An Introduction to Explainable AI

Damien Garreau

Université Côte d'Azur

January 4, 2022

Outline

1. Introduction

2. A game-theory perspective
   - Shapley values
   - SHAP

3. Local approximations
   - Introduction
   - Gradient-based explanations

4. LIME
   - Images
   - Text data
   - Tabular data

5. Future directions

1. Introduction

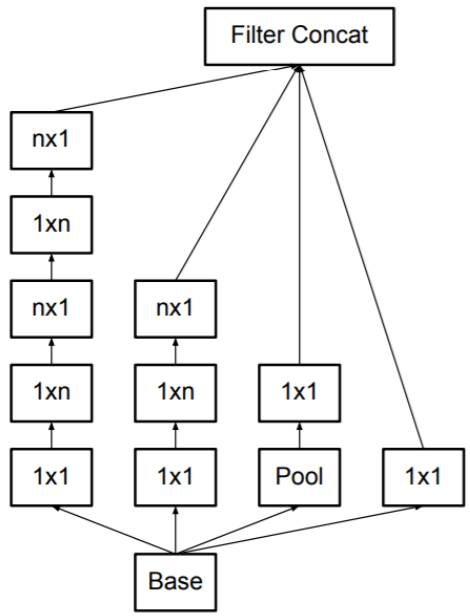
Introduction

- machine learning algorithms are now used for *critical* decisions:
  - medical diagnosis
  - credit scoring
  - college admissions
  - ...
- **Problem:** state-of-the-art = deep neural networks
- we have no idea how a particular decision is made
- often referred to as "black-boxes":
  - billions of parameters (*e.g.*, GPT-3, 175 billions[1])
  - complicated architectures
- **Disclaimer:** in many applications, we only care about performance, and interpretability is not a topic of discussion

[1] Brown et al., *Language models are few-shot learners*, arxiv, 2020

Example: Inception network

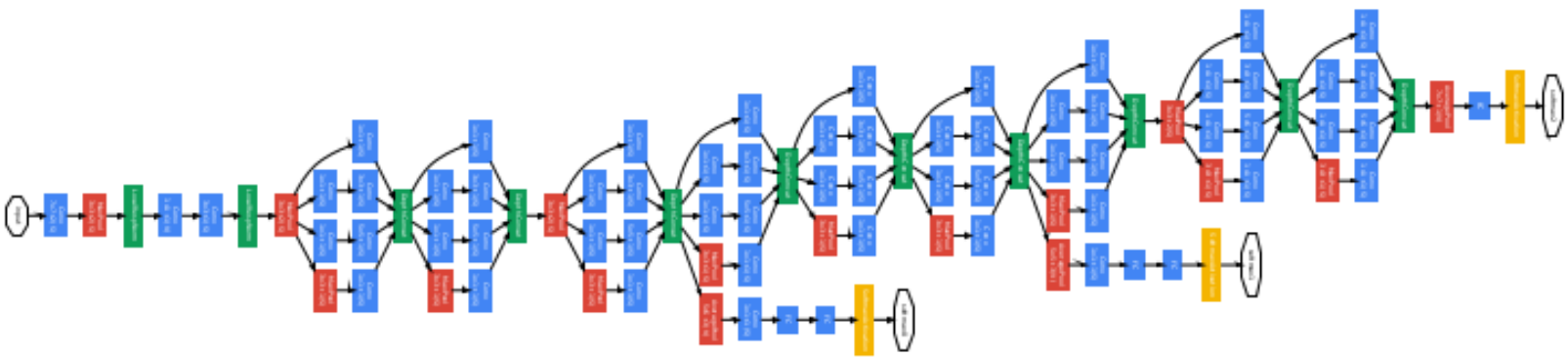
- **Example:** the InceptionV3 network for image classification[2]
- here is one block of the architecture:



[2] Szegedy et al., *Rethinking the inception architecture for computer vision*, CVPR, 2016

Example: Inception network, ctd.

- then stack all these modules (159 layers, 24M parameters)



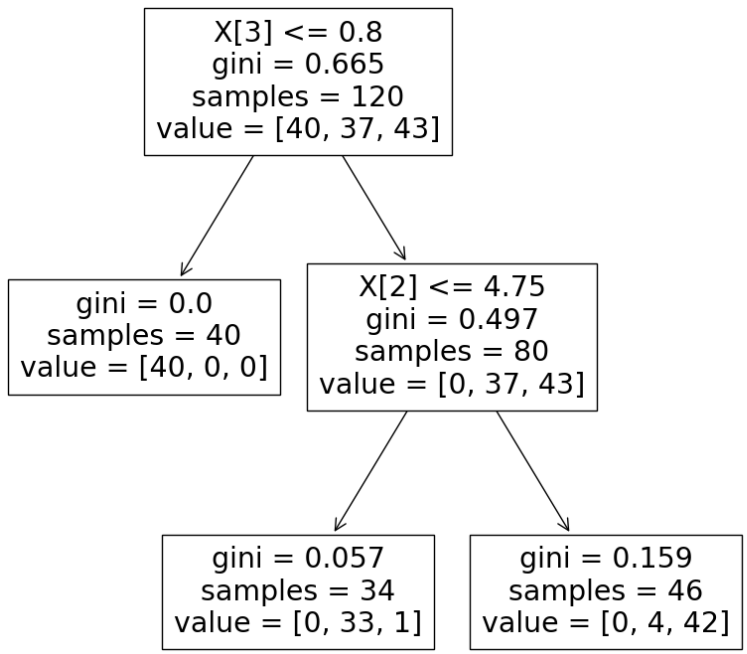
Interpretable models

- stark contrast with *interpretable models*
- **Question:** what is an interpretable model?
- for instance, linear model with small number of parameters
- **Example:** credit score, historical features:[3]
  - x_1 = character
  - x_2 = capital
  - x_3 = collateral
  - x_4 = capacity
  - x_5 = condition
- imagine the model is $f(x) = 0.1x_1 + 5x_2 + 3x_3 + 10x_4 + x_5$
- we *know* that increasing collateral by one unit increases credit score by 3 units

[3] Dastile et al., *Statistical and machine learning models in credit scoring: A systematic literature survey*, Applied Soft Computing, 2020

Interpretable models, ctd.

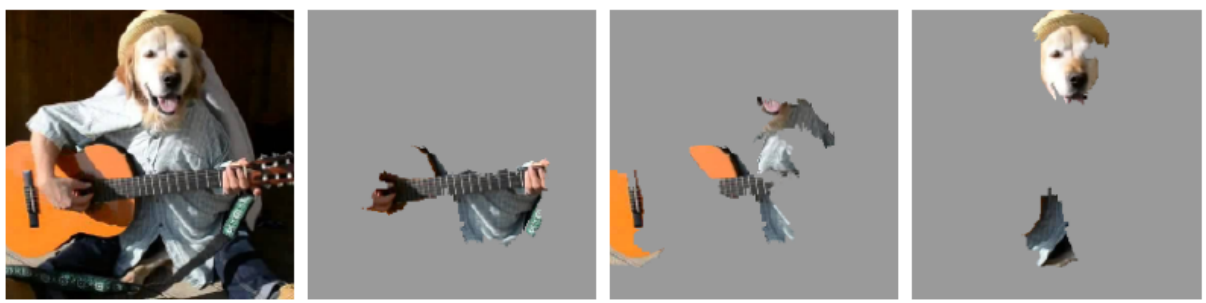
- also the case for small decision trees



- **Problem:** these simple models do not achieve high enough accuracy

Explainability

- **Goal:** explain how a machine learning algorithm takes a particular decision
- **In this talk:** local explanation: explaining an already trained model $f : \mathbb{R}^D \to \mathbb{R}$ (*post hoc* explanation) at an example $\xi \in \mathbb{R}^D$ (*local*)
- in many cases, explanation = interpretable model that looks like f around ξ
- but on the surface, visual clues helping us to understand the model



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

- **Figure:** explaining the Inception network with LIME[4]

[4] Ribeiro et al., *"Why should I trust you?" Explaining the Predictions of Any Classifier*, SIGKDD, 2016

Further motivation

- **As a debugging tool:** sometimes we learn artifacts despite our best efforts
- not detectable looking only at the performance on the test set
- **Example:** learning to detect wolves looking only at the background (snowy)
- interpretability could point out this problem before going into production
- **Coming legal requirement:** political will to regulate automated critical decision making
- in spirit, if you make an automated decision, then you should be able to provide the tools to explain it to the customer
- not a firm legal requirement yet (in Europe)[5]

[5] Wachter, Mittelstadt, Floridi, *Why a right to explanation of automated decision-making does not exist in the general data protection regulation*, International Data Privacy Law, 2017

2. A game-theory perspective

2.1. Shapley values

Shapley values

- **Setting:** D-player game[6]
- characteristic function $v : 2^D \to \mathbb{R}$, gives the *value* of a coalition S
- total sum of gains the members of S can obtain by cooperation
- **Idea:** distribute fairly the total gains to the players, assuming that they all contribute

Definition: Shapley value of player j:

$$\phi_j(v) = \sum_{S \subseteq \{1,\dots,D\} \setminus \{j\}} \frac{|S|!(D - |S| - 1)!}{D!} \left(v(S \cup \{j\}) - v(S)\right) .$$

- **Intuition:** if player j plays much better than the others, then $v(S \cup \{j\})$ consistently higher than $v(S)$, and $\phi_j(v) \gg 0$

[6] Shapley, *A value for n-person game*, Contributions to the theory of games, 1953

Shapley values, ctd.

- Shapley values have nice theoretical properties:
  - *efficiency:* sum of Shapley values = gain of the whole coalition:

$$\sum_j \phi_j(v) = v(\{1, \ldots, D\}) \,.$$

  - *symmetry:* players with the same skills are rewarded equally:

$$\forall S \subseteq \{1, \ldots, D\},\ v(S \cup \{j\}) = v(S \cup \{k\}) \quad \Rightarrow \quad \phi_j(v) = \phi_k(v) \,.$$

  - *linearity:* v and w two characteristic functions, then

$$\forall j \in \{1, \ldots, D\}, \quad \phi_j(v + w) = \phi_j(v) + \phi_j(w) \,.$$

  - *null player:* a player that does not bring anything is not rewarded:

$$\forall j \in \{1, \ldots, D\}, \quad v(S \cup \{j\}) = v(S) \quad \Rightarrow \quad \phi_j(v) = 0 \,.$$

Shapley values, ctd.

- other nice properties:
  - *anonymity*
  - *standalone test*
  - ...
- more interestingly:

Theorem:[7] Shapley values are the only payment rule satisfying efficiency, symmetry, linearity, and null player.

But what is the link with interpretability?

[7] *ibid*

Shapley regression values

- **Setting:** linear regression on a set of D features, we want to look at feature importance for example ξ
- for each subset of features $S \subseteq \{1, \dots, D\}$, retrain a model f_S only using the features in S

Definition:[8] the *Shapley regression value* associated to feature j is given by

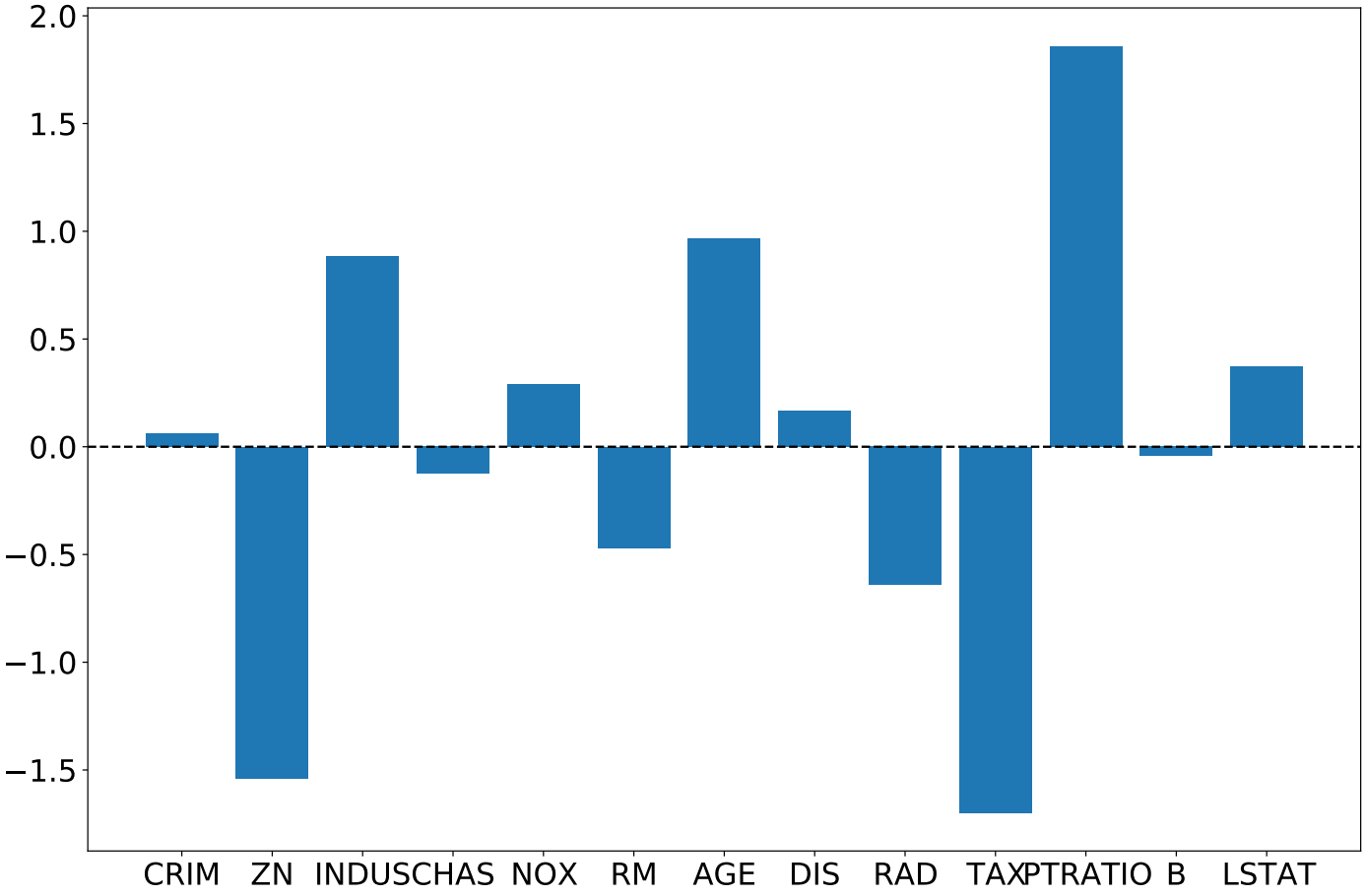
$$\phi_j := \sum_{S \subseteq \{1,\dots,D\} \setminus \{j\}} \frac{|S|!(D - |S| - 1)!}{D!} \left(f_{S \cup \{j\}}(\xi_{S \cup \{j\}}) - f_S(\xi_S) \right) ,$$

where ξ_S is the restriction of ξ to S features.

