d}$ we encode the fact that if one of them is zero then also z is zero. This gives us

$$z \leq x_{i_1} \quad (3.4)$$
$$\vdots$$
$$z \leq x_{i_d}. \quad (3.5)$$

Since z is binary, it has to be zero if one variables on the right hand side is zero. Now we have to encode the fact that if all the x_i are one then also z must be one. We do this as follows

$$z \geq x_{i_1} \dots x_{i_d} - (d-1). \tag{3.6}$$

We notice that when all the x_i are one, then we get $z \geq d - (d-1) = 1$. Since z is binary it implies $z = 1$. We notice that if z and $x_{i_1} \dots x_{i_d}$ are binary, then the conditions we mentioned are not only sufficient but also necessary. This means we didn't put any unnecessary constraint that could exclude the optimal solution from the feasible solutions.

Simple xor constraints

For the xor constraint we try to reproduce the modulo 2 operation with linear constraints. We just need to subtract to $z = x_{i_1} + \ldots + x_{i_d}$ the largest smaller or equal even number. This is encoded with

$$z = x_{i_1} + \ldots + x_{i_d} - 2s, \qquad s \in \mathbb{N}. \tag{3.7}$$

We don't need to put more constraints on s because z must be binary, and this gives us already the right s. Since z and $x_{i_1} \dots x_{i_d}$ are binary the above condition is necessary and sufficient.

3.1.4 Recursive constraints

We found out with experiments that adding redundant constraints to the ILP improved the speed of the solver. For this reason, we have tried to encode the logical and and the logical xor recursively. This gives an ILP with more constraints and also more variables. Notice that we are not adding extra constraints that are not needed to encode z as a logical and or logical xor of variables. That is, we are not narrowing feasible region.

Recursive and constraints

To encode the and constraint recursively we use $d-2$ additional variables. The additional variable t_j is defined as $x_{i_j} \wedge \ldots \wedge x_{i_d}$, as shown in equation 3.8.

$$z = x_{i_1} \wedge x_{i_2} \wedge \underbrace{x_{i_3} \wedge \ldots \wedge \underbrace{x_{i_{d-1}} \wedge x_{i_d}}_{t_{d-1}}}_{t_3}}_{t_2}. \tag{3.8}$$

We write the constraints considering the logical and of two variables at a time, in the following way

$$z = x_{i_1} \wedge t_2 \tag{3.9}$$
$$t_2 = x_{i_2} \wedge t_3 \tag{3.10}$$
$$t_3 = x_{i_3} \wedge t_4 \tag{3.11}$$
$$\vdots$$
$$t_{d-1} = x_{i_{d-1}} \wedge x_{i_d}. \tag{3.12}$$

Each of the $d-1$ constraints above consists actually of 3 simple constraints written as explained in the previous section. For example for $t_2 = x_{i_2} \wedge t_3$ we would be

$$t_2 \leq x_{i_2} \tag{3.13}$$
$$t_2 \leq t_3 \tag{3.14}$$
$$t_2 \geq x_{i_2} + t_3 - 1. \tag{3.15}$$

Recursive xor constraints

We define $t_2, \ldots, t_{d-1}$ similarly as before.

$$z = x_{i_1} \oplus \underbrace{x_{i_2} \oplus \underbrace{x_{i_3} \oplus \ldots \oplus \underbrace{x_{i_{d-1}} \oplus x_{i_d}}_{t_{d-1}}}_{t_3}}_{t_2}. \tag{3.16}$$

The above constraint is also written considering two variables at a time like before

$$z = x_{i_1} \oplus t_2 \tag{3.17}$$
$$t_2 = x_{i_2} \oplus t_3 \tag{3.18}$$
$$t_3 = x_{i_3} \oplus t_4 \tag{3.19}$$
$$\vdots$$
$$t_{d-1} = x_{i_{d-1}} \oplus x_{i_d}. \tag{3.20}$$

There is a simpler way to encode the xor of two variables that doesn't require to introduce the auxiliary variable $t \in \mathbb{N}$. Now we will write the constraints for equalities such as $t_2 = x_{i_2} \oplus t_3$ as follows

$$t_2 \leq 2 - x_{i_2} - t_3 \tag{3.21}$$
$$t_2 \leq x_{i_2} + t_3 \tag{3.22}$$
$$t_2 \geq x_{i_2} - t_3 \tag{3.23}$$
$$t_2 \geq t_3 - x_{i_2}. \tag{3.24}$$

(3.22) encodes that if x_{i_2} and t_3 are both one then t_2 should be equal to zero.
(3.23) encodes that if x_{i_2} and t_3 are both zero then also t_2 should be zero.
(3.24) and (3.35) encode that if the variable on the right hand side with the positive coefficient is one and the other one is zero then t_2 should be equal to one.

3.2 Pseudo-Boolean optimization

Another method we used to solve the optimization problem from 1.2 is pseudo-Boolean optimization, which deals with minimizing or maximizing pseudo-Boolean functions. The function m from our minimization problem (which we have written below for convenience) is exactly of that kind

$$\min_{\epsilon \in \{0,1\}^n, ||\epsilon||_0 = k} m(x_0 + \epsilon). \tag{1.2}$$

However, in this second approach we are unable to enforce that $||\epsilon||_0 = k$. This is because our algorithm takes just a polynomial as an objective and does not support constraints. We have tried to enforce sparsity by adding an L_0 regularizer, in a

similar way to the lasso optimization problem. Therefore the optimization problem we solve is the following

$$\min_{\epsilon \in \{0,1\}^n} m(x_0 + \epsilon) + \alpha ||\epsilon||_0. \tag{3.25}$$

The state of the art algorithms for pseudo-Boolean optimization can be found in [13]. In this thesis we use generalized roof duality with generators, or GRD-gen, also from [13]. This algorithm works for pseudo-Boolean functions m of degree up to 4. It provides a lower bound for the minimum of m and provides a partial solution. That is, GRD-gen will determine some of the entries of the minimizer x^* of m. The number of entries of x^* for which a value will be found is unknown in advance and depends on the particular function m.
This algorithm is used as a subroutine of a branch and bound algorithm implemented in the code of [13], which we also used for our experiments besides the *gurobi* optimizer.
The branch and bound algorithm finds an exact solution to the problem of minimizing m. It keeps an upper bound u of the minimum $m(x^*)$ during the whole run. The algorithm proceeds by exploring all the $\{0,1\}^n$ space and pruning a branch if the lower bound of the minimum provided by GRD-gen for that branch is larger than u.
We applied two little modifications to their code to make it more suitable to our needs

- **Strong branching**: We precompute the energies of each of the variables (which correspond to features) in f. The energy of a feature is defined in 2.1. When the algorithm needs to branch, it will branch on the variable with the largest energy that hasn't been fixed yet.

