Review of LIME interpreting method

Contents

Introduction

This review is the second piece of interpretable machine learning topic. In this one, we focus on LIME, Local Interpretable Model-agnostic Explanations, it is an algorithm to interpret a model by finding an explainable local area around the prediction (Sharma 2018). This local area contains samples close to the investigated sample, we can usually fit an easier model (linear model for instance) into this area and explain the predictions. We will discuss how LIME works.

LIME explanation

Ribeiro, Singh and Guestrin (2016) states that LIME is an algorithm that uses an interpretable model explain classifier or model locally in a faithful way.

Before we explain how LIME works, we need to understand 2 words when we interpret a model, one is Fidelity, it means we want our explanation model to have the lowest unfaithfulness to the original model, or at least in the local area (shown in the Figure 1 below), so in another word, we want our explanation model to have close to the same prediction as the original model.

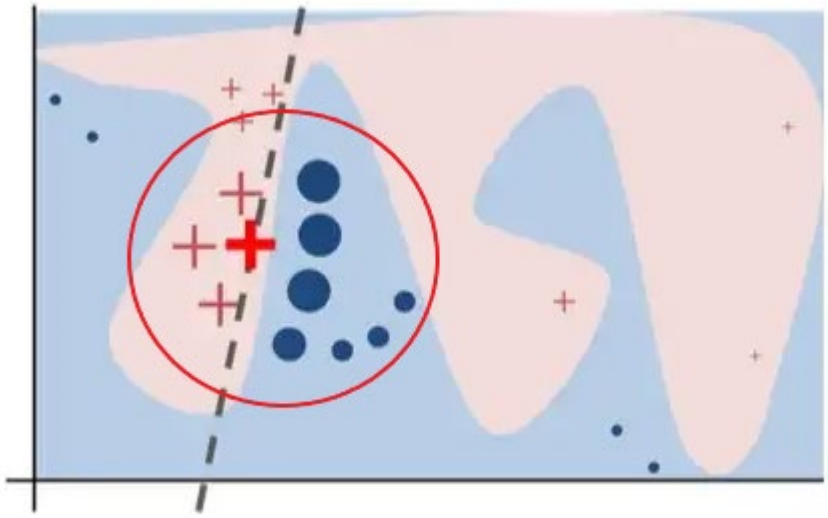


Figure 1 find a local area to form a interpretable model (adapted from Ribeiro, Singh and Guestrin 2016)

The second is interpretability, it means we want the explanation model to be as easy as possible so people can understand.

Then we can further understand the following formular:

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

Formular 1 model unfaithfulness and model complexity (Ribeiro, Singh and Guestrin 2016)

G is all possible easy interpretable model such as linear model, decision tree model, f is the original mode, g is the interpretable model under G, π is the local instances, $\mathcal{L}$ means the unfaithfulness, Ω means the complexity of the model. Normally, the more complex the g is, the less unfaithfulness it has, since it will be close to the original model. The goal is to minimize the unfaithfulness of the interpretable model, while keep the complexity of the model low.

This is how the local model comes in, because if we can find a simple model to only interpret a local area, both complexity and unfaithfulness (in local area) will be low. As Figure 1 shows, the original class labels could not be interpreted by an easy model (see the boundary between red and blue area), however, in the red circled local area the dashed line perfectly separated the 2 class labels (red cross and blue dots). We use the locally weighted square loss as $\mathcal{L}$, to present the unfaithfulness, then we get

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

Equation 2 use distance measure represents unfaithfulness

Lower the faithfulness becomes minimize the locally weighted square loss. We will show how LIME works in the following example.

Interpret Random Forest model on Boston Dataset

For Random Forest model, we are going to use 1000 trees to get average prediction for housing price, the code shown in Appendix A.

```
rf.score(test, labels_test)
```

```
0.8824332524624339
```

As we can see the R^2 is 0.88, which is good.

```
# get categorical feature names from the array, unique value <10 means categorical data.
categorical_features = np.argwhere(np.array([len(set(boston.data[:,x])) for x in range(boston.data.shape[1])]) <= 10).flatten()
```

```
!pip install lime
import lime
import lime.lime_tabular
```

Then we get categorical feature names, build LIME explainer by using *LimeTabularExplainer.*

```
explainer = lime.lime_tabular.LimeTabularExplainer(train, feature_names=boston.feature_names, class_names=['price'],
                                                   categorical_features=categorical_features, verbose=True, mode='regression')
```

We explain the first test result.

```
i = 1
exp = explainer.explain_instance(test[i], rf.predict, num_features=5)
```

```
Intercept 20.032287755327314
Prediction_local [36.21572914]
Right: 30.59949999999986
```

```
exp.show_in_notebook(show_table=True)
```

