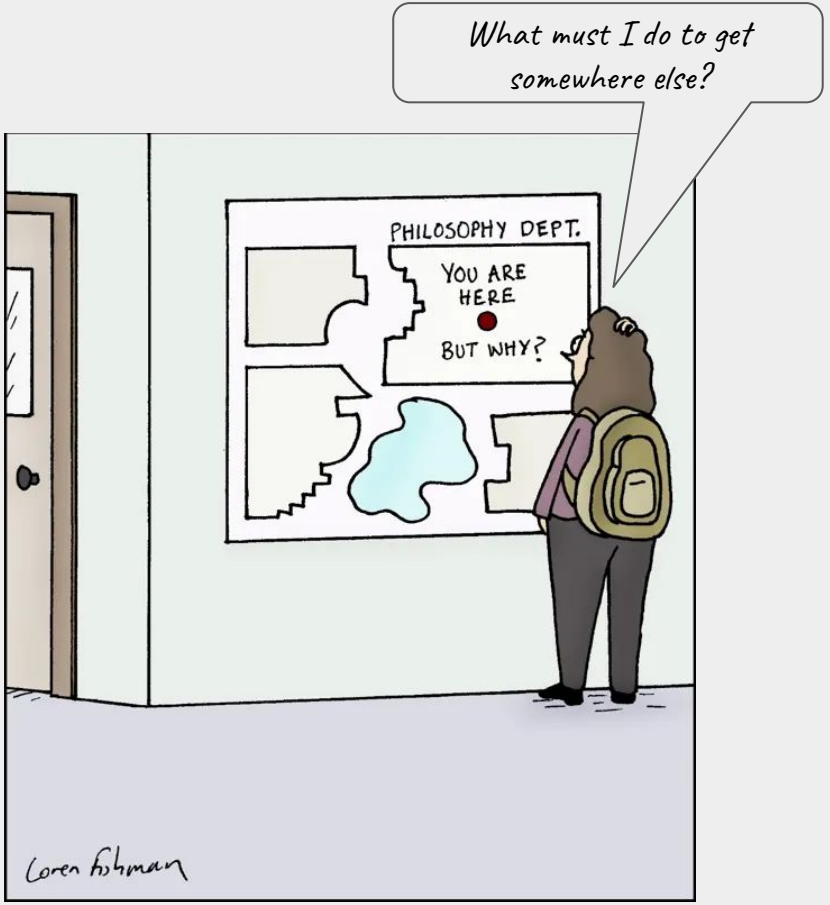
Explainable AI - Lecture 3

Local surrogate model explanations and LIME

norwegian open ai lab
NTNU

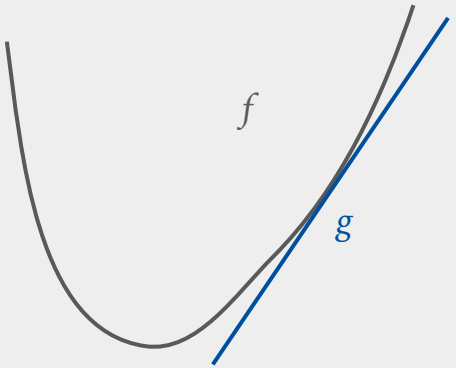
Before we start...

How are the counterfactuals doing?



NOW

Local surrogate model explanations



We already know global surrogate model

What would you say is the main weakness of global surrogate models?

We already know global surrogate model

What would you say is the main weakness of global surrogate models?

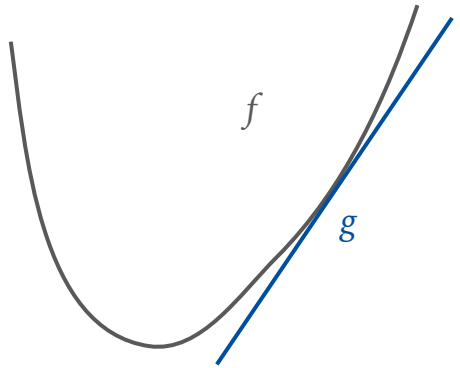
I'd argue that a main weakness is we cannot assume an interpretable model to be able to match the behaviour of a potentially highly complex, non-linear, non-interpretable model in the entire feature space.

This motivates the investigation of *local* surrogate models.

Intuitively: Surrogate models

Look around: Is the ground flat? Yes.

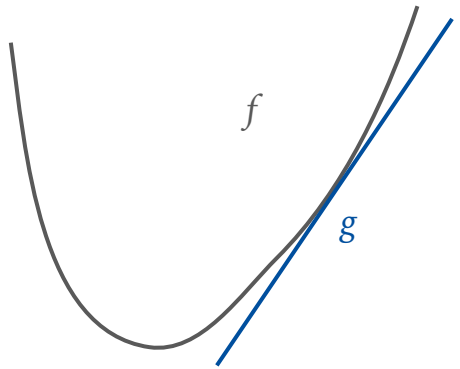
Is the earth flat?



Intuitively: Surrogate models

Look around: Is the ground flat? Yes.

Is the earth flat? No. But: locally, the earth is so flat that we can use Euclidean geometry to measure distances (locally!)



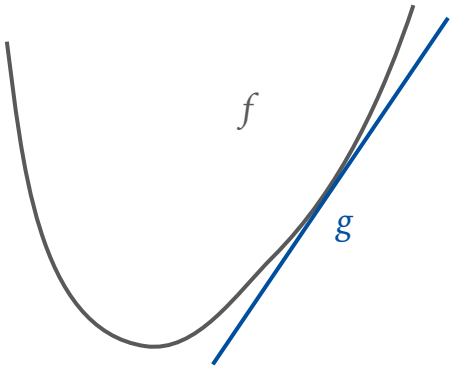
Intuitively: Surrogate models

Look around: Is the ground flat? Yes.

Is the earth flat? No. But: locally, the earth is so flat that we can use Euclidean geometry to measure distances (locally!)

Local surrogate models:

Replace the full non-interpretable model f by an interpretable model g *in a neighborhood.*

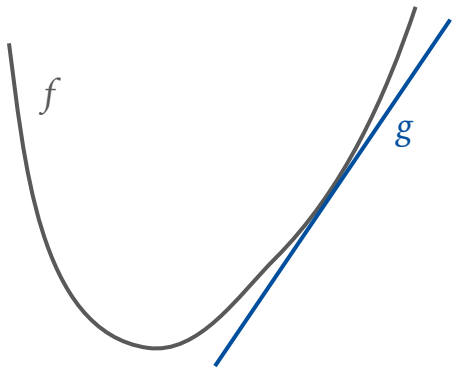


Local surrogate models

We can approximate any function f in a neighborhood x_0 by a polynomial via the Taylor expansion

$$f^{\text{Taylor}}(x) \approx f(x_0) + f'(x_0)(x-x_0) + \mathcal{O}(x^2)$$

$\Rightarrow$ we need the derivative of f around x_0



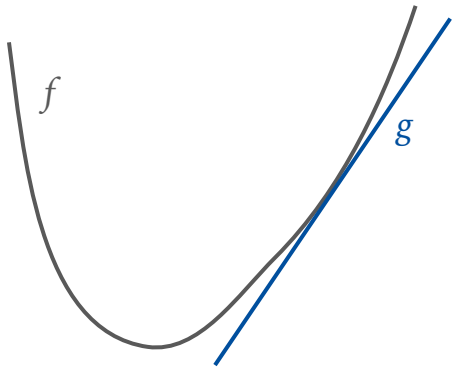
Local surrogate models

We can approximate any function f in a neighborhood x_0 by a polynomial via the Taylor expansion

$$f^{\text{Taylor}}(x) \approx f(x_0) + f'(x_0)(x-x_0) + \mathcal{O}(x^2)$$

$\Rightarrow$ we need the derivative of f around x_0

How do we differentiate our function f ?



Local surrogate models

We can approximate any function f in a neighborhood x_0 by a polynomial via the Taylor expansion

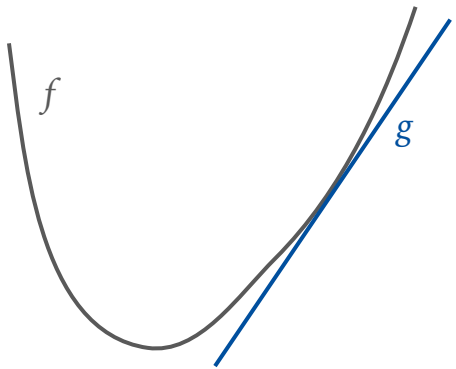
$$f^{\text{Taylor}}(x) \approx f(x_0) + f'(x_0)(x-x_0) + \mathcal{O}(x^2)$$

$\Rightarrow$ we need the derivative of f around x_0

Our f is a neural network / boosted forest / ..., not an analytical function.

$\Rightarrow$ forget that about the Taylor expansion :/

What can we do instead?



Local surrogate models

In a nutshell:

We have a model that needs explaining.

Local surrogate models

In a nutshell:

We have a model that needs explaining.

For a single prediction of this model,

Local surrogate models

In a nutshell:

We have a model that needs explaining.

For a single prediction of this model,

fit an interpretable model around the single prediction. ← *local surrogate model*

Local surrogate models

In a nutshell:

We have a model that needs explaining.

For a single prediction of this model,

fit an interpretable model around the single prediction. ← *local surrogate model*

Use the interpretability of this model as explanation for the full model, *for this prediction*.

Local surrogate models

In a nutshell:

We have a model that needs explaining.

For a single prediction of this model,

fit an interpretable model around the single prediction. ← *local surrogate model*

Use the interpretability of this model as explanation for the full model, *for this prediction*.

In summary: A surrogate model is trained to approximate a single prediction of the full model locally.

Local surrogate models

In a nutshell:

We have a model that needs explaining.

For a single prediction of this model,

fit an interpretable model around the single prediction. ← *local surrogate model*

Use the interpretability of this model as explanation for the full model, around the prediction.

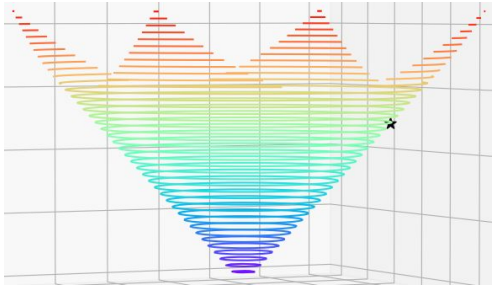
In summary: A surrogate model is trained to approximate single a prediction of the full model locally.

Now, we need

1) to define the neighborhood,
2) to make sure that the surrogate model behaves as we want it to.

Step by step, with notation

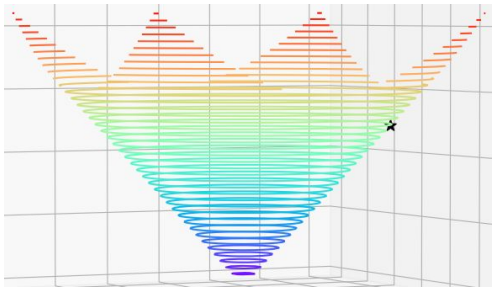
Define a neighborhood π_x around the point x we want to explain.

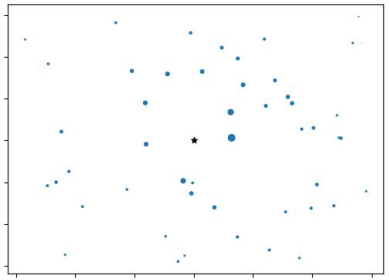


Step by step, with notation

Define a neighborhood π_x around the point x we want to explain.

Sample from this neighborhood, i.e. generate n points x_i





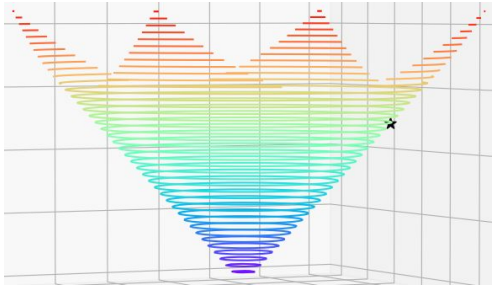
Step by step, with notation

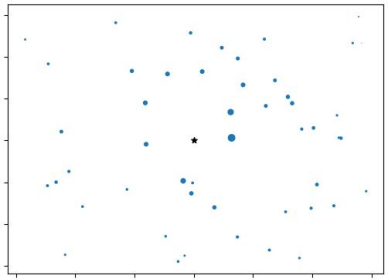
Define a neighborhood π_x around the point x we want to explain.