- **Thresholding**: In our work we are looking for low degree solutions. Therefore, as soon as in a certain branch the number of ones in the solution vector exceeds a threshold h, the branch will be discarded. In all of our experiments we have used $h = 10$.

Chapter 4

Experiments

In this section we show plots regarding various experiments we have made. The plots showing the feature energies and the plots about feature attribution with perturbations are the ones relevant to the topic of explainability. The other plots are needed to justify the soundness of our approach. More details about the experiments and the plots will be given.

4.1 Setup

We briefly provide information about the datasets we used, which come from the UCI Machine Learning Repository.

- **Crimes** [15]: The features are socio-economic data regarding communities within the United States. For a certain community the model has to predict the number of violent crimes per 100.000 inhabitants.

- **Superconductors** [16]: Physical quantities regarding the atoms that constitute a superconductive material have been measured. The features are values describing the statistical distribution of those quantities. The task is to predict the critical temperature of the material, i.e. the temperature under which the material behaves as a superconductor.

- **Mushrooms**: The features are the physical characteristics of mushrooms. The task is to classify them in poisonous or edible.

We transformed the original features in binary features to make the input discrete. For each numerical feature we have 32 quantiles, the lowest quantile is coded by 00000 and the highest by 11111. One numerical feature in the original vector gets mapped to 5 binary features that code the quantile in which the original value lies. For all the datasets we extracted the approximate Fourier representation with the Robust Walsh Hadamard Transform algorithm from [1]. For the *crimes* and *superconductors* datasets we always set $c = 1024$ as number of nonzero Fourier coefficients, except when explicitely stated. For the *mushrooms* dataset we used $c = 164$.
Most of our experiments involve finding a sparse perturbation $\tilde{x}$ of an input x. To find such perturbations we use linear programs written as explained in 3.1 and the pseudo-Boolean optimization algorithm described in 3.2. To solve the linear programs we have used the *gurobi* optimization library.

4.2 Outline

Let b be the the black box and m its approximated Fourier representation.
For each dataset we will show and discuss the following plots:

- **Goodness of Fourier approximation**: we generate 1000 random binary vectors x and for each of them we compute the error $m(x) - b(x)$. We plot in a histogram the value of the error against the number of times we got that much error. This is to show that the approximated Fourier representation we recovered is close to the real one, so that we can find a good perturbation.

- **Effectiveness of the perturbation**: we iterate over all vectors x_i in the test set. We fix a k which is the degree of the perturbation we apply. For each of the x_i, we get a perturbed $\tilde{x_i}$. We compare the change in the output of the black box b with the change in the output of the Fourier representation m. That is, we plot $(b(x_i) - b\tilde{x_i}))^2$ against $(m(x_i) - m(\tilde{x_i}))^2$ in a scatter chart.

  We would like to obtain a plot that has most of the points near the main diagonal, which means that the squared error measured on m is a good approximation of the squared error on b. This would imply that m is good approximation of b.

  Ideally, the points should be as far as possible from the origin, meaning that the squared errors are large and that $\tilde{x}$ leads to completely different predictions.

- **Degree energy**: we want to know how large are the coefficients of the terms of degree d in m. For every degree d, we sum the squares of the coefficients of terms of degree d, as explained in section 2.1. This will give the energy of that degree in m, which gives us an idea about how much terms of degree d contribute to the prediction of the black box. We plot those energies in a histogram where on the x axis we have the degree and on the y axis the energy. If we see that the terms with $d > 4$ are negligible this will justify our approach for strong branching described in section 3.

- **Feature energy**: we want to know how much a certain feature i contributes to the prediction of the black box. For each feature we sum the squares of the coefficients of terms containing that feature. This will give us the energy of that feature in m as explained in section 2.1. This information is represented in a histogram where on the x axis we have the index of the feature and on the y axis its corresponding energy.

- **Feature attribution with perturbations**: we perturb multiple inputs to understand which feature will be changed most times. This gives an idea of the importance of that feature for the prediction. We consider perturbations of a fixed degree k performed with *gurobi* and perturbations from lasso problems solved with the branch and bound algorithm. We apply a perturbation on each of the vectors in the test set and for each of them we report which features have been perturbed. We represent this data in a histogram, where on the x axis we have the index of the feature and on the y axis the number of times it has been perturbed.

4.3 Crimes dataset

From 4.1 we see that the extracted Fourier transform m approximates well the black box function b, since for no one of the randomly generated vectors x the

error $m(x) - b(x)$ is larger than 0.3 in absolute value. The mean squared error $\frac{1}{1000}\sum(m(x) - b(x))^2$ is 0.0065. The R^2 score is 0.387. It also doesn't seem to be a biased approximation, since roughly half of the values are overestimated and half underestimated. More precisely, for 48% of the vectors $m(x) - b(x) > 0$.