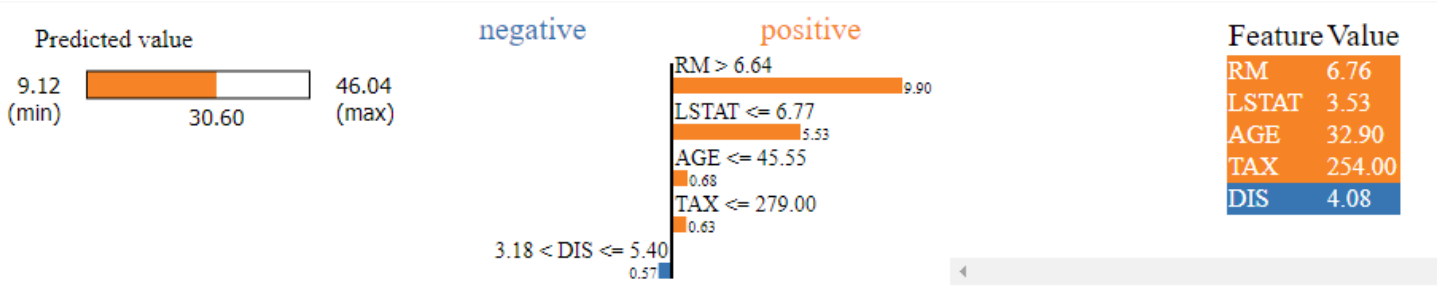


Figure 2 feature importance on first prediction on test data

The feature importance related to the score generated by decision tree model in the local, as Figure 2 shows, the left one show the position of the explained value (30.6) amount all the predicted values. The middle one shows the top 5 features contributed to the explained value, RM being the first, scores 9.9, then LSTAT the second, the score is 5.53. It also shows which category each feature's value falls into, for example RM > 6.64, LSTAT <= 6.67 (LIME automatically

discretize the continuous values into categories). The graph on the right is the actual feature values for the explained instance.

We can also show the details in a list:

```
exp.as_list()
```

```
[('RM > 6.64', 9.903297463058937),
 ('LSTAT <= 6.77', 5.531982701480015),
 ('AGE <= 45.55', 0.6830300162325166),
 ('TAX <= 279.00', 0.6304426113829611),
 ('3.18 < DIS <= 5.40', -0.5653114096848009)]
```

Or show more features in a bar chart (shown in figure 3).

```
%matplotlib inline
i = 1
exp = explainer.explain_instance(test[i], rf.predict, num_features=14)
fig = exp.as_pyplot_figure();
```

```
Intercept 19.88551978167734
Prediction_local [36.87684323]
Right: 30.59949999999986
```

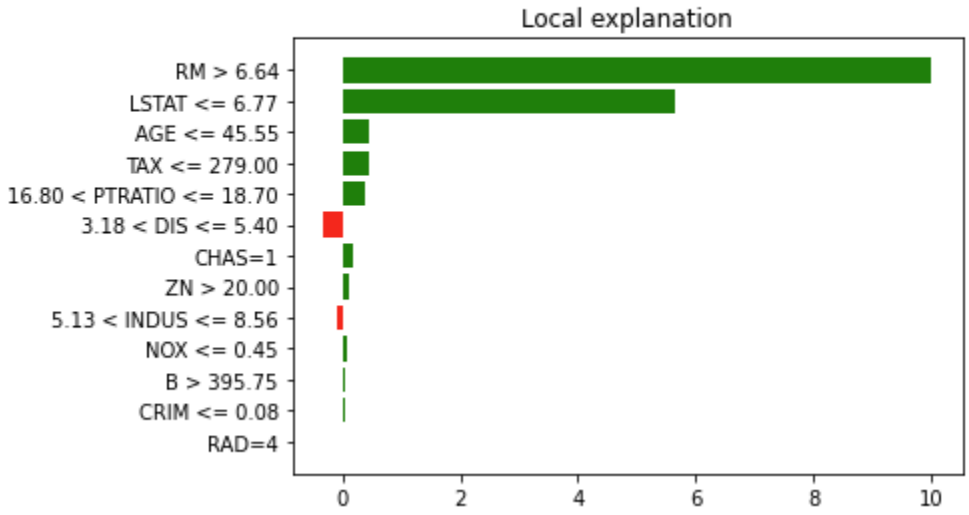


Figure 3 Bar chart for feature contribute for the first prediction in test data.

If we want to explain multiple instances, we can use **SP-LIME** (*submodular_pick* module), which will choose a set of representative instances to explain the whole model via submodular optimization (Ribeiro, Singh and Guestrin 2016).

The submodular pick algorithm will try to choose the instances give the maximum amount of feature coverage in the global scale; a simple explanation shown below:

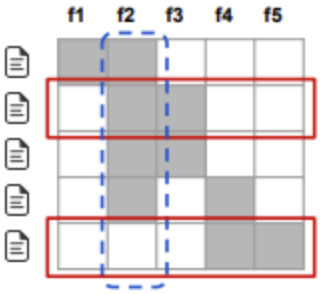


Figure 4 a simple explanation for submodular pick method(picture from Ribeiro, Singh and Guestrin 2016)

We choose the instance row 2 and row 5 (red rectangle) which give the most feature coverage in the whole set. How did it achieve this? First, we find that feature 2 have the largest feature importance since it has present in 4 of the instances (blue dashed rectangle), features 1 and 5 only have 1 instance coverage. If we choose the row 2 and row 5 as representatives, we included 4 of the 5 features, leave only feature 1 out, which is optimum. We used least instances covers the most features (important features). Ribeiro, Singh and Guestrin (2016) also gives the formular to summaries the algorithm:

$$c(V, \mathcal{W}, I) = \sum_{j=1}^{d'} \mathbb{1}_{[\exists i \in V : \mathcal{W}_{ij} > 0]} I_j$$

