XAI 설명 가능한 인공지능 5장 리뷰

작성일시	@2022년 3월 2일 오전 10:41
최종 편집일시	@2022년 3월 5일 오전 6:32
회의 유형	XAI책
작성자	
참가자	
속성	

5.1 대리 분석 개론

대리 분석 기법(Surrogate Analysis)이란 엔지니어링에 사용되던 용어였다. 대리 분석이라는 말에서 알 수 있듯이, 본래의 기능을 흉내내는 간단한 대체재를 만들어서 프로토타입이 동작하는지 판단하는 분석 방법이다.

즉 인공지능 모델이 복잡해 분석이 불가능한 경우, 유사한 기능을 가진 인공지능 모델 여러 개를 대리로 만들어 본래의 모델을 분석하는 기법이다.

(= 근사치 모델, 반응 표면 기법, 에뮬레이터)

분석해야하는 모델을 f라고 하고, 대리 분석 모델은 g라고 할 때, 모델 g를 결정하는 조건은 3가지다.

1. 모델 f 보다 학습하기 쉬워야 한다.
2. 설명 가능해야한다.
3. 모델 f 를 유사하게 흉내낼 수 있으면 된다.

모델 g를 학습하는데 2가지 방식이 존재

1. 글로벌 대리 분석 - 학습 데이터 일부 또는 전체를 사용해서 학습
2. 로컬 대리 분석 - 학습 데이터 하나를 해석하는 과정

5.1.1 글로벌 대리 분석 (Global Surrogate)

글로벌 대리분석 (global surrogate)이란 전체 학습 데이터를 사용해 블랙박스 함수 f 를 따라하는 유사 함수 g를 만들고 g를 해석 가능하도록 변조하는 방법

- 잘 알려진 XAI 알고리즘과 각각의 특성

알고리즘	선형성 (Linearity)	단조함수 유무 (Monotone)	PDP Interaction	목표
선형 회귀	있음	단조함수	불가능	회귀
로지스틱 회귀	없음	단조함수	불가능	분류
의사 결정 트리	없음	일부	가능	회귀,분류
나이브 베이즈	없음	단조함수	불가능	분류
K-최근접 이웃	없음	단조함수 아님	불가능	분류,회귀

글로벌 대리 분석 수행 과정

1. 데이터 집합 X 선택
2. 선택한 데이터 집합 X에 대해 f 의 예측 결과를 구함
3. XAI 모델을 구한다
4. 데이터 집합 X로 모델 g를 학습
5. 데이터 X에 대한 f의 결과와 g의 결과를 비교해, 두 모델이 최대한 유사한 결과를 내도록 튜닝
6. 설명 가능한 모델 g를 XAI 기법을 사용해 해석

Measure Function R-스퀘어(Squared) 방식

- $R^2 = 1 - SSE/SST = 1 - \sum_{i=1}^{n}(\hat{y_*^{(i)}} - \hat{y^{(i)}})^2 / \sum_{i=1}^{n}(\hat{y^{(i)}} - \hat{y})^2$

R-스퀘어 값이 1에 가까워질수록 모델 g와 모델 f가 유사하다는 의미

R-스퀘어 값이 0에 가까워질수록 모델 g가 모델 f를 설명하고있다고 보기 어렵다.

글로벌 대리 분석의 장단점

- 장점

1. 유연하다 - 다양한 XAI 기법 적용 가능 및 블랙박스 모델 f를 이해하지 않아도 measure function으로 모델 f가 어떻게 학습했는지 확인 가능
2. 직관적이다 - 대리 구현에 사용되는 머신러닝의 구현이 쉽고 설명이 간단하기 때문

- 단점 (주의할 점)

1. 모델 g는 모델 f를 유사하게 설명하는 모델이어서 사실 상 해석 방향에 결함이 있을 수 있다.
2. Measure 함수의 설명 가능성 판단 기준이 주관적이다.(ex) R-스퀘어 결정지수
3. 데이터 X가 전체일 수도, 일부일 수도 있기에 데이터X가 전체 데이터와 비교했을 때 편향되었을 위험이 존재.

5.1.2 로컬 대리 분석 (Local Surrogate)

로컬 대리 분석은 데이터 하나에 대해 블랙박스가 해석하는 과정을 분석하는 기법

- LIME (Local Interpretable Model-agnostic Explanations) - 학습 기법과 관계없이 모델을 설명할 수 있는 로컬 설명 가능 모델

5.2 LIME

LIME이란 모델이 현재 데이터의 어느 영역을 집중해서 분석했고, 어떤 영역을 분류 근거로 사용했는지 알려주는 XAI 기법.