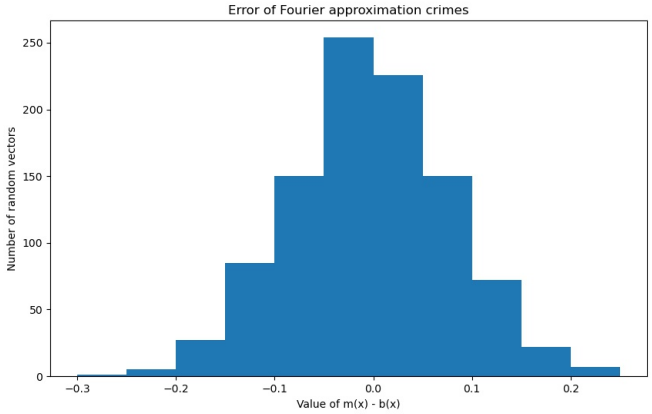

Figure 4.1: Value of the error on the x axis and the number of random vectors on the y axis. Bins of size 0.05

In 4.2 most of the points lie close to the $y = x$ line. More precisely, the $y = x$ line approximates the points with a R^2 score of 0.79. This suggests that our m is good. However we notice that many of the points lie near to the origin, that is, $m(x) \approx m(\tilde{x})$ and $b(x) \approx b(\tilde{x})$. This means that after the perturbation the output of neither m nor b changes significantly. In 4.2 we see near the origin a cluster of aligned points parallel to the x axis that have approximately $y = 0$. This is an issue because it means an effective perturbation on the Fourier approximation is not effective in changing the black box prediction, which remains about the same.

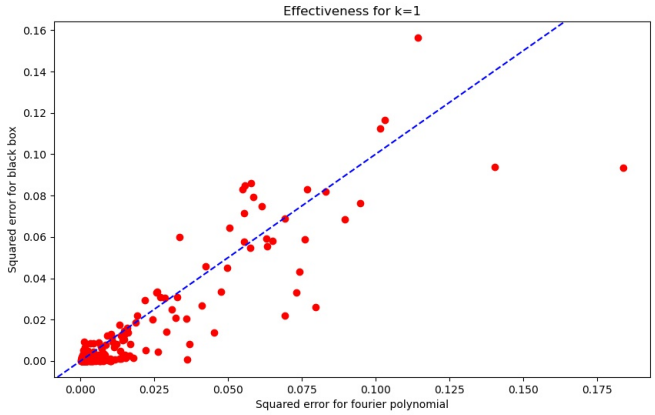

Figure 4.2: $(m(x) - m(\tilde{x}))^2$ on the x axis and $(b(x) - b(\tilde{x}))^2$ on the y axis. $\tilde{x}$ results from a perturbation with $k = 1$.

In 4.3 we see that all the points are lying below the $y = x$ diagonal and near the x axis, which means that the squared error for the Fourier approximation m is much larger compared to the one for the black box function b.

With $k = 2$ the squared error for m is larger, probably because we are choosing $\tilde{x}$ from a larger space, so we can find more easily inputs for which m is a bad approximation of b. That is $\tilde{x}$ is not such an effective perturbation as the large value of $(m(x) - m(\tilde{x}))^2$ makes it seem.

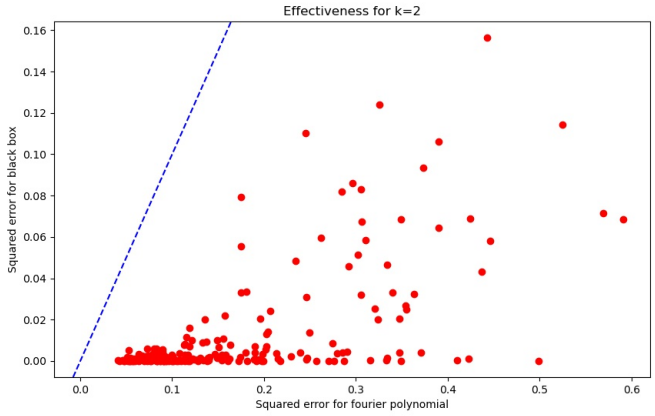

Figure 4.3: $(m(x) - m(\tilde{x}))^2$ on the x axis and $(b(x) - b(\tilde{x}))^2$ on the y axis. $\tilde{x}$ results from a perturbation with $k = 2$.

The recovered Fourier approximation for the crimes dataset is a degree 10 polynomial. In 4.4 we see that low degree terms have most of the energy and they give a good approximation of m. This justifies the approach used for the branch and bound algorithm, where we consider only frequencies with degree $d \leq 4$ and omit all the others. The constant term had by far the highest energy but we omitted it in the plot for clarity.

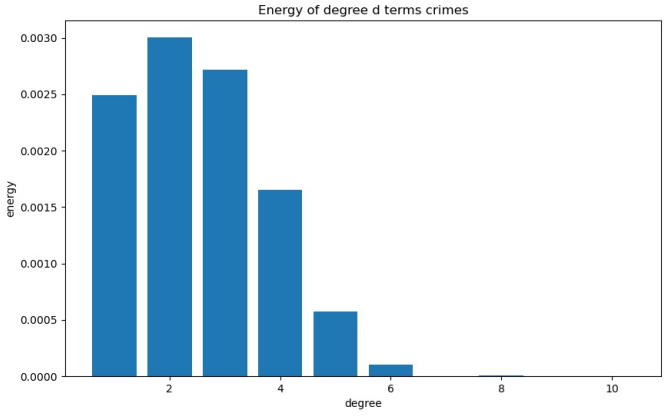

Figure 4.4: Overall energy of the terms of degree d

In 4.5 we have plotted the *energy* of feature i, which is obtained by summing the squares of all the coefficients of frequencies containing i. We conjecture that this will tell us how much does a certain feature influence predictions.
Plot 4.6 is generated by applying a $k = 1$ perturbation on each vector x_i in the test set of our gradient boosted tree model. Such a perturbation flips just one bit of the input. For each feature we keep track how many times it has been perturbed.