Sample from this neighborhood, i.e. generate n points x_i.

Fit an interpretable model g to the points x_i.

(this is the "make an interpretable model in a neighborhood")





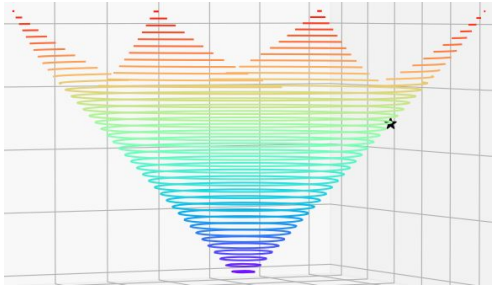
Step by step, with notation

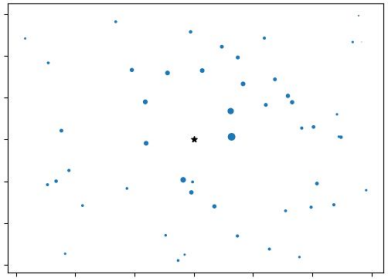
Define a neighborhood π_x around the point x we want to explain.

Sample from this neighborhood, i.e. generate n points x_i.

Fit an interpretable model g to the points x_i.

What should we use as targets when fitting g ?





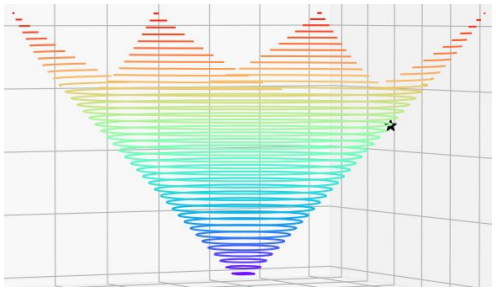
Step by step, with notation

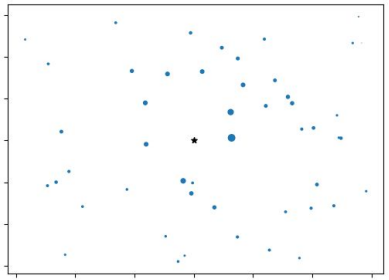
Define a neighborhood π_x around the point x we want to explain.

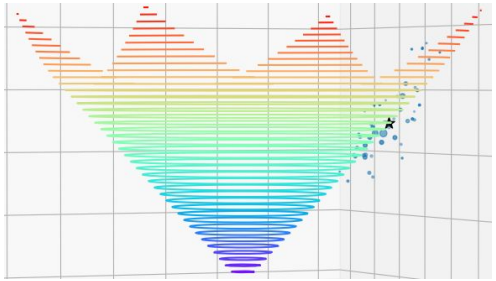
Sample from this neighborhood, i.e. generate n points x_i.

Fit an interpretable model g to the points x_i. *What should we use as targets when fitting g?*

The predictions $f(x_i)$ - because **the point of g is to approximate f .**





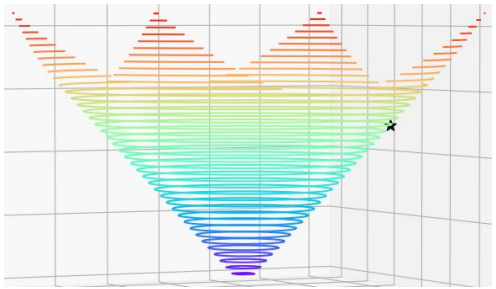


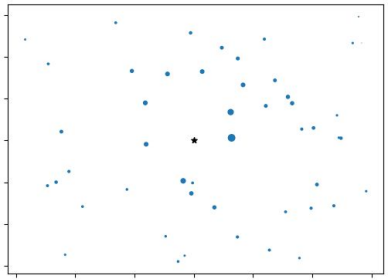
Step by step, with notation

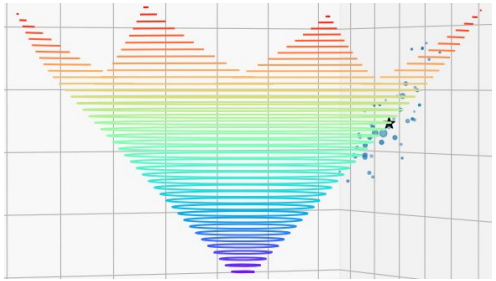
Generate n points x_i in a neighborhood, and make labels for these points by using f to predict on the x_i.

Fit an interpretable model g to these points and labels.

Choose g such that the complexity $\mathbf{\Omega}(g)$ is minimised ← *why?*





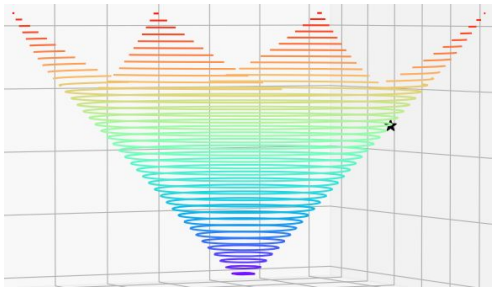


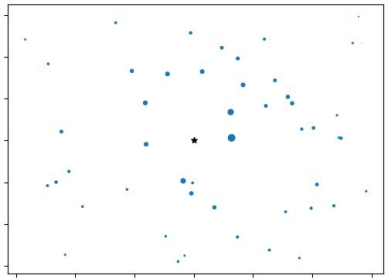
Step by step, with notation

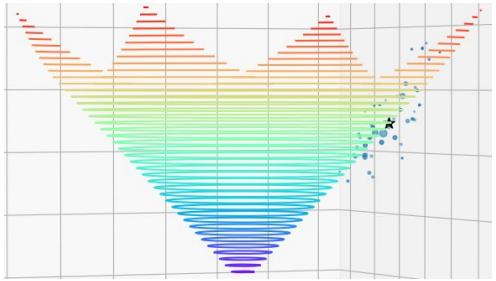
Generate n points x_i in a neighborhood, and make labels for these points by using f to predict on the x_i.

Fit an interpretable model g to these points and labels.

Choose g such that the complexity $\mathbf{\Omega}(g)$ is minimised ← keep g interpretable





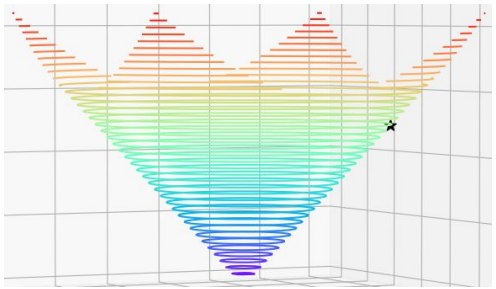


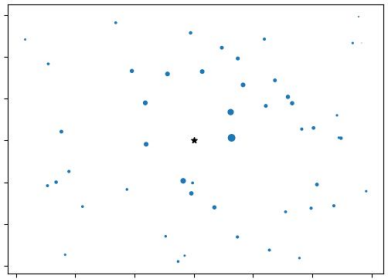
Step by step, with notation

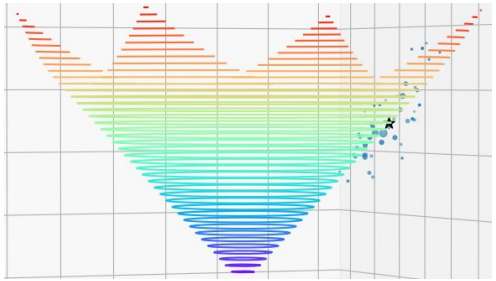
Generate n points x_i in a neighborhood, and make labels for these points by using f to predict on the x_i.

Fit an interpretable model g to these points and labels.

Choose g such that the loss $\mathcal{L}(f, g, \pi_x)$ is minimised ← *Why?*





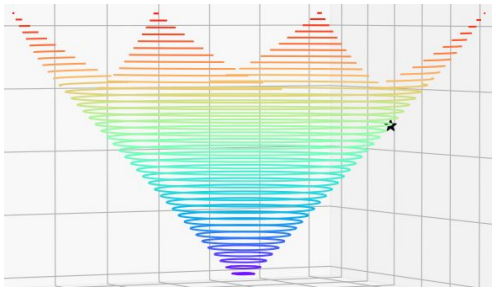


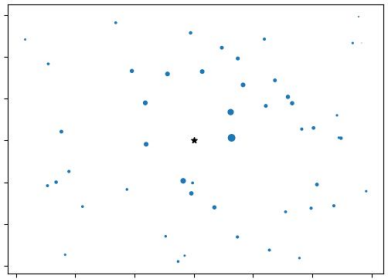
Step by step, with notation

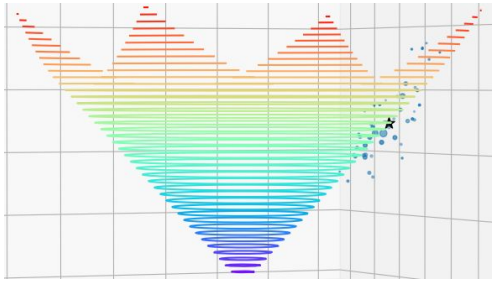
Generate n points x_i in a neighborhood, and make labels for these points by using f to predict.

Fit an interpretable model g to these points and labels.

Choose g such that the loss $\mathcal{L}(f, g, \pi_x)$ is minimised ← f and g must behave similarly in the neighborhood of x.





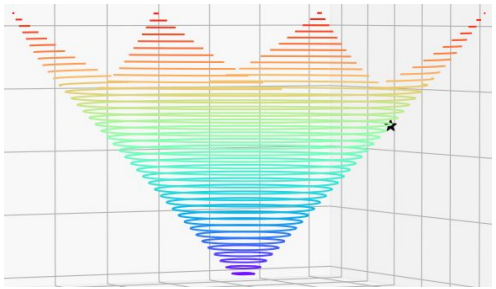


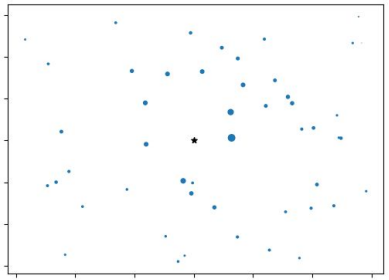
Step by step, with notation

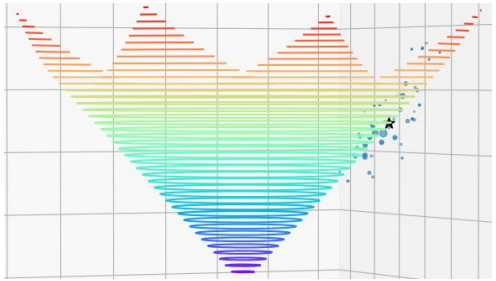
Generate n points x_i in a neighborhood, and make labels for these points by using f to predict.