[8] Lipovetsky and Conklin, *Analysis of regression in game theory approach*, Applied Stochastic Models in business and industry, 2001

Shapley regression values

- **Example:** Boston housing dataset ($n = 404$ training points, $D = 13$)
- the output for a given example can be represented as



Shapley sampling values

- there are two main problems with this approach:
  - *computational cost* $= \mathcal{O}\left(2^D\right)$
  - *retraining* the model each time
- a first solution: *Shapley sampling values*[9]
  - subsample in the sum over all subsets
  - instead of retraining the model, mimic the removal a variables by randomly sampling over the training set
- in other words, replace $f_S(\xi_S)$ by

$$\mathbb{E}\left[f(x) \mid x_S = \xi_S\right] .$$

- f can now be any model, provided that we can query efficiently

[9] Štrumbelj and Kononenko, *Eplaining models and individual predictions with feature contributions*, Knowledge and information systems, 2014

2.2. SHAP

SHAP

- still very costly to test *all the coalitions*
- **Clever idea:** linear regression on the presence / absence of features
- define interpretable features $z \in \{0,1\}^d$, with $d \leq D$ (you can imagine $d = D$ if you want)
- $h_\xi : \{0,1\}^d \to \mathbb{R}^D$ mapping function such that $h_\xi(\mathbf{1}) = \xi$

Definition-theorem (kernel SHAP)[10]**:** ϕ minimizes

$$\sum_{z \in \{0,1\}^d} \frac{d-1}{\binom{d}{|z|} \cdot |z| \cdot (d - |z|)} \left(f(h_\xi^{-1}(z)) - \phi^\top z \right)^2 .$$

[10] Lundberg and Lee, *A Unified Approach to Interpreting Model Predictions*, NeurIPS, 2017

SHAP, ctd.

- Shapley values can be computed using weighted linear regression
- computational cost: $\mathcal{O}\left(2^d + d^3\right)$
- **Remark:** not practical if $d \gg 1$
- in that case, subsample: $z_1, \ldots, z_n$ i.i.d. Bernoulli $\in \{0,1\}^d$ and minimize for $\phi \in \mathbb{R}^d$

$$\sum_{i=1}^{n} \pi_i \cdot \left(f(h_\xi^{-1}(z_i)) - \phi^\top z_i \right)^2 ,$$

with

$$\pi_i := \frac{d-1}{\binom{d}{|z_i|} \cdot |z_i| \cdot (d - |z_i|)} .$$

SHAP additional theory

- **Question:** what happens in simple cases?
- let us say that f is linear, that is,

$$f(x) := \sum_{j=1}^{d} \lambda_j x_j + b\,.$$

Corollary:[11] If f is linear, then $\phi_0 = b$ and

$$\phi_j = \lambda_j(\xi_j - \overline{x}_j)\,,$$

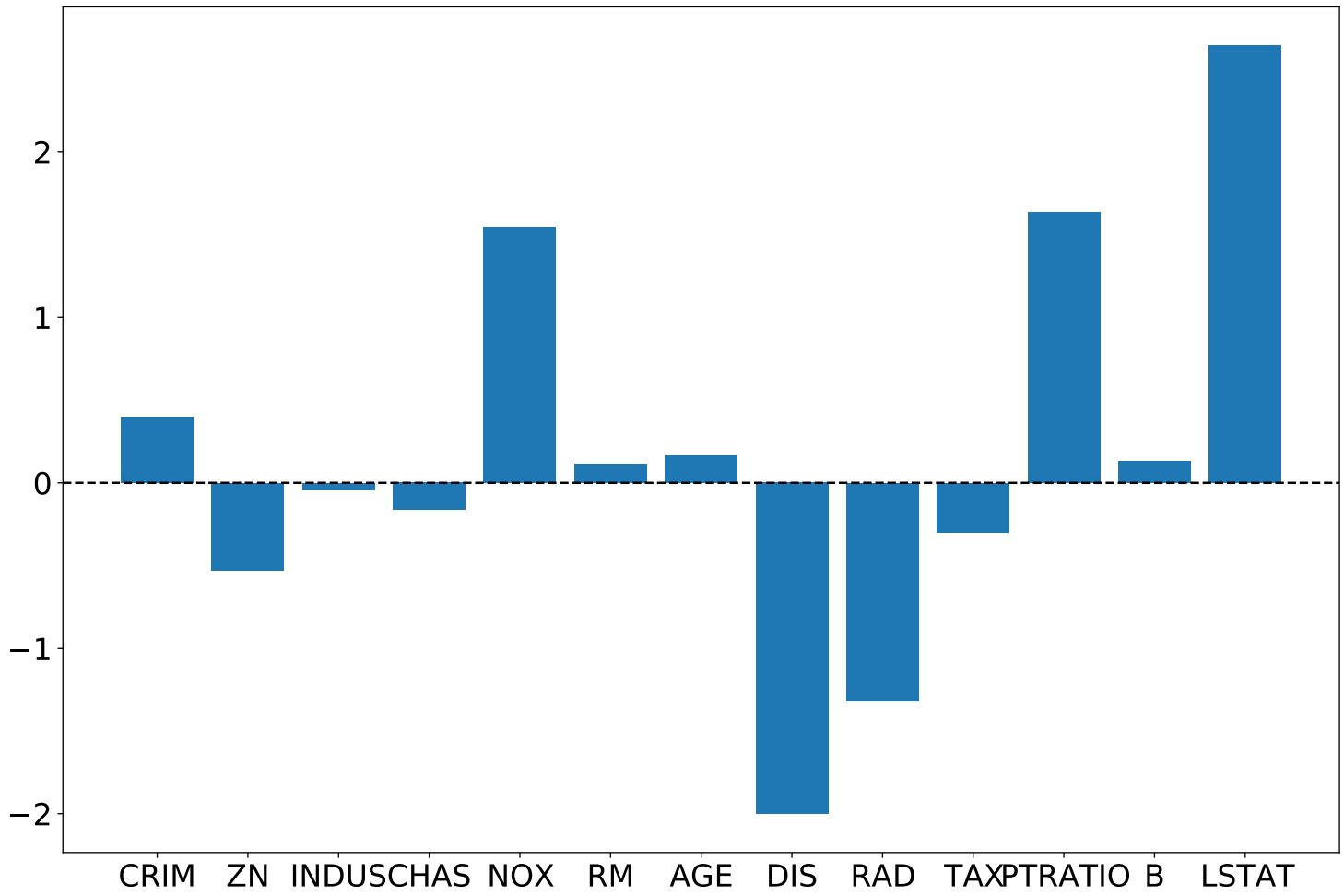
where $\overline{x}_j$ is the mean of feature j on the dataset.

- good sanity check: our interpretability method should output something meaningful for a linear model

[11] ibid

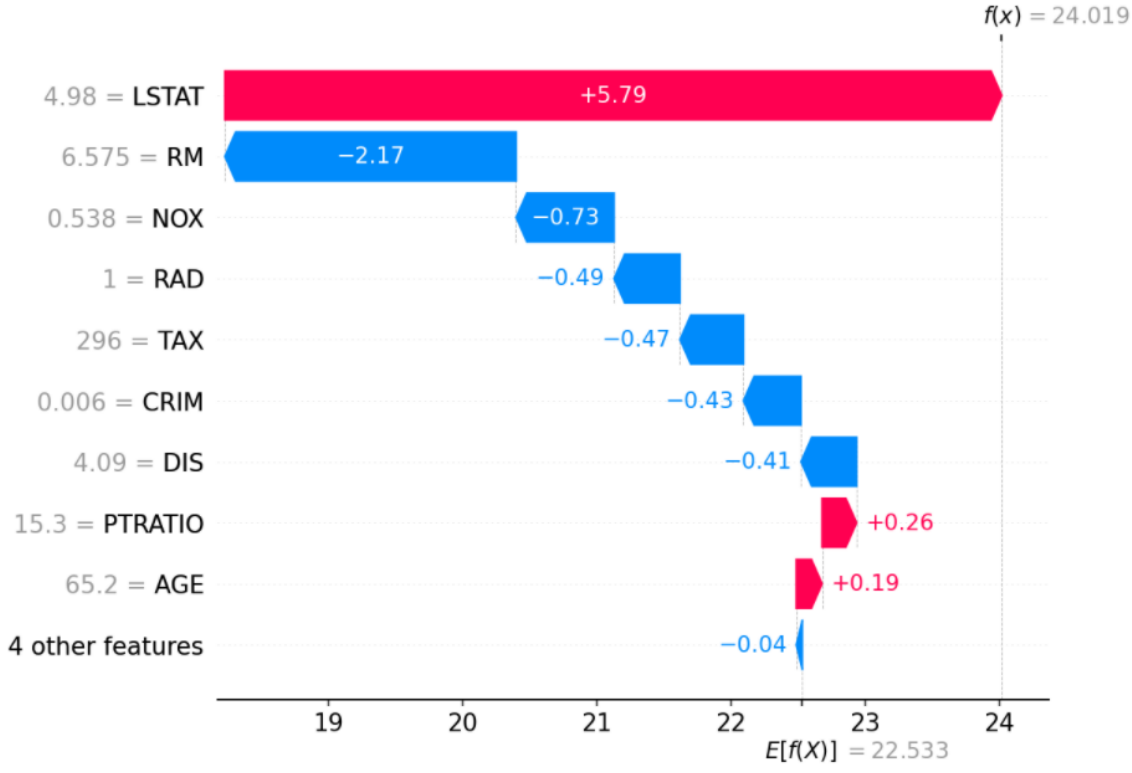
SHAP, tabular example

- **Example:** interpreting a linear model on the Boston dataset:



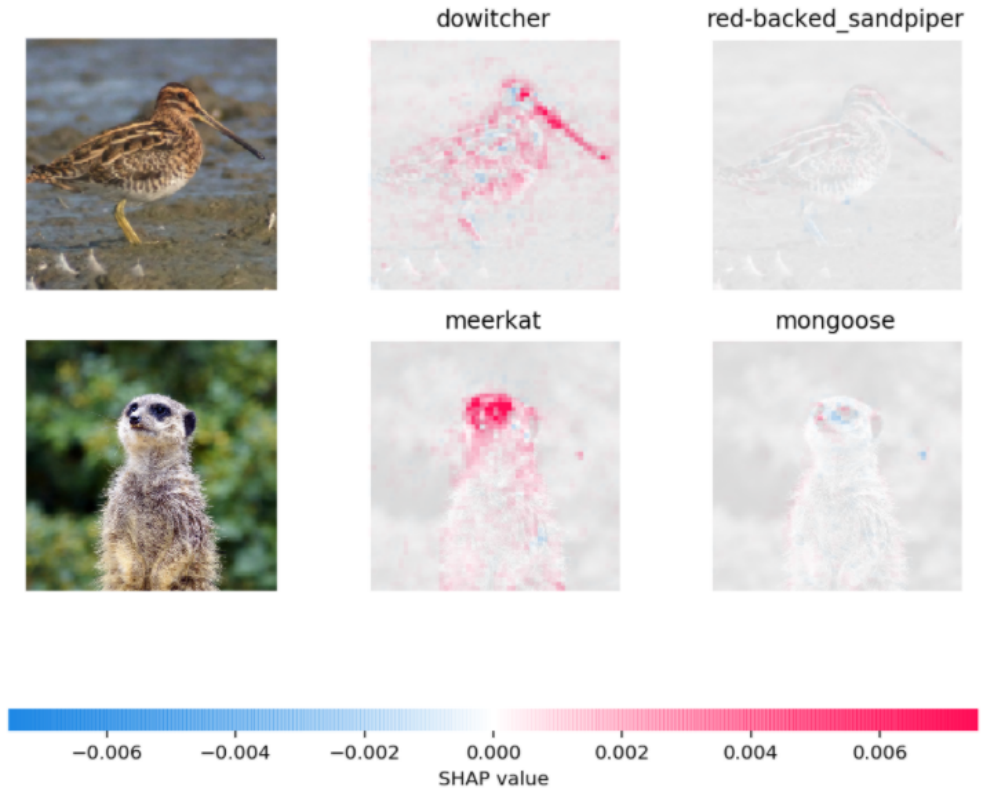
SHAP, tabular example

- you can also use the `shap` Python package
- really nice visualizations:



SHAP, image example

- of course, SHAP is not restricted to tabular data
- **Example:** explaining the predictions of VGG16 for two classes with expected gradients (approximate SHAP)



Conclusion

Advantages:

- based on solid theory
- SHAP can be used on any model
- can take advantage of specific architectures:
  - *TreeSHAP*[12] (tree-based predictors)
  - *DeepSHAP* (DeepLIFT[13] + Shapley values)

Inconvenients:

- costly to run[14]
- not easy to read if many features

---

[12] Lundberg et al., *Consistent individualized feature attribution for tree ensembles*, arxiv, 2018

[13] Shrikumar et al., *Learning important features through propagating activation differences*, ICML, 2017

[14] improving the efficiency is work in progress, *e.g.*, Covert and Lee, *Improving KernelSHAP: Practical Shapley Value Estimation via Linear Regression*, arxiv, 2021

3. Local approximations

3.1. Introduction

Introduction

- **General idea:** machine learning model = complicated function of the inputs
- approximate this function by a first order approximation