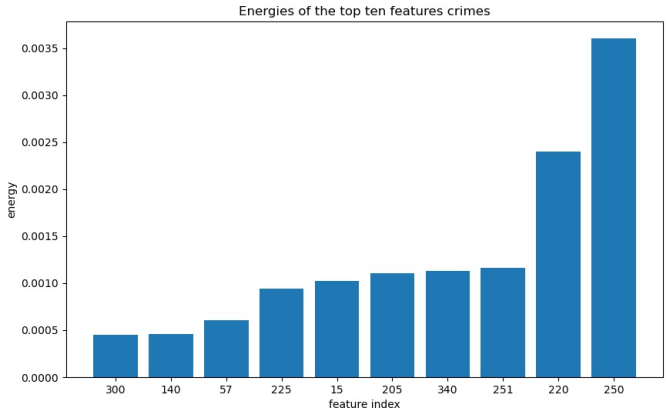

Figure 4.5: Overall energy of the terms containing feature number i. Only the top ten features are shown

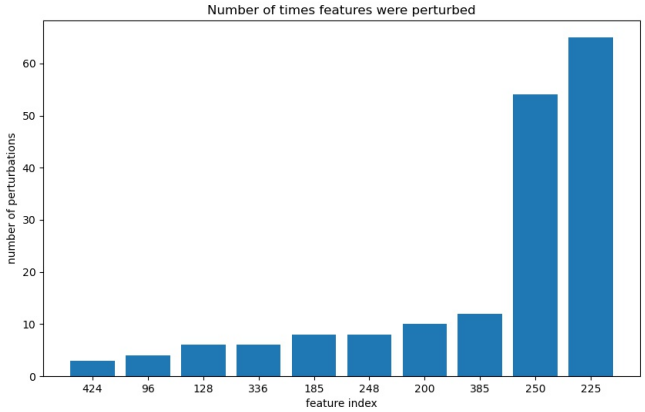

Figure 4.6: How many times feature i has been modified in a $k = 1$ perturbation. Only the top ten features are shown

In a similar way we generated the plot 4.7 but we applied perturbations with $k = 2$.

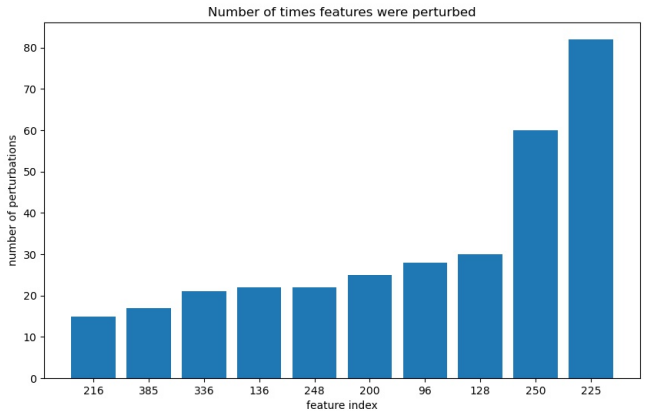

Figure 4.7: How many times feature i has been modified in a $k = 2$ perturbation. Only the top ten features are shown

Finally we include the histogram for the perturbations found with the branch and bound technique, where we have chosen $\alpha = 0.025$ as regularization coefficient for the lasso. Here the number of features perturbed for a given input varies.

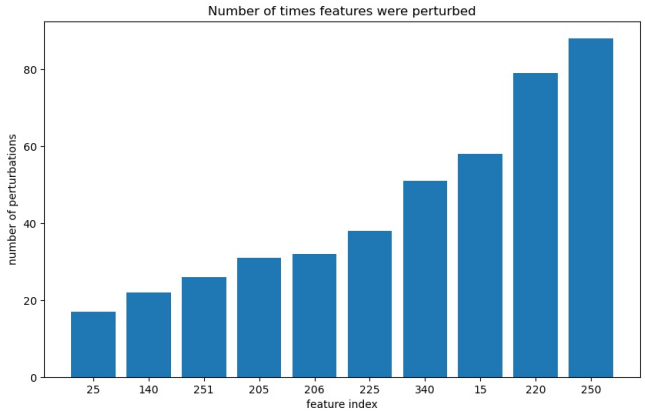

Figure 4.8: How many times feature i has been modified in the perturbation resulting from a lasso with regularization coefficient $\alpha = 0.025$. Only the top ten features are shown

We notice that some features are in common among the plots 4.5, 4.6, 4.7, 4.8, like 250, 225. Those seem to be the most important features for this dataset. In table 4.3 we map feature numbers to the value they correspond to. From the table we see that 225 corresponds to percent of kids 4 and under in two parent households and 250 corresponds to the percentage of kids born to never married.
Because of the way we binarized the numerical features, we would expect always the most significant bits to be perturbed, which in our case are the ones with index multiple of 5. Also we would expect the second most significant digit (with index equal to 1 modulo 5) to be perturbed only if also the most significant digit has already been perturbed.

This is the case in 4.8, where almost all the perturbed features have an index which is a multiple of 5, except from 251, which appears because 250 had already been perturbed.

Index	Feature meaning
15	percentage of population that is caucasian
25	percentage of population that is of hispanic heritage
57	percentage of people living in areas classified as urban
96	median family income
128	per capita income for people with 'other' heritage
136	number of people under the poverty level
140	percentage of people under the poverty level
185	percentage of people 16 and over employed in management or professional occupations
200	percentage of females who are divorced
205	percentage of population who are divorced
206	percentage of population who are divorced
216	percentage of families (with kids) that are headed by two parents
220	percentage of kids in family housing with two parents
225	percent of kids 4 and under in two parent households
248	number of kids born to never married
250	percentage of kids born to never married
251	percentage of kids born to never married
300	percent of people who speak only English
336	percent of people in owner occupied households
340	percent of persons in dense housing (more than 1 person per room)
385	percent of occupied housing units without phone
424	rental housing - upper quartile rent

