6. Perturbation-based approaches

Perturbation-based approaches

- **General idea:** perturb the input, see how the output behaves
- many ways of doing that, need to be specific
- we will cover LIME[29], SHAP[30], and Anchors[31]
- different operating procedure depending on the data-type
- methods in this section are **post-hoc, local**
- **Notation:** f = trained model, ξ = example to explain

[29] Ribeiro et al., *"Why should i trust you?" Explaining the predictions of any classifier*, ACM SIGKDD, 2016
[30] Lundberg, Lee, *A unified approach to interpreting model predictions*, NeurIPS, 2017
[31] Ribeiro et al., *Anchors: High-precision model-agnostic explanations*, AAAI, 2018

6.1. LIME for images

Image LIME

- **Example:** image classification, $f(x)$ = likelihood for image to contain a truck
- 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: trailer_truck (35.2%)

LIME explanation

Aparté: superpixels

- **Superpixels** = group of pixels of an image, generally sharing some similar color / texture
- many algorithms to do that, let us explain how **quickshift**[32] operates
- **Example:** quickshift with default hyperparameters applied on a truck image from the ILSVRC dataset (281×500):

[32] Vedaldi and Soatto, *Quick shift and kernel methods for mode seeking*, ECCV, 2008

Alternative approaches

- many other approaches
- just to name a few:

Felzenszwalb[2]

SLIC[3]

compact watershed[4]

[2] Felzenszwalb and Huttenlocher, *Efficient graph-based image segmentation*, International Journal of Computer Vision, 2004

[3] Achanta et al., *SLIC superpixels compared to state-of-the-art superpixel methods*, IEEE Transactions on Pattern Analysis and Machine Intelligence, 2012

[4] Neubert and Protzel, *Compact watershed and preemptive SLIC: On improving trade-offs of superpixel algorithms*, ICPR, 2014

Quickshift

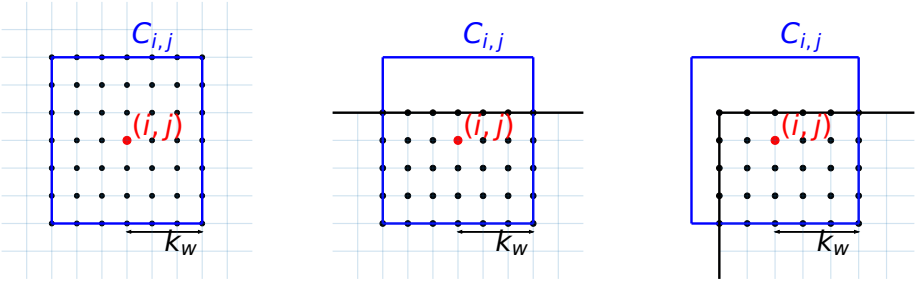
- **Notation:** our image ξ has height H and width W
- pixel location are denoted by $(i, j) \in I := [H] \times [W]$
- the $\xi_{i,j}$ are the pixel (RGB) values
- we see each pixel as a point in $\mathbb{R}^5$ (2 spacial coordinates, 3 color coordinates)
- **Two main steps:**
  1. *density estimation:* give a value $P_{i,j}$ to each pixel depending on the local properties of the image at (i, j)
  2. *graph construction:* build a directed graph $\mathcal{G}_0(P)$
- **Final output:** the superpixels are the connected components of the graph $\mathcal{G}_0(P)$

Density estimation I

- for a fixed (i, j), we do not want to look at the whole image
- (square) lookup area:

$$C_{i,j} := \{(u, v) \in [H] \times [W], \quad \text{s.t.} \quad |i - u| \wedge |j - v| \leq k_w\}$$

- k_w is the `kernel_width` parameter (15 by default)

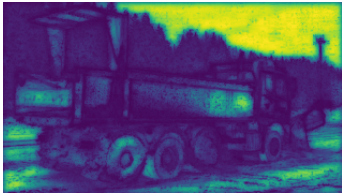


Density estimation II

- for all $(i,j) \in [H] \times [W]$, compute

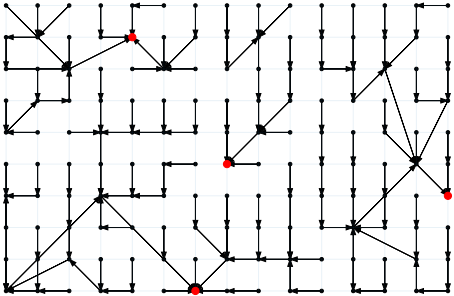
$$P_{i,j} := \sum_{(u,v)\in C_{i,j}} \exp\left(\frac{-(i-u)^2 - (j-v)^2 - \|\xi_{i,j} - \xi_{u,v}\|^2}{2k_s^2}\right)$$

- k_s is the `kernel_size` hyperparameter ($k_w/3 = 5$ by default)
- small i.i.d. Gaussian noise is added to break ties ($\sigma_0 = 10^{-5}$)



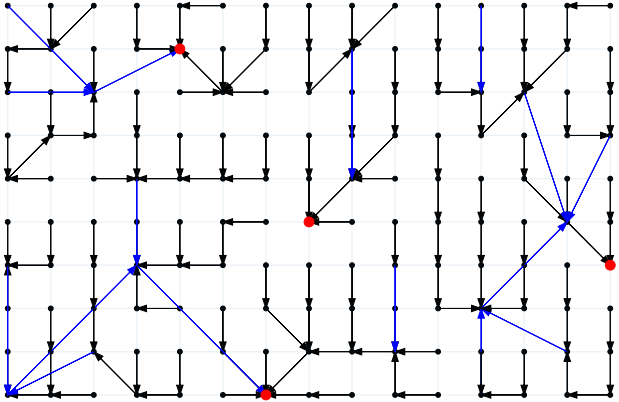
Graph construction I

- for each $(i,j) \in I$, at most one directed edge coming out of (i,j)
- draw an edge if
  1. $(u,v) \in C_{i,j}$
  2. $P_{i,j} < P_{u,v}$
  3. $(i-u)^2 + (j-v)^2 + \|\xi_{i,j} - \xi_{i,v}\|^2$ minimal
- no edge if $P_{i,j}$ achieves the max on $C_{i,j}$
- **Intuition:** find approximate local maxima of the density



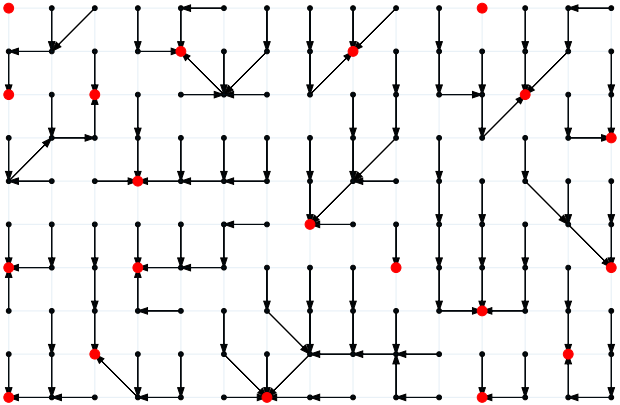
Graph construction II

- next, remove all edges with length greater than d_m (in blue below)
- d_m is the `max_dist` hyperparameter ($= 10$ by default)



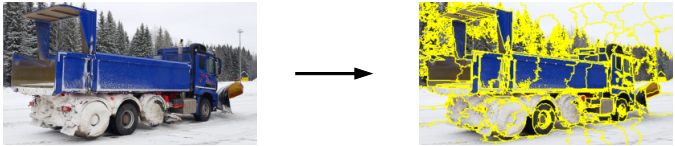
Graph construction III

- final output: connected components of $\mathcal{G}_0(P)$



Back to LIME, step 1: superpixels

- **Interpretable features:** superpixels $J_1, \ldots, J_d$ = sets of pixel indices
- $\cup_k J_k = [H] \times [W]$ and for all $j \neq k$, $J_k \cap J_\ell = \emptyset$
- by default, *quickshift*[33] is used



- in this example, $D = 299 \times 299 \times 3 = 268,203$ and $d = 65$ superpixel

[33] 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:

x_1

x_2

...

x_n

- 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.