5.2.2 배경 이론

어떤 이미지가 주어졌을 때, 이미지 내의 특정 관심 영역을 x라고 하고, 초점 주변으로 관심 있는 영역을 키워나갈 때, 기준 x로부터 동일한 정보를 가지고 있다고 간주할 수 있는 영역을 π_x라고 하자. 이때 π_x를 슈퍼 픽셀이라고 한다.

이미지 전체를 입력으로 받아, 특정 사람일 확률을 반환하는 블랙박스 모델을 f, 해당 이미지 자체가 아닌 슈퍼 픽셀 π_x의 마스킹 정보 m_x를 입력으로 받아 f(π_x)와 동일한 값을 반환하도록 학습한 해석 가능한 모델을 g라고 하자

- $explanation(x) = argmin_{g \in G} L(f, g, \pi_x)$

LIME 논문에서는 단순한 선형결합 모델 g(m1,m2,...) = $w_1 x_1 + w_2 x_2 + ...$ 을 사용해 마스크 m_x로 대표되는 각 슈퍼 픽셀 π_x의 영향의 정도를 w_x로 파악할 수 있다.

L은 손실함수로, 슈퍼 픽셀 π_x에 대해 분류 모델 f 의 예측 결과와 마스킹 데이터 m_x에 대한 회귀모델 g의 검증 결과를 비교해 유사성 계산.

LIME의 decision boundary 찾는 방법

배경이론에서의 계산 방식과 같다고 본다.

LIME은 입력 이미지 x 근처를 조사하며 모델 f의 예측 결과를 구한다.

→ 그림 5.12를 보면, LIME이 이미지 근방을 샘플링해서, 원래 이미지를 입력으로 받은 f의 출력과 슈퍼 픽셀들로 일부만 선택한 이미지를 입력으로 받은 f의 출력이 비슷한 결과를 내는지 확인해서 입력 이미지와 근처 이미지가 얼마나 유사한지 구하고 이를 통해 g를 구할 수 있다. g는 이미지 내에 어떤 부분이 예측에 긍정적인 영향을 주는 부분인지, 부정적인 영향을 주는 부분인지 찾아내는 것이기에 위 내용이 g를 구하는데 중요하게 작용

LIME은 한 이미지와 그 근처 이미지들에 대해 탐색하므로, 해석 가능한 함수인 선형 함수만을 가지고도 결정 경계를 표현 가능

LIME의 장점

1. 머신러닝 알고리즘과 관계없이 XAI를 적용할 수 있다.
2. 매트릭스로 표현 가능한 데이터인 텍스트나 이미지에 대해 작동하는 XAI 기법
3. 다른 XAI 기법과 비교했을때 매우 가볍다.

LIME의 단점

1. 슈퍼 픽셀 알고리즘에 따라 마스킹 데이터가 달라진다.