4.4 Superconductors

The extracted Fourier representation for the superconductors dataset gives a mean squared error of 0.0853 and the percentage of overestimated values is 48.5%. The R^2 score for the approximation is 0.666.
For $k = 1$ on the superconductors dataset the line $y = x$ fits the data with an R^2 coefficient of -1.129. Some of the points lie near the $y = x$ line and far from the origin.
This means they have with a large squared error both for the perturbation on the Fourier approximation and on the black box.
However we see a large cluster parallel to the x axis, for which the y coordinate is small. Those points correspond to inputs for which the Fourier approximation doesn't give a good local approximation of the black box.
With $k = 2$ we get a plot in which all the points lie under the $y = x$ line, as it was the case for the crimes dataset. We suspect that, also in this case, it is because the attack is chosen from a larger space. This makes it easier to find a $\tilde{x}$ for which m fails to approximate the black box b.
For this dataset the extracted Fourier approximation has degree 6. From plot 4.12 we see that terms of degrees 5 and 6 have little energy. The predictions of the black box seem to be determined by the terms with degree less or equal than four. This justifies our use of the optimization techniques from section 3.2.
Comparing figure 4.13 and 4.14, we see that most of the features are in common between the two plots. Namely 256, 288, 24, 296, 304, 268.

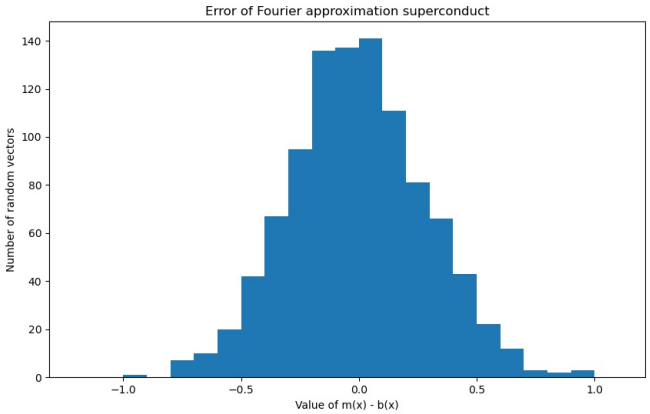

Figure 4.9: Value of the error on the x axis and the number of random vectors on the y axis. Bin size of 0.1.

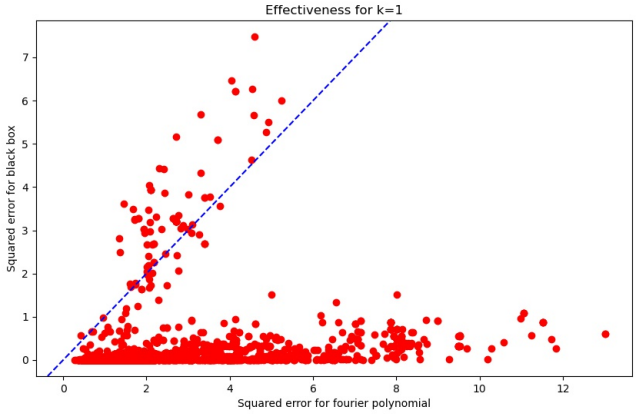

Figure 4.10: $(m(x)-m(\tilde{x}))^2$ on the x axis and $(b(x)-b(\tilde{x}))^2$ on the y axis. $\tilde{x}$ results from a perturbation with $k = 1$.

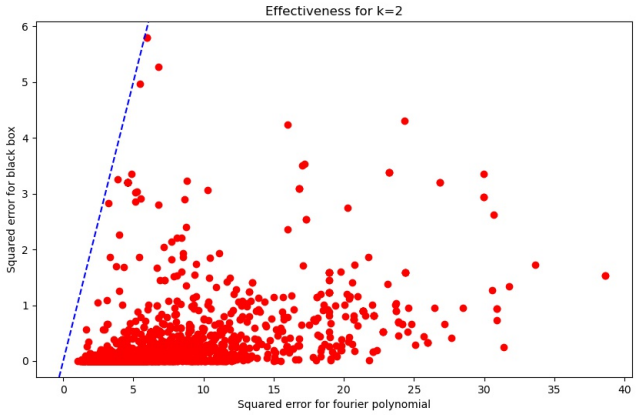

Figure 4.11: $(m(x)-m(\tilde{x}))^2$ on the x axis and $(b(x)-b(\tilde{x}))^2$ on the y axis. $\tilde{x}$ results from a perturbation with $k = 2$.

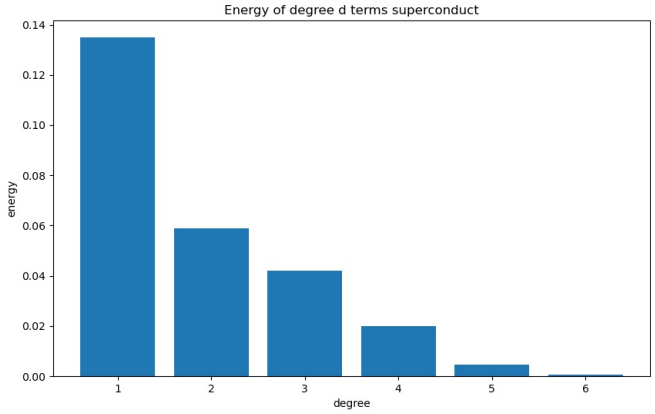

Figure 4.12: Overall energy of the terms of degree d

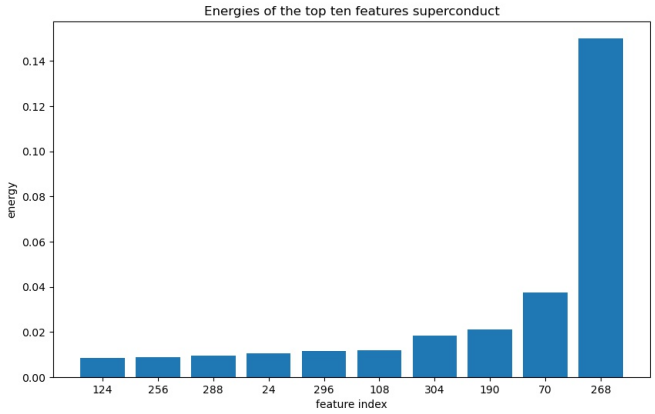

Figure 4.13: Overall energy of the terms containing feature number i. Only the top ten features are shown