Fit an interpretable model g to these points and labels. Choose g such that

1) the complexity $\Omega(g)$ is minimised ← *g is interpretable*
2) the loss $\mathcal{L}(f, g, \pi_x)$ is minimised ← *f and g behave similarly in the neighborhood of x*

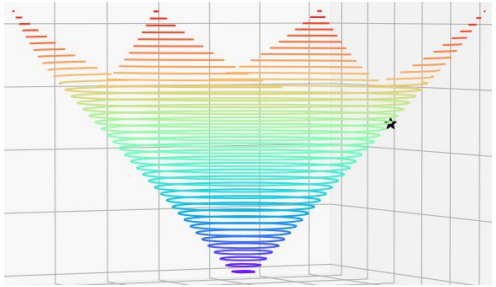




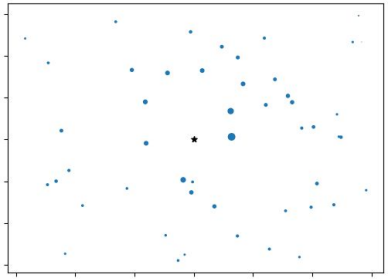


Local surrogate models at a glance

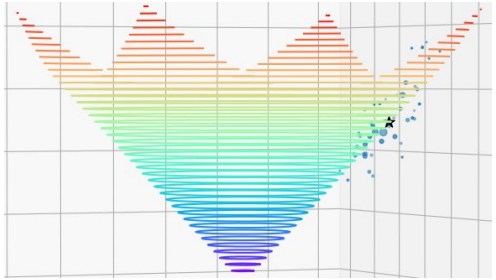
Given model f and point x_0 to explain



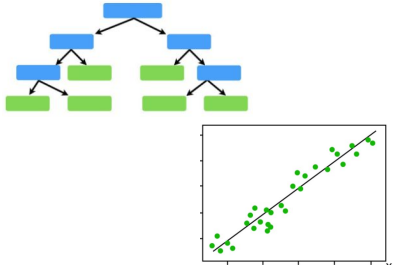
Generate random points weighted by distance to x_0



Use f to predict targets for generated points

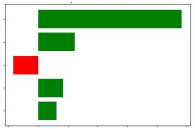


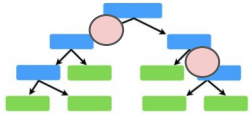
Choose interpretable model class G, and fit g to generated data



Use g for explanation

$$y = \beta_0 + \beta_1 x$$



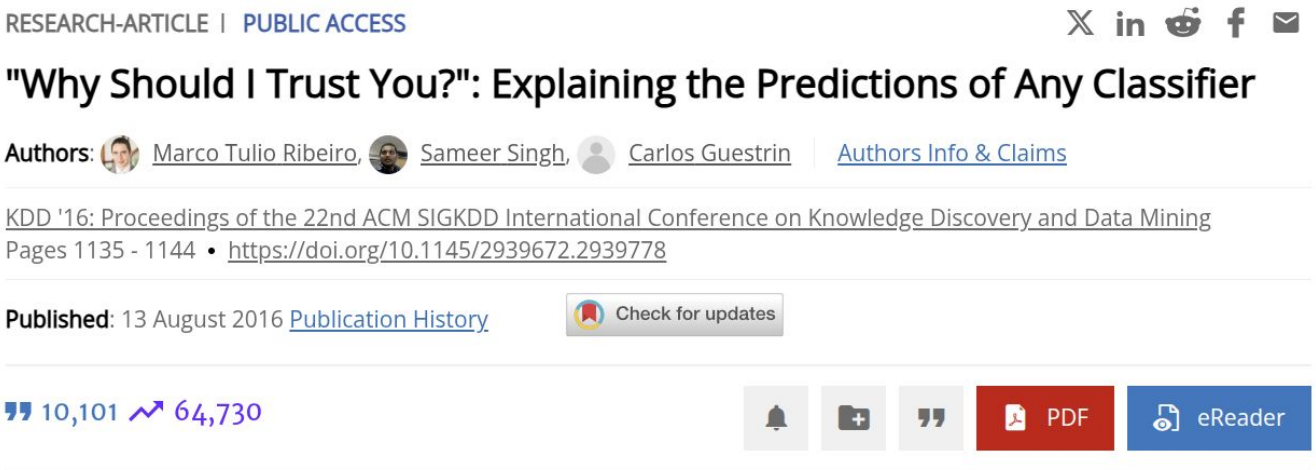


norwegian open ai lab NTNU

Today's explanation method

is one of the classics (perhaps *the* classic), pretty old and widely cited.

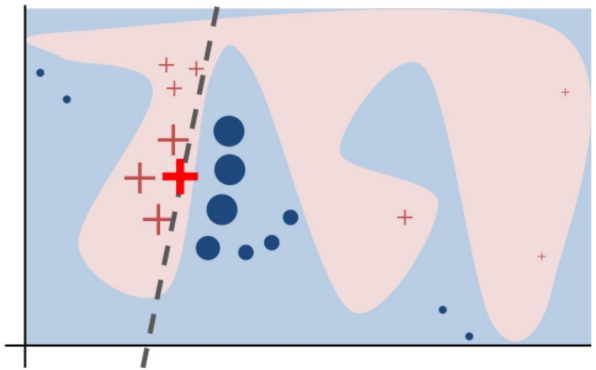
LIME: **L**ocal **I**nterpretable **M**odel-agnostic **E**xplanation



LIME

LIME: **L**ocal **I**nterpretable **M**odel-agnostic **E**xplanation

Generates explanations by learning a surrogate model per neighborhood π_x around x



$$\underset{g \in G}{\operatorname{argmin}} \quad \mathcal{L}(f, g, \pi_x) + \Omega(g)$$

"Why Should I Trust You?": Explaining the Predictions of Any Classifier
DOI:10.1145/2939672.2939778

LIME - Local Interpretable Model-agnostic Explanation

Where do we place this in the taxonomy?

Post-hoc?

Model-agnostic or model-specific?

Local or global?

Post-hoc methods	Model-agnostic	Model-specific
Local		
Global		

LIME - Local Interpretable Model-agnostic Explanation

Where do we place this in the taxonomy?

Post-hoc: *We got a model and want to explain its behaviour.*

Model-agnostic or model-specific?

Local or global?

Post-hoc methods	Model-agnostic	Model-specific
Local		
Global		

LIME - Local Interpretable Model-agnostic Explanation

Where do we place this in the taxonomy?

Post-hoc: *We got a model and want to explain its behaviour.*

Model-agnostic: *We use only the model's predictions to train a surrogate model.*

Local or global?

Post-hoc methods	Model-agnostic	Model-specific
Local		
Global		

LIME - Local Interpretable Model-agnostic Explanation

Where do we place this in the taxonomy?

Post-hoc: *We got a model and want to explain its behaviour.*

Model-agnostic: *We use only the model's predictions to train a surrogate model.*

Local: *We care about one prediction, i.e. one data instance.*

Post-hoc methods	Model-agnostic	Model-specific
Local		
Global		

LIME - Local Interpretable Model-agnostic Explanation

Where do we place this in the taxonomy?

Post-hoc: *We got a model and want to explain its behaviour.*

Model-agnostic: *We use only the model's predictions to train a surrogate model.*

Local: *We care about one prediction, i.e. one data instance.*

Post-hoc methods	Model-agnostic	Model-specific
Local	*LIME*	
Global		

Get started quickly

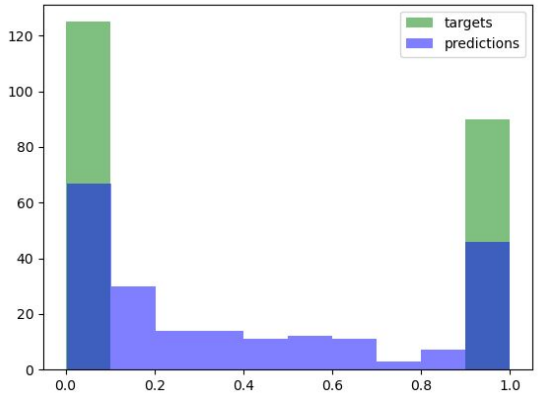
pip install lime

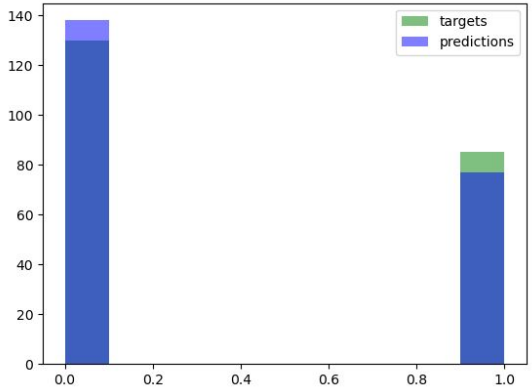
Let's get some data and a model...

We'll use the Titanic dataset as example

Features: `Passenger class`, `Sex` and `Age`

Model: XGBoost Classifier or Regressor





titanic.

Who survived?

The goal of this dataset is to predict who might have survived the titanic distaster. There was certainly an element of luck involved in surviving, it seems some groups of people were more likely to survive than others.

This dataset was originally found in **kaggle**.

NAME	DESCRIPTION
survived	Did this passenger survive.
pclass	Passenger class. Can be 1, 2 or 3.
Name	Name of the passenger.
sex	Sex of the passenger.
age	Age of the passenger.
fare	The amount paid for a ticket.
sibsp	The number of siblings on board.
parch	The number of parents on board.

You can download the dataset from the commandline.

```
wget https://calmcode.io/static/data/titanic.csv
```

Models predicting survival

A classifier and a regressor (although this is a classification problem).

The model behaviours and explanations will turn out to be similar, but you'll get familiar with the syntax for LIME on tabular data for both classification and regression.

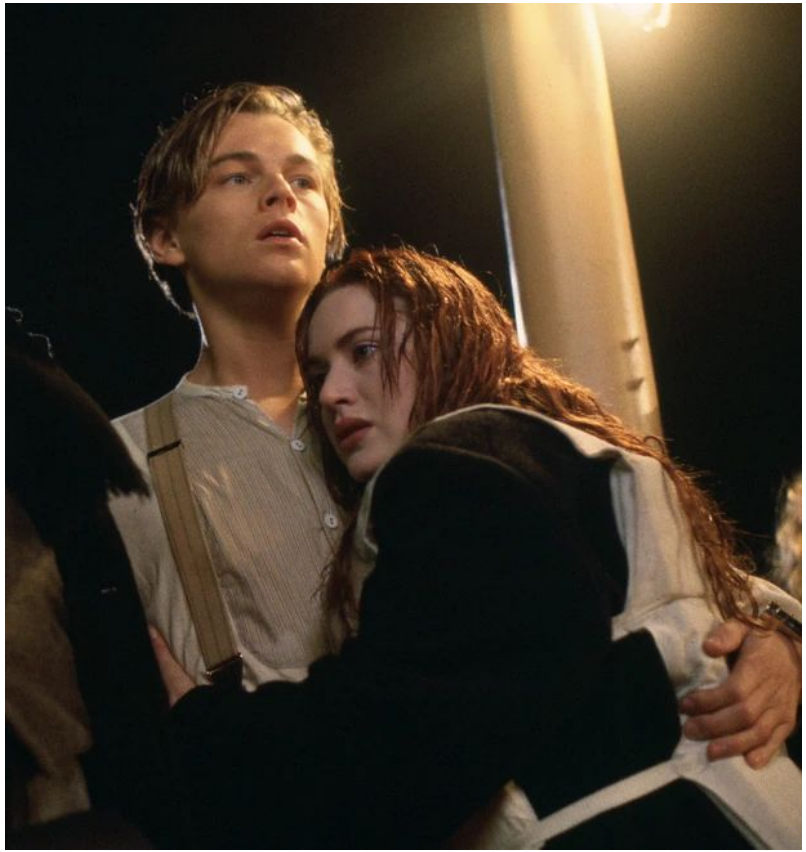
```
classifier = xgboost.XGBClassifier()
classifier.fit(X_train, y_train)

probs = classifier.predict(X_test)

print("Model accuracy: ", accuracy(y_test, probs))
```

```
Model accuracy:  0.786046511627907
```

```
regressor = xgboost.XGBRegressor(random_state=50)
regressor.fit(X_train, y_train)
probs = regressor.predict(X_test)
classes = [0 if _p<0.5 else 1 for _p in probs]
print("Model accuracy: ", accuracy(y_test, classes))
```

```
Model accuracy:  0.7674418604651163
```

Data instances to explain



```
X_jack = np.array([3,1,21])
X_rose = np.array([1,0,17])
```

Why does Jack die?

Why does Rose survive?

Let's ask LIME.