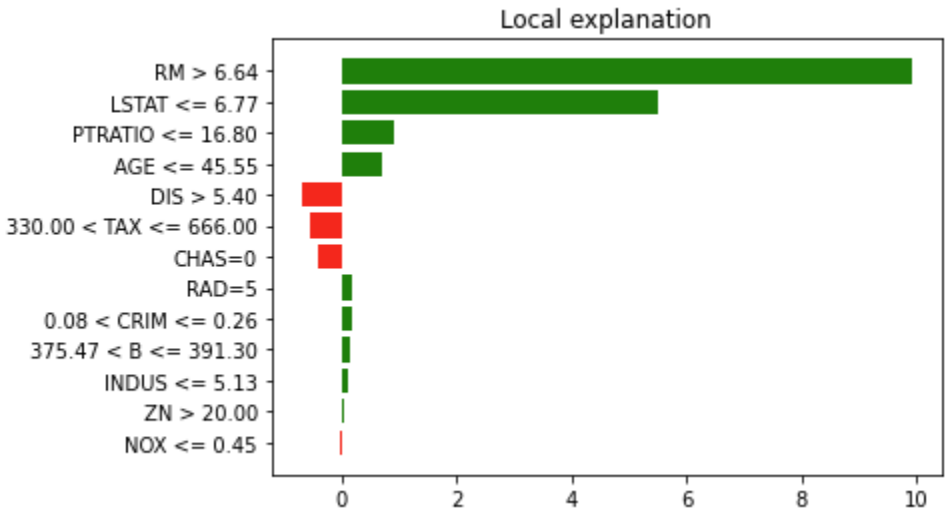
$$Pick(\mathcal{W}, I) = \underset{V, |V| \leq B}{\operatorname{argmax}}\, c(V, \mathcal{W}, I)$$

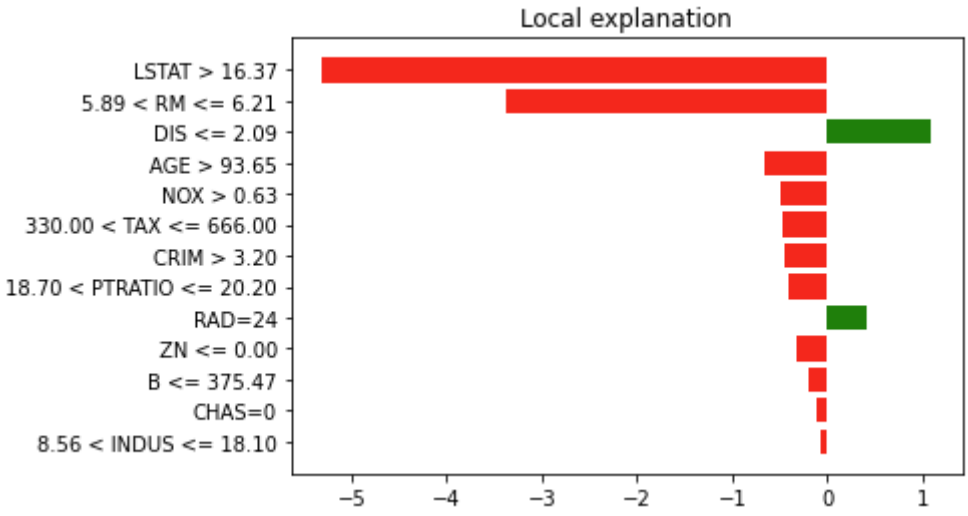
V is the coverage set we trying to choose. W is the local importance for each instance. B is the seleted instances.

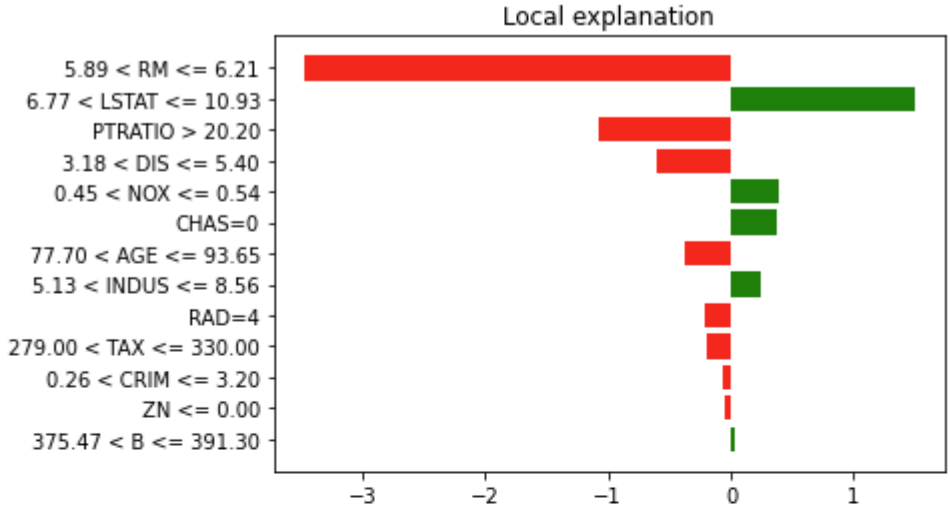
The following code randomly choose 20 instances to explain, only 5 explanations with show 14 features.

