6.5. 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[49] to generate the missing words,...
  - **deterministic replacements:** get word embedding and replace by word having similar embeddings,[50]...

---

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

[50] 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 $\boxed{\text{great}}$ and so is the food the service is $\boxed{\text{not}}$ $\boxed{\text{bad}}$ prices are $\boxed{\text{fine}}$

the selection $\boxed{\text{on}}$ the menu is great and so is the food the $\boxed{\text{service}}$ is not bad prices $\boxed{\text{are}}$ 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 $\boxed{\text{great}}$ and so is the food the service is $\boxed{\text{not}}$ $\boxed{\text{bad}}$ prices are $\boxed{\text{fine}}$

the selection on the menu is great and so is $\boxed{\text{the}}$ food the $\boxed{\text{service}}$ 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 $\boxed{\text{great}}$ and so is the food the service is $\boxed{\text{not}}$ $\boxed{\text{bad}}$ prices are $\boxed{\text{fine}}$

$\boxed{\text{the}}$ selection on the menu $\boxed{\text{is}}$ great and $\boxed{\text{so}}$ $\boxed{\text{is}}$ the $\boxed{\text{food}}$ the $\boxed{\text{service}}$ $\boxed{\text{is}}$ not bad $\boxed{\text{prices}}$ 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

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

- **Remark:** in practice, shorter anchors have higher coverage
- **Why?** think one common word: contain in many documents
- in the other direction, whole sentence $\to$ 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:**

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

- **Additional caveat:** if ξ has length b, $|\mathcal{A}| = 2^b$...
- **What is done in practice:** use KL-UCB[51] to approximately solve $(\star)$

[51] Kaufmann and Kalyanakrishnan, *Information complexity in bandit subset selection*, COLT, 2013

Visualizing ($\star$)

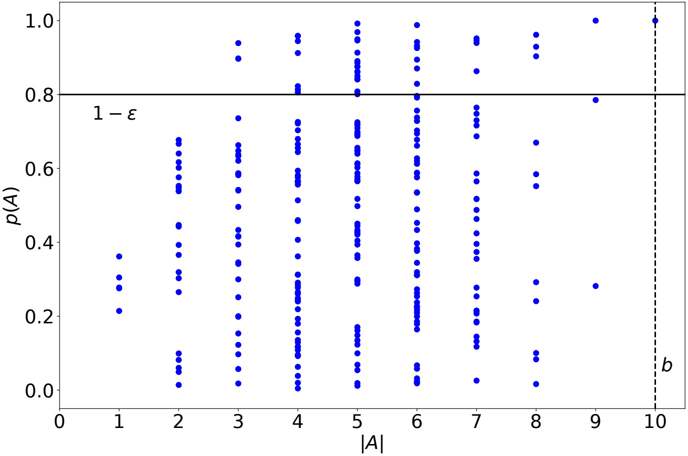


Figure: all anchors for a given example / classifier represented in the $|A|$ / $p(A) = \mathrm{Prec}(A)$ space

Visualizing ($\star$)

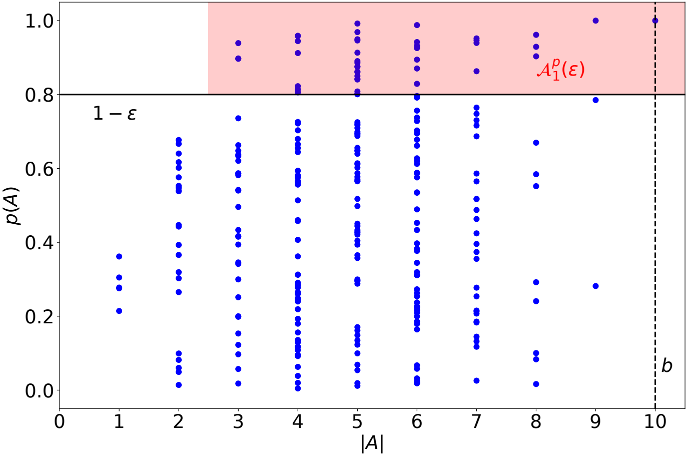


Figure: selecting $\mathcal{A}_1^p(\varepsilon)$, set of all anchors with evaluation higher than $1-\varepsilon$