LIME tabular explainer

```
# Instantiate explainer objects
lime_regressor = LimeTabularExplainer(X_train,
                                      feature_names=features,
                                      verbose=True,
                                      mode='regression')

lime_classifier = LimeTabularExplainer(X_train,
                                       feature_names=features,
                                       verbose=True,
                                       mode='classification')
```

LIME tabular explainer for classifier

```
exp_lime = lime_classifier.explain_instance(X_jack, classifier.predict_proba, num_features=3)

#exp_lime.show_in_notebook()
# Workaround with same kwargs as show_in_notebook
html = exp_lime.as_html()
display(HTML(html))
```

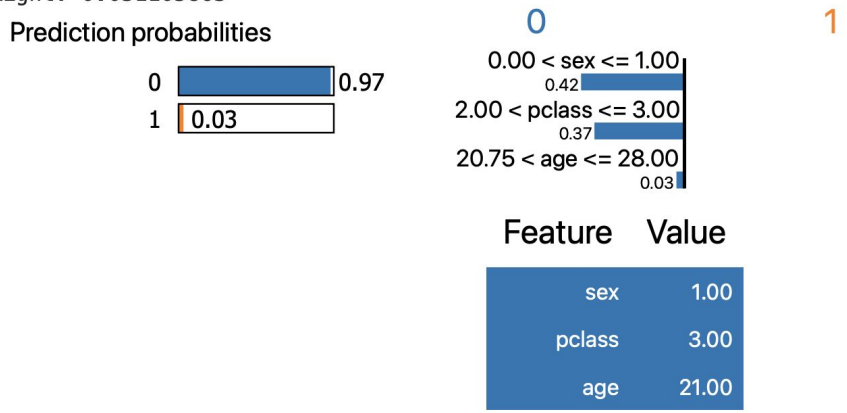
You can use show_in_notebook if IPython plays along, or use this workaround.

arguments to the explain_instance method:

- feature values of data point to explain
- prediction function (of the model to explain)
- number of the top features to include

```
exp_lime = lime_classifier.explain_instance(X_jack, classifier.predict_proba, num_features=3)

#exp_lime.show_in_notebook()
# Workaround with same kwargs as show_in_notebook
html = exp_lime.as_html()
display(HTML(html))
```

```
Intercept 0.9094895416420348
Prediction_local [0.08620924]
Right: 0.031105863
```



Intercept: the constant in the linear surrogate model's prediction for the test array.

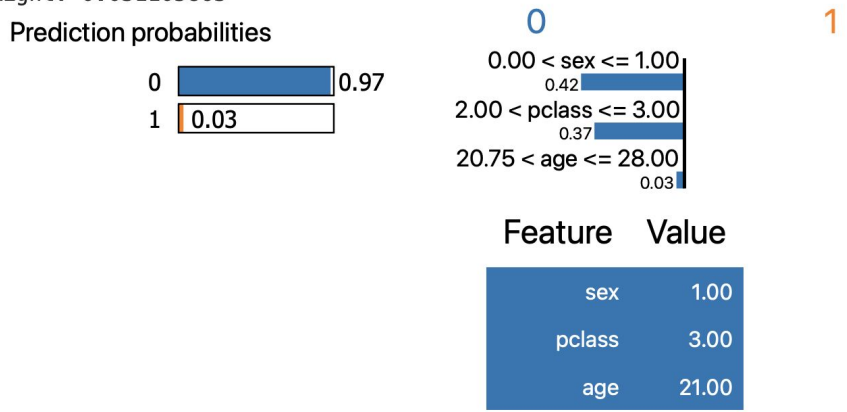
Prediction_local: the surrogate model prediction, trained on the perturbed samples using LIME's top num_features.

Right: full model prediction for the test array

Visualisations: blue = negative; orange = positive contributions

```
exp_lime = lime_classifier.explain_instance(X_jack, classifier.predict_proba, num_features=3)

#exp_lime.show_in_notebook()
# Workaround with same kwargs as show_in_notebook
html = exp_lime.as_html()
display(HTML(html))
```

```
Intercept 0.9094895416420348
Prediction_local [0.08620924]
Right: 0.031105863
```



The feature importances represent contribution to the deviation from the intercept (= the global baseline)

```
0.91 - (0.42 + 0.37 + 0.03) = 0.09
```

```
intercept - contributions = surrogate model prediction
```

```
exp_lime = lime_regressor.explain_instance(X_jack, regressor.predict, num_features=3)
html = exp_lime.as_html()
display(HTML(html))
```

```
Intercept 0.8005407852612024
Prediction_local [0.08794572]
Right: 0.19780497
```

`intercept - contributions = surrogate model prediction`

Now. Why does Jack die?

```
exp_lime = lime_regressor.explain_instance(X_jack, regressor.predict, num_features=3)
html = exp_lime.as_html()
display(HTML(html))
```

```
Intercept 0.8005407852612024
Prediction_local [0.08794572]
Right: 0.19780497
```

Why does Jack die?

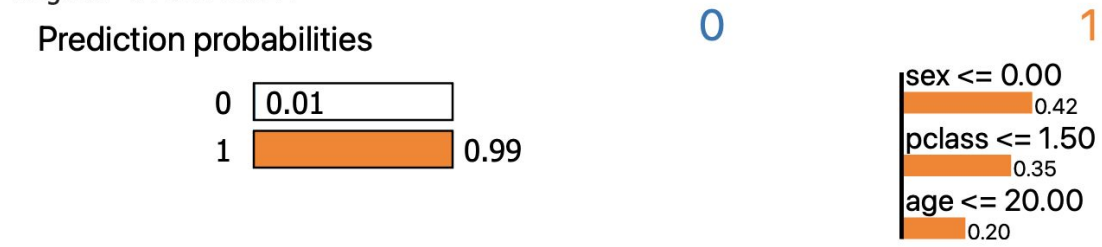
The classifier and regressor explanations agree. He dies because of

1) His sex
2) His passenger class
3) His age

According to LIME, he has nothing pulling him towards survival.

```
exp_lime = lime_classifier.explain_instance(X_rose, classifier.predict_proba, num_features=3)
html = exp_lime.as_html()
display(HTML(html))
```

```
Intercept 0.1024037066888634
Prediction_local [1.07273219]
Right: 0.99145824
```

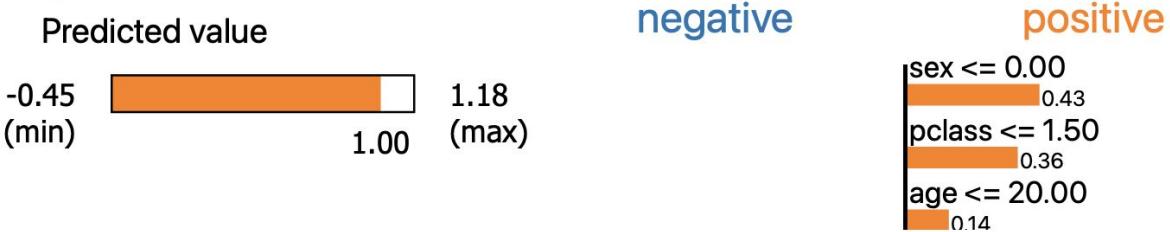


The two models agree on Rose's survival.

Why does Rose survive?

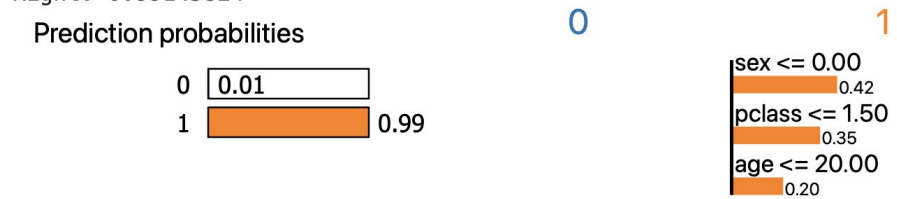
```
exp_lime = lime_regressor.explain_instance(X_rose, regressor.predict, num_features=3)
html = exp_lime.as_html()
display(HTML(html))
```

```
Intercept 0.11071881363284836
Prediction_local [1.03663708]
Right: 1.0034986
```



```
exp_lime = lime_classifier.explain_instance(X_rose, classifier.predict_proba, num_features=3)
html = exp_lime.as_html()
display(HTML(html))
```

```
Intercept 0.1024037066888634
Prediction_local [1.07273219]
Right: 0.99145824
```

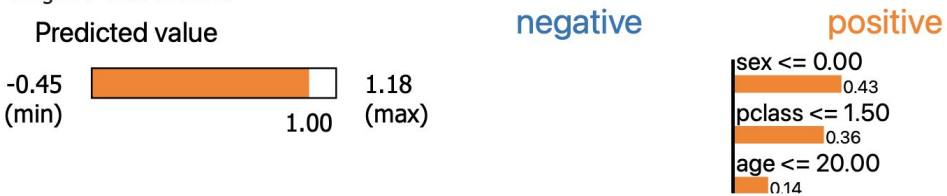


Why does Rose survive?

1) Her gender
2) Her age
3) Her passenger class

```
exp_lime = lime_regressor.explain_instance(X_rose, regressor.predict, num_features=3)
html = exp_lime.as_html()
display(HTML(html))
```

```
Intercept 0.11071881363284836
Prediction_local [1.03663708]
Right: 1.0034986
```



According to LIME, there's no reason for her to die :)

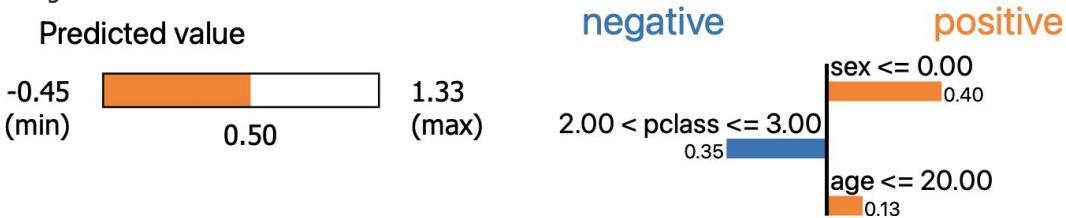
Let's look at a less clear-cut case

```
X_kid = np.array([3, 0, 16])
regressor.predict([X_kid])
```

```
array([0.5019737], dtype=float32)
```

```
exp_lime = lime_regressor.explain_instance(X_kid, regressor.predict, num_features=3)
html = exp_lime.as_html()
display(HTML(html))
```

```
Intercept 0.38487297404954196
Prediction_local [0.56132847]
Right: 0.5019737
```



What pulls this individual towards life/death?

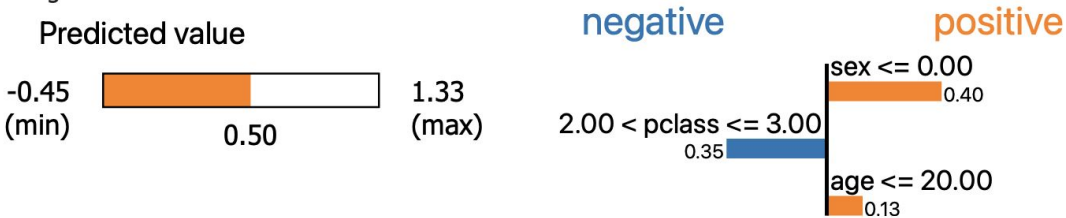
Let's look at a less clear-cut case

```
X_kid = np.array([3, 0, 16])
regressor.predict([X_kid])
```

```
array([0.5019737], dtype=float32)
```

```
exp_lime = lime_regressor.explain_instance(X_kid, regressor.predict, num_features=3)
html = exp_lime.as_html()
display(HTML(html))
```

```
Intercept 0.38487297404954196
Prediction_local [0.56132847]
Right: 0.5019737
```



What pulls this individual towards life/death?