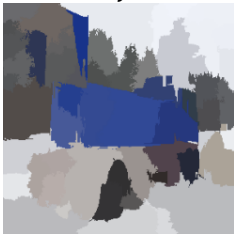
- **Idea:** compute the *mean* of the image on each superpixel
- formally,

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

- channel-wise if RGB image

segmented ξ

$\overline{\xi}$



Step 2: sampling, ctd.

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

x_1 x_2 ... x_n

- **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 $z_i \in \{0,1\}^d$
- $z_{i,j} = 1 \Leftrightarrow$ superpixel j is "switched on"
- $\mathbf{1}$ corresponds to ξ
- formally, for all $1 \leq i \leq n$, 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 $u, v \in \mathbb{R}^d$, define the *cosine distance* between u and v by

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

Step 3: weights, ctd.

- **Idea:** give more weight to samples near ξ

original image ξ

x_1

x_2

- many superpixels "switched off" in x_2
- $\Rightarrow$ far away from the original image
- $\Rightarrow$ small weight

Step 4: query

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

$y_1 = f(x_1) = 0.01$

$y_2 = f(x_2) = 0.04$

...

$y_n = f(x_n) = 0.55$

- cost $= \mathcal{O}(n)$ (n calls to the model)
- generally the main computational cost
- **Remark:** can be parallelized

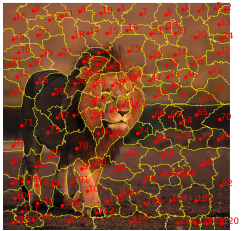
Step 5: local surrogate model

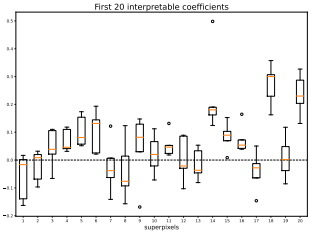
- finally, train a *local surrogate model*
- by default, *weighted* ridge regression:

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

  with $\lambda > 0$ a *regularization constant*
- each superpixel receives a coefficient β_j
- **Intuition:** if $\beta_j \gg 0$, superpixel j has a positive influence on 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:** other penalty can be used (LASSO)

Visualization





- **Figure:** (left) superpixels with ids; (middle) output of LIME; (right) boxplot with LIME coefficients for each superpixel, several runs, orange horizontal bar = mean

Properties of LIME for images

- **Notation:** in the following, Z random variable such that the Z_is are i.i.d. Z
- associated X, π: X is a random variable with distribution given LIME perturbation process

Proposition:[34] There exist (explicit) constants c_d, σ_1, σ_2, and σ_3 such that,

$$\forall j \in [d], \quad \hat{\beta}_j \longrightarrow \beta_j := c_d^{-1}\left\{\sigma_1 \mathbb{E}\left[\pi f(X)\right] + \sigma_2 \mathbb{E}\left[\pi Z_j f(X)\right] + \sigma_3 \sum_{\substack{k=1 \\ k \neq j}}^{d} \mathbb{E}\left[\pi Z_k f(X)\right]\right\}.$$

- **Intuition:** LIME explanations **stabilize** when number of perturbed examples $\to +\infty$

[34] Garreau, Mardaoui, *What does LIME really see in images?*, ICML, 2021

Consequences

- **First consequence:** up to noise from the sampling, the explanations are linear in the model:

$$\hat{\beta}(f+g) \approx \hat{\beta}(f) + \hat{\beta}(g)\,.$$

- good property: each model contributes proportionally
- **Second consequence:** 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 in the image
- this corresponds to our intuition

Consequences

- **Question:** what happens for linear models?
- let us set

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

Proposition:[35] assume that f is linear. Then

$$\forall j \in [d], \quad \beta_j = \sum_{u \in J_j} \lambda_u \cdot (\xi_u - \overline{\xi}_u)\,,$$

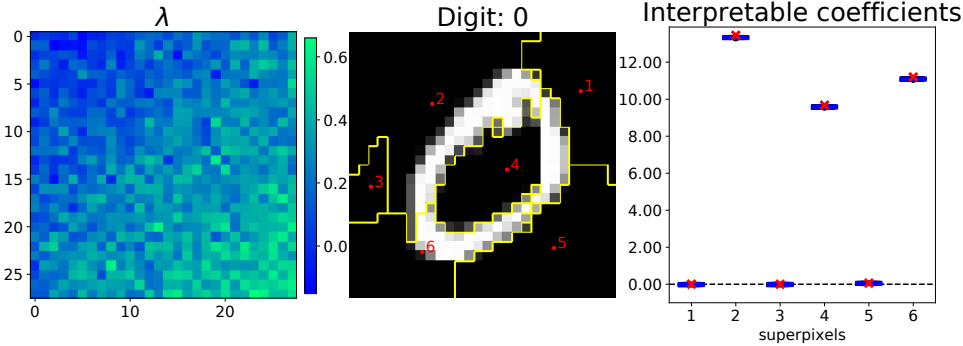
where $\overline{\xi}$ is the replacement image.

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

[35] Garreau, Mardaoui, *ibid.*

Linear models, ctd.

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

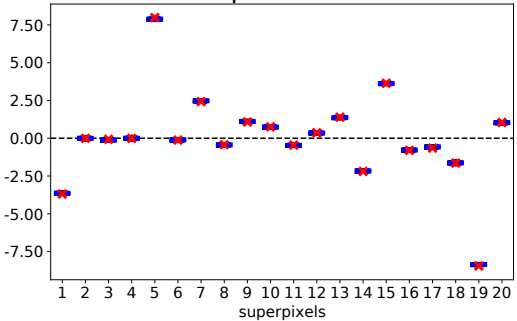


Linear models, ctd.

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

segmentation

first 20 interpretable coefficients



Limitations of image LIME

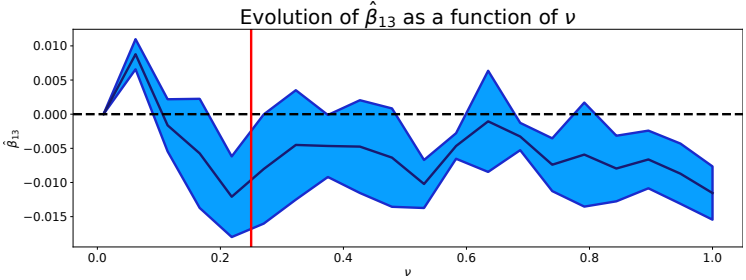
- **Sampling:** if the superpixel is very similar to the replacement superpixel...
- ...switching it on and off does not change much to the prediction
- LIME cannot learn that this pixel is important for the prediction
- even though it may be!

predicted: Band_Aid (25.4%)

LIME explanation

Limitations of image LIME

- **Choice of hyperparameters,** *e.g.*, choice of ν
- **Question:** what happens when we vary it?