Visualizing $(\star)$

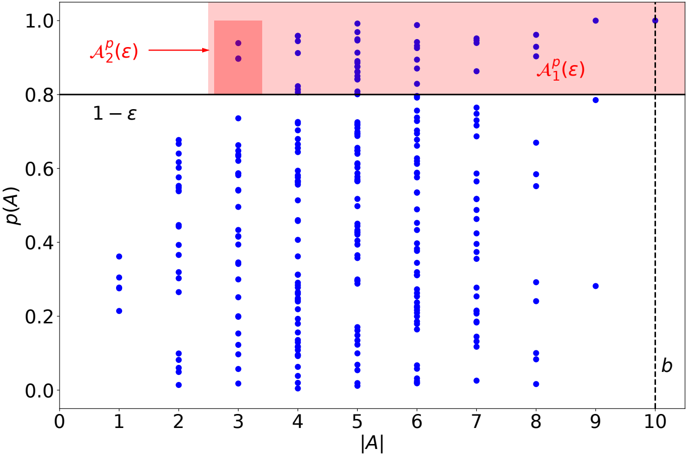


Figure: selecting $\mathcal{A}_2^p(\varepsilon)$, anchors with $p(A) \geq 1-\varepsilon$ and minimal length

Visualizing ($\star$)

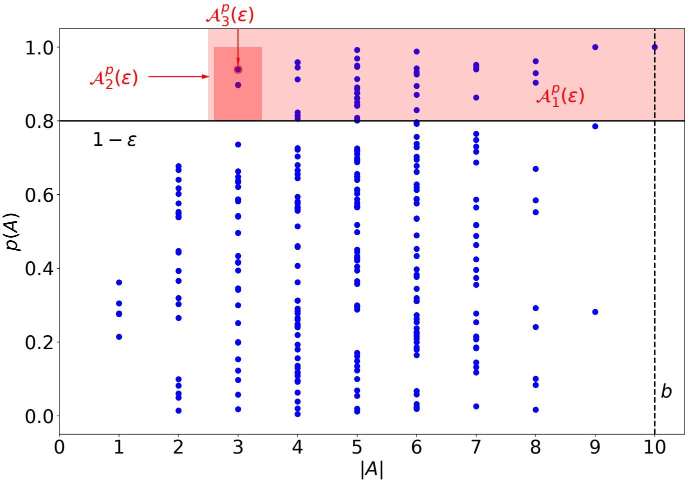


Figure: selecting $\mathcal{A}_3^p(\varepsilon)$, anchors with $p(A) \geq 1-\varepsilon$, minimal length, and maximal $p(A)$

Summary

- **Anchors** = rule selection via random perturbation
- interpretable features = subset of the words
- post-hoc, local method
- local rules have a global flavor
- very costly to run
- version for other data-types exist[52]
- some theoretical analysis (indicator and linear models)[53]

[52] `https://github.com/marcotcr/anchor`
[53] Lopardo, Precioso, Garreau, *A sea of words: an in-depth analysis of Anchors for text data*, AISTATS, 2023

6.6. A game-theoretical perspective

Shapley values

- **Setting:** D-player game[54]
- 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 $\phi_j(v) \gg 0$

[54] 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:

$$\sum_j \phi_j(v) = v([D])\,.$$

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

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

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

$$\forall j \in [D], \quad \phi_j(v + w) = \phi_j(v) + \phi_j(w)\,.$$

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

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

Shapley values, ctd.

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

Theorem:[55] Shapley values are the only payment rule satisfying efficiency, symmetry, linearity, and null player.

- **Question:** connection with interpretability?
- we can see f as the reward and a subset of features as the player

[55] *ibid*

Shapley regression values

- **Example:** linear model
- for each subset of features $S \subseteq [D]$, retrain a model f_S only using the features in S

Definition:[56] the *Shapley regression value* associated to feature j is given by

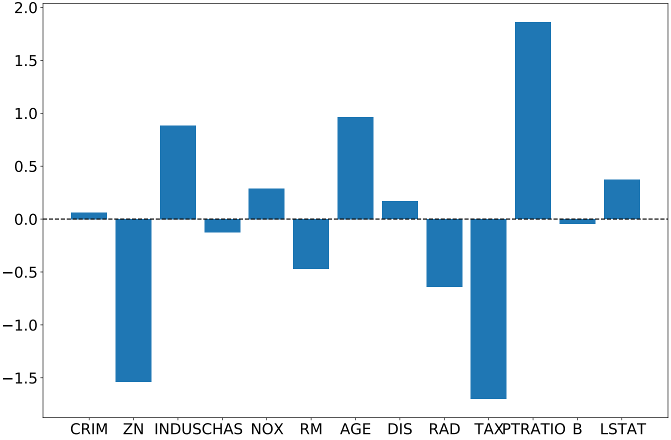
$$\phi_j := \sum_{S \subseteq [D] \setminus \{j\}} \frac{|S|!(D - |S| - 1)!}{D!} \left(f_{S \cup \{j\}}(\xi_{S \cup \{j\}}) - f_S(\xi_S) \right) ,$$

where ξ_S is the restriction of ξ to S features.