1) Being female pulls towards survival
2) Travelling third class pulls away from survival
3) Being young pulls towards survival

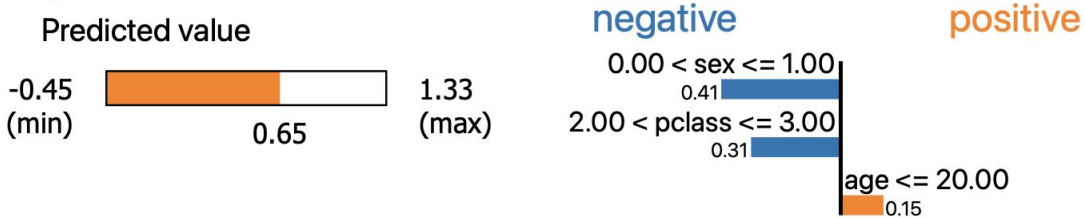
One more instance...

```
X_kid = np.array([3, 1, 9])
regressor.predict([X_kid])
```

```
array([0.6534467], dtype=float32)
```

```
exp_lime = lime_regressor.explain_instance(X_kid, regressor.predict, num_features=3)
html = exp_lime.as_html()
display(HTML(html))
```

```
Intercept 0.7624183531066626
Prediction_local [0.18837236]
Right: 0.6534467
```



What pulls this individual towards life/death?

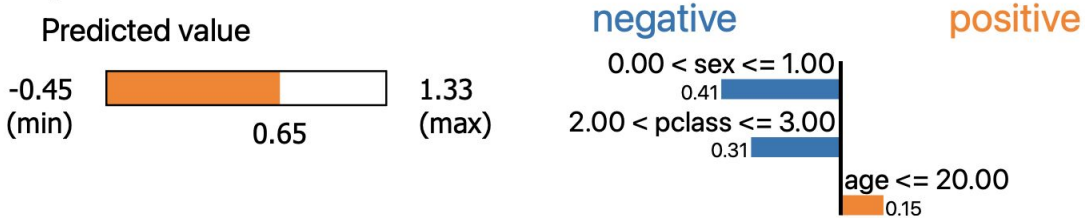
One more instance...

```
X_kid = np.array([3, 1, 9])
regressor.predict([X_kid])
```

```
array([0.6534467], dtype=float32)
```

```
exp_lime = lime_regressor.explain_instance(X_kid, regressor.predict, num_features=3)
html = exp_lime.as_html()
display(HTML(html))
```

```
Intercept 0.7624183531066626
Prediction_local [0.18837236]
Right: 0.6534467
```



What pulls this individual towards life/death?

1) Being male pulls away from survival
2) Travelling third class pulls away from survival
3) Being young pulls towards survival

Next, let's look under the hood

How to define the neighborhood π_x

How to keep the feature space interpretable

How to run LIME on other data types

The neighborhood

The neighborhood

LIME currently defines the neighborhood using a Gaussian radial basis function (RBF) kernel.
This is a function that measures the similarity between data points based on their Euclidean distance:

$$RBF(x^{(i)}) = \exp\left(-\frac{||x^{(i)} - x^{(ref)}||^2}{kw}\right)$$

This function calculates a proximity measure based on two data points.

What are the two data points in our case?

The neighborhood

LIME currently defines the neighborhood using a Gaussian radial basis function (RBF) kernel.
This is a function that measures the similarity between data points based on their Euclidean distance:

$$RBF(x^{(i)}) = \exp\left(-\frac{||x^{(i)} - x^{(ref)}||^2}{kw}\right)$$

This function calculates a proximity measure based on two data points.

In our case, $x^{(ref)}$ is the data point we want to explain, and $x^{(i)}$ is some point potentially in the neighborhood.

The neighborhood

LIME currently defines the neighborhood using a Gaussian radial basis function kernel to calculate proximity:

$$RBF(x^{(i)}) = \exp\left(-\frac{||x^{(i)} - x^{(ref)}||^2}{kw}\right)$$

The kernel width in the denominator effectively determines the size of the neighborhood.

A small *kw* leads to only close points being included; a large *kw* includes points farther away.

What is a good number for *kw*?

The neighborhood

From the LIME git: https://github.com/marcotcr/lime in the file `lime_tabular.py`:

```
if kernel_width is None:
    kernel_width = np.sqrt(training_data.shape[1]) * .75
kernel_width = float(kernel_width)

if kernel is None:
    def kernel(d, kernel_width):
        return np.sqrt(np.exp(-(d ** 2) / kernel_width ** 2))
```

the kernel width is 0.75 times the square root of the number of columns of the training data.

This 0.75 is a *choice*, probably empirical, but still not necessarily good in every case.

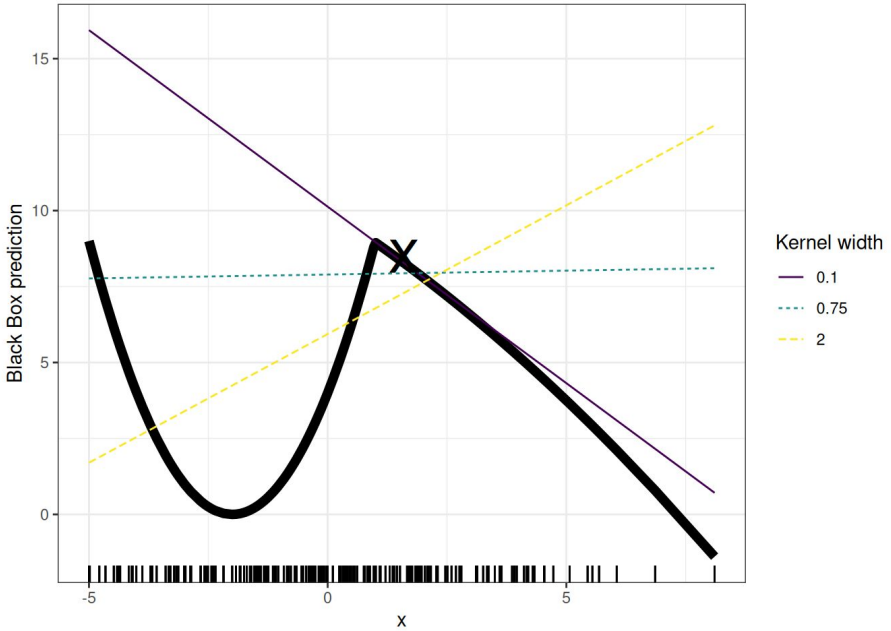
The neighborhood

Say the full model is represented by the black line.

The purple, green and yellow lines represent linear surrogate models using three different kernel widths.

These three surrogate models represent different behaviours, and thus produce different explanations.

The purple and yellow will produce opposing explanations.



Example from https://christophm.github.io/interpretable-ml-book

The neighborhood

The choice of neighborhood is an unsolved problem when using LIME with tabular data.

Always try different kernels and settings.

LIME currently samples the neighborhood by assuming

- a Gaussian distribution for each feature
- feature independence (no feature correlation).

This can lead to unlikely data points.

Interpretable space

Model space vs human-interpretable space

The model space isn't necessarily interpretable for humans.

For the Titanic data, we know what `Pclass`, `sex`, `age`, etc mean.

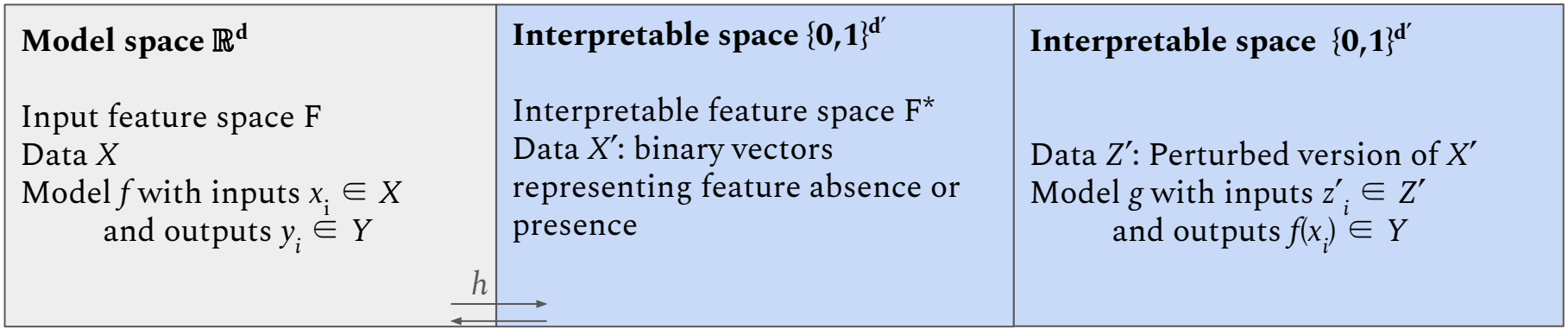
But what about pixels for an image model or tokens for a language model?

LIME distinguishes between the *model space* and the *human-interpretable space*.

Model space vs human-interpretable space

The model space isn't necessarily interpretable for humans.

LIME distinguishes between the *model space* and the *human-interpretable space*.

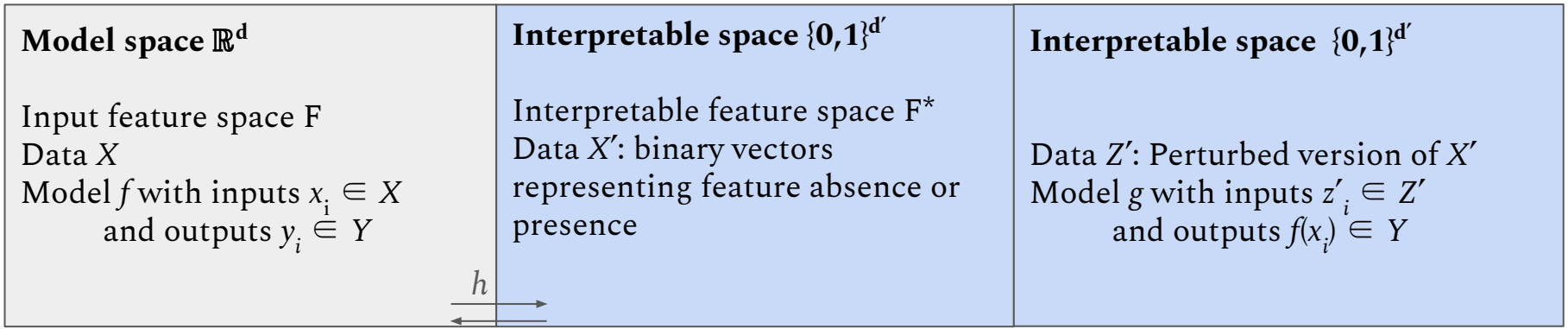


I know this interpretable space stuff is annoying and you might be tempted not to think too hard about it. RESIST THAT URGE. This is considered common XAI knowledge, and it returns in SHAP, SAGE, ...

Model space vs human-interpretable space

The model space isn't necessarily interpretable for humans.

LIME distinguishes between the *model space* and the *human-interpretable space*.



I know that this interpretable space stuff is annoying and that it's tempting not to think too hard about it. RESIST THAT URGE. This is considered common XAI knowledge, and it returns in SHAP, SAGE, ...

Model space vs human-interpretable space

x_i: the data point to be explained, in the input space of the original model f

x'_i: the interpretable representation of x, typically a binary vector representing presence or absence of human-meaningful components (pixels groups in an image, words in a sentence).

LIME creates perturbations around point x'_i to learn a the surrogate model g:

Model space vs human-interpretable space

x_i: the data point to be explained, in the input space of the original model f

x'_i: the interpretable representation of x, typically a binary vector representing presence or absence of human-meaningful components (pixels groups in an image, words in a sentence).

LIME creates perturbations around point x'_i to learn a the surrogate model g:

In the interpretable space, perturbed samples z' are generated by flipping some entries in x'_i.

z' is mapped back to the original feature space to get z and the model prediction $f(z)$.

The pairs z', $f(z)$ are weighted by proximity to x_i and used to train the surrogate model g.