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

Summary

- LIME for images
  - creates **interpretable features as superpixels**
  - randomly perturbs ξ
  - builds a **local (linear) surrogate model**
  - **output** = **coefficients** of the linear model
- makes sense in the limit, especially for linear models
- although sampling and hyperparameter choice can create issues
- many other similar methods, *e.g.*, RISE[36]

[36] Petsiuk, Das, Saenko, [37], BMVC 2018

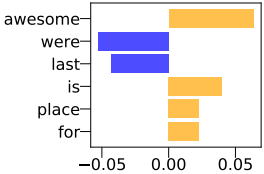
6.2. LIME for text data

Text LIME

- **Example:** sentiment analysis, $f(x)$ = likelihood to be a positive text
- on a high level, very similar operating procedure (differences in **bold**):
  1. **extract** d **unique words from** ξ;
  2. create a number of **perturbed samples** (= new documents) $x_1, \ldots, x_n$;
  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 individual words.
- generally, highlight in the original document the (top 5) positive words:

Explaining a prediction with LIME

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



Step 1: interpretable features

- **Similar idea:** need interpretable features
- **First step of any NLP pipeline:** tokenization
- = transform text in ordered sequence of elements of the dictionary
- usually gets rid off punctuation, capital letters, creates *special tokens*:

  `UNK` = unknown word, `BOS` = beginning of sentence, `EOS` = end of sentence, etc.

- **Remark (i):** unique words of the English language $\rightarrow D \approx 10^5$
- **Remark (ii):** many other possibilities, *e.g.*, Byte Pair Encoding[38]
- but unique words are more **interpretable** by the user

[38] Sennrich, Hadow, and Birch, *Neural machine translation of rare words with subword units*, preprint, 2015

Step 1: interpretable features

- **Example:** in Pytorch:

```
import torchtext
from torchtext.data import get_tokenizer

tokenizer = get_tokenizer("basic_english")
tokens = tokenizer("this is a sentence")
```

- output:

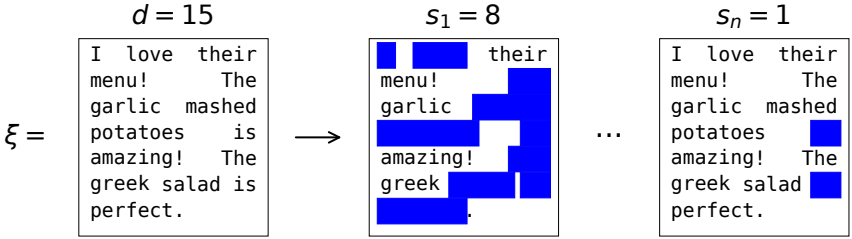
```
['this', 'is', 'a', 'sentence']
```

- we let $\mathcal{D}$ be the dictionary, $\mathcal{D} = \{w_1, w_2, \ldots, w_D\} \cong [D]$
- we let $\mathcal{D}_\ell$ be the **local** dictionary, without loss of generality

$$\mathcal{D}_\ell = \{w_1, w_2, \ldots, w_d\} \cong [d] \,.$$

Step 2: perturbed examples

- **Similar idea:** remove words at random
- "removing" = replacing by the `UNK` token
- "at random:"
  - for each $i \in [n]$, choose s_i words to remove (s_i uniformly distributed on $[d]$)
  - choose S_i a subset of $[d]$ of cardinality s_i (uniformly at random)
  - $\forall j \in S_i$, remove *all occurrences* of w_j from ξ
- associated binary features: $z_1, \ldots, z_n \in \{0,1\}^d$
- as before, $\mathbb{1}$ corresponds to ξ (no words removed)
- **Example:**



Step 3: weights

- **Same idea:** give small weight to perturbed examples far from ξ
- exact same definition:

$$\forall i \in [n], \quad \pi_i := \exp\left(\frac{-\delta(\mathbb{1}, z_i)^2}{2\nu^2}\right),$$

  with δ the cosine distance and $\nu > 0$ *bandwidth parameter*
- by default, $\nu = 0.25$
- **Intuition:**
  - if x_i close to ξ, then $d_{\cos}(\mathbb{1}, z_i) \approx 0$ and $\pi_i \approx 1$
  - if xi far away from ξ, then $d_{\cos}(\mathbb{1}, z_i) \approx 1$ and $\pi_i \approx 0$

Step 4-5: queries and local surrogate model

- queries and surrogate model as before:

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

  with $\lambda > 0$ a *regularization constant*
- each word receives a coefficient β_j
- **Intuition:** if $\beta_j \gg 0$, word j has a positive influence on the prediction
- **Computational cost:** n queries, then ridge for $n \times d$ data with $n \gg d$: $\mathcal{O}\left(d^2 n\right)$

Properties

- same limit behavior: when number of perturbed samples $\to +\infty$,

$$\hat{\beta}_j \longrightarrow \beta_j \,,$$

  with explicit β_j
- more can be said if we assume **TF-IDF vectorization**[39,40]
- assume corpus $\mathcal{C}$ with N documents
- **Vectorizer:** transform documents into vectors $\in \mathbb{R}^D$
- **TF-IDF** = product of two terms
  - TF: Term Frequency = how frequent a word is in the document
  - IDF: Inverse Document Frequency = how uncommon the word is in the corpus

[39] Luhn, *A statistical approach to mechanized encoding and searching of literary information*, IBM Journal of research and development, 1957

[40] Jones, *A statistical interpretation of term specificity and its application in retrieval*, Journal of documentation, 1972

TF-IDF

- formally:

Definition: let x be a document. We define the (normalized) TF-IDF transform of x as the vector $\phi(x) \in \mathbb{R}^D$ with components

$$\forall j \in [D], \quad \phi(x)_j = \frac{m_j v_j}{\sqrt{\sum_{j=1}^{D} m_j^2 v_j^2}},$$

where

- $m_j =$ number of occurrences of word w_j in x
- $v_j = \log\left(\frac{N+1}{N_j+1}\right) + 1$ where $N_j =$ number of documents in $\mathcal{C}$ containing w_j

- **Remark:** normalization is `scikit-learn` default, other choices are possible

Linear model

- assume that

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

Proposition:[41] assume that f is linear in the TF-IDF transform of the input. Then

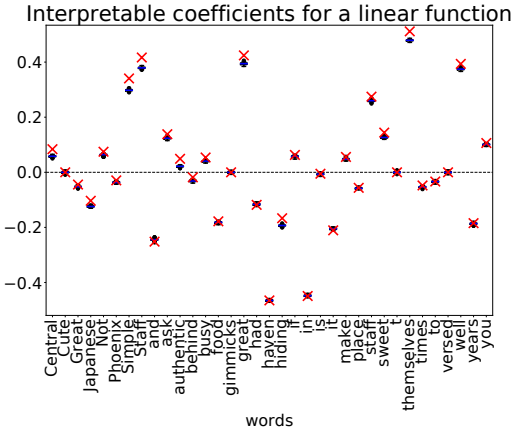
$$\forall j \in [d], \qquad \beta_j \approx 1.36 \cdot \lambda_j \cdot \phi(\xi)_j \,.$$