Theorem (Taylor, order one): let f be smooth enough in the neighborhood of $\xi \in \mathbb{R}^D$. Then

$$f(x) = f(\xi) + \nabla f(\xi)^\top (x - \xi) + o\left(\|x - \xi\|\right) ,$$

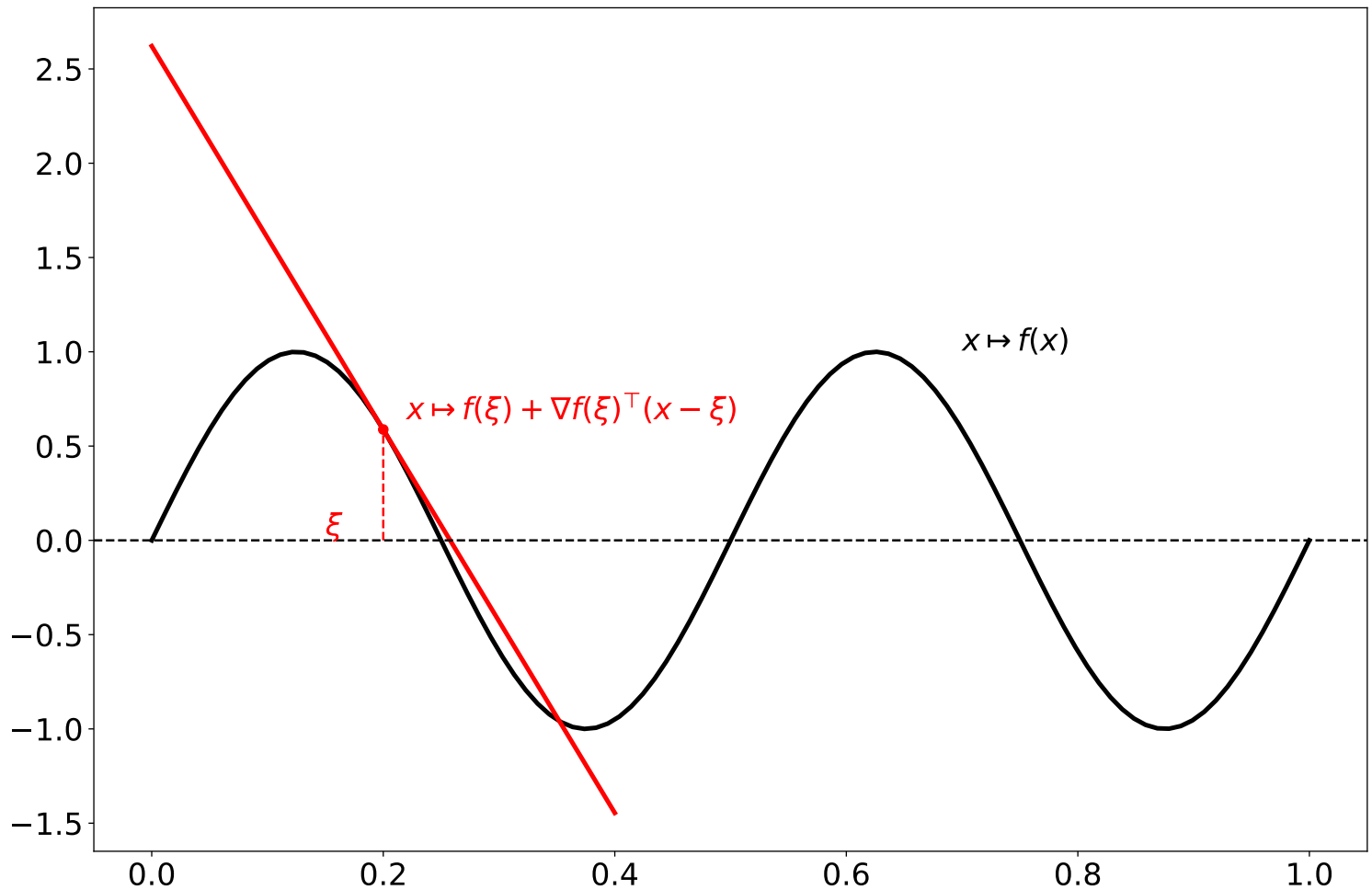
where $\nabla f(\xi)$ is the *gradient* of f at ξ.

- **Reminder:** gradient = vector of $\mathbb{R}^D$ defined by

$$\forall 1 \leq j \leq D, \qquad (\nabla f(\xi))_j = \left.\frac{\partial f(x)}{\partial x_j}\right|_{x=\xi} .$$

Linear approximation in dimension 1

- **Example:** linear approximation in dimension 1:



3.2. Gradient-based explanations

Gradient explanation

- **Simple idea:** take the gradient of the function to explain at the point of interest:

$$\phi_j = (\nabla f(\xi))_j = \left.\frac{\partial}{\partial x_j} f(x)\right|_{x=\xi} .$$

- generally referred to as *gradient explanation*[15] or *saliency maps*
- **Intuition:** tells us how much a change in each input dimension would change the the prediction *in a small neighborhood around* ξ
- computational cost $= \mathcal{O}(1)$ if the model is "tensorflow-compatible"
- **Remark:** in this talk, one evaluation of the model costs $\mathcal{O}(1)$

[15] Erhan et al., *Visualizing higher-layer features of a deep network*, tech. report, 2009

Linear model

- **Question:** what happens for a linear model?
- recall that we set $f(x) := \sum_{k=1}^{d} \lambda_k x_k$
- easy to compute:

$$\begin{aligned}\frac{\partial f(x)}{\partial x_j} &= \frac{\partial}{\partial x_j} \sum_{k=1}^{d} \lambda_k x_k \\ &= \sum_{k=1}^{d} \frac{\partial}{\partial x_j} \lambda_k x_k \\ \frac{\partial f(x)}{\partial x_j} &= \lambda_j \,.\end{aligned}$$

- we retrieve the coefficients of the model:

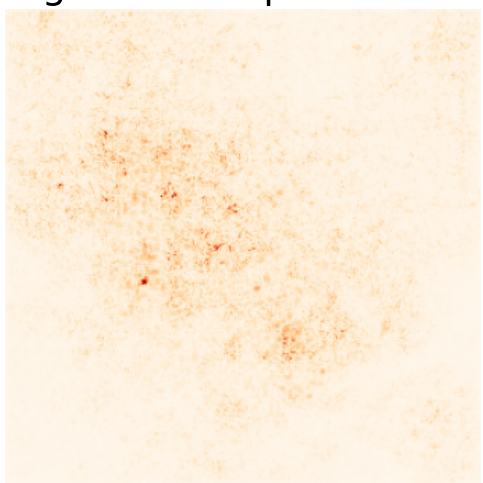
$$\phi_j = \lambda_j \,.$$

Gradient explanation, ctd.

- **Example:** image classification on ILSVRC with InceptionV3:

predicted: quail (22.6%)

gradient explanation



- quite *noisy*, but we can see that the network is using the relevant part of the image
- also possible to look at *positive* and *negative* influence

Gradient times input

- additional idea: weight the partial derivatives by the pixel values
- product of the gradient and the input pixel-wise:[16]

$$\phi_j = (\xi \odot \nabla f(\xi))_j = \xi_j \cdot \frac{\partial f(\xi)}{\partial x_j} \,.$$

- **Intuition:** clear on black and white images, in general provides smoother results
- computational cost $= \mathcal{O}(D)$
- for a linear model:

$$f(x) = \sum_{j=1}^{D} \lambda_j x_j + b \quad \Rightarrow \quad \phi_j = \lambda_j \xi_j \,.$$

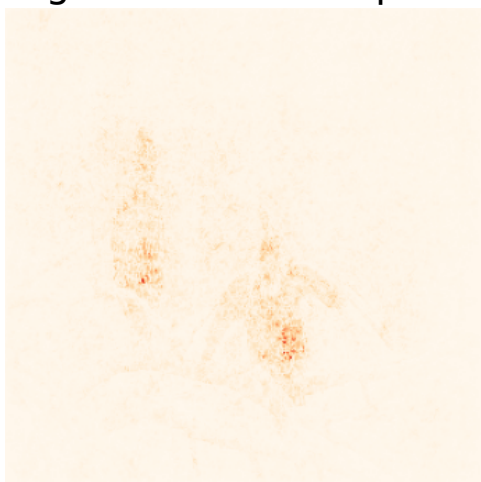
[16] Shrikumar et al., *Learning important features through propagating activation differences*, ICML, 2017

Gradient times input, ctd.

- **Example:** image classification on ILSVRC with InceptionV3

predicted: quail (22.6%)

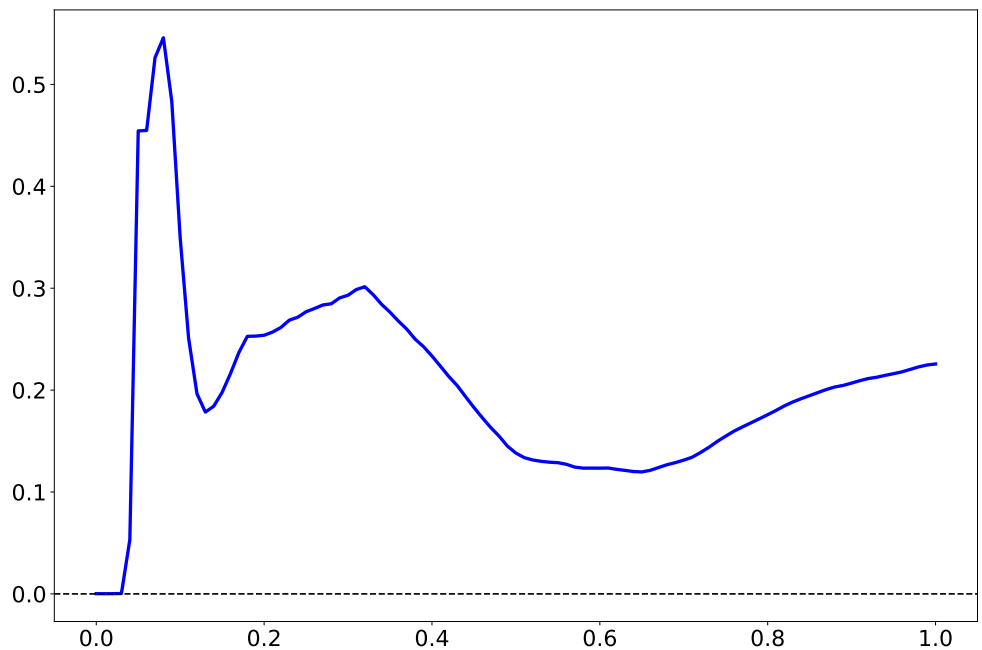
gradient times input



- much *smoother*, as promised
- still difficult to read in some cases

Integrated gradients

- two well-known problems with the decision function of a deep neural net:
  - can be quite *irregular*
  - can *saturate*, especially for examples with high confidence
- **Example:** InceptionV3 prediction for the class 'quail' on a path from 0 to ξ



Integrated gradients, ctd.

- **Idea:**[17] average the gradient between a given reference and the point of interest in order to:
  1. smooth the irregularities
  2. get information even if saturation
- formally, if ξ_0 is a reference image,

$$\phi = (\xi - \xi_0) \odot \int_0^1 \frac{\partial f(\xi_0 + \alpha(\xi - \xi_0))}{\partial \xi} \mathrm{d}\alpha \,.$$

- of course, we have no way to compute the previous integral
- Monte-Carlo approximation:

$$\int_0^1 \frac{\partial f(\xi_0 + \alpha(\xi - \xi_0))}{\partial \xi} \mathrm{d}\alpha \approx \frac{1}{m} \sum_{i=1}^{m} \frac{\partial f(\xi_0 + \frac{i}{m}(\xi - \xi_0))}{\partial \xi} \,.$$

- computational cost $= \mathcal{O}(mD)$ ($m = 20$ gives good results)

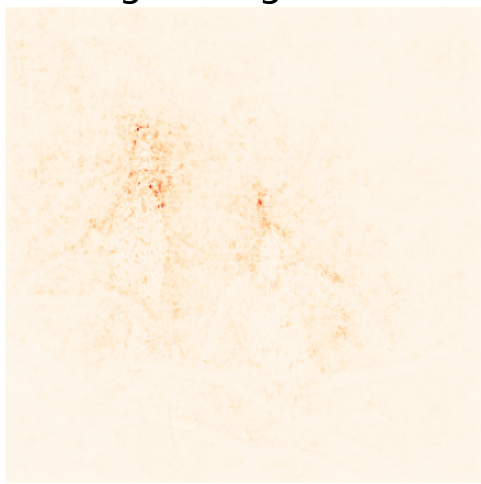
[17] Sundararajan et al., *Axiomatic attribution for deep networks*, ICML 2017

Integrated gradients, ctd.

- **Example:** image classif. on ILSVRC with InceptionV3, reference image = 0

predicted: quail (22.6%)

integrated gradients



- usually less "visual diffusion"
- main critic: similar to an *edge detector*[18]

[18] Adebayo et al., *Sanity checks for saliency maps*, NeurIPS 2018

Integrated gradients theory

- **Question:** what happens for a linear model?
- recall that we set $f(x) = \sum_{k=1}^{d} \lambda_k x_k$
- in that case:

$$\begin{aligned}
\phi_j &= (\xi_j - \xi_{0,j}) \cdot \int_0^1 \frac{\partial f(\xi_0 + \alpha(\xi - \xi_0))}{\partial \xi_j} \mathrm{d}\alpha \\
&= (\xi_j - \xi_{0,j}) \cdot \int_0^1 \frac{\partial}{\partial \xi_j} (\lambda_j (\xi_{0,j} + \alpha(\xi_j - \xi_{0,j}))) \mathrm{d}\alpha \\
&= (\xi_j - \xi_{0,j}) \cdot \int_0^1 \lambda_j \alpha \mathrm{d}\alpha \\
\phi_j &= \frac{1}{2} (\xi_j - \xi_{0,j}) \lambda_j \,.
\end{aligned}$$

- up to constants, we recover gradient $\times$ input

Related methods

- numerous extensions, a number of them relying on the specific architecture of the model (a neural network):
  - *guided backpropagation*[19]
  - *GradCAM*[20]
  - *SmoothGrad*[21]

[19] Springenberg et al., *Striving for simplicity: the all convolutional net*, arxiv, 2015
[20] Selvaraju et al., *Grad-CAM: why did you say that?*, arxiv, 2016
[21] Smilkov et al., *SmoothGrad: removing noise by adding noise*, arxiv, 2017

Conclusion

Pros:

- simplicity: easy to compute if your model falls into an automatic differentiation framework
- seems to make sense for simple models (especially linear)

Cons:

- not always very informative
- structure of the input more important than the gradient in many cases
- what to do when the model is not differentiable?