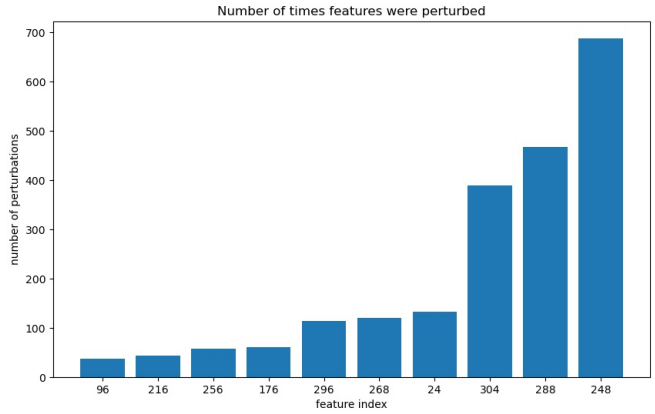

Figure 4.14: How many times feature i has been modified in a $k = 1$ perturbation. Only the top ten features are shown

The intersection of the set of most attacked features from 4.13, 4.14, 4.15 still contains five elements, namely 256, 288, 24, 296, 304. This means that the three metrics used in those graphs tend to agree on which features are most important.

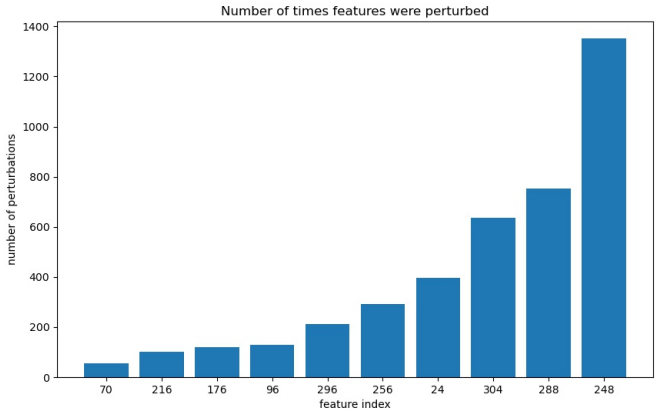

Figure 4.15: How many times feature i has been modified in a $k = 2$ perturbation. Only the top ten features are shown

No one of the five mentioned features appears in figure 4.16. In this case solving the lasso problem delivers perturbations that modify completely different features.

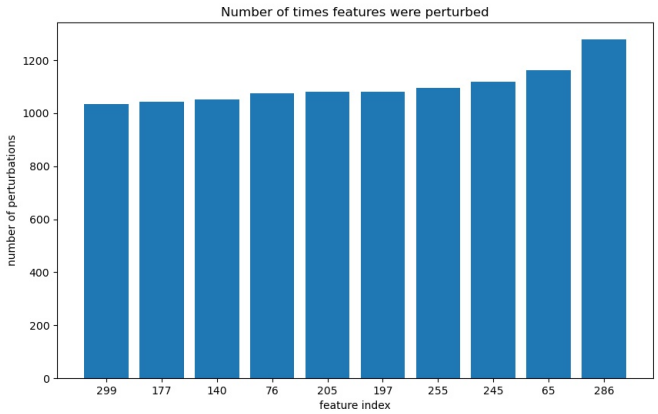

Figure 4.16: How many times feature i has been modified in the perturbation resulting from a lasso with regularization coefficient $\alpha = 0.025$. Only the top ten features are shown

Index	Feature meaning
24	weighted geometric mean of the atomic mass
65	geometric mean of the first ionization energy
70	weighted geometric mean of the first ionization energy
76	entropy of the first ionization energy
96	standard deviation of the first ionization energy
108	mean of the atomic radius
124	weighted geometric mean of the atomic radius
140	entropy of the first ionization energy
176	entropy of the density
177	entropy of the density
190	weighted range of the density
197	standard deviation of the density
205	mean of the electron affinity
216	geometric mean of the electron affinity
245	standard deviation of the electron affinity
248	standard deviation of the electron affinity
255	mean of the fusion heat
256	mean of the fusion heat
268	geometric mean of the fusion heat
286	range of the fusion heat
288	range of the fusion heat
296	standard deviation of the fusion heat
299	standard deviation of the fusion heat
304	weighted standard deviation of the fusion heat

4.5 Mushrooms

For the experiment of figure 4.17 we have a mean squared error of 0.035. The Fourier approximation overestimates the value of 52.3% of the inputs. The coefficient of determination is 0.766.

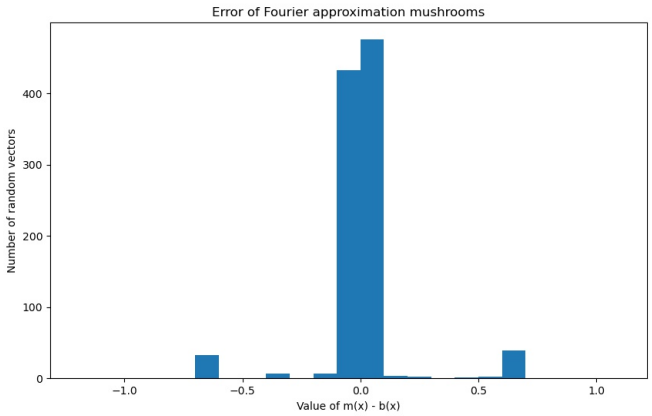

Figure 4.17: Value of the error on the x axis and the number of random vectors on the y axis. Bin size of 0.1.

For $k = 1$ in the mushrooms dataset the coefficient of determination is -0.1177.

For $k = 2$ in the mushrooms dataset the coefficient of determination is -0.6. The squared error on the black box seems to be either always 1 or 0, because most of the points lie either on the $y = 0$ line or on the $y = 1$ line. This suggests that the model predicts always 0 or 1 and therefore is too confident about its predictions.