- **In other words:** up to a numerical constant, we approximately recover

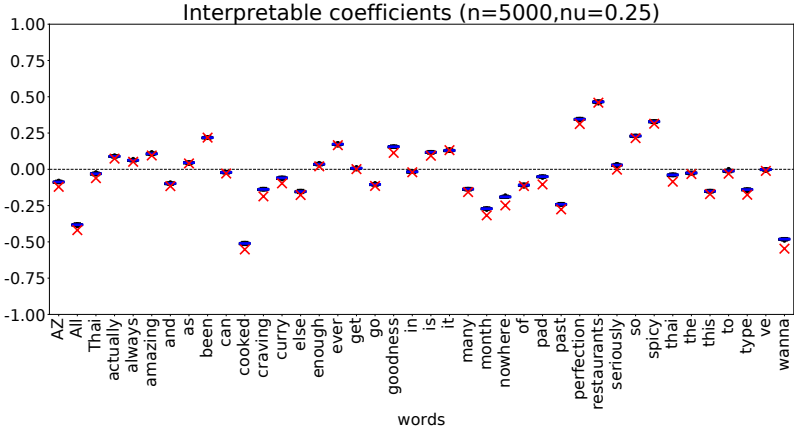
coefficient in the linear model $\times$ input coordinate

[41] Mardaoui, Garreau, *An Analysis of LIME for Text Data*, AISTATS, 2021

Linear models: example



Linear models: example



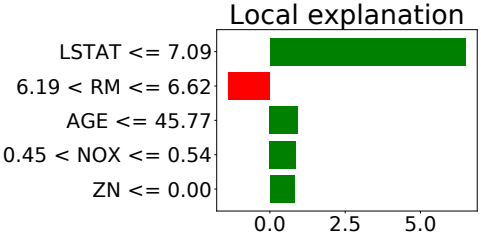
Summary

- LIME for text
  - creates **interpretable features as words**
  - perturbs ξ by randomly removing words
  - builds a **local (linear) surrogate model**
  - **output** = **coefficients** of the linear model
- makes sense in the limit, especially for linear models
- sampling and hyperparameter choice can create issues

6.3. LIME for tabular data

LIME for tabular data

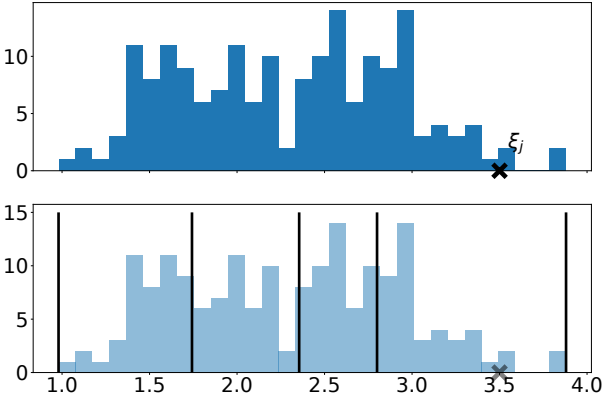
- **Example:** house pricing prediction



- similar high-level operation (differences in **bold**):
  1. **create interpretable features**
  2. **sample** n perturbed samples $x_1, \ldots, x_n$ from ξ
  3. weight the x_is
  4. train a local surrogate model
- one notable difference: assume that we have calibration set $\mathcal{X}_{\text{train}}$ at our disposal

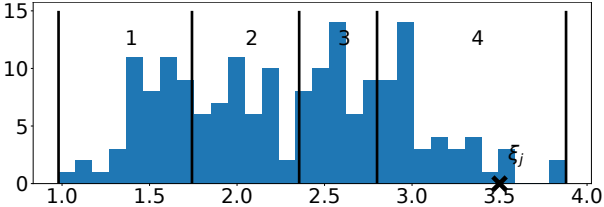
Step 1: interpretable features

- project $\mathcal{X}_{\text{train}}$ on each coordinate and split in p boxes with equal number of observations
- by default, $p = 4$ (quartile discretization)



Step 1: the special bin

- on each coordinate j, there is a *special bin* containing ξ_j
- $b_j^\star$ = index of the special bin
- **Example:** here $b_j^\star = 4$



Step 2: sampling

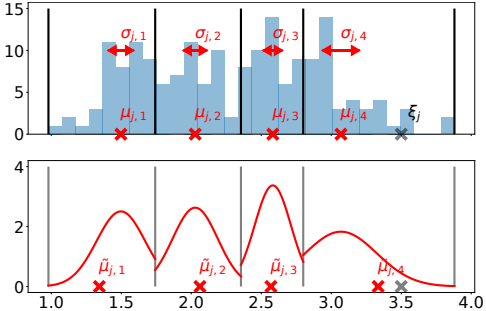
- **Reminder:** we are now sampling $x_1, \ldots, x_n$
- by default $n = 5,000$
- for each new example i, sample box ids $b_i \in \{1, \ldots, p\}^d$
- sampling is independent, uniform
- *interpretable features* defined as $z_{i,j} = 1$ iff $b_{i,j} = b_j^\star$
- **In plain words:** $z_{i,j} = 1$ if x_i falls in the same bin as ξ along feature j

Step 2: data generation

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

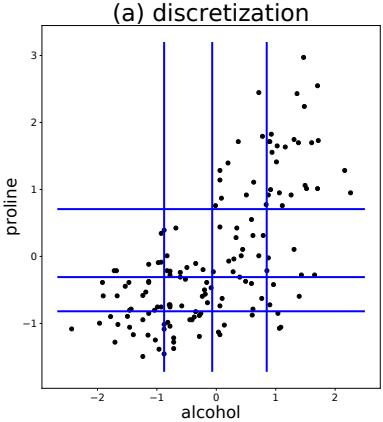
$$\rho_{j,b}(t) := \frac{1}{\sigma_{j,b}\sqrt{2\pi}} \cdot \frac{\exp\left(\frac{-(t-\mu_{jb})^2}{2\sigma_{j,b}^2}\right)}{\Phi(r_{j,b}) - \Phi(\ell_{j,b})} \mathbb{1}_{t\in[q_{j,b-1},q_{j,b}]}\,,$$

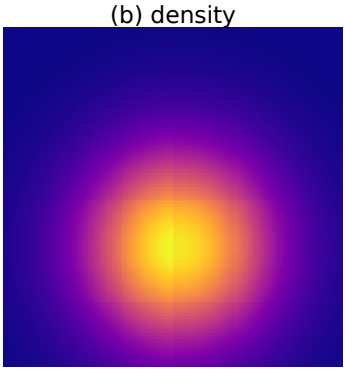
where μ and σ are computed on $\mathcal{X}_{\text{train}}$



Step 2: summary

- **Example:** visualizing the sampling density on a 2D slice of the Wine dataset





Step 3: weights

- **Reminder:** $z_{i,j} = 1 \Leftrightarrow x_i$ falls in the same bin as ξ on feature j
- again, $\mathbf{1}$ corresponds to ξ
- **Definition:**

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

- default bandwidth in this case is $\sqrt{0.75d}$
- **Intuition:** if x_i "far from ξ" then small weight
- **Remark:** not actually true (point can be very far!)