4. LIME

Introduction

- another possibility: perturbative approach
- focus on **L**ocal **I**nterpretable **M**odel-agnostic **E**xplanations (LIME[22])
- in truth, several versions of the method, depending on the nature of the data:
  - *images*
  - *text data*
  - *tabular data*
- complicated operating procedure, but very popular
- we start with image data: as before, $\xi \in \mathbb{R}^D$ an image to explain and $f : \mathbb{R}^D \to [0, 1]$ the model
- **Example:** f is the prediction for a certain class given by InceptionV3

[22] Ribeiro et al., *"Why should I trust you?" Explaining the Prediction of any Classifier*, SIGKDD, 2016

4.1. Images

Image LIME

- on a high level, Image LIME operates as follows:
  1. decompose ξ in d *superpixels* (small, homogeneous patches);
  2. create a number of *perturbed samples* (= new images) $x_1, \ldots, x_n$;
  3. *weight* the perturbed samples;
  4. *query* the model, getting predictions $y_i = f(x_i)$;
  5. build a *local surrogate model* $\hat{\beta}_n$ fitting the y_is on the presence or absence of superpixels.
- generally, highlight in the original image the (top 5) positive superpixels:

predicted: quail (22.6%)

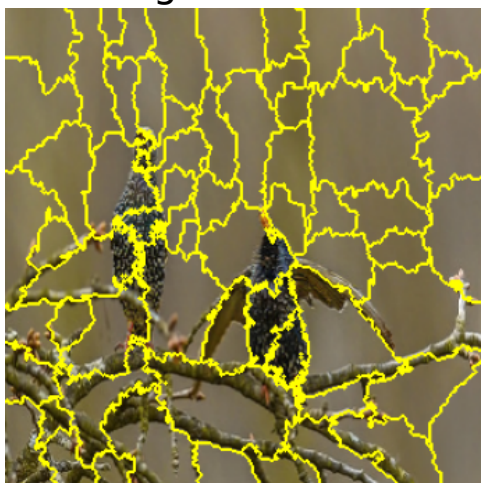
LIME explanation

Step 1: superpixels

- by default, *quickshift* algorithm[23]

ξ

segmentation

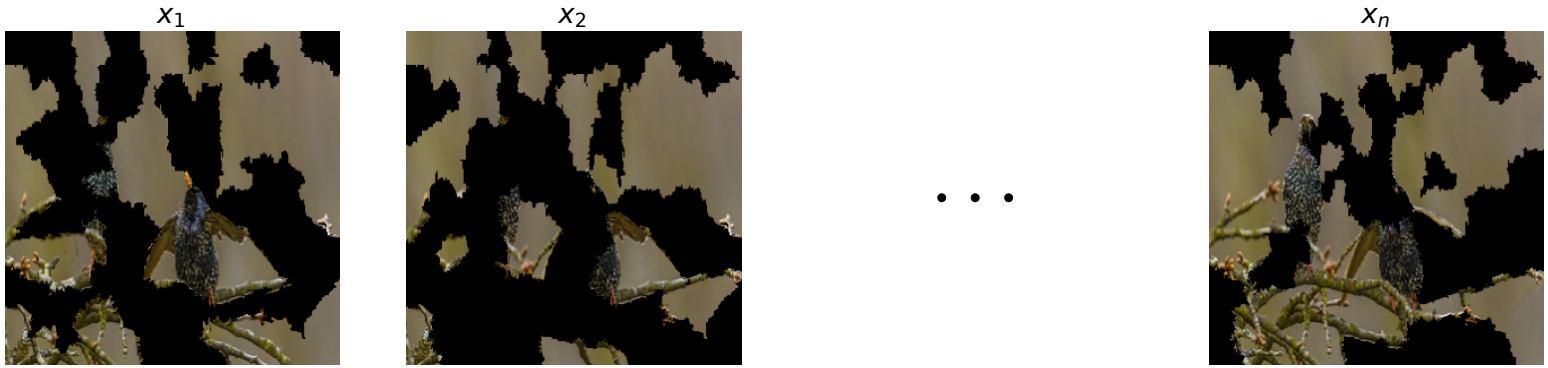


- superpixels $J_1, \ldots, J_d$ = disjoint sets of pixel indices
- in this example, $D = 299 \times 299 \times 3 = 268,203$ and $d = 65$
- corresponds to the h_ξ mapping in SHAP

[23] Vedaldi and Soatto, *Quickshift and kernel methods for mode seeking*, ECCV, 2008

Step 2: sampling

- **Simple idea:** take a replacement color (say black), randomly switch on and off the superpixels:



- by default, $n = 5000$
- **Potential problem:** black can be a meaningful color for the model (we are trying to *remove* a feature)

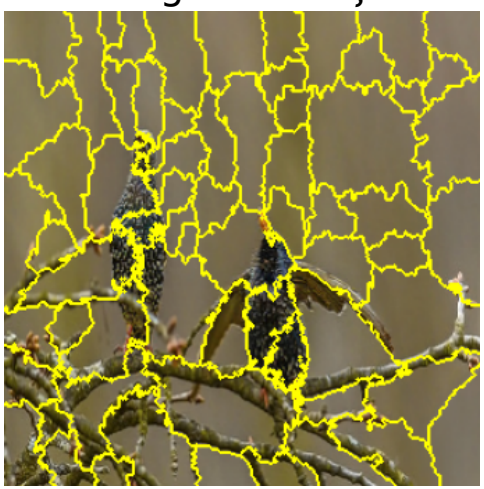
Step 2: sampling, ctd.

- **Idea:** compute the *mean* of the image on each superpixel
- formally,

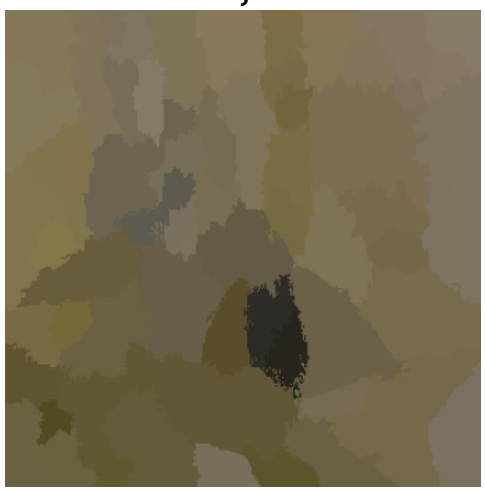
$$\forall u \in J_k, \qquad \overline{\xi}_u = \frac{1}{|J_k|} \sum_{u \in J_k} \xi_u \,.$$

- channel-wise if RGB image

segmented ξ

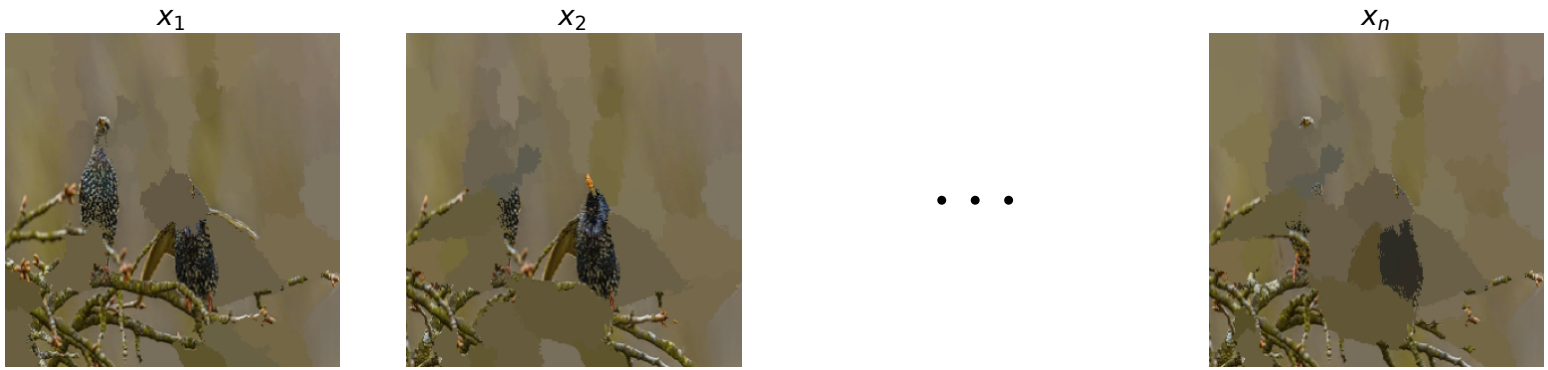


$\overline{\xi}$



Step 2: sampling, ctd.

- **Same idea:** replace superpixels randomly by corresponding superpixel of $\overline{\xi}$



- **Intuition:** replace the superpixel with something non-informative, but not too far away from the local pixel distribution
- this is the *default* behavior

Step 3: weights

- to each perturbed sample x_i corresponds a binary vector $z_i \in \{0,1\}^d$
- $z_{i,j} = 1$ iff superpixel j is turned on
- $\mathbf{1}$ corresponds to ξ
- **Idea:** give more weight to samples near ξ
- formally, x_i receives the weight

$$\pi_i := \exp\left(\frac{-\delta(z_i, \mathbf{1})^2}{2\nu^2}\right),$$

where δ is the *cosine distance* and $\nu > 0$ is a *bandwidth parameter* (default value $= 0.25$)

Definition: for any two vectors $u, v \in \mathbb{R}^d$, we define the *cosine distance* between u and v by

$$\delta(u, v) := 1 - \frac{u^\top v}{\|u\| \cdot \|v\|}.$$

Step 4: query

- compute $y_i = f(x_i)$ for every $i \in \{1, \dots, n\}$

$y_1 = f(x_1) = 0.38$ $y_2 = f(x_2) = 0.03$ $y_n = f(x_n) = 0.0$



$\dots$

- cost $= \mathcal{O}(n)$ (n calls to the model)
- **Remark:** can be costly, but can be parallelized

Step 5: local surrogate model

- finally, train a local surrogate model
- by default, *weighted ridge regression*:

$$\hat{\beta}_n \in \underset{\beta \in \mathbb{R}^{d+1}}{\arg\min} \left\{ \sum_{i=1}^{n} \pi_i (y_i - \beta^\top z_i)^2 + \lambda \|\beta\|^2 \right\},$$

  with $\lambda > 0$ a *regularization constant*
- **Intuition:** if superpixel j receives a large positive weight, superpixel j important for the prediction
- **Computational cost:** n queries, then ridge for $n \times d$ data with $n \gg d$: $\mathcal{O}\left(d^2 n\right)$
- **Remark (i):** when no penalty is used, falls into the SHAP framework[24]
- **Remark (ii):** a lot of flexibility in the LIME framework, another model / penalty could be used

[24] Lundberg and Lee, *A unified Approach to Interpreting Model Predictions, NeurIPS, 2017*

Image LIME theory

- **Question:** does Image LIME make sense for simple models?
- surprisingly complicated question, some simplifications:
  - $\lambda = 0$ (no penalty);
  - f is bounded.
- then it is possible to show[25] that, when $n \to +\infty$,

$$\hat{\beta}_n \xrightarrow{\mathbb{P}} \beta \,,$$

  where $\beta \in \mathbb{R}^{d+1}$ is a vector depending only on f, ξ, and ν
- **Notation:** in the following, z random variable such that the z_is are i.i.d. z, associated x, π

[25] G. and Mardaoui, *What does LIME really see in images?*, arxiv, 2021

Image LIME theory, ctd.