The kernel is used to weigh perturbed samples z by distance to x_i; closer points count more when fitting g.

$$\mathcal{L}(f, g, \pi_x) = \sum_{z, z' \in \mathcal{Z}} \pi_x(z) \left(f(z) - g(z')\right)^2$$

Model space vs human-interpretable space

Summary:

x is the instance in the space of the model f (the space F)

x' is the human-interpretable binary version

z' are binary perturbations of x'

z is the corresponding realization in the feature space of the model f

$$\mathcal{L}(f, g, \pi_x) = \sum_{z, z' \in \mathcal{Z}} \pi_x(z) \left(f(z) - g(z') \right)^2$$

LIME - pseudocode

```
# given point x, model f, number of features k

x_prime = h(x)                                  # transform x to x' in interpretable binary space

z_prime = sample_binary_neighborhood(x_prime)   # generate points z' around x'

z  = map_to_model_space(z_prime)                # map z' back to model space by turning features off / on

y  = f(z)                                       # calculate model predictions on z

w  = kernel_distances(x, z)                     # get kernel weights by proximity of z to x

g = fit_weighted_sparse_linear(z_prime, y, w, k)   # fit surrogate model on z'

explanation = top_k_coeffs(g_hat, k)            # use the k top most important features as explanation
```

This might make more sense once you see it in action

LIME on image data

To understand this part of the lecture, it's useful to study the Jupyter Notebook image_LIME.ipynb

1. We need an image classification model
   We'll use resnet18 from pytorch, trained on Imagenet

2. We need images and functions for preprocessing them
   You'll get these from me

3. LIME needs a predict function that takes in a datapoint and returns one or more predictions
   You'll get this from me

LIME on image data. Step 1: Imports and model

```
import random, ast
import numpy as np
import matplotlib.pyplot as plt
import torch
import torchvision
from torchvision import models, transforms
import os

from PIL import Image
from lime import lime_image
from skimage.segmentation import mark_boundaries
```

See info and class indices at https://deeplearning.cms.waikato.ac.nz/user-guide/class-maps/IMAGENET/

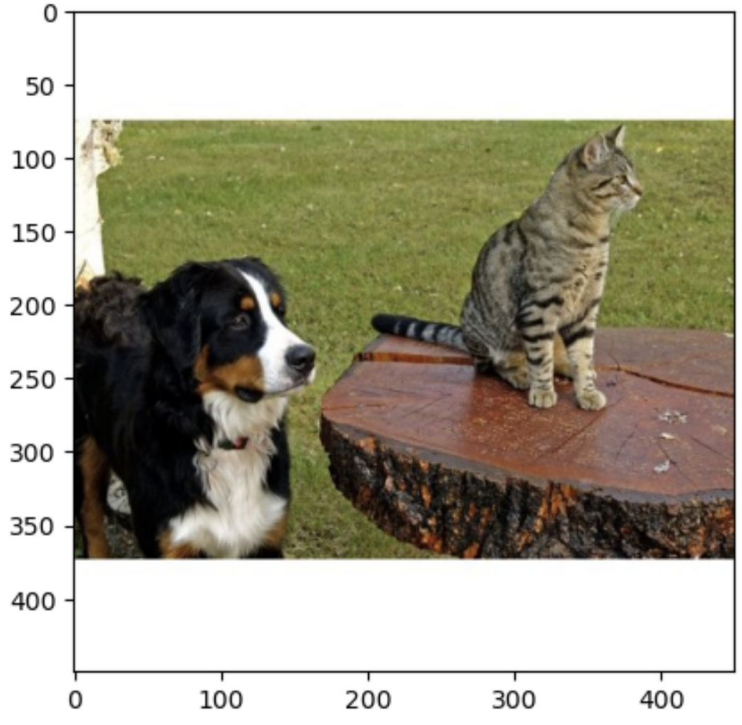
```
model = models.resnet18(weights=models.ResNet18_Weights.IMAGENET1K_V1)
```

LIME on image data. Step 2: Images and transformations

```
with open("imagenet1000_clsidx_to_labels.txt", "r", encoding="utf-8") as f:
    text = f.read()
labels_dict = ast.literal_eval(text)
```

```
def get_image(path):
    with open(os.path.abspath(path), 'rb') as f:
        with Image.open(f) as img:
            return img.convert('RGB')


def get_pil_transform():
    transf = transforms.Compose([
        transforms.Resize((256, 256)),
        transforms.CenterCrop(224)
    ])

    return transf

def get_preprocess_transform():
    normalize = transforms.Normalize(mean=[0.485, 0.456, 0.406],
                                     std=[0.229, 0.224, 0.225])
    transf = transforms.Compose([
        transforms.ToTensor(),
        normalize
    ])

    return transf

pill_transf = get_pil_transform()
preprocess_transform = get_preprocess_transform()
```

```
img = get_image('../data/cat_dog.png')
plt.imshow(img)
plt.show()
```



LIME on image data. Step 2.5: A prediction function for LIME

LIME needs a specific classification function.

Input: numpy array of images where each image is an `ndarray` of shape `(channel, height, width)`.

Output: numpy array of shape `(image index, classes)` where each value in the array should be probability for that image-class combination.

```
def prediction_function(images):
    model.eval()
    images_transformed = torch.stack(tuple(preprocess_transform(i) for i in images), dim=0)

    device = "cuda" if torch.cuda.is_available() else "cpu"
    model.to(device)
    model.eval()

    logits = model(images_transformed)
    preds = logits.softmax(dim=1)
    return preds.detach().cpu().numpy()
```

LIME on image data. Step 3: The explainer

```
explainer = lime_image.LimeImageExplainer()
explanation = explainer.explain_instance(np.array(pill_transf(img)),
                                         prediction_function,
                                         # number of labels (with highest prob) to show
                                         top_labels=5,
                                         # size of the neighborhood to learn the linear model
                                         num_samples=1000)
```

```
100%|██████████| 1000/1000 [04:22<00:00,  3.80it/s]
```

We use the `LimeImageExplainer()` for images. It takes as arguments:

1. The data point (transformed and as a numpy array)
2. The prediction function (see previous slide)

and optionally (among many more):

top_labels: how many labels to show (where those with the highest probabilities are shown)
num_samples: how many neighborhood points to generate and train the surrogate model on

LIME on image data. Step 3: The explainer

```
explanation.top_labels
```

```
[np.int64(240), np.int64(239), np.int64(241), np.int64(238), np.int64(281)]
```

```
top_classes = [labels_dict[_label] for _label in explanation.top_labels]
print(top_classes)
```

```
['Appenzeller', 'Bernese mountain dog', 'EntleBucher', 'Greater Swiss Mountain dog', 'tabby, tabby cat']
```

```
image, mask = explanation.get_image_and_mask(explanation.top_labels[1],
                                             # only include superpixels contributing positively
                                             positive_only=True,
                                             # number of superpixels to show
                                             num_features=6,
                                             # turn off the non-explanation part
                                             hide_rest=True)
img_boundary = mark_boundaries(image/255.0, mask)
plt.imshow(img_boundary)
plt.show()
```

Explain the top label according to the model (Appenzeller).

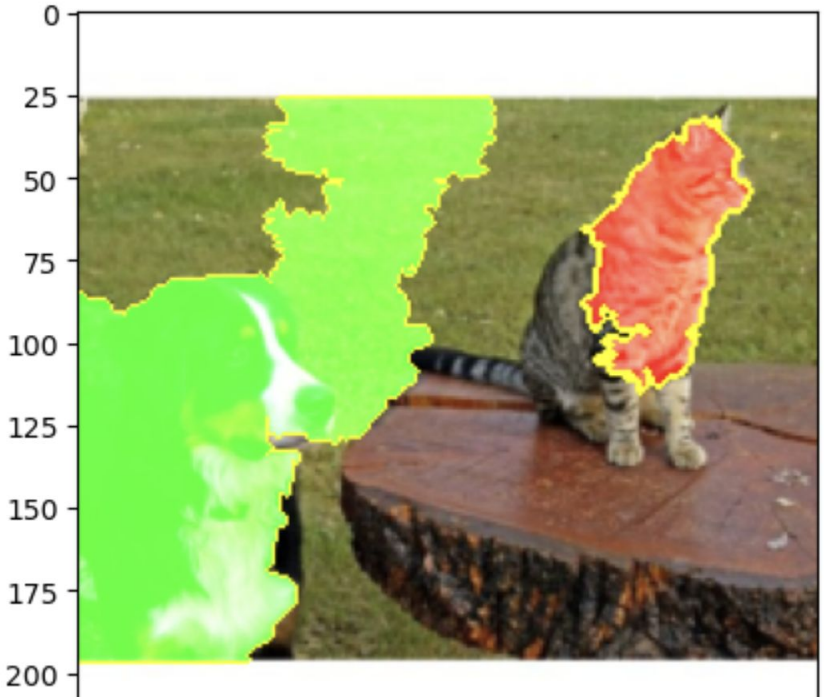
Show only the parts contributing towards this class.

Show `num_features` superpixels

Hide the rest of the image

```
image, mask = explanation.get_image_and_mask(explanation.top_labels[1],
                                             positive_only=False,
                                             negative_only=False,
                                             num_features=7,
                                             hide_rest=False)
img_boundary = mark_boundaries(image/255.0, mask)
plt.imshow(img_boundary)
```

```
<matplotlib.image.AxesImage at 0x354cfb410>
```

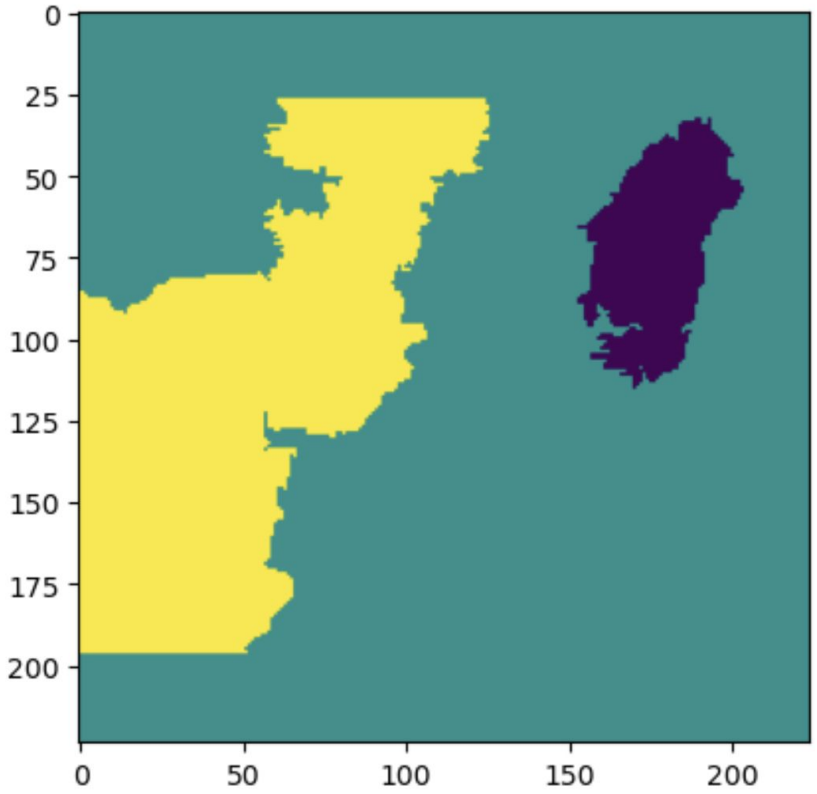


Including the negatively contributing features through

`negative_only = False`

we see that the model has understood that the cat is not relevant for classifying this dog.

```
plt.imshow(mask)
plt.show()
```




```
np.unique(np.array(mask))
```

```
array([-1,  0,  1])
```

The mask contains:

1 for positively contributing features
0 for features not part of the explanation
-1 for negatively contributing features

```
image, mask = explanation.get_image_and_mask(explanation.top_labels[1],
                                             # only include superpixels contributing positively
                                             positive_only=True,
                                             # number of superpixels to show
                                             num_features=3,
                                             # turn off the non-explanation part
                                             hide_rest=False)
img_boundary = mark_boundaries(image/255.0, mask)
plt.imshow(img_boundary)
plt.show()
```

At this point, the interpretable space makes more sense:

The interpretable features are pixel groups.

This explanation includes 3 interpretable features (pixel groups).

This plot still shows only those pixel groups / interpretable features contributing positively to the top label, which is the correct class.

The interpretable space and the surrogate model

In the interpretable space, we have two feature values:

- 1 → superpixel is **present** (original pixels kept when using the full model to predict)
- 0 → superpixel is **absent** (masked, typically replaced by a baseline such as the mean color)

The interpretable space and the surrogate model

The linear model's features are interpretable binary indicators of superpixel presence.