Step 4: surrogate model

- same surrogate model:

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

- **Intuition:** if $\left(\hat{\beta}_n\right)_j \gg 0$, then falling in the same bin as ξ on feature j probably important to predict $f(\xi)$
- **Remark:** many implementations do not consider the discretization step...
- ...but this is the default behavior if you rely on the official repository

Properties: limit behavior

- similar properties, in particular:

Proposition:[42] there exist a constant c depending on p and ν such that, when the number of perturbed samples goes to $+\infty$,

$$\hat{\beta}_j \longrightarrow \beta_j = c^{-1}\left[\frac{-pc}{pc-1}\mathbb{E}\left[\pi f(X)\right] + \frac{p^2c^2}{pc-1}\mathbb{E}\left[\pi Z_j f(X)\right]\right],$$

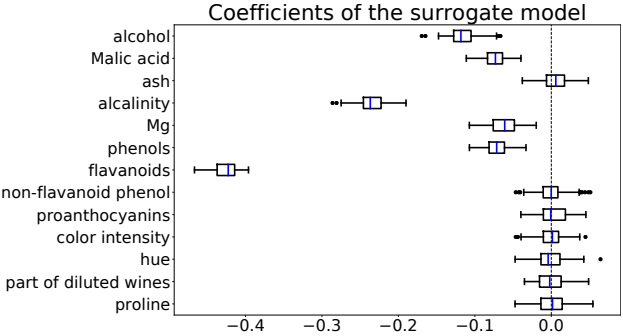
expectation taken w.r.t. $(X, \pi, Z) =$ distribution of (x_i, π_i, z_i).

- explanations stabilize when $n \to +\infty$

[42] Garreau, von Luxburg, *Explaining the explainer: A first theoretical analysis of LIME*, AISTATS, 2020

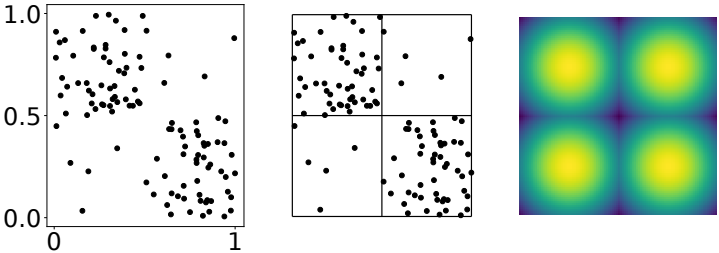
Properties: ignoring unused features

- **Question:** what happens if f is actually not using feature j to make predictions?
- one can show that $\beta_j^f = 0$ for all such j
- **Example:** kernel ridge regressor trained without the last 6 features



Limitations: sampling

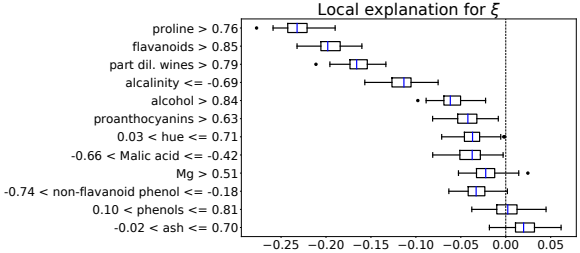
- after the split the **dependency between features is lost** (here $p = 2$)
- **Example:** *left panel:* training data, *middle panel:* quantiles on each feature, *right panel:* density of the LIME sampling

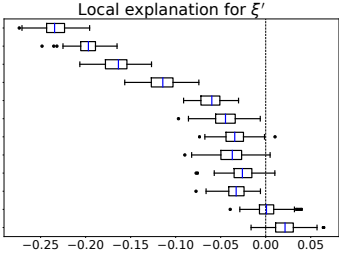


- we query the model in (nearly) empty areas

Limitations: discretization

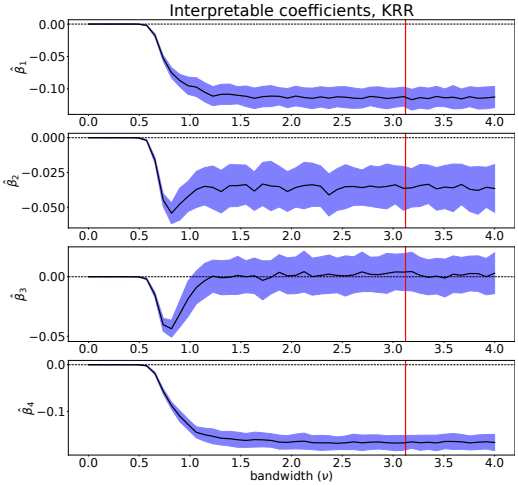
- $\mathcal{L}(X, \pi, Z)$ identical for any other ξ' in the same d-dimensional box
- **Consequence:** in the previous expression, ξ "disappears"
- up to noise, explanations are the same for examples falling into the same box
- **Example:** Wine dataset again





Limitations: hyperparameters choice

- **Same issue with the bandwidth:** (ξ is fixed, ν varies)



Limitations: linear models

- **Reminder:**

$$\forall x \in \mathbb{R}^d, \qquad f(x) = \sum_{j=1}^{d} \lambda_j x_j \,.$$

Proposition: assume that f is linear. Then

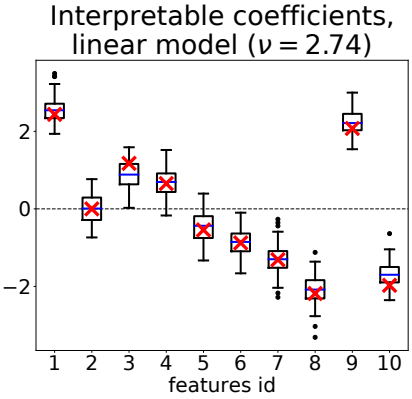
$$\beta_j = \lambda_j \times \frac{1}{p-1} \sum_{b=1}^{p} (\tilde{\mu}_{j,b_j^\star} - \tilde{\mu}_{j,b}) \,,$$

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

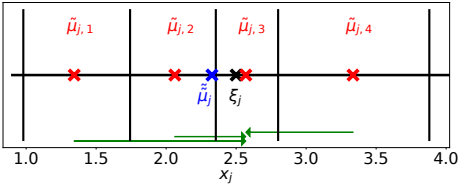
Limitations: linear models

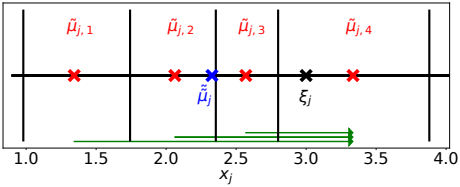
- **Example:** arbitrary linear model in dimension 10



Limitations: linear models

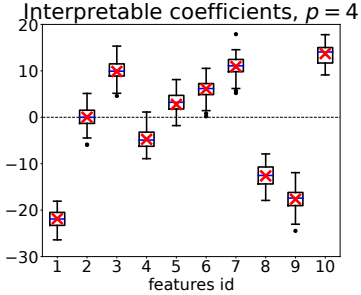
- **Recall:** $\beta_j = \lambda_j \times \frac{1}{p-1} \sum_{b=1}^{p} (\tilde{\mu}_{j,b_j^\star} - \tilde{\mu}_{j,b})$
- **In pictures:** coef = sum of the green arrows

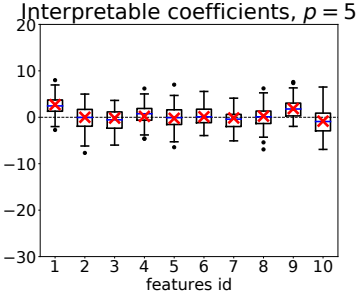




Limitations: linear models

- **Consequence:** one can artificially bring β_j to zero
- change, for instance, the number of boxes
- **Example:** same f, same ξ, going from $p = 4$ to $p = 5$