- moreover, the expression of β is *explicit*:

Proposition: There exist constants c_d, σ_1, σ_2, and σ_3 such that

$$\beta_j^f = c_d^{-1}\left\{\sigma_1 \mathbb{E}\left[\pi f(x)\right] + \sigma_2 \mathbb{E}\left[\pi z_j f(x)\right] + \sigma_3 \sum_{\substack{k=1 \\ k\neq j}}^{d} \mathbb{E}\left[\pi z_k f(x)\right]\right\}.$$

- c_d, σ_1, σ_2, and σ_3 can be exactly computed and do not depend on f
- simple expression in the *large bandwidth limit* ($\nu \to +\infty$):

$$\beta_j \approx 2\left(\mathbb{E}\left[f(x) \mid z_j = 1\right] - \mathbb{E}\left[f(x)\right]\right).$$

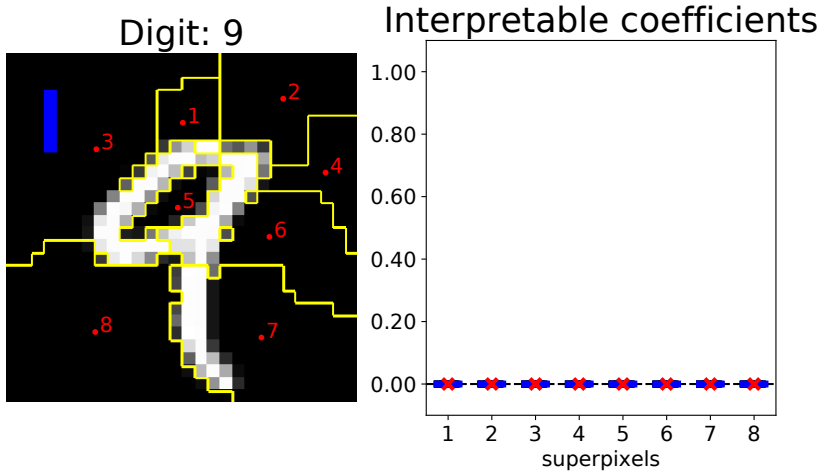
- **Intuition:** large value if the model takes significantly larger values when superpixel j is present

Shape detection

- we can be more precise for specific models, for instance shape detectors:

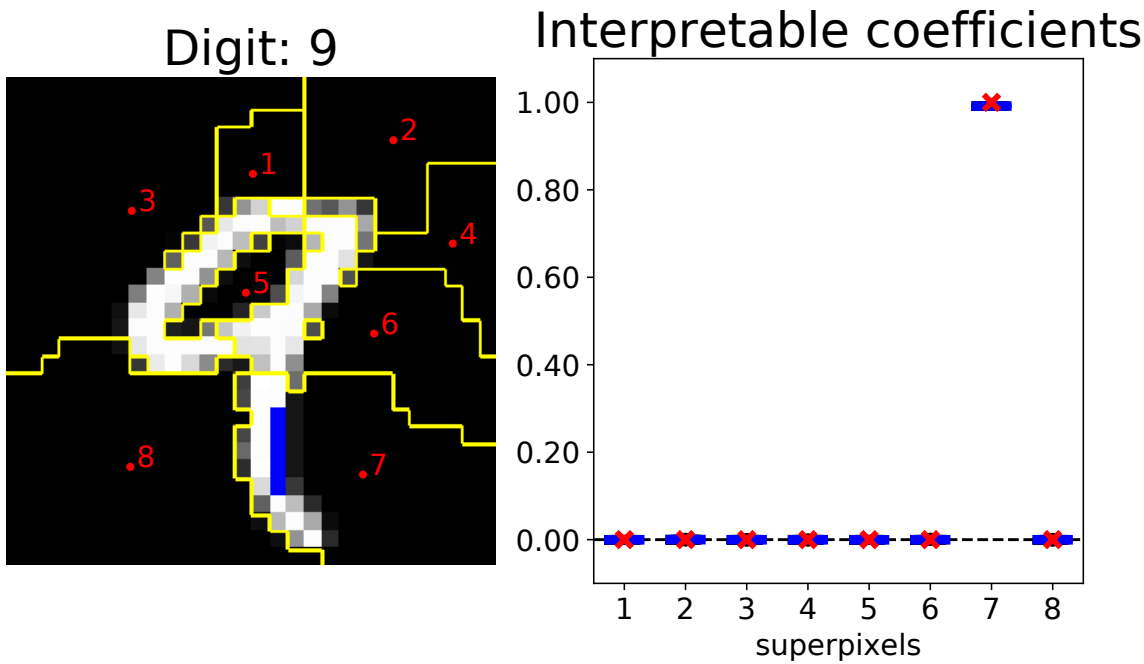
$$f(x) = \prod_{u \in \mathcal{S}} \mathbb{1}_{x_u > \tau} \,,$$

  where $\mathcal{S}$ is a given set of pixels and τ is a positive threshold
- f takes value 1 if the shape $\mathcal{S}$ is lit up in image x
- then LIME approximately counts how many time $\mathcal{S}$ intersects the superpixels (if $f(\xi) = 1$)
- **Example:** rectangular shape, MNIST dataset, zero replacement:



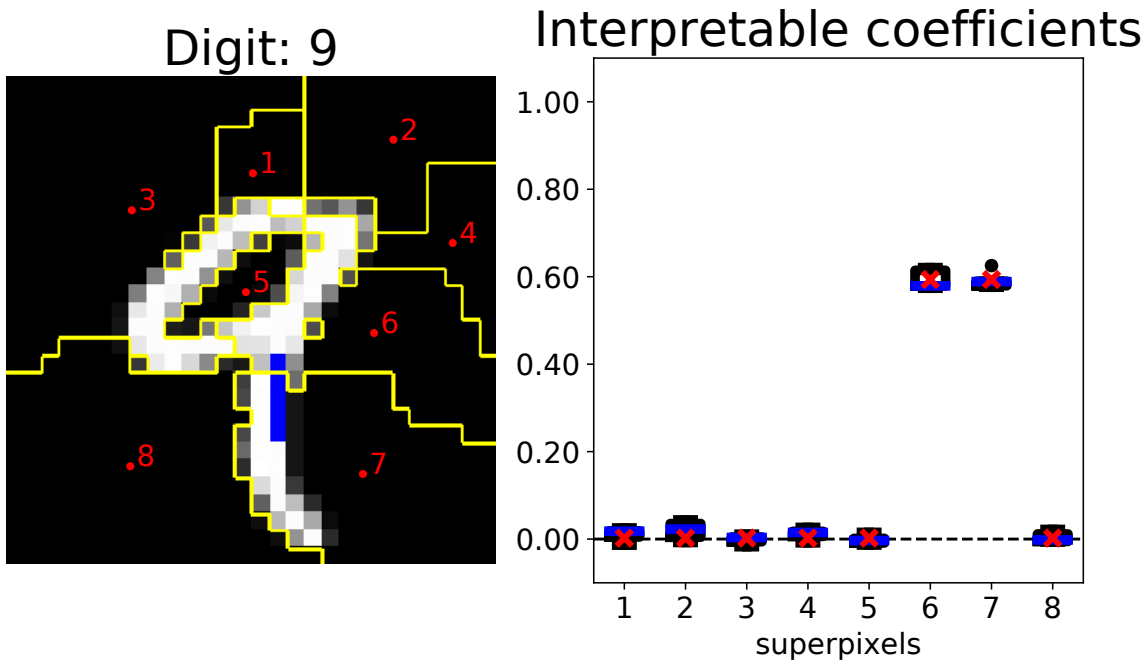
Shape detection, ctd.

- **Example:** same digit, $\mathcal{S}$ intersects one superpixels:



Shape detection, ctd.

- **Example:** same digit, $\mathcal{S}$ intersects two superpixels:



- this seems to make a lot of sense!

Linear models

- **Question:** what about linear models?
- recall that we set

$$f(x) = \sum_{u=1}^{D} \lambda_u x_u + b\,,$$

with $\lambda \in \mathbb{R}^D$ and $b \in \mathbb{R}$

Proposition: assume that f is linear. Then

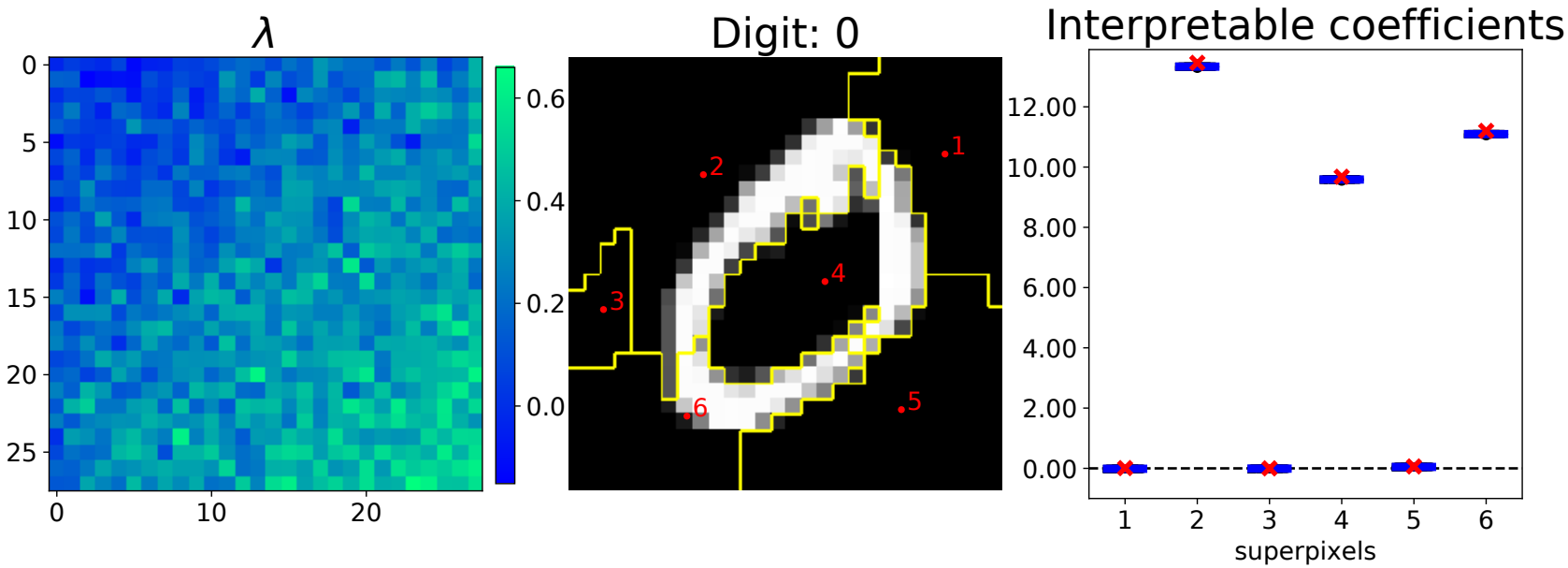
$$\beta_j = \sum_{u \in J_j} \lambda_u \cdot (\xi_u - \overline{\xi}_u)\,,$$

where $\overline{\xi}$ is the replacement image.

- **Intuition:** sum of gradient $\times$ input on the superpixels

Linear models, ctd.

- **Example:** linear function on MNIST with arbitrary coefficients:



- again, seems to make a lot of sense

More complex models

- difficult to extend the analysis
- **However,** if we replace f by a linear approximation, we see empirically that

$$\beta_j \approx \sum_{u \in J_j} \mathrm{IG}_u \cdot (\xi_u - \overline{\xi}_u)\,,$$

where IG is the integrated gradients previously defined

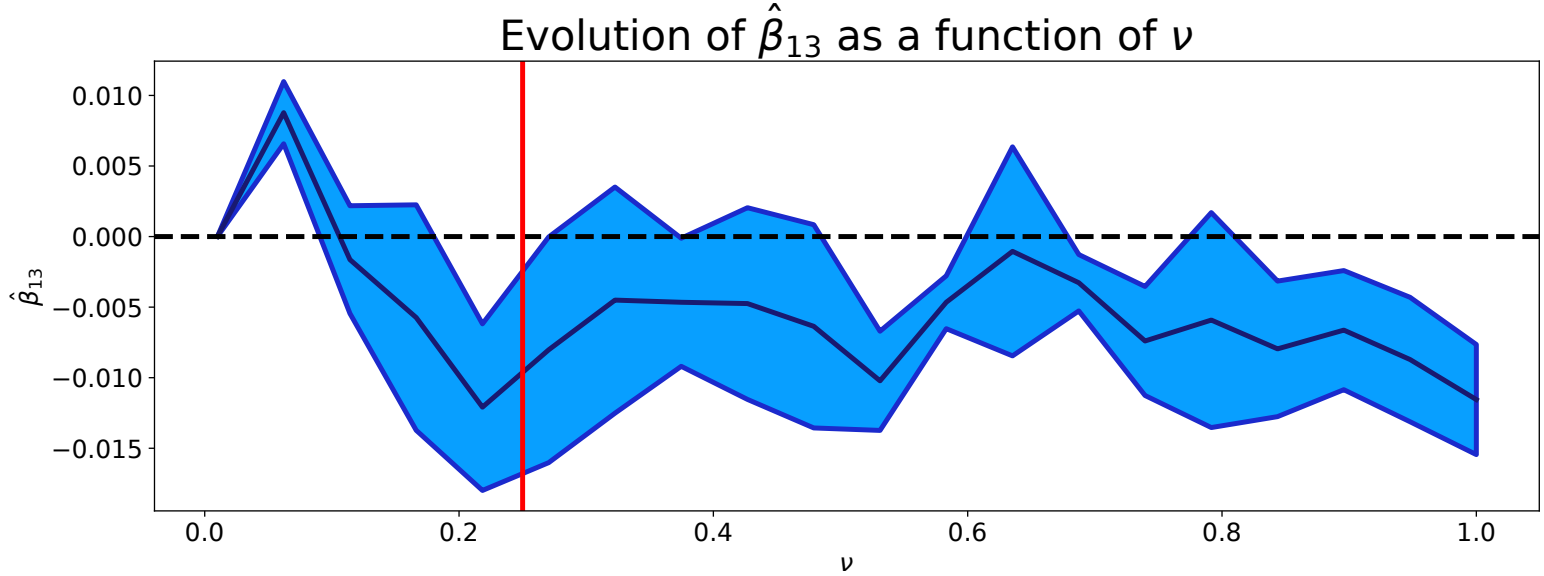
LIME

int. gradient

linear approx.

Bandwidth cancellation

- ν is essentially the only free parameter of the method
- what happens when we vary it?



- **Figure:** explanation for superpixel 13, ILSVRC dataset, InceptionV3 model, 10 repetitions for each ν
- default if 0.25 (in red)
- **Undesirable behavior:** explanations change sign when ν changes

4.2. Text data

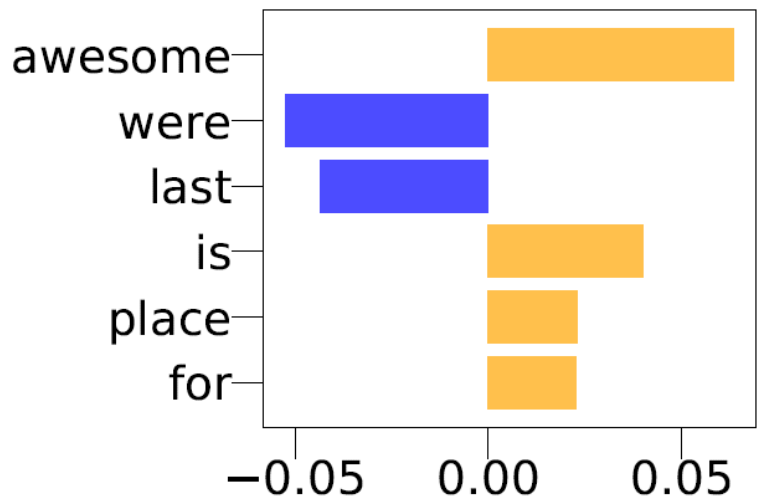
Text LIME

- **Question:** what happens with text data?
- same principle, but key difference: no superpixels, data goes through a vectorizer first
- formally, vectorizer $=$ function ϕ that transforms any text into a vector of $\mathbb{R}^D$
- D is the size of the dictionary (for instance)
- d is the size of the *local* dictionary
- Text LIME operates as follows:
  1. calibrate ϕ;
  2. create n perturbed samples;
  3. weight the perturbed samples;
  4. query the model;
  5. build a local surrogate model.