The local, linear surrogate model is of the form:

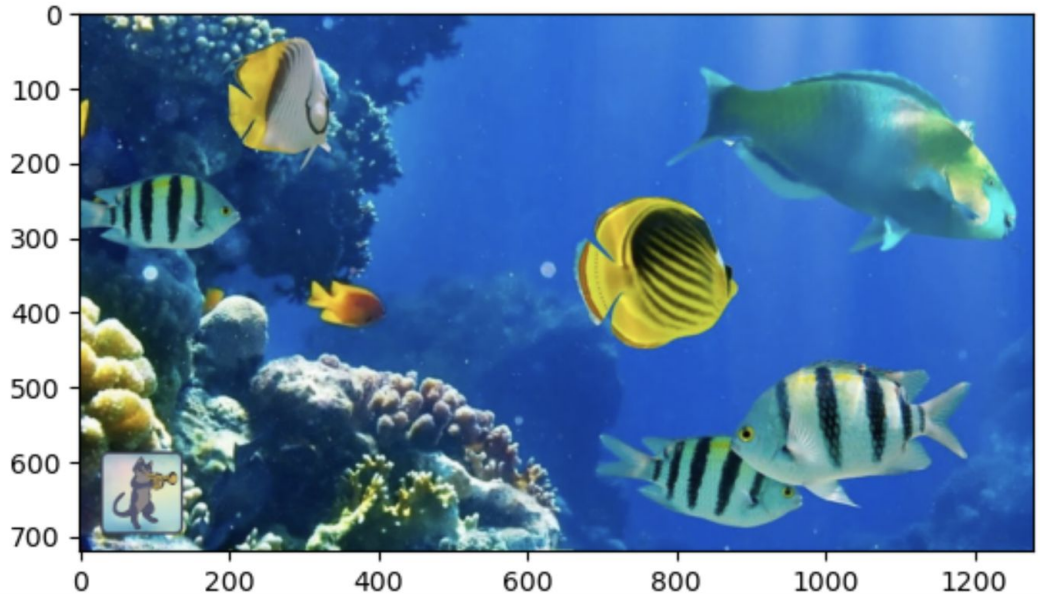
$$g(z') = \beta_0 + \sum_{i=1}^{K} \beta_i z_i'$$

- K = number of superpixels (number of features included),
- $z_i \in \{0,1\}$ indicates presence/absence of superpixel i,
- the coefficients β_i are used to represent the **local feature importance** of superpixel i for the prediction.

A few special cases to make you think

```
img = get_image('../data/aquarium.jpg')
plt.imshow(img)

explainer = lime_image.LimeImageExplainer()
explanation = explainer.explain_instance(np.array(pill_transf(img)),
                                         prediction_function,
                                         top_labels=5,
                                         num_samples=1000)

top_classes = [labels_dict[_label] for _label in explanation.top_labels]
print(top_classes)
```

```
  0%|          | 0/1000 [00:00<?, ?it/s]
['coral reef', 'scuba diver', 'rock beauty, Holocanthus tricolor', 'jellyfish', 'anemone fish']
```

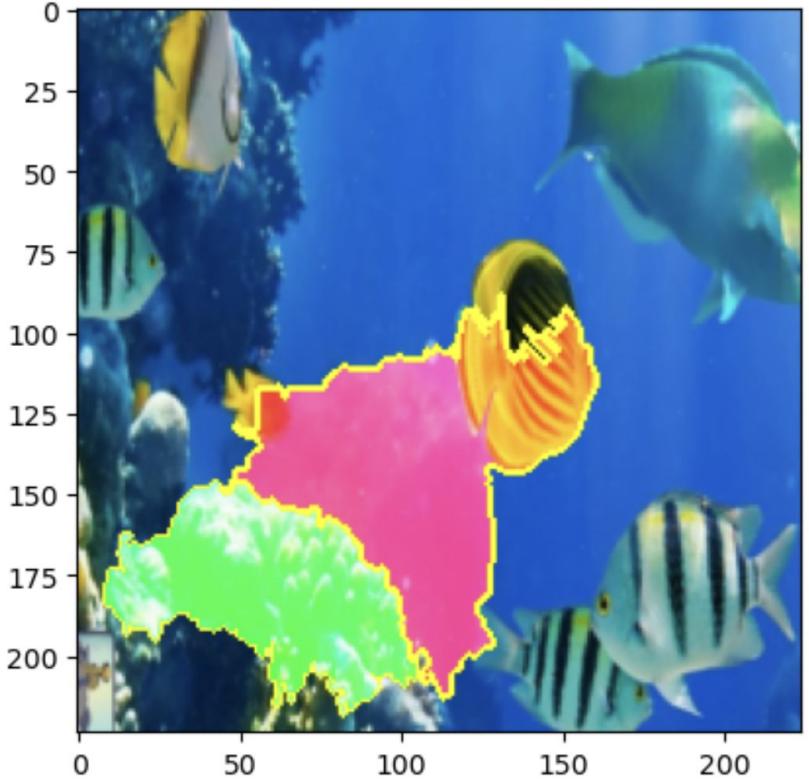


There is a coral reef on the image, so this looks good.

What about the explanation?

Why is it a coral reef?

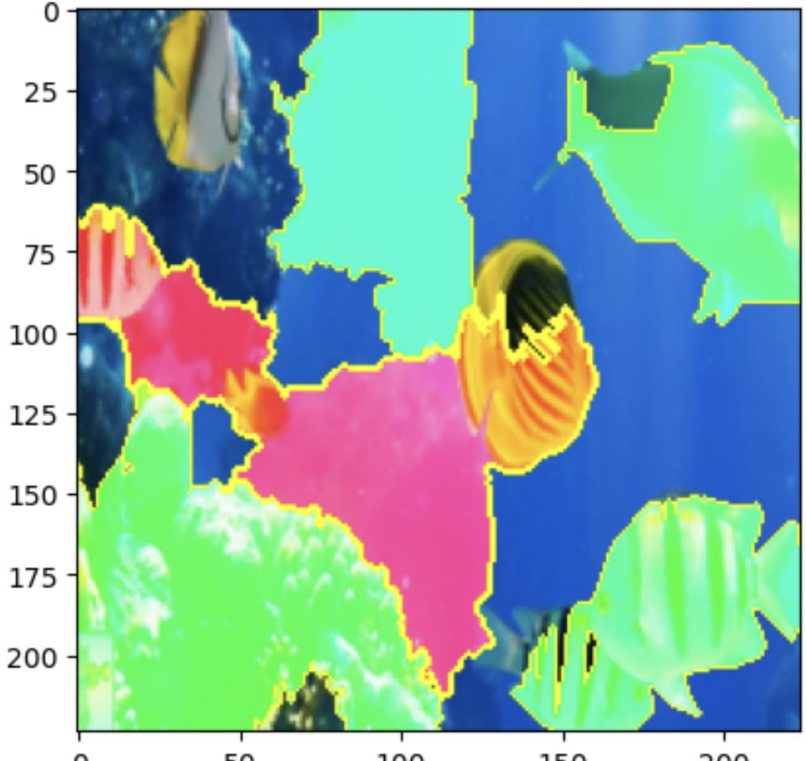
```
image, mask = explanation.get_image_and_mask(explanation.top_labels[0],
                                             positive_only=False,
                                             num_features=3,
                                             hide_rest=False)
img_boundary = mark_boundaries(image/255.0, mask)
plt.imshow(img_boundary)
plt.show()
```



Why is it a coral reef?

Still looks good. The reef is highlighted.

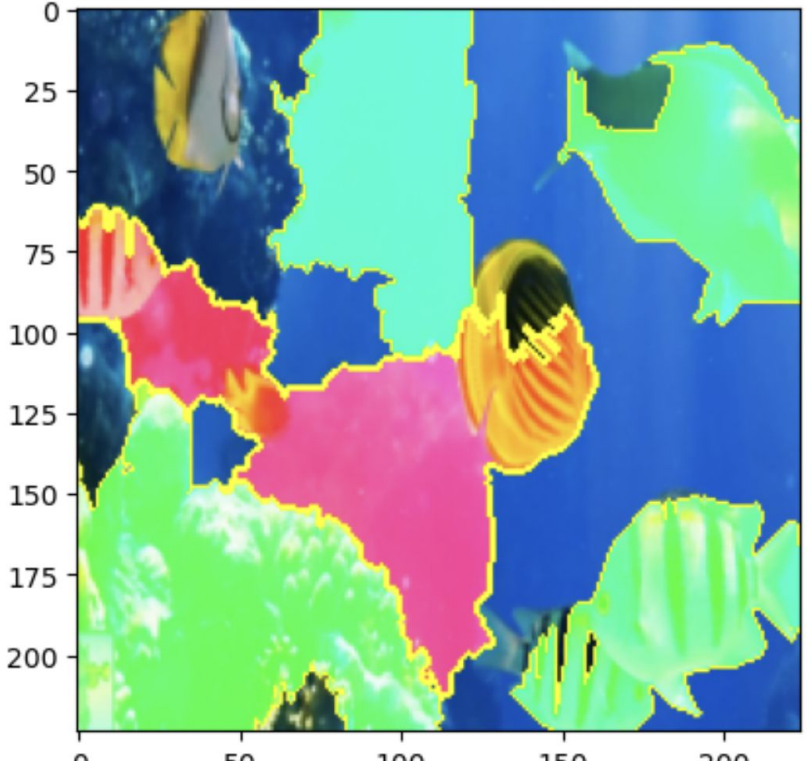
```
image, mask = explanation.get_image_and_mask(explanation.top_labels[0],
                                             positive_only=False,
                                             num_features=10,
                                             hide_rest=False)
img_boundary = mark_boundaries(image/255.0, mask)
plt.imshow(img_boundary)
plt.show()
```



Wait a minute!

What do you think about this?

```
image, mask = explanation.get_image_and_mask(explanation.top_labels[0],
                                             positive_only=False,
                                             num_features=10,
                                             hide_rest=False)
img_boundary = mark_boundaries(image/255.0, mask)
plt.imshow(img_boundary)
plt.show()
```



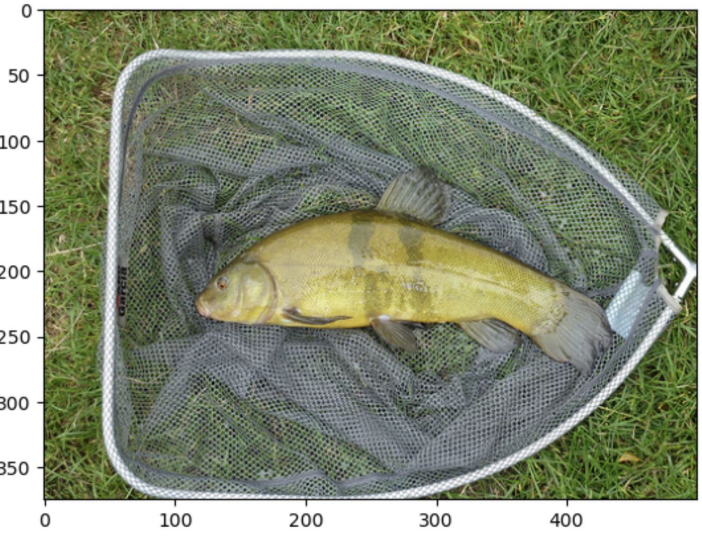
Wait a minute!

Turning on more features, we see that the model might be confused about what a coral reef is.