[56] 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

- several problems with this approach:
  - *computational cost* $= \mathcal{O}\left(2^D\right)$
  - *retraining* the model each time
- a first solution: *Shapley sampling values*[57]
  - 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 we can estimate this last quantity efficiently

[57] 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)[58]**:** define ϕ as the minimizer of

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

[58] 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 $\phi \in \mathbb{R}^d$

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

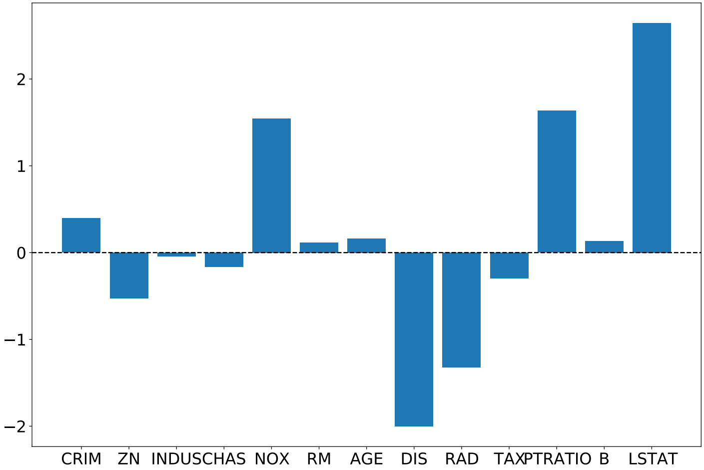
  with

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

- **Remark:** very similar to LIME

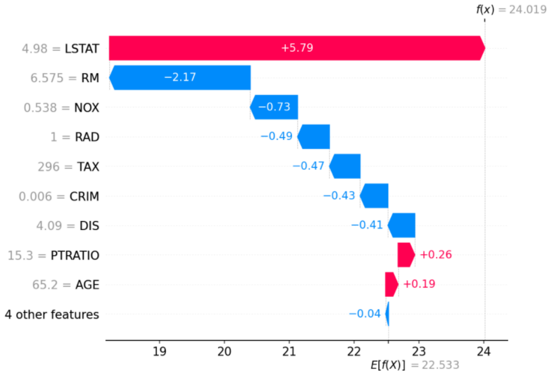
Kernel SHAP, tabular example

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



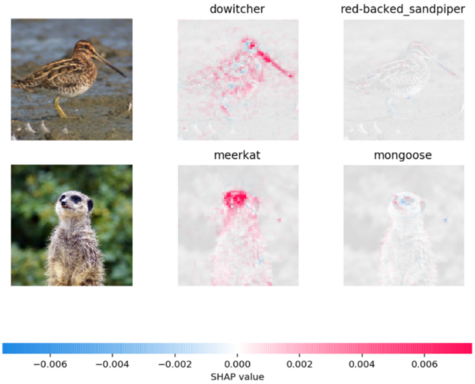
Kernel SHAP, tabular example

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



Extensions

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



Summary

- Kernel SHAP can be used on any model
- specialized versions for specific architectures:
  - *TreeSHAP*[59] (tree-based predictors)
  - *DeepSHAP* (DeepLIFT[60] + Shapley values)

Inconvenients:

- costly to run[61]
- not easy to read if many features

[59] Lundberg et al., *Consistent individualized feature attribution for tree ensembles*, arxiv, 2018
[60] Shrikumar et al., *Learning important features through propagating activation differences*, ICML, 2017
[61] improving the efficiency is work in progress, *e.g.*, Covert and Lee, *Improving KernelSHAP: Practical Shapley Value Estimation via Linear Regression*, AISTATS, 2021

7. Gradient-based approaches

7.1. Model agnostic methods

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 differentiable 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