Summary

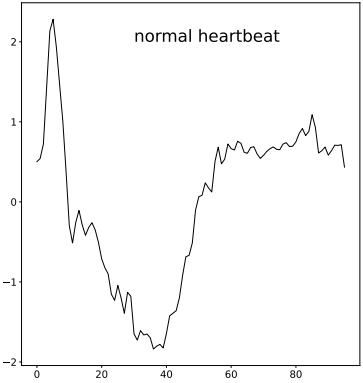
- LIME for tabular data
  - creates **interpretable features** as one-hot vectors
  - perturbs ξ by randomly perturbing feature values
  - builds a **local (linear) surrogate model**
  - **output** = **coefficients** of the linear model
- many issues coming from the discretization
- more and more typical to get rid of the discretization...
- ...but then it looks like gradient[43]

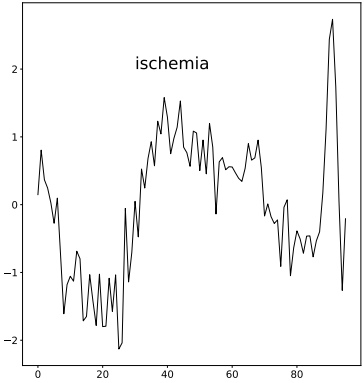
[43] Agarwal et al., *Towards the Unification and Robustness of Perturbation and Gradient Based Explanations*, ICML, 2021

6.4. LIMESegment

Time series classification

- **Time series:** ordered sequence of T observations
- **Example:**[44] ECG from one heartbeat, detect ischemia or not

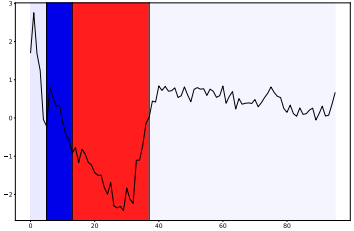




[44] Olszewski, *Generalized feature extraction for structural pattern recognition in time-series data*, Carnegie Mellon, 2001

LIMESegment

- **Idea:**[45] adapt the LIME framework to time series
- similar high-level operation (differences in **bold**):
  1. **create interpretable features**
  2. **sample** n perturbed samples $x_1, \ldots, x_n$ from ξ
  3. **weight** the x_is
  4. train a local surrogate model
- **Output:** highlight important parts of the time-series



[45] Sivill, Flach, *LIMESegment: Meaningful, Realistic Time Series Explanations*, AISTATS, 2022

Step 1: interpretable features

- **Interpretable features:** homogeneous segments in the time series
- standard problem (usually called *change-point detection*[46])
- proposed methodology: NNSegment
- relies on **normalized cross correlation:** let w_s be a fixed window size,

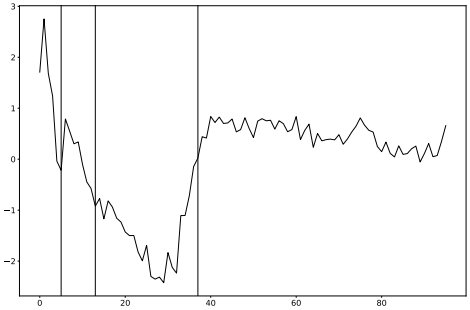
$$\forall s_1, s_2 \in [T - w_s], \qquad \psi(x_{s_1:(s_1+w_s)}, x_{s_2:(s_2+w_s)}) := \frac{\widehat{\mathrm{Cor}}(x_{s_1:(s_1+w_s)}, x_{s_2:(s_2+w_s)})}{\sqrt{\widehat{\mathrm{Var}}(x_{s_1:(s_1+w_s)})\widehat{\mathrm{Var}}(x_{s_2:(s_2+w_s)})}} .$$

- NNSegment computes all pairwise distances between segments
- then group them by nearest neighbor affinity
- further refinement looking at the signal to noise ratio

[46] Truong, Oudre, Vayatis, *Selective review of offline change-point detection methods*, Signal Processing, 2020

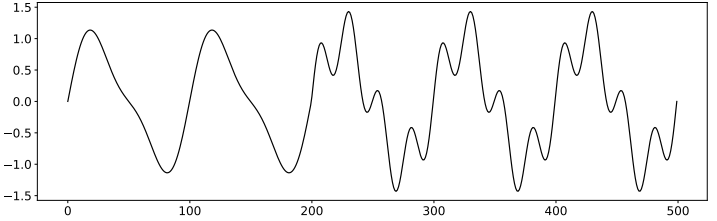
Step 1: interpretable features

- **Output:** segmented signal
- number of segments is user-specified (here 4, that is, 3 breaks)



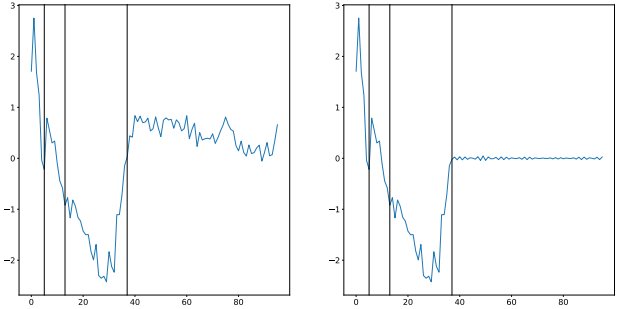
Step 2: perturbed examples

- **Idea:** identify background signal in the spectral domain
- **Discrete Short Time Frequency Transform (STFT):** $\rightarrow$ time-frequency domain
- **Example:** (local) spectrogram of superposition of sine waves



Step 2: perturbed samples

- identify a persistent frequency, map it back via inverse STFT
- **Example:** perturbing the last segment of the signal



Step 3: weights

- similar idea: exponential weights depending on a distance
- **Issue:** Euclidean distance between the z_i does not reflect distance between signals
- **Dynamic time warping (DTW):**[47] distance between signals taking alignment into account
- formally,

$$\mathrm{DTW}(x, x')^2 := \min_{\pi \in P(x,x')} \sum_{(i,i') \in \pi} d(x_i, x'_{i'}) \,,$$

  where π is an *admissible path*
- namely:
  - $\pi_1 = (1, 1)$ (beginning of signals matched together);
  - $\pi_K = (S, T)$ (end of signals matched together);
  - writing π_k as (i_k, i'_k), both i and i' are non-decreasing.

[47] Bellman, Kalaba, *On adaptive control processes*, IRE Transactions on Automatic Control, 1959

Summary

- then surrogate model as before (ridge), coefficients given as importance
- again, depends a lot on the segmentation / sampling scheme
- no existing theoretical analysis
- many other methods[48]

[48] see Theissler et al., *Explainable AI for Time Series Classification: A review, taxonomy and research directions*, for an overview

6.5. A game-theoretical perspective: Shapley values

Shapley values

- **Setting:** D-player game[49]
- 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:

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

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

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

Properties

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

But what is the link with interpretability?