Text LIME, ctd.

- as before, display the top 5 coefficients
- **Example:** sentiment analysis

Update! Went back last night for dinner, this place is still awesome. I had the Las Vegas Rolls, they were pure deep fried goodness.



- orange = positive influence for the prediction, blue = negative influence

Step 1: TF-IDF transform

- by default, **TF-IDF** transform is used, product of:
  - TF = **T**erm **F**requency = how frequent a word is in the document;
  - IDF = **I**nverse **D**ocument **F**requency = how rare the word is in the corpus
- formally,

Definition: given a corpus $\mathcal{C}$ of N documents built on a dictionary of size D, the TF-IDF transform of a document δ is given by

$$\forall j \in \{1, \dots, D\}, \qquad \phi(\delta)_j := \frac{m_j v_j}{\sqrt{\sum_{k=1}^{D} m_k^2 v_k^2}},$$

where m_j is the number of occurrences of word j in δ and $v_j := \log(N+1)/(N_j+1) + 1$, with N_j the number of documents containing word j.

Step 1: TF-IDF transform, ctd.

- **Intuition:** $\phi_j \gg 0$ if word j is frequent in δ but not in $\mathcal{C}$
- **Example:**

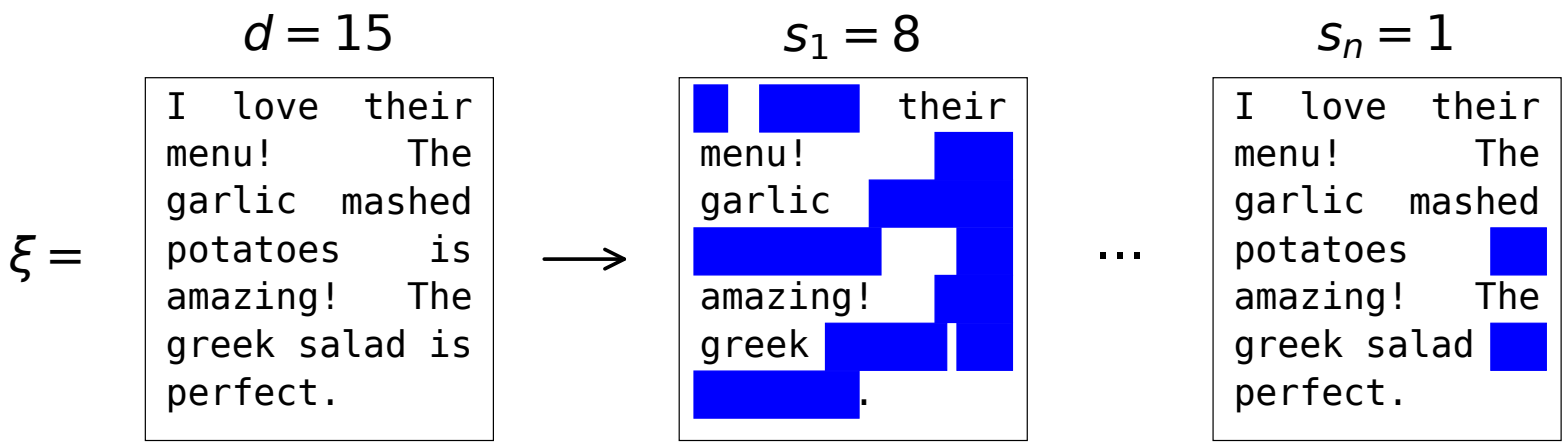
```
Love       this
place.     Great
staff!
Friendly,    fun
and
knowledgable.
The    food    is
always     good.
```

$$\begin{bmatrix} 0.1 \\ 0.05 \\ \cdot \\ \cdot \\ \cdot \\ 0.01 \\ 0.03 \end{bmatrix} \begin{matrix} \text{Love} \\ \text{food} \\ \\ \\ \\ \text{good} \\ \text{is} \end{matrix} \in S^{D-1}$$

- **Remark:** the v_js can be pre-computed on $\mathcal{C}$ (calibration step)

Step 2: sampling

- **Idea:** remove words at random in ξ
- more precisely: ξ has d distinct words $w_1, \ldots, w_d$
- for each $1 \leq i \leq n$, choose s_i words to remove, with s_i uniformly distributed on $\{1, \ldots, d\}$
- choose (uniformly) S_i random subset of $\{1, \ldots, d\}$ with $|S_i| = s_i$
- $\forall j \in S_i$, remove *all occurrences* of w_j from ξ
- **Example:**



Step 3-5: weights and surrogate model

- **Step 3:** compute the weights: same definition, with z_i indicating this time if a word is present (1) or not (0)
- **Step 4:** query the model: for each $i \in \{1, \ldots, n\}$, get

$$y_i = f(\phi(x_i)) \,.$$

- only difference: we must go through the discretizer
- **Step 5:** surrogate model: as before,

$$\hat{\beta}_n \in \underset{\beta \in \mathbb{R}^{d+1}}{\arg\min} \left\{ \sum_{i=1}^{n} \pi_i (y_i - \beta^\top z_i)^2 + \lambda \|\beta\|^2 \right\} .$$

Text LIME theory

- **Question:** does Text LIME make sense for simple models?
- same result holds[26]: when $n \to +\infty$,

$$\hat{\beta}_n \xrightarrow{\mathbb{P}} \beta\,,$$

where $\beta \in \mathbb{R}^{d+1}$ is a vector depending only on f, ξ, ϕ and ν

Proposition: there exist constants c_d, σ_1, σ_2, and σ_3 such that

$$\beta_j = c_d^{-1}\left\{\sigma_1 \mathbb{E}\left[\pi f(\phi(x))\right] + \sigma_2 \mathbb{E}\left[\pi z_j f(\phi(x))\right] + \sigma_3 \sum_{\substack{k=1 \\ k\neq j}}^{d} \mathbb{E}\left[\pi z_k f(\phi(x))\right]\right\}.$$

[26] Mardaoui and G., *An Analysis of LIME for Text Data,* AISTATS, 2021

Text LIME theory, ctd.

- same conclusion holds, *e.g.*, linearity of explanations, large bandwidth behavior
- more complicated to get simple models computations
- **Problem:** $\phi(\xi)$ changes in a non-trivial fashion when words are deleted from ξ
- we can still get some insights, for starters: *small decision trees*
- **Example:**

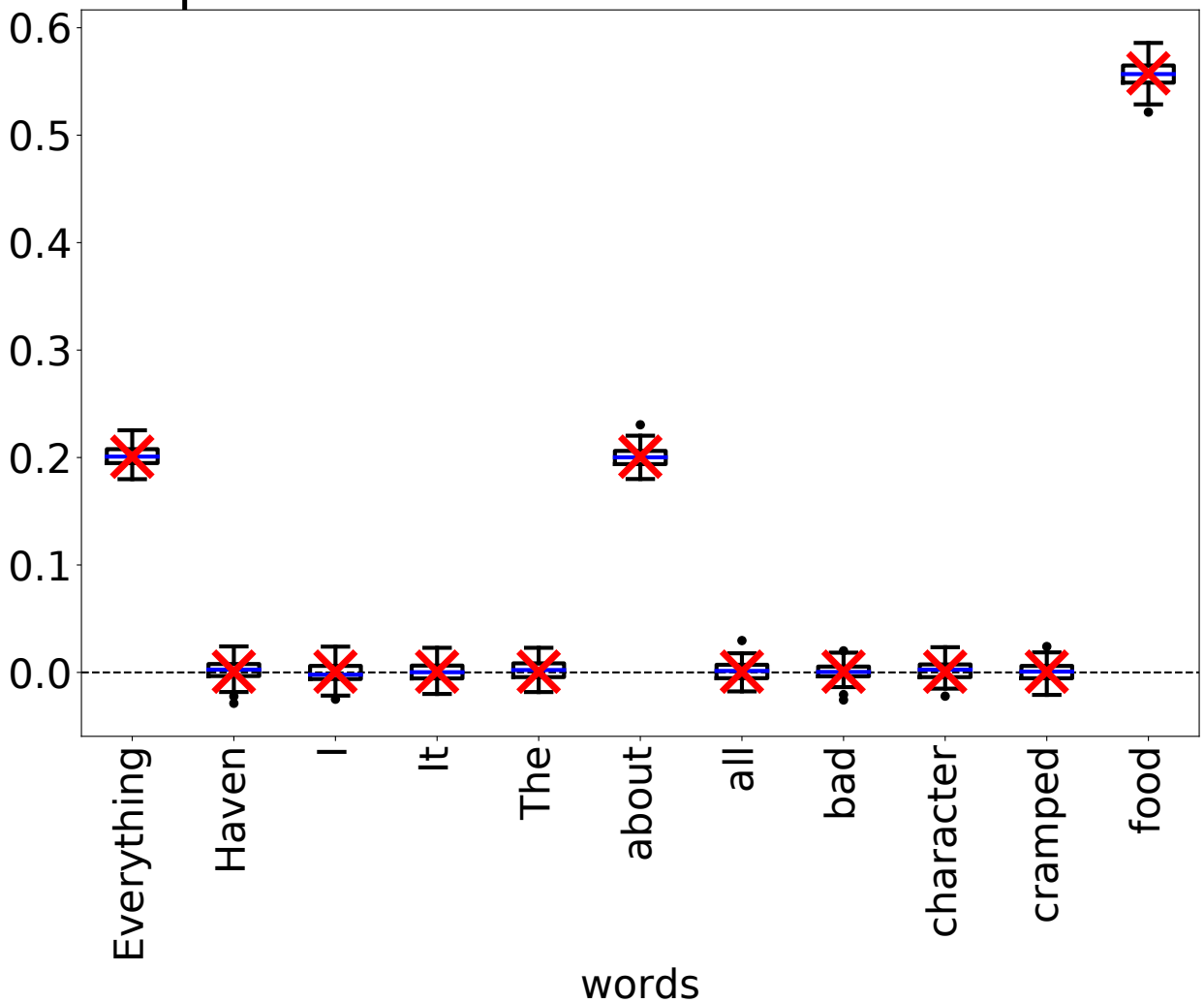
$$f(\phi(x)) = \mathbb{1}_{w_1 \in x} + (1 - \mathbb{1}_{w_1 \in x}) \cdot \mathbb{1}_{w_2 \in x} \cdot \mathbb{1}_{w_3 \in x} \,.$$

- **Intuition:** model gives high values if the document contains w_1 or w_2 and w_3
- for this kind of trees, LIME explanation for word $w_j \approx 1/$ depth of w_j in the tree

Decision trees

- **Example:**

Interpretable coefficients for a decision tree



Linear models

- **What about linear models?**
- that is,

$$f(\phi(x)) = \sum_{j=1}^{d} \lambda_j \phi(x)_j \,,$$

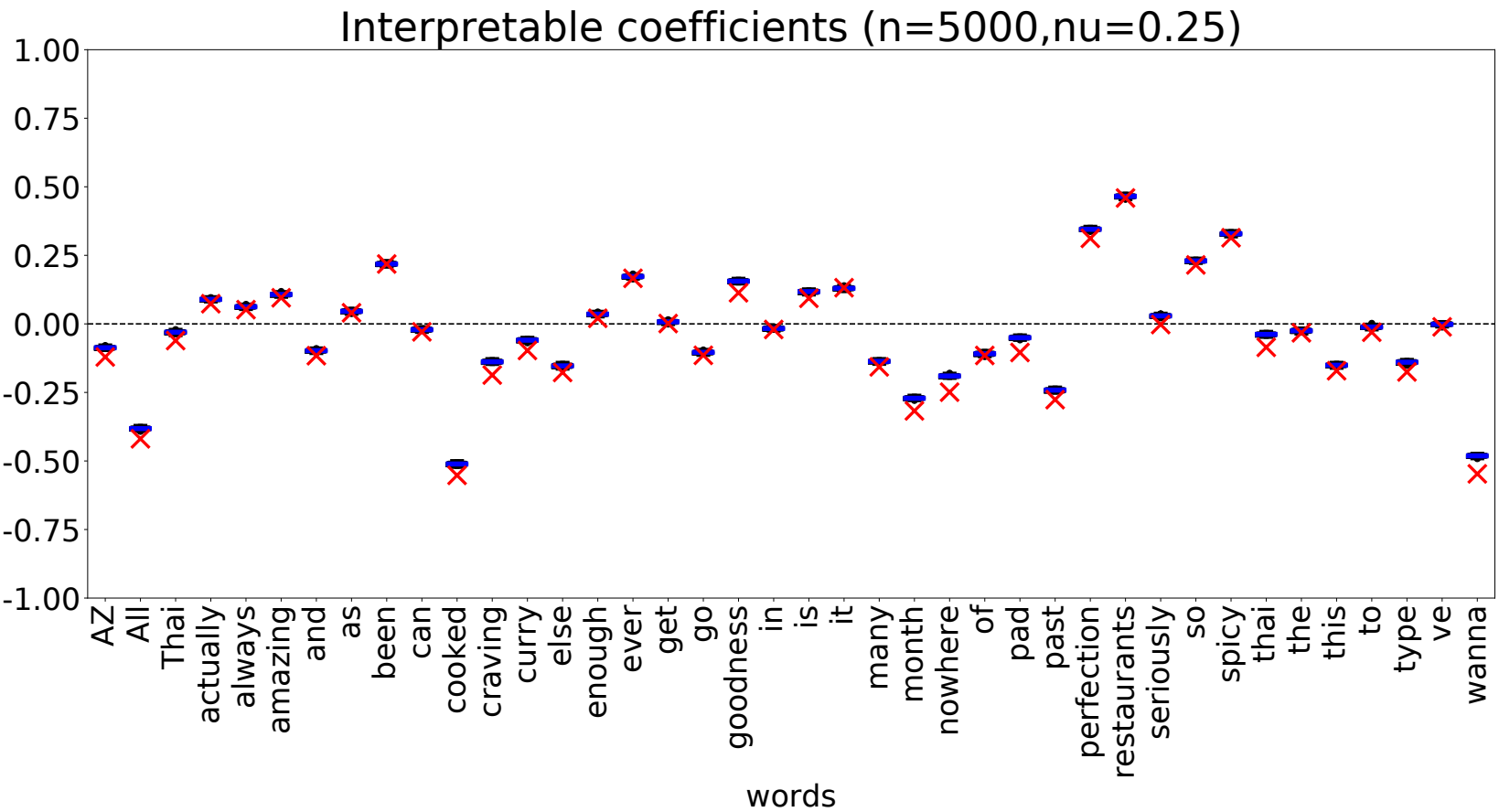
  with $\lambda \in \mathbb{R}^d$
- in that case, it is possible to show that

$$\beta_j \approx 1.36 \cdot \lambda_j \cdot \phi(\xi)_j \,.$$

- again, LIME seems to behave like gradient $\times$ input (at least approximately)

Linear models, ctd.