실습 2. 텍스트 데이터에 LIME 적용하기

```
# 예제 5.1 sklearn 패키지의 20 news groups 데이터 세트를 가져오는 코드

from sklearn.datasets import fetch_20newsgroups

newsgroups_train = fetch_20newsgroups(subset='train')
newsgroups_test = fetch_20newsgroups(subset='test')

# making class names shorter
class_names = [x.split('.')[-1] if 'misc' not in x else '.'.join(x.split('.')[-2:]) for x in newsgroups_train.target_names]

print(class_names)

class_names[3] = 'pc.hardware'
class_names[4] = 'mac.hardware'
print(class_names)
```

```
# 예제 5.2 제보 기사의 카테고리를 분류하는 모델을 만들고 F1-점수를 측정하는 코드

import sklearn
import sklearn.metrics
from sklearn.naive_bayes import MultinomialNB

# TF-IDF를 사용해서 문서를 숫자 벡터로 변환하는 전처리 과정
vectorizer = sklearn.feature_extraction.text.TfidfVectorizer(lowercase=False)
train_vectors = vectorizer.fit_transform(newsgroups_train.data)
test_vectors = vectorizer.transform(newsgroups_test.data)

# learning
nb = MultinomialNB(alpha=.01)
nb.fit(train_vectors, newsgroups_train.target)

# testing
pred = nb.predict(test_vectors)
sklearn.metrics.f1_score(newsgroups_test.target, pred, average='weighted')
```

```
# 예제 5.3 파이프라인 기술을 사용해 테스트 데이터 인덱스 0번에 데이터 벡터라이저와 카테고리 분류를 한꺼번에 수행하는 과정

from sklearn.pipeline import make_pipeline
pipe = make_pipeline(vectorizer, nb)
predict_classes = pipe.predict_proba([newsgroups_test.data[0]]).round(3)[0]

print(predict_classes)
```

```
# 예제 5.4 데이터 분류 결과의 가독성을 높이기 위해 출력을 수정하는 코드

rank = sorted(range(len(predict_classes)), key=lambda i: predict_classes[i], reverse=True)
for rank_index in rank:
    print('[{:>5}] \t{:<3}\tclass ({:.1%})'.format(rank.index(rank_index) + 1, rank_index, predict_classes[rank_index]))
```

```
# 예제 5.5 LIME 텍스트 설명체를 선언하는 코드

! pip install lime

from lime.lime_text import LimeTextExplainer
explainer = LimeTextExplainer(class_names=class_names)
```

```
# 예제 5.6 LIME의 explain_instance 메서드에 필요한 최소한의 파라미터를 넣었다

exp = explainer.explain_instance(newsgroups_test.data[0], pipe.predict_proba, top_labels=1)

# 예제 5.7 LIME이 잘 작동하는지 확인하기 위한 메서드

exp.available_labels()

# 예제 5.8 explainer가 0번 테스트 데이터를 해석한 결과를 iPython 노트북으로 출력하는 코드

exp.show_in_notebook(text=newsgroups_test.data[0])
```

```
# 예제 5.9 테스트 데이터 5번을 LIME 알고리즘에 입력하는 코드

from lime.lime_text import LimeTextExplainer

idx = 5

explainer = LimeTextExplainer(class_names=class_names)
exp = explainer.explain_instance(newsgroups_test.data[idx], pipe.predict_proba, top_labels=1)
predict_classes = pipe.predict_proba([newsgroups_test.data[idx]]).round(3)[0]
rank = sorted(range(len(predict_classes)), key=lambda i: predict_classes[i], reverse=True)

print('Document id: %d' % idx)
print('Predicted class: %s' % class_names[nb.predict(test_vectors[idx]).reshape(1,-1)[0,0]])
print('True class: %s' % class_names[newsgroups_test.target[idx]])
print(predict_classes)

print(rank)
print ('Explanation for class %s' % class_names[rank[0]])
print ('\n'.join(map(str, exp.as_list(rank[0]))))

exp.show_in_notebook(text=newsgroups_test.data[idx])
```

```
# 예제 5.10 scikit-image 패키지를 사용해서 올리베티 얼굴 데이터를 로드하고 확인하는 코드

import numpy as np
import matplotlib.pyplot as plt
from skimage.color import gray2rgb, rgb2gray

from skimage.util import montage

from sklearn.datasets import fetch_olivetti_faces
faces = fetch_olivetti_faces()

# make each image color so lime_image works correctly

X_vec = np.stack([gray2rgb(iimg) for iimg in faces.data.reshape((-1, 64, 64))],0)
y_vec = faces.target.astype(np.uint8)

%matplotlib inline

fig, ax1 = plt.subplots(1,1, figsize = (8,8))
ax1.imshow(montage(X_vec[:,:,:,0]), cmap='gray', interpolation = 'none')
ax1.set_title('All Faces')
ax1.axis('off')
```

```
# 예제 5.11 이미지 데이터 한 장을 그리는 코드

index = 93
plt.imshow(X_vec[index], cmap='gray')
plt.title('{} index face'.format(index))
plt.axis('off')
```

```
# 예제 5.12 텐서플로를 이용한 분류 모델을 LIME에서 사용할 수 있게 컨벤션을 맞추는 코드

def predict_proba(image):
    return session.run(model_predict, feed_dict={preprocessed_image: image})
```

```
# 예제 5.13 sklearn 패키지에 있는 train_test_split 함수를 사용해서 X_vec과 y_vec으로부터 학습용과 테스트용 데이터세트를 분리하는 코드
```

```
from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(X_vec, y_vec, train_size=0.70)
```

```
# 예제 5.14 MLP가 학습할 수 있도록 이미지 전처리를 수행하는 파이프라인 생성

from sklearn.pipeline import Pipeline
from sklearn.neural_network import MLPClassifier
class PipeStep(object):
    """ Wrapper for turning functions into pipeline transforms (no-fitting) """
    def __init__(self, step_func):
        self._step_func=step_func
    def fit(self,*args):
        return self
    def transform(self,X):
        return self._step_func(X)

makegray_step = PipeStep(lambda img_list: [rgb2gray(img) for img in img_list])
flatten_step = PipeStep(lambda img_list: [img.ravel() for img in img_list])
simple_pipeline = Pipeline([
    ('Make Gray', makegray_step),
    ('Flatten Image', flatten_step),