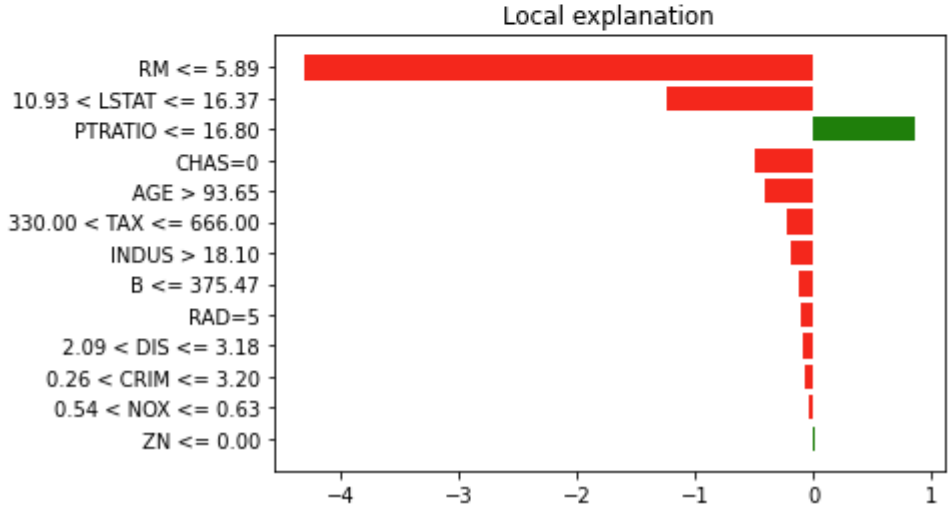
```
import warnings
from lime import submodular_pick
sp_obj = submodular_pick.SubmodularPick(explainer, train, rf.predict, sample_size=20, num_features=14, num_exps_desired=5)
```

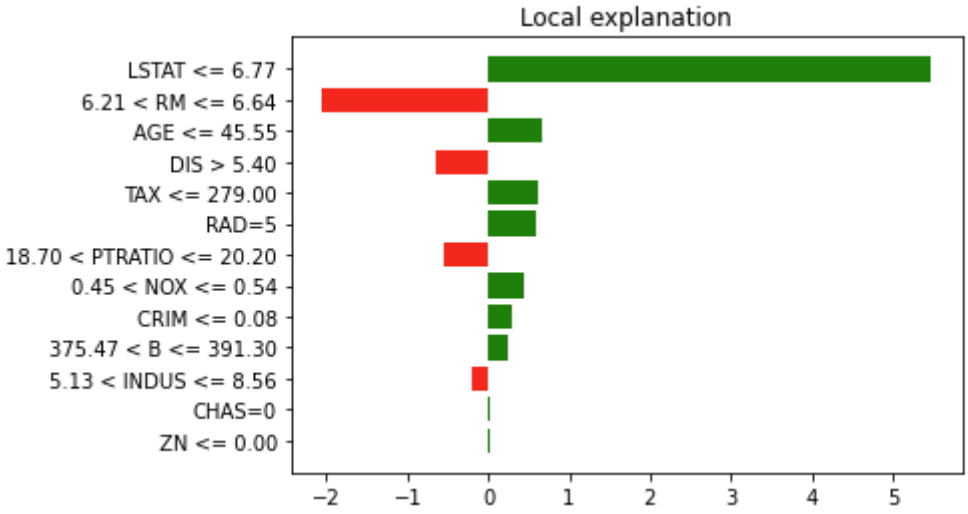
```
[exp.as_pyplot_figure() for exp in sp_obj.sp_explanations];
```




Local explanation
LSTAT > 16.37
5.89 < RM <= 6.21
DIS <= 2.09
AGE > 93.65
NOX > 0.63
330.00 < TAX <= 666.00
CRIM > 3.20
18.70 < PTRATIO <= 20.20
RAD=24
ZN <= 0.00
B <= 375.47
CHAS=0
8.56 < INDUS <= 18.10
−5
−4
−3
−2
−1
0
1


Local explanation
5.89 < RM <= 6.21
6.77 < LSTAT <= 10.93
PTRATIO > 20.20
3.18 < DIS <= 5.40
0.45 < NOX <= 0.54
CHAS=0
77.70 < AGE <= 93.65
5.13 < INDUS <= 8.56
RAD=4
279.00 < TAX <= 330.00
0.26 < CRIM <= 3.20
ZN <= 0.00
375.47 < B <= 391.30
−3
−2
−1
0
1


Local explanation
RM <= 5.89
10.93 < LSTAT <= 16.37
PTRATIO <= 16.80
CHAS=0
AGE > 93.65
330.00 < TAX <= 666.00
INDUS > 18.10
B <= 375.47
RAD=5
2.09 < DIS <= 3.18
0.26 < CRIM <= 3.20
0.54 < NOX <= 0.63
ZN <= 0.00
−4
−3
−2
−1
0
1



LIME interpretation for text data

Interpretation model could also help us identify problems in our original model.