- **Example:** arbitrary linear model:



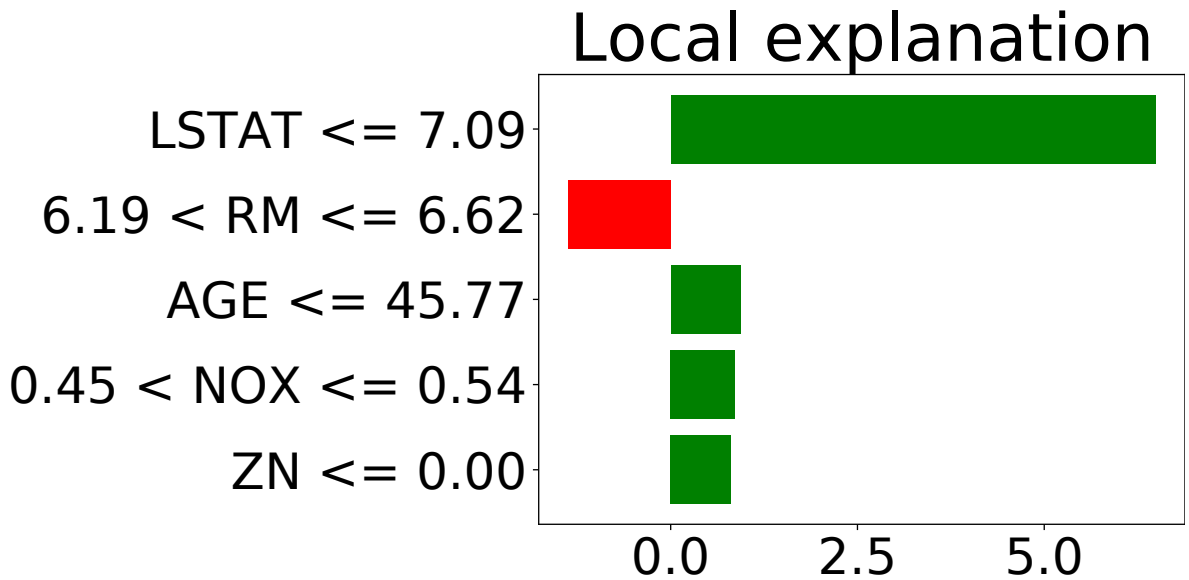
4.3. Tabular data

Tabular LIME

- **Question:** what happens with tabular data, *i.e.*, unstructured data in $\mathbb{R}^d$?
- key difference: discretization of the inputs *via* empirical quantiles
- ongoing debate: does it make sense to discretize the inputs?
- this talk = default version, with discretization
- need a training set $\mathcal{X}$
- high-level operation:
  1. create p boxes long each coordinate corresponding to the quantiles;
  2. sample box ids at random;
  3. sample truncated Gaussian in the boxes;
  4. weight;
  5. train a local surrogate model.

Tabular LIME, example

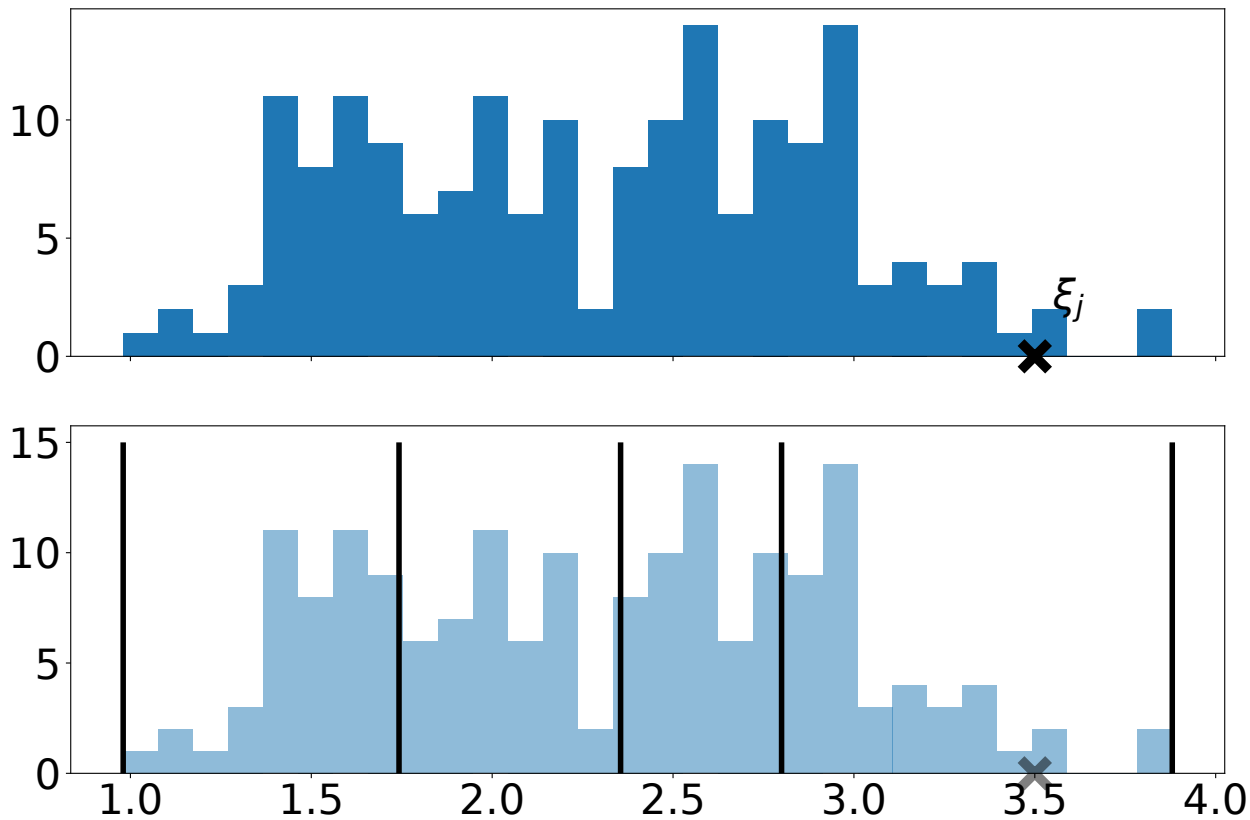
- **Example:** explaining a random forest classified on the Boston housing dataset



- green = positive influence on the prediction, red = negative influence on the prediction

Step 1: discretization

- project all the data on each coordinate and split in p boxes with equal number of observations
- by default, $p = 4$



Steps 2: bin sampling

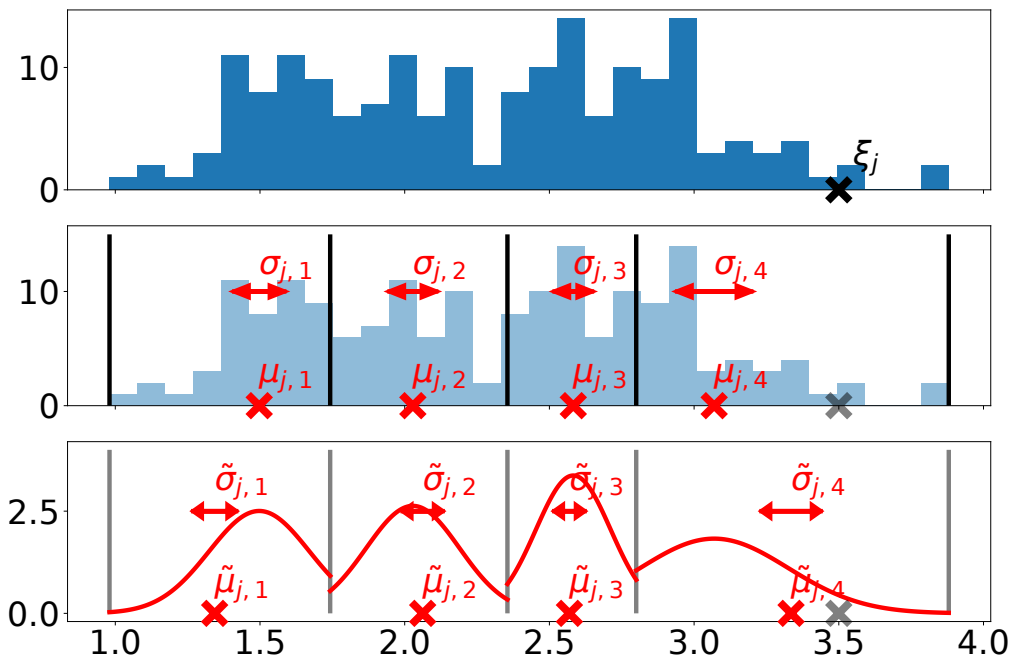
- for each new example i, sample box ids $b_i \in \{1, \ldots, p\}^d$ chosen uniformly at random
- on each coordinate j, there is a *special bin* containing ξ_j
- let $b_j^\star$ be the index of the special bin
- the interpretable features are defined as follows: $z_{i,j} = 1$ iff $b_{i,j} = b_j^\star$
- in other words, for tabular data, presence of the feature = falling into the same box as ξ along dimension j

Step 3: data generation

- on each bin, sample a *truncated Gaussian*:

$$\rho_{j,b}(t) := \frac{1}{\sigma_{j,b}\sqrt{2\pi}} \cdot \frac{\exp\left(\frac{-(t-\mu_{jb})^2}{2\sigma_{j,b}^2}\right)}{\Phi(r_{j,b}) - \Phi(\ell_{j,b})} \mathbb{1}_{t\in[q_{j,b-1},q_{j,b}]} ,$$

where μ and σ are computed on $\mathcal{X}$



Steps 4-5: weights and surrogate model

- other definition for the weights:

$$\pi_i := \exp\left(\frac{-\|\mathbf{1} - z_i\|^2}{2\nu^2}\right) .$$

- default bandwidth in this case is $0.75d$
- again, fit a linear model on the z_is with weighted ridge regression:

$$\hat{\beta}_n \in \underset{\beta \in \mathbb{R}^{d+1}}{\arg\min} \left\{ \sum_{i=1}^{n} \pi_i (y_i - \beta^\top z_i)^2 + \lambda \|\beta\|^2 \right\} .$$

Tabular LIME theory

- **Question:** does Tabular LIME make sense for simple models?
- same result as before[27]: when $n \to +\infty$,

$$\hat{\beta}_n \xrightarrow{\mathbb{P}} \beta\,,$$

with $\beta \in \mathbb{R}^{d+1}$ depending only on f, ξ, p, and ν.

Proposition: there exist a constant c depending on p and ν such that

$$\beta_j = c^{-1}\left\{\frac{-pc}{pc-1}\mathbb{E}\left[\pi f(x)\right] + \frac{p^2c^2}{pc-1}\mathbb{E}\left[\pi z_j f(x)\right]\right\}\,.$$

[27] G. and von Luxburg, *Explaining the explainer: a first theoretical analysis of LIME*, AISTATS, 2020; G. and von Luxburg, *Looking Deeper into Tabular LIME*, arxiv, 2020

Tabular LIME theory, ctd.

- what about linear models?

Proposition:[28] assume that f is linear. Then

$$\beta_j = \frac{\lambda_j}{p-1} \sum_{b=1}^{p} (\tilde{\mu}_{j,b_j^\star} - \tilde{\mu}_{j,b}) \,,$$

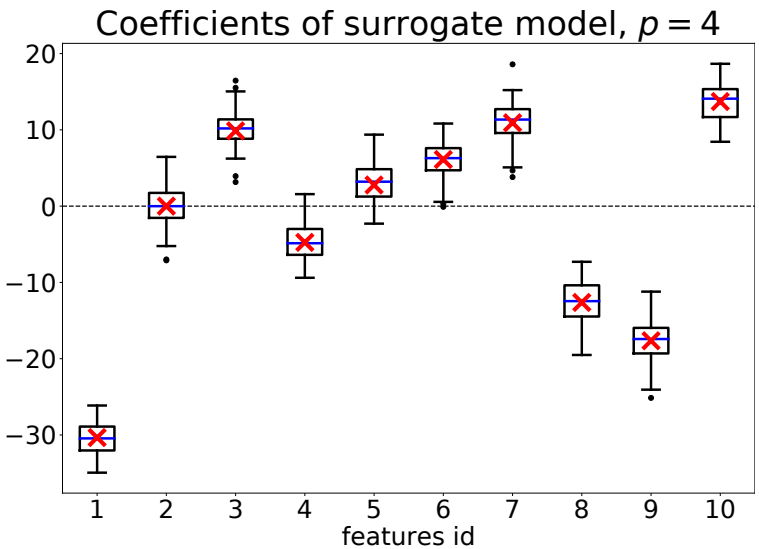
where $\tilde{\mu}_{j,b}$ is the mean of the truncated Gaussian on bin b for dimension j.