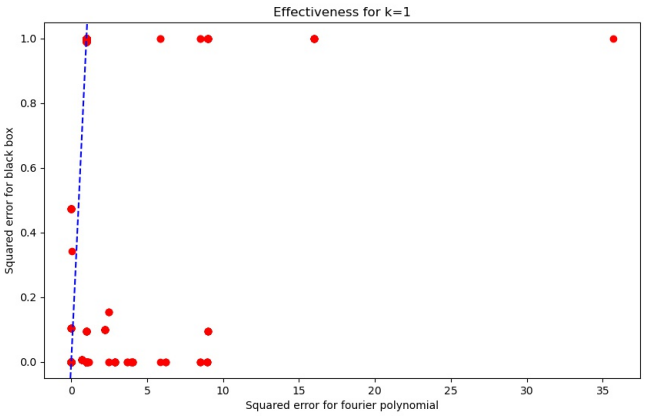

Figure 4.18: $(m(x) - m(\tilde{x}))^2$ on the x axis and $(b(x) - b(\tilde{x}))^2$ on the y axis. $\tilde{x}$ results from a perturbation with $k = 1$.

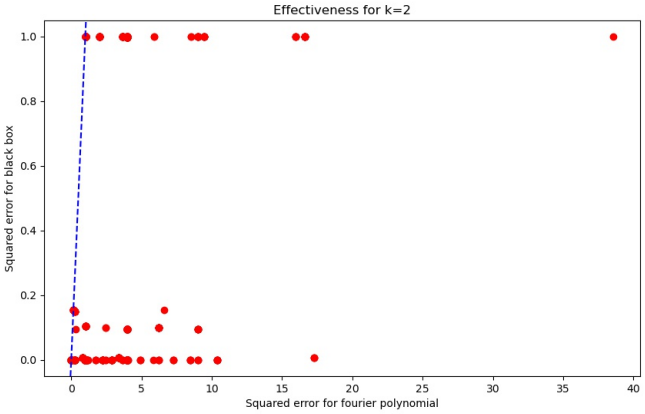

Figure 4.19: $(m(x) - m(\tilde{x}))^2$ on the x axis and $(b(x) - b(\tilde{x}))^2$ on the y axis. $\tilde{x}$ results from a perturbation with $k = 2$.

The extracted Fourier representation of the trained model has degree 6. Also in this case we see that terms of degree 5 and 6 don't have much energy, hence justifying the use of using branch and bound with GRD-gen.
Only 5 different features are perturbed in figure 4.22. All of them (40, 47, 16, 11) except from 45 appear also among the features with most energy in figure 4.21.
Features with indices 40, 47, 16, 11 appear also in figure 4.7. This plot shows also features from figure 4.21, that is 4, 9, 12.
Figure also shows features 40, 47, 16, 11. Moreover shows also features 4, 9, 12 which are in common with figure 4.6 and figure 4.21.

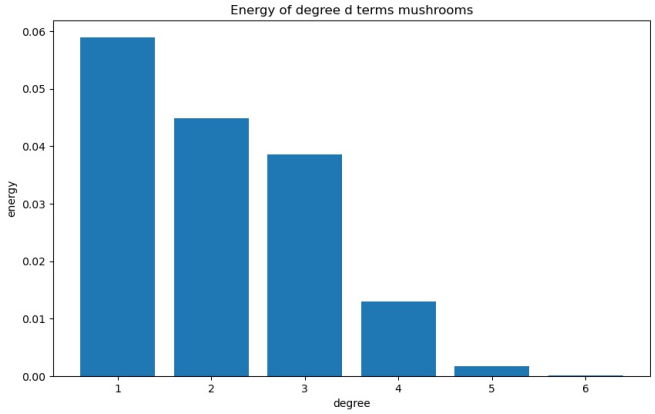

Figure 4.20: Overall energy of the terms of degree d

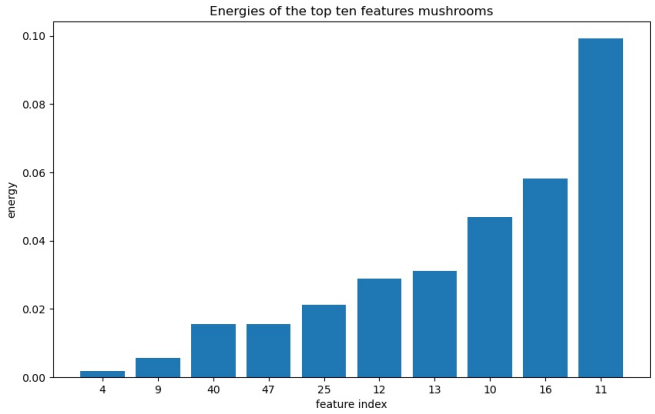

Figure 4.21: Overall energy of the terms containing feature number i. Only the top ten features are shown

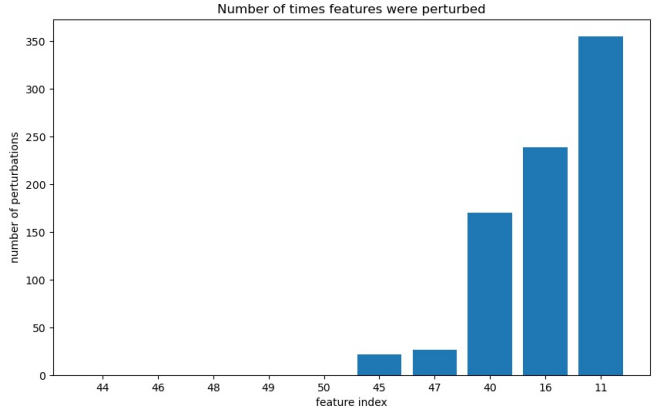

Figure 4.22: How many times feature i has been modified in a $k = 1$ perturbation. Only the top ten features are shown