In the following example, LIME use SVM to interpret text data which is predicted by Random Forest classifier (code is from Jamesmyatt 2019, full code is at Appendix B).

```
print(sklearn.metrics.f1_score(newsgroups_test.target, pred, average='binary'))
```

```
0.9230769230769231
```

```
exp.show_in_notebook()
```

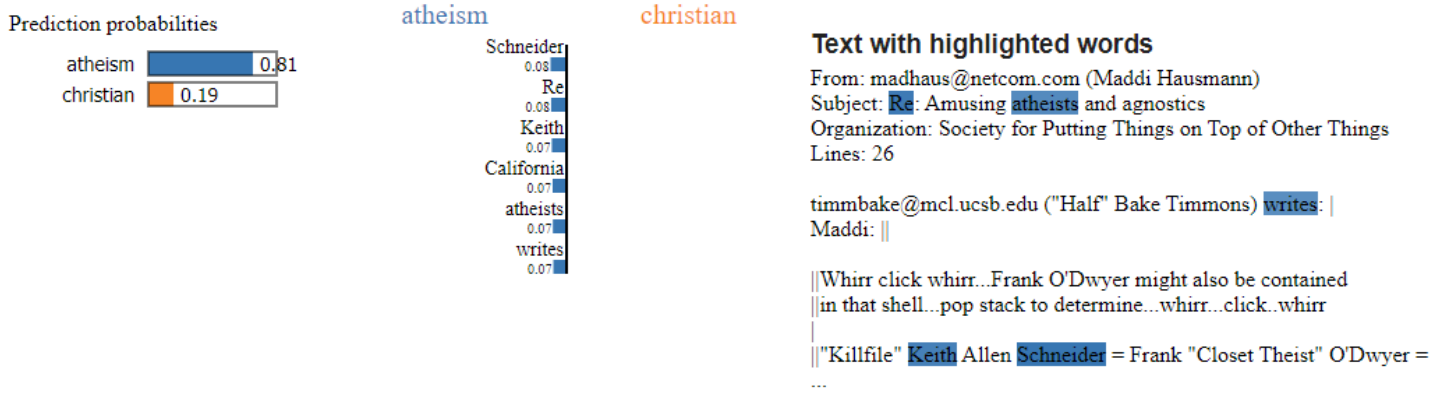


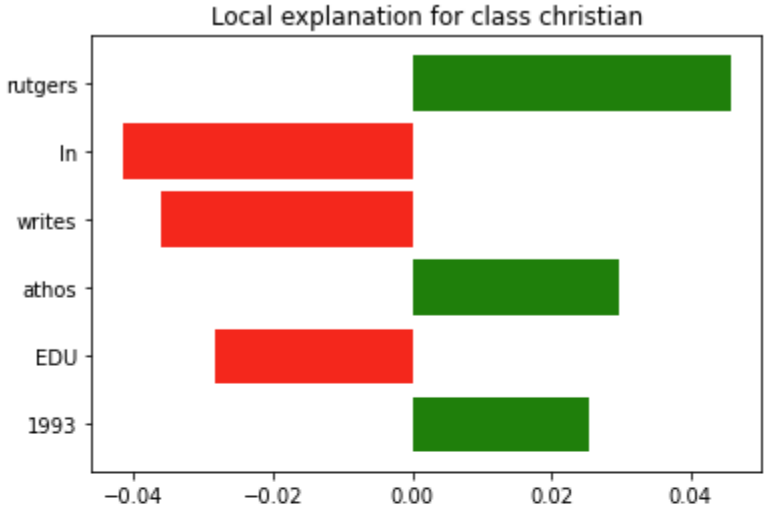
Figure 5 interpreting text data use LIME

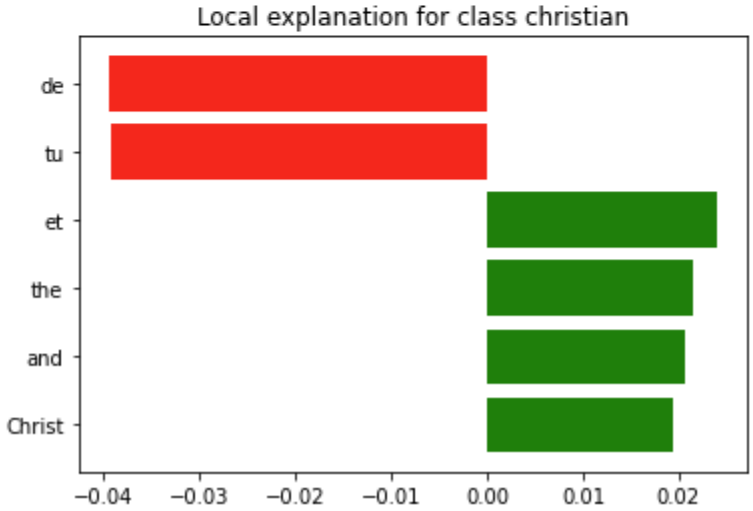
Even thou the predictions are very accurate (f1 score is 0.92), but when we try to explain an instance, we realized something was not right. From Figure 4, we can see that the words which contributed to the class label don't make much sense: 'Schneider, 'Keith and 'writes' contribute to **atheism**?