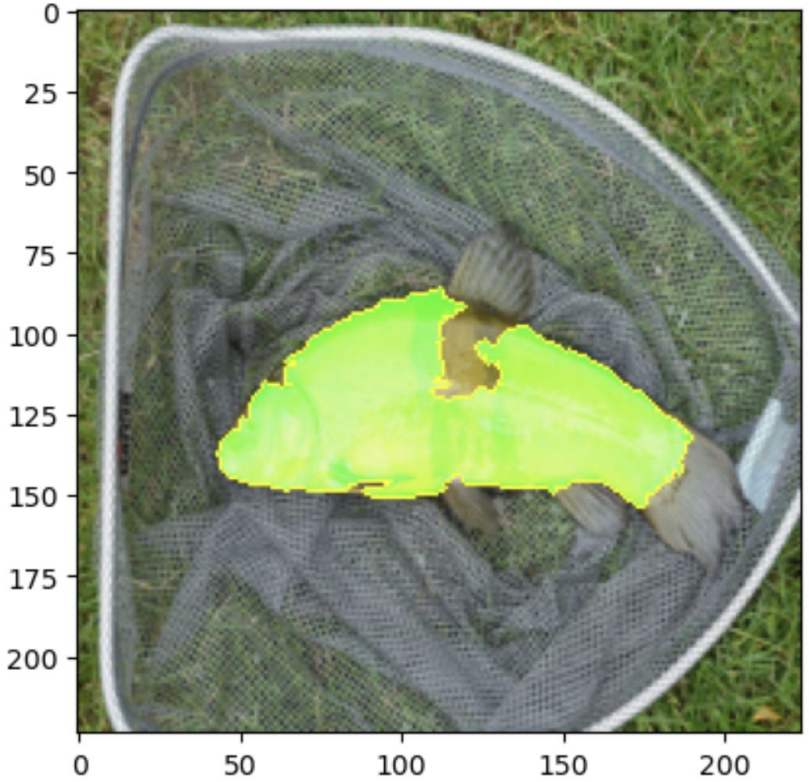
It is not unlikely that the coral reef class correlates strongly with colorful stuff and fish in general.

```
img = get_image('../data/fish.jpg')
plt.imshow(img)

explainer = lime_image.LimeImageExplainer()
explanation = explainer.explain_instance(np.array(pill_transf(img)),
                                         prediction_function,
                                         top_labels=5,
                                         num_samples=1000)

top_classes = [labels_dict[_label] for _label in explanation.top_labels]
print(top_classes)
```

```
100%|██████████| 1000/1000 [04:01<00:00,  4.15it/s]
['tench, Tinca tinca', 'bolete', 'armadillo', 'isopod', 'mud turtle']
```

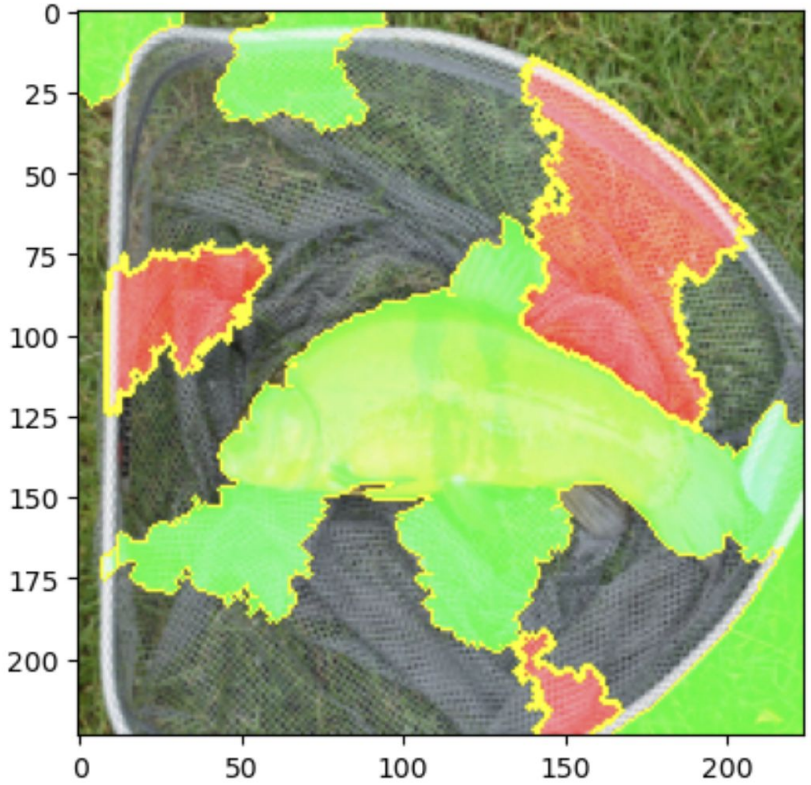


```
image, mask = explanation.get_image_and_mask(explanation.top_labels[0],
                                             positive_only=False,
                                             num_features=3,
                                             hide_rest=False)
img_boundary = mark_boundaries(image/255.0, mask)
plt.imshow(img_boundary)
plt.show()
```



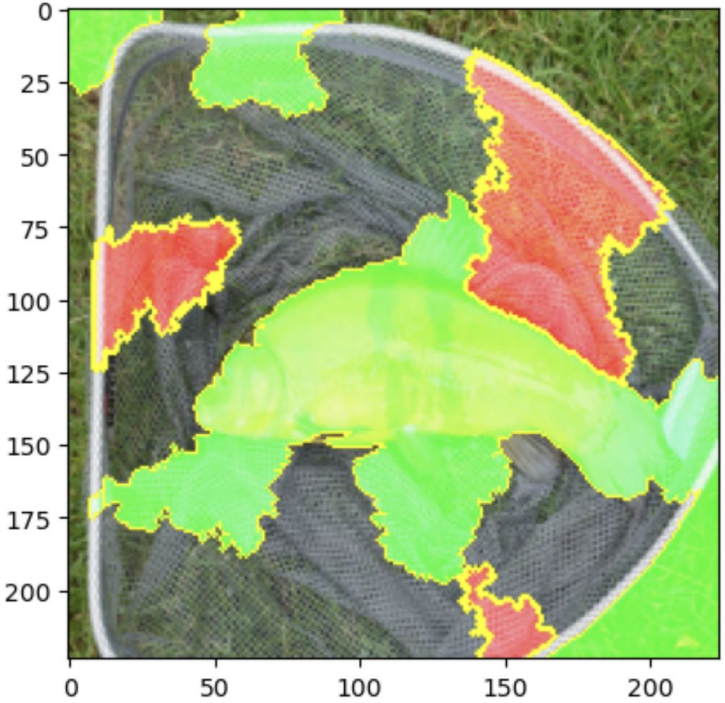
This looks solid; the top 3 features see only the fish

```
image, mask = explanation.get_image_and_mask(explanation.top_labels[0],
                                             positive_only=False,
                                             num_features=15,
                                             hide_rest=False)
img_boundary = mark_boundaries(image/255.0, mask)
plt.imshow(img_boundary)
plt.show()
```



The top 3 features see only the fish

… but turning on more features tells us that it's a bit random what speaks for and what speaks against.



In the homework, you should play around with the parameters to see how it changes the explanation, and think about what the behaviour might tell you about the full model, the surrogate model and the LIME parameters.

Speaking of homework...

Homework :)

Import the LIME library, and

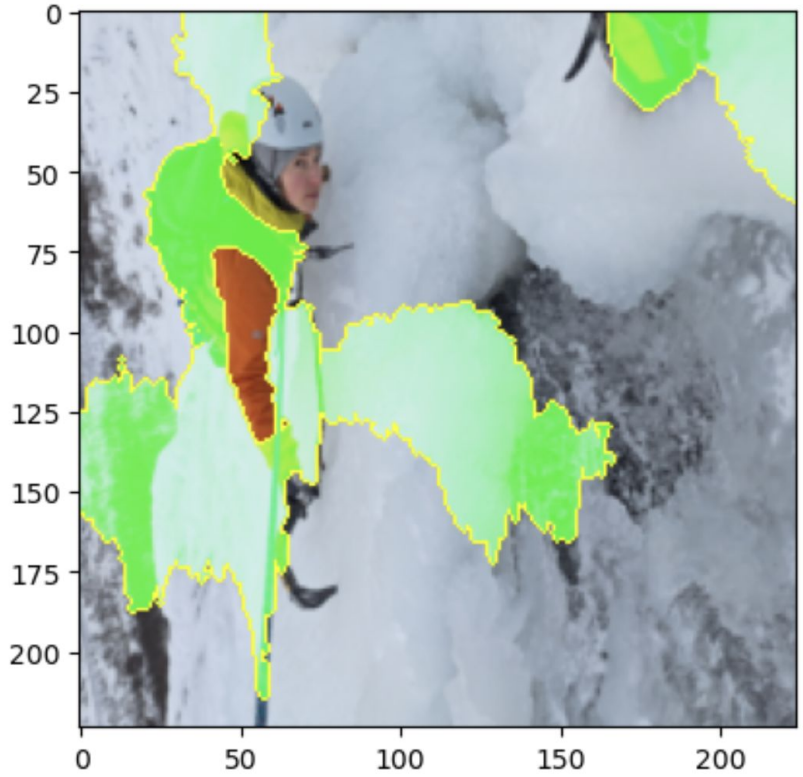
1) Train a model on the Titanic data or use the one in the titanic_LIME notebook. Use LIME to generate explanations for the tabular data.
2) Find an image classification model, or use the one in the image_LIME notebook. Use LIME to generate explanations for the aquarium, cat_dog and fish images. Play around with the parameters until you get an explanation you like.
3) Take your own image and repeat point 2) on that.

```
img = get_image('../data/inga_ice.jpg')
plt.imshow(img)
plt.show()
```


```
top_classes = [labels_dict[_label] for _label in explanation.top_labels]
print(top_classes)
```

```
['ski', 'shovel', 'snowmobile', 'snowplow, snowplough', 'ski mask']
```

```
image, mask = explanation.get_image_and_mask(explanation.top_labels[0],
                                             positive_only=False,
                                             num_features=5,
                                             hide_rest=False)
img_boundary = mark_boundaries(image/255.0, mask)
plt.imshow(img_boundary)
plt.show()
```



Imagenet probably doesn't know what an ice axe is

RESEARCH-ARTICLE | PUBLIC ACCESS

"Why Should I Trust You?": Explaining the Predictions of Any Classifier

Authors:  Marco Tulio Ribeiro,  Sameer Singh,  Carlos Guestrin | Authors Info & Claims

KDD '16: Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining
Pages 1135 - 1144 • https://doi.org/10.1145/2939672.2939778

from the abstract:

Understanding the reasons behind predictions is quite important in **assessing trust**,

Such understanding also provides **insights into the model**, which can be used to transform an untrustworthy model or prediction into a **trustworthy** one.

LIME explains the predictions of any classifier in an **interpretable and faithful** manner, by learning an interpretable model locally around the prediction.

Strengths

What do you like about LIME?

Strengths

LIME is easy to use, and available in Python and R.

LIME is one of the few methods that works for tabular data, images and text.

All surrogate model explanations benefit from the significant availability of interpretable models, and literature on their use and interpretation.

The fidelity measure between the full and the surrogate model provides us with an estimate of how reliable the explanation is in the neighborhood.

Interpretable features are easier for the recipient to understand than non-interpretable features (which many models are trained on).

Weaknesses

What do you not like so much about LIME?

Weaknesses

The choice of neighborhood is an unsolved problem when using LIME with tabular data. This is probably the main weakness, and it is important to find an appropriate kernel width.

Data points are sampled from a Gaussian distribution, ignoring the correlation between features. This can lead to unlikely data points, used to train g.

Repeating the sampling process can significantly change the explanations (just try in the provided notebook).

With many free parameters, LIME explanations can be manipulated to hide bias and other model weaknesses (just try in the provided notebook).

Have fun finding importances of tabular and interpretable image features :)

See you tomorrow!

The class of interpretable models

The class of interpretable models G

https://github.com/marcotcr/lime

Linear regression can be chosen as an interpretable surrogate model. In advance, you have to select K, the number of features you want to have in your interpretable model. The lower K, the easier it is to interpret the model. A higher K potentially produces models with higher fidelity. There are several methods for training models with exactly K features. A good choice is Lasso. A Lasso model with a high regularization parameter lambda yields a model without any feature. By retraining the Lasso models with slowly decreasing lambda, one after the other, the features get weight estimates that differ from zero. If there are K features in the model, you have reached the desired number of features. Other strategies are forward or backward selection of features. This means you either start with the full model (= containing all features) or with a model with only the intercept and then test which feature would bring the biggest improvement when added or removed, until a model with K features is reached.

The complexity of the explanation model has to be defined in advance. This is just a small complaint because, in the end, the user always has to define the compromise between fidelity and sparsity.