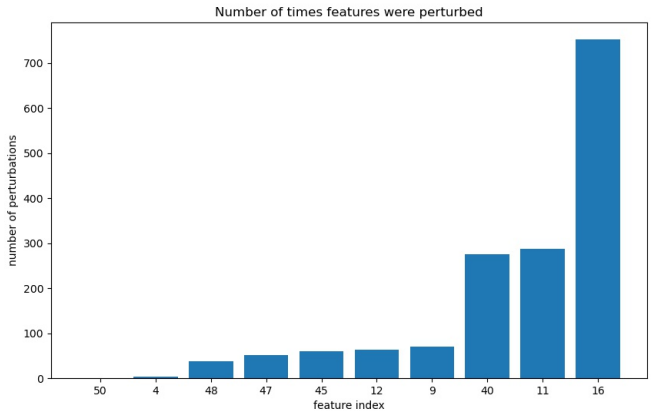

Figure 4.23: How many times feature i has been modified in a $k = 2$ perturbation. Only the top ten features are shown

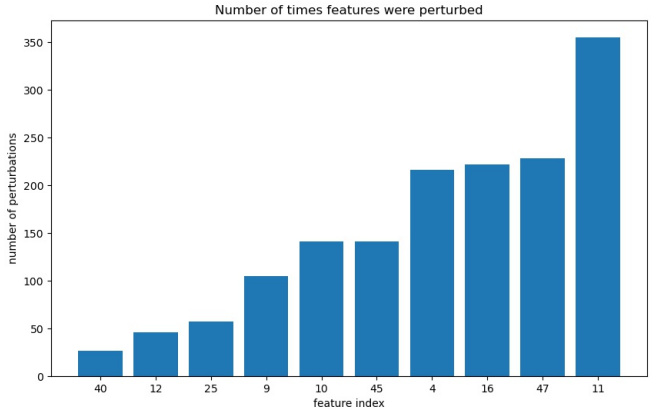

Figure 4.24: How many times feature i has been modified in the perturbation resulting from a lasso with regularization coefficient $\alpha = 0.025$. Only the top ten features are shown

Bibliography

[1] A. Amrollahi, A. Zandieh, M. Kapralov, and A. Krause, "Efficiently learning fourier sparse set functions," in *Advances in Neural Information Processing Systems*, H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett, Eds., vol. 32. Curran Associates, Inc., 2019.

[2] P. Stobbe and A. Krause, "Learning fourier sparse set functions," in *Proceedings of the Fifteenth International Conference on Artificial Intelligence and Statistics*, ser. Proceedings of Machine Learning Research, N. D. Lawrence and M. Girolami, Eds., vol. 22. La Palma, Canary Islands: PMLR, 21–23 Apr 2012, pp. 1125–1133.

[3] I. Haviv and O. Regev, *The Restricted Isometry Property of Subsampled Fourier Matrices*, pp. 288–297.

[4] R. Scheibler, S. Haghighatshoar, and M. Vetterli, "A fast hadamard transform for signals with sublinear sparsity in the transform domain," *IEEE Transactions on Information Theory*, vol. 61, no. 4, pp. 2115–2132, 2015.

[5] E. J. Candes and M. B. Wakin, "An introduction to compressive sampling," *IEEE Signal Processing Magazine*, vol. 25, no. 2, pp. 21–30, 2008.

[6] E. Hazan, A. Klivans, and Y. Yuan, "Hyperparameter optimization: a spectral approach," in *International Conference on Learning Representations*, 2018.

[7] M. Ribeiro, S. Singh, and C. Guestrin, ""why should I trust you?": Explaining the predictions of any classifier," in *Proceedings of the 2016 Conference of the North American Chapter of the Association for Computational Linguistics: Demonstrations*. San Diego, California: Association for Computational Linguistics, Jun. 2016, pp. 97–101.

[8] M. Sundararajan, A. Taly, and Q. Yan, "Axiomatic attribution for deep networks," in *Proceedings of the 34th International Conference on Machine Learning - Volume 70*, ser. ICML'17. JMLR.org, 2017, p. 3319–3328.

[9] J. V. Jeyakumar, J. Noor, Y.-H. Cheng, L. Garcia, and M. Srivastava, "How can i explain this to you? an empirical study of deep neural network explanation methods," in *Advances in Neural Information Processing Systems*, H. Larochelle, M. Ranzato, R. Hadsell, M. F. Balcan, and H. Lin, Eds., vol. 33. Curran Associates, Inc., 2020, pp. 4211–4222.

[10] C. Olah, A. Satyanarayan, I. Johnson, S. Carter, L. Schubert, K. Ye, and A. Mordvintsev, "The building blocks of interpretability," *Distill*, 2018, https://distill.pub/2018/building-blocks.

[11] P. Yang, J. Chen, C.-J. Hsieh, J. ling Wang, and M. I. Jordan, "Greedy attack and gumbel attack: Generating adversarial examples for discrete data," *ArXiv*, vol. abs/1805.12316, 2020.

[12] Q. Lei, L. Wu, P.-Y. Chen, A. Dimakis, I. S. Dhillon, and M. J. Witbrock, "Discrete adversarial attacks and submodular optimization with applications to text classification," in *Proceedings of Machine Learning and Systems*, A. Talwalkar, V. Smith, and M. Zaharia, Eds., vol. 1, 2019, pp. 146–165.

[13] F. Kahl and P. Strandmark, "Generalized roof duality," *Discrete Applied Mathematics*, vol. 160, no. 16, pp. 2419–2434, 2012.

[14] R. de Wolf, "A brief introduction to fourier analysis on the boolean cube." *Theory of Computing, Graduate Surveys*, vol. 1, pp. 1–20, 01 2008.

[15] M. Redmond and A. Baveja, "A data-driven software tool for enabling cooperative information sharing among police departments," *European Journal of Operational Research*, vol. 141, no. 3, pp. 660–678, 2002.

[16] K. Hamidieh, "A data-driven statistical model for predicting the critical temperature of a superconductor," *arXiv: Applications*, 2018.