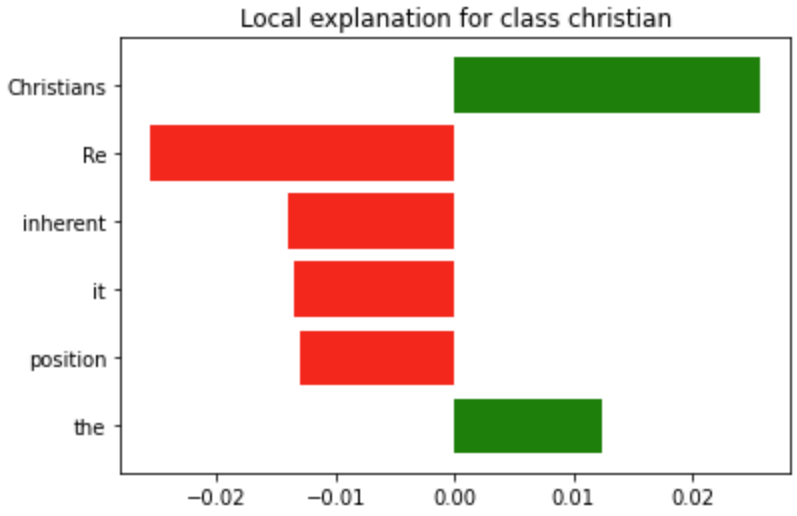
We can show more examples:

```
from lime import submodular_pick
sp_obj = submodular_pick.SubmodularPick(explainer, newsgroups_test.data, c.predict_proba, sample_size=3, num_features=6, num_exps_desired=3)
```

```
[exp.as_pyplot_figure(label=exp.available_labels()[0]) for exp in sp_obj.sp_explanations];
```







Clearly some of these words cannot be used to predict “Atheism” or “Christianity”.

In this example, we used LIME to identify the problem of the prediction, we should not rely on the original model to predict this text data (Ribeiro, Singh and Guestrin 2016).

LIME interpretation for image

Machine learning algorithm could identify if there is a cat or dog in a image, our interpretation task will be explain why the model made such prediction.

Before we introduce LIME interpreting on image predictions, we need to understand an important concept: **Superpixcels**. Superpixels means grouping pixels that similar in color and other low-level properties together, which will heavily reduce the number of primitives for subsequent algorithms (Ren and Malik, 2003, as cited in Stutz, Hermans, Leibe in 2018). This segmentation process makes image processing faster and more meaningful.

So, think superpixcels(areas of the image) like important features that contribute to the model’s prediction, our task is to find the specific area to explain why the prediction has been made.

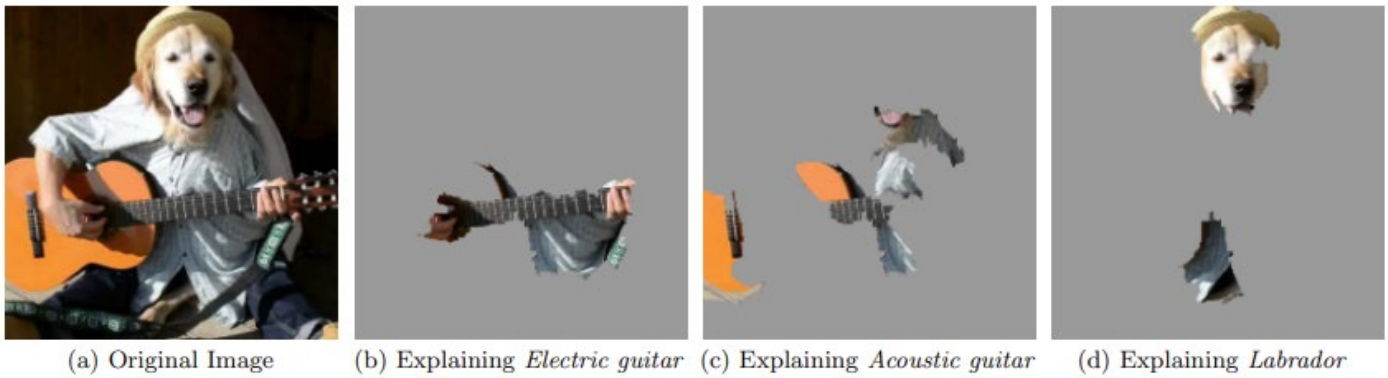


Figure 6 explaining different superpixcels could contribute to different prediction outcomes (image from Ribeiro, Singh and Guestrin, 2016)

We will use a code example from Aditya Bhattacharya (link see reference) to demonstrate how to use LIME to interpret.

First, we predict an image data by using a pre-trained **Xception** model in Keras. Xception is a convolutional neural network that has 71 layers.

```
import warnings
warnings.filterwarnings("ignore")
import os
os.environ['TF_CPP_MIN_LOG_LEVEL'] = '3'

import numpy as np
import matplotlib.pyplot as plt
import matplotlib.cm as c_map
from IPython.display import Image, display
import tensorflow as tf
from tensorflow import keras
from tensorflow.keras.applications.xception import Xception, preprocess_input, decode_predictions
from tensorflow.keras.preprocessing import image

import lime
from lime import lime_image
from lime import submodular_pick

from skimage.segmentation import mark_boundaries

np.random.seed(123)
```

```
def load_image_data_from_url(url):
    '''
    Function to load image data from online
    '''
    # The local path to our target image
    image_path = keras.utils.get_file(
    "shark.jpg", url
    )

    display(Image(image_path))
    return image_path
url = "https://images.unsplash.com/photo-1560275619-4662e36fa65c?ixlib=rb-1.2.1&ixid=MnwxMjA3fDB8MHxwaG90by1wYWdlfHx8fGVufDB8fHx8&auto=format&fit=crop&w=1200&q=80"
image_path = load_image_data_from_url(url = url)
```