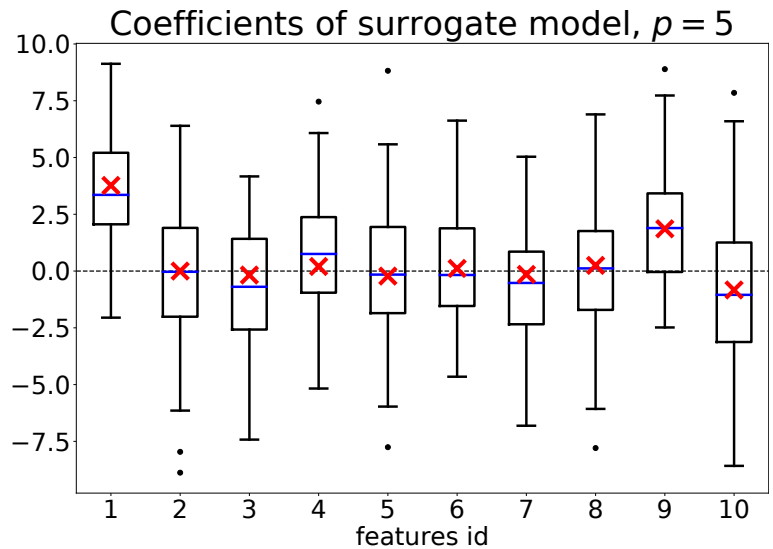
- intuitively, $\tilde{\mu}_{j,b} \approx \mu_{j,b}$ (the mean of the data)
- thus coefficient of the linear model $\times$ barycenter

[28] ibid

Cancellation phenomenon

- not good news: we can artificially cancel out the explanations





- **Figure:** linear f with uniform training data. Odd number of boxes cancels out the barycenter term.

More insights from the theory

- as for images and text, nice consequences:
  - linearity of the explanations:

$$\beta^{f+g} = \beta^f + \beta^g \,;$$

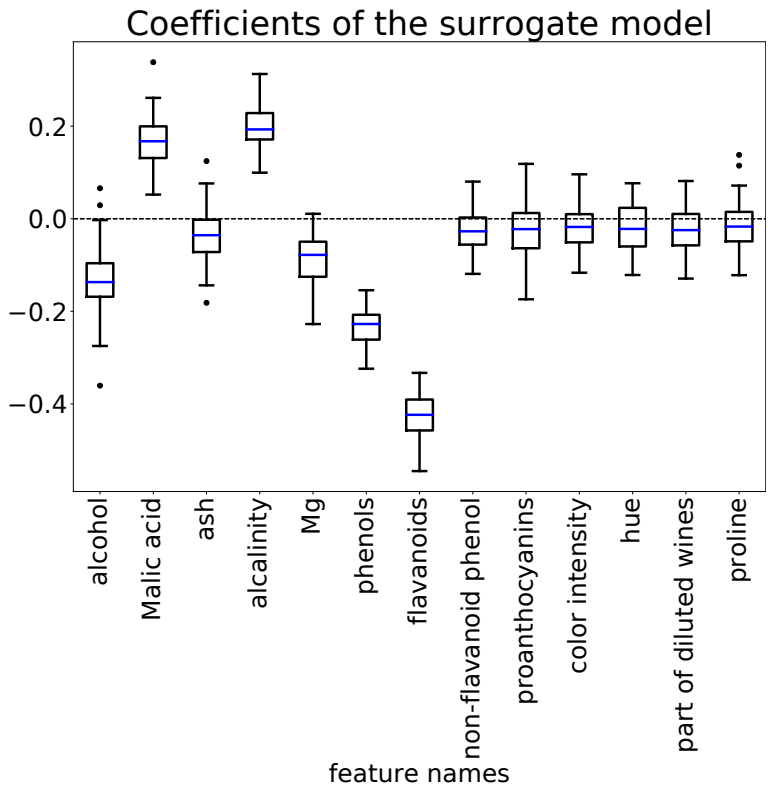
  - large bandwidth behavior:

$$\beta_j \approx \frac{p}{p-1}\left(\mathbb{E}\left[f(x) \mid b_j = b_j^\star\right] - \mathbb{E}\left[f(x)\right]\right);$$

  - ignoring unused coordinates
- but also some problems:
  - bandwidth cancellation (again);
  - explanations only depend on the box ids of the example to explain;
  - artifacts

Ignoring unused coordinates

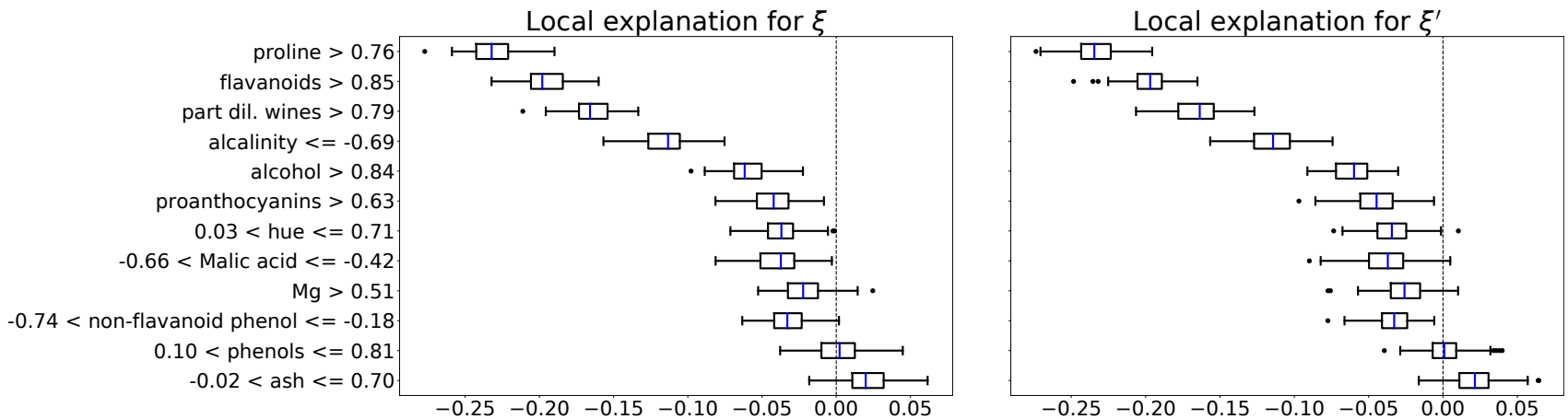
- **Example:** Wine dataset, training a kernel ridge regressor on a subset of the features



- up to noise coming from the sampling, LIME explanations are 0 for the unused coordinates

Explanations only depend on the box ids

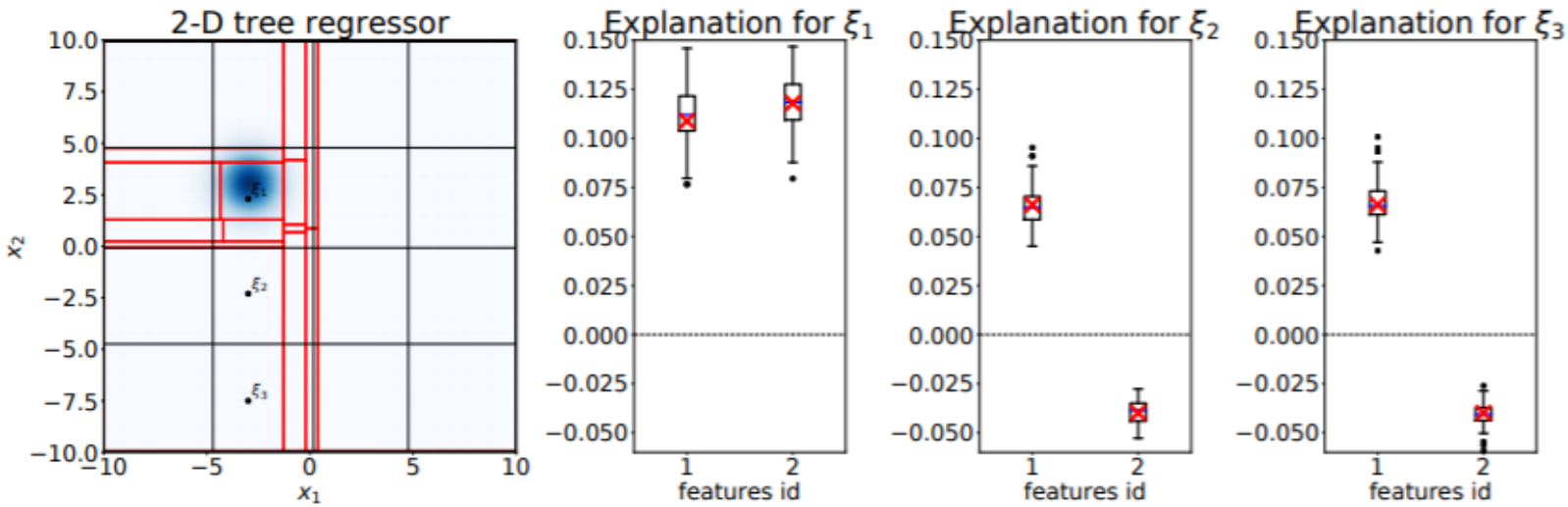
- **Example:** Wine dataset, ξ and ξ' fall into the same d-dimensional box



- up to noise coming from the sampling, LIME explanations are the same!

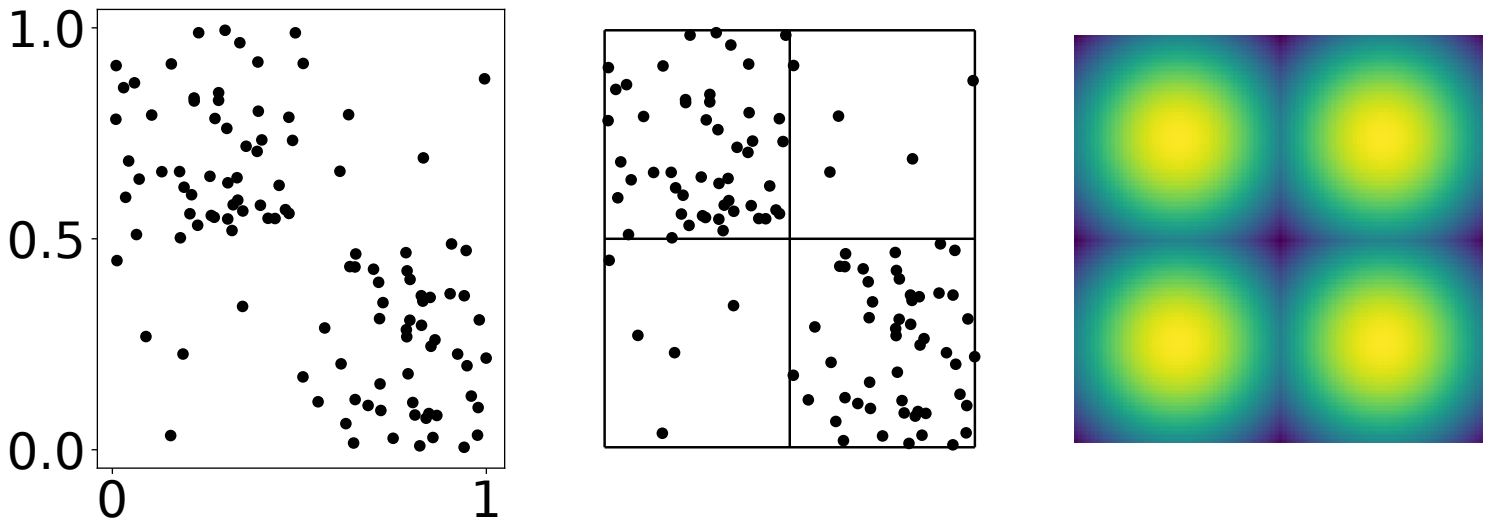
Artifacts

- when computing the weights, $\|\mathbf{1} - z_i\|$ give equal weight to samples that can be very far away
- **Example:** explaining a 2D kernel regressor trained on the blue bump



Problem with the sampling

- after the split the dependency between features is lost (here $p = 2$)



- **Figure:** *left panel:* training data, *middle panel:* quantiles on each feature, *right panel:* density of the LIME sampling.

Conclusion

Advantages:

- results easy to interpret
- moderate computational cost
- some guarantees for text and images

Drawbacks:

- different methods for each data type
- undesirable behaviors, especially for tabular data

Related work

- *if-then* rules[29]
- extensions to other data types:
  - *time series*[30]
  - *survival analysis*[31]
  - ...
- addressing the stability issue[32]

---

[29] Ribeiro et al., *Anchors: high-precision model-agnostic explanations*, AAAI, 2018

[30] Mishra et al., *Local Interpretable Model-Agnostic Explanations for Music Content Analysis,* ISMIR, 2017

[31] Kovalev and Utkin, *SurvLIME: a method for explaining machine learning survival models*, Knowledge-based systems, 2020

[32] Visani et al., *Statistical stability indices for LIME: obtaining reliable explanations for machine learning models*, arxiv, 2020

5. Future directions

Theoretical guarantees for explainable AI

- **Idea:** if our method does not work on a linear model, do we really trust it on a deep neural network?
- lots of work to be done
- hard to keep up with the flow of new papers

Fixing current methods

- for instance the sampling scheme in Tabular LIME
- also sub-optimal for images in some situations:

predicted: Band_Aid (25.4%)

LIME explanation

- **What is happening here?**
- the band-aid is contained in a superpixel, and the mean on the superpixel is very similar to the superpixel itself
- when sampling, it does not make a difference.

General conclusion

- **Disclaimer:** not an exhaustive view, look at the resources if you want to know more!
- explainability is a new concept
- already many methods proposed
  - Shapley values;
  - gradient-based;
  - LIME
- nice visual results, but do they make sense?

Thank you for your attention!

Resources

- an online book about interpretability: christophm.github.io/interpretable-ml-book/
- the NeurIPS 2020 tutorial on explainability: www.youtube.com/watch?v=dxPIkyo0eoI&ab_channel=ArtificialIntelligence
- NeurIPS 2017 debate opposing LeCun, Weinberger, Caruana, and Simard on the topic: www.youtube.com/watch?v=93Xv8vJ2acI&ab_channel=TheArtificialIntelligenceChannel