[50] *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 $S \subseteq [D]$, retrain a model f_S only using the features in S

Definition:[51] the *Shapley regression value* associated to feature j is given by

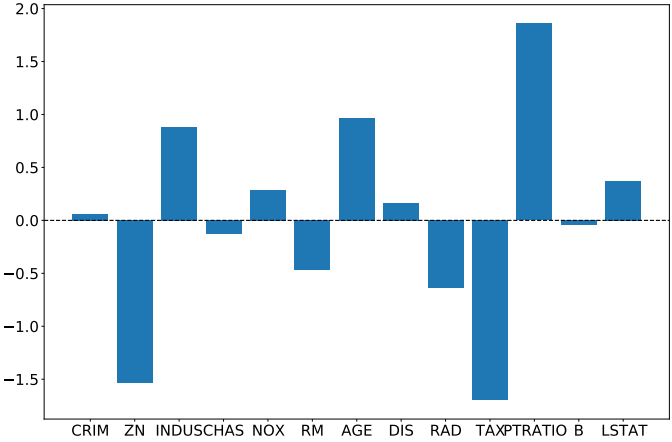
$$\phi_j := \sum_{S \subseteq \{1,\dots,D\} \setminus \{j\}} \frac{|S|!(D-|S|-1)!}{D!} \left(f_{S \cup \{j\}}(\xi_{S \cup \{j\}}) - f_S(\xi_S) \right) ,$$

where ξ_S is the restriction of ξ to S features.

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

Shapley regression values

- **Example:** output for linear regressor on Boston housing dataset



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*[52]
  - 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

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

Kernel SHAP

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

Definition (kernel SHAP)[53]**:** define ϕ as the minimizer of

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

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

Kernel SHAP

- can be seen 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 $\phi \in \mathbb{R}^d$

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

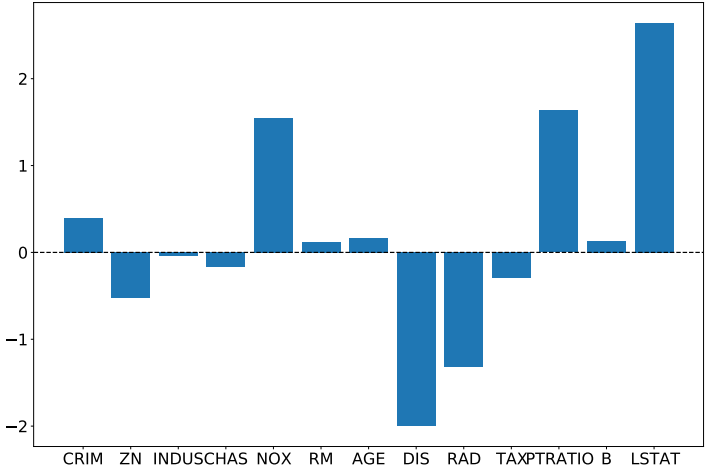
with

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

- **Remark:** very similar to LIME

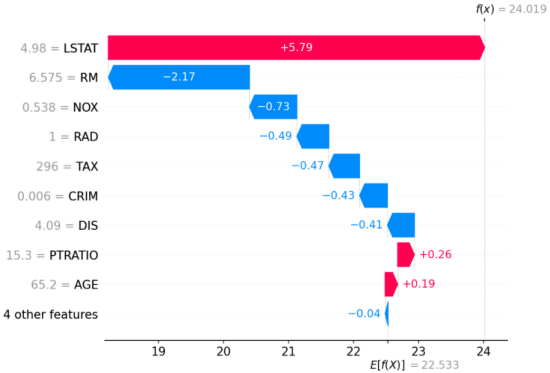
SHAP, tabular example

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



SHAP, tabular example

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



Kernel SHAP properties

- assume f is linear, that is,

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

Corollary:[54] If f is linear, then $\phi_0 = b$ and

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

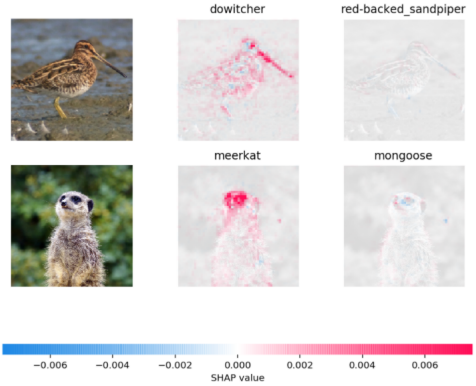
where $\overline{x}_j$ is the mean of feature j on the dataset.

- we recover the coefficients of the linear model multiplied by the (normalized) input

[54] ibid

Extensions

- Kernel SHAP is not restricted to tabular data
- **Example:** explaining the predictions of VGG16 for two classes



Summary

Advantages:

- Kernel SHAP can be used on any model
- can take advantage of specific architectures:
  - *TreeSHAP*[55] (tree-based predictors)
  - *DeepSHAP* (DeepLIFT[56] + Shapley values)

Inconvenients:

- costly to run[57]
- not easy to read if many features

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

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

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

6.6. Anchors

Notation and first definitions

- **Back to text:** ξ = document to explain = ordered sequence of tokens $(\xi_1, \ldots, \xi_T)$, f = classifier

Definition: we define an *anchor* A as an ordered subset of the words of ξ. We let $\mathcal{A}$ be the set of all possible *non-empty* anchors.

- two key definitions:
  1. *precision* = probability of same classification knowing that the document contains A
  2. *coverage* = how many documents in the dataset contain A
- one-sentence summary: find anchor with prescribed precision and maximal coverage

```
The selection on the
menu is great, and so
is the food! The
service is not bad,
prices are fine.
```

$\Longrightarrow$

$$\mathrm{Prec}(A) = 0.97$$
$$\mathrm{Cov}(A) = 0.12$$

How precision is computed

- **Formal definition:**

$$\mathrm{Prec}(A) := \mathbb{P}_A\left(f(X) = f(\xi)\right) ,$$

  where X is a random perturbation of ξ containing all words in A
- **Question:** what is the distribution of "X given A" in this definition?
  - **default implementation:** i.i.d. Bernoulli for each word not in A to decide removal, replace by `UNK` token if removed (more on that later)
  - **generative model:** for instance, using BERT[58] to generate the missing words,...
  - **deterministic replacements:** get word embedding and replace by word having similar embeddings,[59]...

[58] Devlin, Chang, Lee, Toutanova, *BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding*, Proc. ACL, 2019

[59] Ribeiro, Singh, Guestrin, *"Why should I trust you?" Explaining the prediction of any classifier*, ACM SIGKDD, 2016

Sampling mechanism

The selection on the menu is great, and so is the food! The service is not bad, prices are fine.

the selection on the menu is great and so is the food the service is not bad prices are fine

the selection on the menu is great and so is the food the service is not bad prices are fine

the selection on the menu is great and so is the food the service is not bad prices are fine

the selection UNKthe menu is great and so is the food the UNK is not bad prices UNK fine

Sampling mechanism

The selection on the menu is great, and so is the food! The service is not bad, prices are fine.

the selection on the menu is great and so is the food the service is not bad prices are fine

the selection on the menu is great and so is the food the service is not bad prices are fine

the selection on the menu is great and so is the food the service is not bad prices are fine

the selection on the menu is great and so is UNK food the UNK is not bad prices are fine

Sampling mechanism

The selection on the menu is great, and so is the food! The service is not bad, prices are fine.

the selection on the menu is great and so is the food the service is not bad prices are fine

the selection on the menu is great and so is the food the service is not bad prices are fine

the selection on the menu is great and so is the food the service is not bad prices are fine

UNK selection on the menu UNKgreat and UNKUNKthe UNK the UNK UNKnot bad UNK are fine