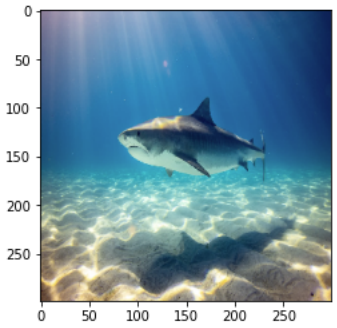
Figure 7 image for modeling

```
IMG_SIZE = (299, 299)
def transform_image(image_path, size):
    '''
    Function to transform an image to normalized numpy array
    '''
    img = image.load_img(image_path, target_size=size)
    img = image.img_to_array(img)# Transforming the image to get the shape as [channel, height, width]
    img = np.expand_dims(img, axis=0) # Adding dimension to convert array into a batch of size (1,299,299,3)
    img = np.double(img/255.0) # normalizing the image to keep within the range of 0.0 to 1.0

    return img

normalized_img = transform_image(image_path, IMG_SIZE)
```

```
from tensorflow.keras.applications.xception import Xception
model = Xception(weights="imagenet")
```

```
def get_model_predictions(data):
    model_prediction = model.predict(data)
    print(f"The predicted class is : {decode_predictions(model_prediction, top=1)[0][0][1]}")  # top means how many top-guesses to return
    return decode_predictions(model_prediction, top=1)[0][0][1]

plt.imshow(normalized_img[0])
pred_orig = get_model_predictions(normalized_img)
```

```
1/1 [==============================] - 1s 879ms/step
The predicted class is : tiger_shark
```



We then show the top 5 predicts for the give image:

```
# top 5 predicts
model_prediction = model.predict(normalized_img)
top5_pred = decode_predictions(model_prediction, top=5)[0]
for pred in top5_pred:
    print(pred[1])
```

```
1/1 [==============================] - 0s 363ms/step
tiger_shark
great_white_shark
scuba_diver
coral_reef
stingray
```

We can clearly see the top prediction is a tiger shark.

Then we use LIME's image explainer.

```
# explain the prediction
explainer = lime_image.LimeImageExplainer()
```

```
# explain_instance: First, we generate neighborhood data by randomly perturbing features from the instance.
# We then learn locally weighted linear models on this neighborhood data to explain each of the classes in an interpretable way.
exp = explainer.explain_instance(normalized_img[0],
                                 model.predict,
                                 top_labels=5,
                                 hide_color=0,
                                 num_samples=1000)   # num_samples = size of the neighborhood to learn the linear model
```

Explain_instance will first generate neighborhood data by randomly perturbing features from the instance, then learn locally weighted linear models on this neighborhood data to explain each of the classes in an interpretable way.

The segments differences are shown in below:

```
plt.imshow(exp.segments)
plt.axis('off')
plt.show()
```

Now, we can show the Area (Superpixels) that contribute to the tiger shark prediction. We use *exp.get_image_and_mask* to return an image (3d numpy array) and a mask (2d numpy array). Then use *imshow* and *mark_boundaries* to create the image. This will show a clear boundary that highlight the area we need. Choose positive only to only show positive superpixels.

```
def generate_prediction_sample(exp, exp_class, weight = 0.1, show_positive = True, hide_background = True):
    '''
    Method to display and highlight super-pixels used by the black-box model to make predictions
    '''
    #  use 'get_image_and_mask' to return a image - 3d numpy array and a mask - 2d numpy array that can be used with skimage.segmentation.mark_boundaries
    image, mask = exp.get_image_and_mask(exp_class,  # label to explain
                                         positive_only=show_positive,   # if True, only take superpixels that positively contribute to the prediction of the label.
                                         num_features=6,          # number of superpixels to include in explanation
                                         hide_rest=hide_background,
                                         min_weight=weight     # minimum weight of the superpixels to include in explanation
                                        )
    plt.imshow(mark_boundaries(image, mask))    # creates an image from a 2-dimensional numpy array
    plt.axis('off')
    plt.show()

# show without background
generate_prediction_sample(exp, exp.top_labels[0], show_positive = True, hide_background = True)
```

We can also show with background by setting *hide_background* to True.

```
# show with background
generate_prediction_sample(exp, exp.top_labels[0], show_positive = True, hide_background = False)
```

Or we can highlight the positive superpixel region by setting *show_positive* to False. Now the area become green.

```
# green highlight the positive region
generate_prediction_sample(exp, exp.top_labels[0], show_positive = False, hide_background = False)
```

To get a bigger picture of how important each superpixel is (figure 8).

We first turn the explained class into a dictionary (key,value) then we define a vectorized function translate every element in numpy array according to key. So we can form a heat map contain all values for each superpixcel.

```
# heat-map to show how important each super-pixel is.
def explanation_heatmap(exp, exp_class):
    '''
    Using heat-map to highlight the importance of each super-pixel for the model prediction
    '''
    dict_heatmap = dict(exp.local_exp[exp_class])
    # Define a vectorized function which takes a nested sequence of objects or numpy arrays as inputs and returns a single numpy array or a tuple of numpy arrays
    # Translate every element in numpy array according to key
    heatmap = np.vectorize(dict_heatmap.get)(exp.segments)
    plt.imshow(heatmap, cmap = 'RdBu', vmin  = -heatmap.max(), vmax = heatmap.max())
    plt.colorbar()
    plt.show()

explanation_heatmap(exp, exp.top_labels[0])
```