    ('MLP', MLPClassifier(
        activation='relu',
        hidden_layer_sizes=(400, 40),
    random_state=1))
])
```

```
# 예제 5.15 학습 데이터를 MLP가 있는 파이프라인에 붓는 코드


simple_pipeline.fit(X_train, y_train)
```

```
# 예제 5.16 classification_report를 사용해서 모델 성능을 테스트하는 코드

def test_model(X_test, y_test):
  pipe_pred_test = simple_pipeline.predict(X_test)
  pipe_pred_prop = simple_pipeline.predict_proba(X_test)

  from sklearn.metrics import classification_report
  print(classification_report(y_true=y_test, y_pred = pipe_pred_test))

test_model(X_test, y_test)
```

```
# 예제 5.17 Normalizer 전처리 과정을 추가해서 MLP를 학습시키는 코드

from sklearn.pipeline import Pipeline
from sklearn.preprocessing import Normalizer
from sklearn.neural_network import MLPClassifier
class PipeStep(object):
    """ Wrapper for turning functions into pipeline transforms (no-fitting) """
    def __init__(self, step_func):
        self._step_func=step_func
    def fit(self,*args):
        return self
    def transform(self,X):
        return self._step_func(X)


makegray_step = PipeStep(lambda img_list: [rgb2gray(img) for img in img_list])
flatten_step = PipeStep(lambda img_list: [img.ravel() for img in img_list])
```

```
simple_pipeline = Pipeline([
    ('Make Gray', makegray_step),
    ('Flatten Image', flatten_step),
    ('Normalize', Normalizer()),
    # add Normalizer preprocessing step
    ('MLP', MLPClassifier(
        activation='relu',
        hidden_layer_sizes=(400, 40),
        random_state=1)),
])

simple_pipeline.fit(X_train, y_train)

test_model(X_test, y_test)
```

```
# 예제 5.18 필자가 찾은 최적의 파이프라인 조합

simple_pipeline = Pipeline([
    ('Make Gray', makegray_step),
    ('Flatten Image', flatten_step),
    ('Normalize', Normalizer()),
    # add Normalizer preprocessing step
    ('MLP', MLPClassifier(
        activation='relu',
        alpha=1e-7,
        epsilon=1e-6,
        hidden_layer_sizes=(800, 120),
        random_state=1)),
])

simple_pipeline.fit(X_train, y_train)

test_model(X_test, y_test)
```

```
# 예제 5.19 LIME의 이미지 설명체와 이미지 분할 알고리즘을 선언하는 코드

from lime import lime_image
from lime.wrappers.scikit_image import SegmentationAlgorithm

explainer = lime_image.LimeImageExplainer()

# segmenter algorithms: quickshift(default), slic, felzenszwalb
segmenter = SegmentationAlgorithm(
        'slic',
        n_segments=100,
        compactness=1,
        sigma=1)
```

```
# 예제 5.20 테스트 0번 이미지에 대해 설명 모델을 구축하는 코드

%%time

olivetti_test_index = 0

exp = explainer.explain_instance(
    X_test[olivetti_test_index],
    classifier_fn = simple_pipeline.predict_proba,
    top_labels=6,
    num_samples=1000,
    segmentation_fn=segmenter)
```

```
# 예제 5.21 올리베티 데이터 0번을 설명체에 통과시켜 XAI를 수행하는 코드

from skimage.color import label2rgb
# set canvas
fig, ((ax1, ax2), (ax3, ax4)) = plt.subplots(2,2, figsize = (8, 8))

# show positive segments
temp, mask = exp.get_image_and_mask(
    y_test[olivetti_test_index],
    positive_only=True,
    num_features=8,
    hide_rest=False)

ax1.imshow(
        label2rgb(mask, temp, bg_label = 0),
        interpolation = 'nearest')

ax1.set_title('Positive Regions for {}'.format(y_test[olivetti_test_index]))

# show all segments
temp, mask = exp.get_image_and_mask(
    y_test[olivetti_test_index],
    positive_only=False,
    num_features=8,
    hide_rest=False)

ax2.imshow(
    label2rgb(4 - mask, temp, bg_label = 0),
    interpolation = 'nearest')

ax2.set_title('Positive/Negative Regions for {}'.format(y_test[olivetti_test_index]))

# show image only
ax3.imshow(temp, interpolation = 'nearest')
ax3.set_title('Show output image only')

# show mask only
ax4.imshow(mask, interpolation = 'nearest')
ax4.set_title('Show mask only')
```

```
# 예제 5.22 올리베티 얼굴 테스트 데이터 0번(3번 인물)으로부터 추가 설명을 출력하는 코드

def show_extra_info(test, test_index):

    pipe_pred_test = simple_pipeline.predict(test)
    pipe_pred_prop = simple_pipeline.predict_proba(test)
    # now show them for each class
    fig, m_axs = plt.subplots(2,6, figsize = (12,4))
    for i, (c_ax, gt_ax) in zip(exp.top_labels, m_axs.T):
        temp, mask = exp.get_image_and_mask(
                i,
                positive_only=True,
                num_features=12,
                hide_rest=False,
                min_weight=0.001)

        c_ax.imshow(
                label2rgb(mask,temp, bg_label = 0),
                interpolation = 'nearest')
        c_ax.set_title('Positive for {}\nScore:{:2.2f}%'.format(i, 100*pipe_pred_prop[test_index, i]))

        c_ax.axis('off')

        face_id = np.random.choice(np.where(y_train==i)[0])

        gt_ax.imshow(X_train[face_id])
        gt_ax.set_title('Example of {}'.format(i))
        gt_ax.axis('off')

show_extra_info(X_test, 1)
```

```
# 그림 5.36용

show_extra_info(X_test, 25)

# 그림 5.37용

show_extra_info(X_test, 101)
```