Estimating $\mathrm{Prec}(A)$

- wlog, one can assume that $f(\xi) = 1$
- thus
$$\mathrm{Prec}(A) := \mathbb{P}_A\left(f(X) = 1\right) .$$
- **Remark:** of course, impossible to compute in practice (too costly with `UNK` replacement, worse with BERT)
- **Solution:** Monte-Carlo estimate:
$$\widehat{\mathrm{Prec}}_n(A) := \frac{1}{n}\sum_{i=1}^{n} \mathbb{1}_{f(X_i)=1} ,$$
where X_i i.i.d. draw from X
- in practice, $n = 10$

Coverage

- **Formal definition:** let $\mathcal{C}$ be a given set of documents. For any anchor A, we define

$$\mathrm{Cov}(A) := |\{\delta \in \mathcal{C} \quad \text{s.t.} \quad \forall w \in A, w \in \delta\}| \ .$$

- **Remark:** in practice, shorter anchors have higher coverage
- **Why?** think one common word: contain in many documents
- in the other direction, whole sentence $\rightarrow$ only contained in one document
- since $\mathrm{Cov}(A)$ costly to compute, Anchors minimizes $|A|$ instead of maximizing $\mathrm{Cov}(A)$

Summary

- let $\varepsilon > 0$ be some tolerance threshold (by default, $\varepsilon = 0.05$)
- **What is described originally:**

$$\underset{A \in \mathcal{A}}{\text{Maximize}} \mathrm{Cov}(A) \quad \text{subject to} \quad \mathrm{Prec}(A) \geq 1 - \varepsilon \,.$$

- **What the actual goal is:**

$$\underset{A \in \mathcal{A}}{\text{Minimize}} \, |A| \quad \text{subject to} \quad \widehat{\mathrm{Prec}}_n(A) \geq 1 - \varepsilon \,. \qquad (\star)$$

- **Additional caveat:** if ξ has length b, $|\mathcal{A}| = 2^b$...
- **What is done in practice:** use KL-UCB[60] to approximately solve $(\star)$

[60] Kaufmann and Kalyanakrishnnan, *Information complexity in bandit subset selection*, COLT, 2013

Visualization

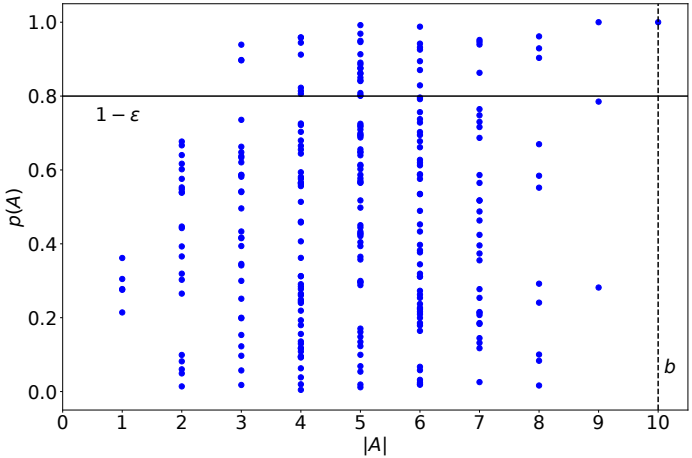


Figure: all anchors for a given example / classifier represented in the $|A|$ / $p(A) = \mathrm{Prec}(A)$ space

Visualization

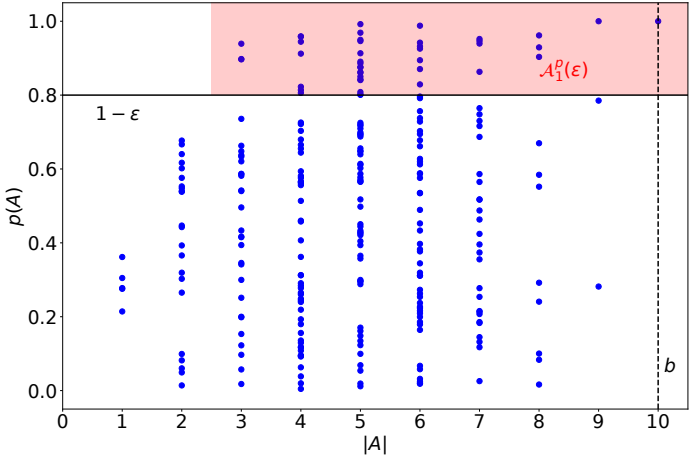


Figure: selecting $\mathcal{A}_1^p(\varepsilon)$, set of all anchors with evaluation higher than $1-\varepsilon$

Visualization

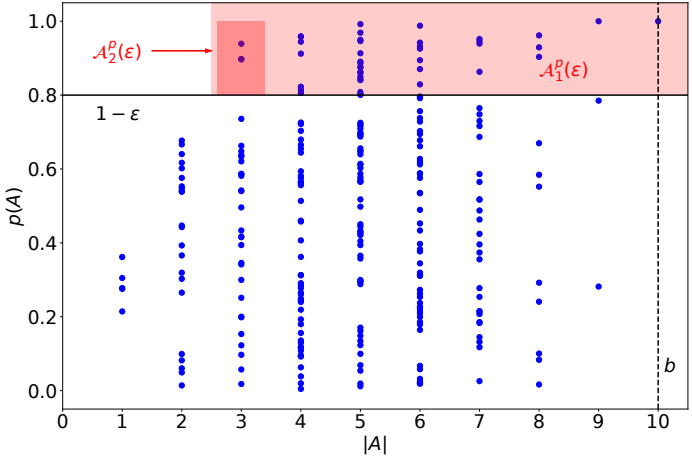


Figure: selecting $\mathcal{A}_2^p(\varepsilon)$, anchors with $p(A) \geq 1-\varepsilon$ and minimal length

Visualization

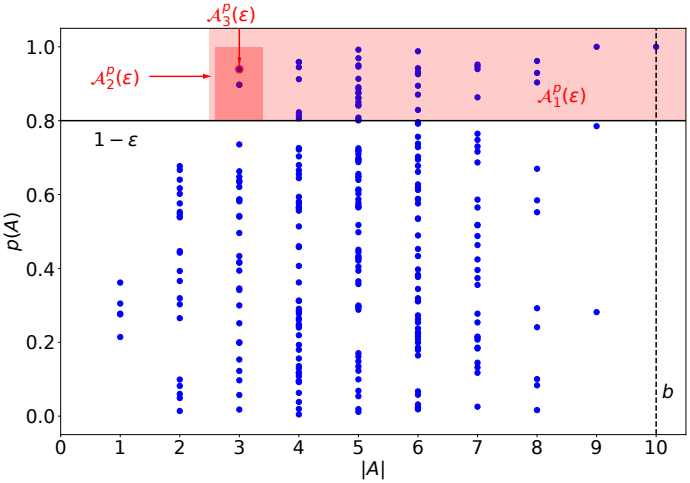


Figure: selecting $\mathcal{A}_3^p(\varepsilon)$, anchors with $p(A) \geq 1-\varepsilon$, minimal length, and maximal $p(A)$

Summary

- rule selection via perturbation
- very costly to run (combinatorics)
- some properties (recover coefficients of linear models)[61]

[61] Lopardo, Precioso, Garreau, *A sea of words: an in-depth analysis of Anchors for text data*, AISTATS, 2023