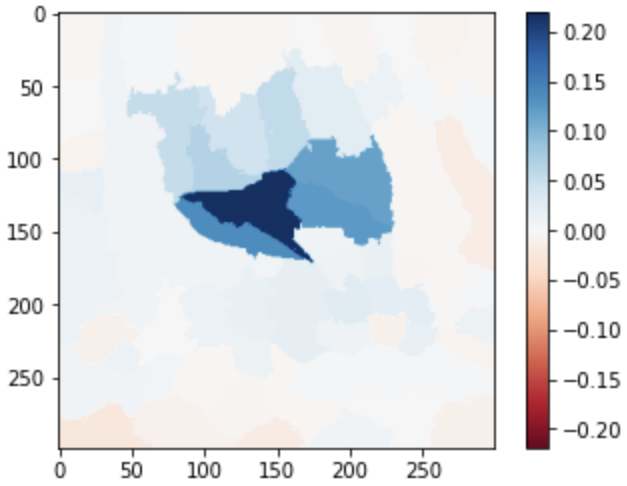


Figure 8 Heat map shows the importance of each superpixels

We can see the negative values have no impact on the prediction, while positive ones decide the area with prediction.

Conclusion

In this review, we discussed another powerful interpretable machine learning method LIME. Firstly, we demonstrated how it can explain a complex model locally and how we can use SP-LIME to pick multiple instances represent the whole data. secondly, we looked an example to interpret predictions on text data, the result showed us the words used for prediction were not meaningful, hence, the original model is not suitable for the given task. Lastly, we explored how LIME can explain image classifiers by highlighting the important features(superpixels). By knowing how the model predicted the outcome, we can make our model more robust and trustworthy.

Reference

Sharma, A 2018, *Decrypting your Machine Learning model using LIME,* Towards Data Science, viewed 30 November, https://towardsdatascience.com/decrypting-your-machine-learning-model-using-lime-5adc035109b5.

Ribeiro, M T & Singh, S & Guestrin, C, 2016, '"Why Should I Trust You?" Explaining the Predictions of Any Classifier', *in Proceedings of the 22nd ACM SIGKDD international conference on knowledge discovery and data mining*, pp. 1135-1144).

Jamesmyatt 2019, *Regression explainer with boston housing prices dataset,* Github, viewed 30 November, 2022, https://github.com/marcotcr/lime/blob/master/doc/notebooks/Submodular%20Pick%20examples.ipynb.

Stutz, D, Hermans, A and Leibe, B, 2018. 'Superpixels: An evaluation of the state-of-the-art'. *Computer Vision and Image Understanding*, *166*, pp.1-27.

Bhattacharya, A 2022, *How to Explain Image Classifiers Using LIME,* Towards Data Science, viewed 5 December, <https://towardsdatascience.com/how-to-explain-image-classifiers-using-lime-e364097335b4>.

Appendix:

A:

Code for build Random Forest model (Jamesmyatt 2019)

```
from sklearn.datasets import load_boston
import sklearn.ensemble
import numpy as np
```

```
boston = load_boston()
```

```
rf = sklearn.ensemble.RandomForestRegressor(n_estimators=1000)
```

```
train, test, labels_train, labels_test = sklearn.model_selection.train_test_split(boston.data, boston.target, train_size=0.80, random_state=42)
```

```
rf.fit(train, labels_train)
```

```
RandomForestRegressor(n_estimators=1000)
```

```
rf.score(test, labels_test)
```

```
0.8824332524624339
```

```
print('Random Forest MSError', np.mean((rf.predict(test) - labels_test) ** 2))
```

```
Random Forest MSError 8.621614370784307
```

```
print('MSError when predicting the mean', np.mean((labels_train.mean() - labels_test) ** 2))
```

```
MSError when predicting the mean 75.04543037399255
```

B:

Interpreting text data using LIME

```
# run the text explainer example notebook, up to single explanation
import sklearn
import numpy as np
import sklearn
import sklearn.ensemble
import sklearn.metrics

from sklearn.datasets import fetch_20newsgroups
categories = ['alt.atheism', 'soc.religion.christian']
newsgroups_train = fetch_20newsgroups(subset='train', categories=categories)
newsgroups_test = fetch_20newsgroups(subset='test', categories=categories)
class_names = ['atheism', 'christian']

vectorizer = sklearn.feature_extraction.text.TfidfVectorizer(lowercase=False)
train_vectors = vectorizer.fit_transform(newsgroups_train.data)
test_vectors = vectorizer.transform(newsgroups_test.data)

rf = sklearn.ensemble.RandomForestClassifier(n_estimators=500)
rf.fit(train_vectors, newsgroups_train.target)

pred = rf.predict(test_vectors)
print(sklearn.metrics.f1_score(newsgroups_test.target, pred, average='binary'))

!pip install lime
from lime import lime_text
from sklearn.pipeline import make_pipeline
c = make_pipeline(vectorizer, rf)

from lime.lime_text import LimeTextExplainer
explainer = LimeTextExplainer(class_names=class_names,verbose=True,random_state=42)

idx = 2
exp = explainer.explain_instance(newsgroups_test.data[idx], c.predict_proba, num_features=6)
print('Document id: %d' % idx)
print('Probability(christian) =', c.predict_proba([newsgroups_test.data[idx]])[0,1])
print('True class: %s' % class_names[newsgroups_test.target[idx]])
```