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

Gradient explanation

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

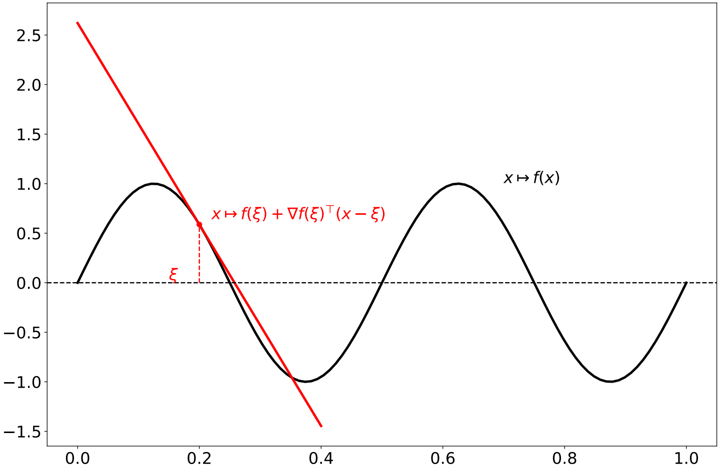
$$\phi_j = (\nabla f(\xi))_j = \frac{\partial}{\partial x_j} f(x) \Big|_{x=\xi} .$$

- generally referred to as *gradient explanation*[62], *sensivity map*, or *saliency map*
- **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 "PyTorch-compatible"
- **Remark:** in this talk, one evaluation of the model costs $\mathcal{O}(1)$
- **Beware:** gradient with respect to the input, not the parameters!

[62] Baehrens et al., *How to Explain Individual Classification Decisions*, JMLR, 2010

In low dimensions

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



Linear model

- **Usual question:** what happens for a linear model?
- that is, $f(x) := \sum_{k=1}^{d} \lambda_k x_k$
- in that case, simple answer:

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

- we retrieve the coefficients of the model:

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

Gradient explanation for images

- in this context, particular instance of *saliency maps*[63]
- input variables = pixel values
- each pixel has typically 3 channels
- when taking gradient with respect to input, pixel $\xi_{(i,j)}$ is attributed

$$(\nabla f(\xi))_{(i,j)} = \left(\frac{\partial f}{\partial x_{(i,j),\text{red}}}(\xi), \frac{\partial f}{\partial x_{(i,j),\text{green}}}(\xi), \frac{\partial f}{\partial x_{(i,j),\text{blue}}}(\xi) \right)^\top \in \mathbb{R}^3 .$$

- **Question:** how to visualize this as a heatmap?
- typical to display the maximum

$$\max \left(\frac{\partial f}{\partial x_{(i,j),\text{red}}}(\xi), \frac{\partial f}{\partial x_{(i,j),\text{green}}}(\xi), \frac{\partial f}{\partial x_{(i,j),\text{blue}}}(\xi) \right) .$$

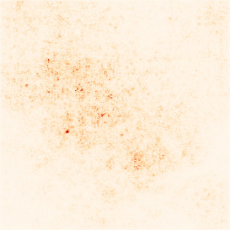
[63] Simonyan, Vedaldi, Zisserman, *Deep Inside Convolutional Networks: Visualising Image Classification Models and Saliency Maps*, preprint, 2013

Gradient explanation for images

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

predicted: quail (22.6%)

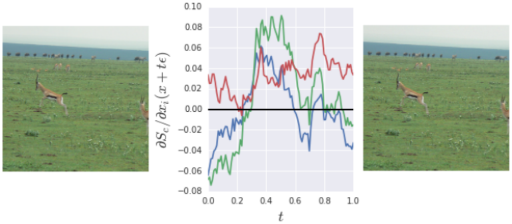
gradient explanation



- 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

Issues with the gradient

- **Main issue** with score functions of deep neural net: can be quite irregular at small scale
- **Example:** gradient fluctuations between similar images[64]



- $\Rightarrow$ purely gradient-based explanations can be quite noisy

[64] Smilkov et al., *SmoothGrad: removing noise by adding noise*, arxiv, 2017

Gradient times input

- many works trying to address this issue
- **First idea:** gradient times input[65]
- namely, partial derivatives multiplied by feature values

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

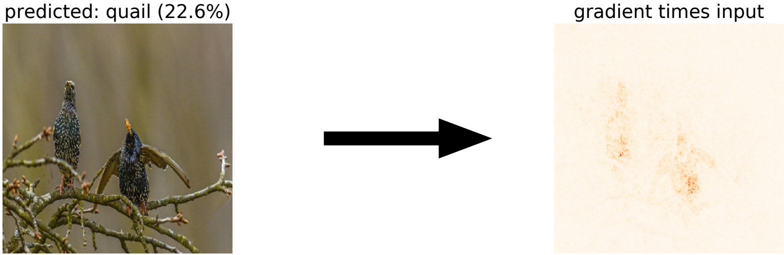
- **Intuition:** smoother explanations
- computational cost $= \mathcal{O}(D)$
- for a linear model:

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

[65] Shrikumar et al., *Not just a black box: Learning important features through propagating activation differences*, ICML, 2016

Gradient times input for images

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



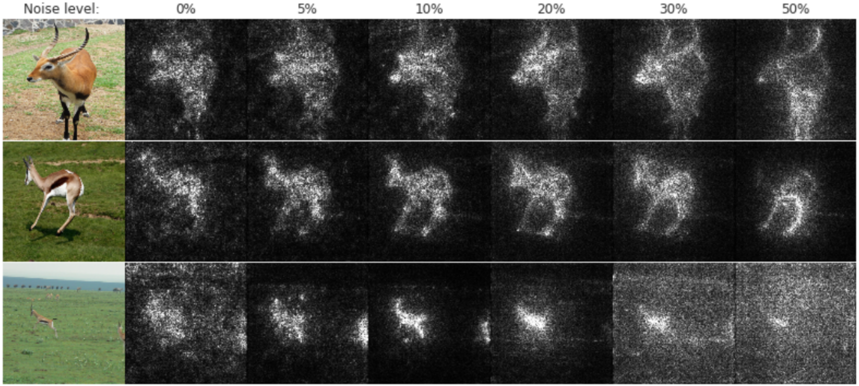
- much *smoother*, as promised
- still difficult to read in some cases

SmoothGrad

- another effort in this direction: *SmoothGrad*[66]
- **Idea:** average local fluctuations
- $\to$ sample many gradients in the surrounding and take the mean
- namely,
$$\forall j \in [D], \qquad \phi_j := \frac{1}{n}\sum_{i=1}^{n} \left(\nabla f(\xi + \varepsilon_i)\right)_j \,,$$
  where ε_i i.i.d. $\mathcal{N}\left(0, \sigma^2 \, \mathrm{I}_d\right)$
- Smilkov recommends taking σ as $10 - 20\%$ of input range (see next slide)
- **Intuition:** adding noise is equivalent to regularization[67]

[66] Smilkov et al., *SmoothGrad: removing noise by adding noise*, arxiv, 2017
[67] Bishop, *Training with noise is equivalent to Tikhonov regularization*, Neural Computation, 1995

SmoothGrad, influence of σ



- **Figure:** SmoothGrad with σ as percentage of the range

SmoothGrad, behavior for large n

- when $n \to +\infty$, according to SLLN:

$$\phi_j \xrightarrow{\text{a.s.}} \mathbb{E}\left[\frac{\partial f}{\partial x_j}(\xi + \varepsilon)\right] ,$$

  where $\varepsilon \sim \mathcal{N}\left(0, \sigma^2 \mathrm{I}\right)$
- set $g := \frac{\partial f}{\partial x_j}$
- let us assume that f is "nice" around ξ:

$$g(\xi + \varepsilon) \approx g(\xi) + \varepsilon^\top \nabla g(\xi) .$$

- taking expectation in the previous display yields

$$\mathbb{E}\left[g(\xi + \varepsilon)\right] \approx g(\xi) + \mathbb{E}\left[\varepsilon^\top \nabla g(\xi)\right] \approx g(\xi) = \frac{\partial f}{\partial x_j}(\xi) .$$

- we recover (approximately) the gradient

Integrated gradients

- **Another idea:**[68] average gradients between given reference and point of interest
- 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:

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

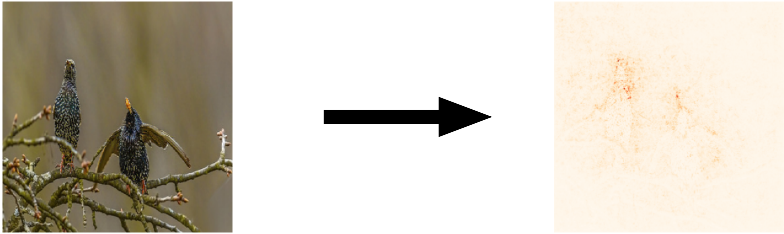
- computational cost $= \mathcal{O}(mD)$ ($m = 20$ gives good results)

[68] 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*[69]

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

Integrated gradients meets linear model

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