5.3 SHAP (SHapley Additive exPlanations)

5.3.1 배경 이론

SHAP는 섀플리 값 (Shapley value)과 피처 간 독립성을 핵심 아이디어로 사용하는 XAI 기법

- 섀플리 값은 전체 성과를 창출하는 데 각 참여자가 얼마나 공헌했는지를 수치로 표현
- 각 사람 or 피처의 기여도는 그 사람의 기여도를 제외했을 때, 전체 성과의 변화 정도

수식

- $\phi_i(v) = \sum_{S \in Ni}(|S|!(n - |S| - 1)!/n!)(v(S \cup i) - v(S))$

-수식이 정확하게 적히지 않아 책 148페이지 참조하길 바람

- ϕ_i : i 데이터에 대한 shapley value
- n : 참여자
- S : 총 그룹에서 i번째 인물을 제외한 모든 집합
- $v(S)$: i번째 인물을 제외하고 나머지 부분 집합이 결과에 공헌한 기여도
- $v(S \cup (i))$: i번째 인물을 포함한 전체 기여도

그림 5.40을 보면 오른쪽 화살표로 가는 것은 원점으로부터 $f(x)$가 높은 예측 결과를 내도록 도움을 주는 요소이며, 왼쪽 화살표는 그 반대이다.

Shapley value가 음수이면 이는 특정 피처가 예측에 부정적인 영향을 주는 것이라고 해석할 수 있다.

피처에 대한 XAI AIgorithm

1. 피처 중요도 (Feature Importance)

- 예측에 가장 큰 영향을 주는 변수를 Permutation 하면서 찾는 기법

(Permutation이란 피처를 임의의 값으로 바꾸고, 그 변화가 예측에 주는 영향력을 계측하는것)

- 그러나 피처 간의 의존성을 간과하기에 피처 간 상관관계가 존재하는 모델은 피처 중요도의 사용을 지양해야한다.

2. 부분 의존성 플롯(PDP)

- 관심 피처를 조정한 값을 모델에 투입해 예측값을 구하고 평균을 낸다.
- 그러나 3차원까지의 관계만 표시할 수 있을 뿐, 만약 4번째 feature가 예측에 큰 영향을 미친다면 이는 결과가 왜곡될 수 있다. → Feature의 영향력이 과대평가 될 수 있다.

3. SHAP

- 피처의 의존성까지 고려하며 모델 영향력 계산
- 그러나 결측(missing)을 시뮬레이션 해야하므로 계산시간이 오래걸리고 피처의 추가,삭제가 빠른 모델을 설명하기에 적합하지 않다.

  → 이는 outlier의 등장에 취약하다.

실습 4의 내용을 읽으면 이해하기 쉽다.

실습 5. 보스턴 주택 가격 결정 요소 구하기

```
# 예제 5.23 SHAP 모듈로부터 보스턴 데이터세트를 불러와서 학습용과 테스트용 데이터세트로 분리하는 코드
!pip install shap

import shap
from sklearn.model_selection import train_test_split

X, y = shap.datasets.boston()

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=1)

X_train[:10]
```

```
# 예제 5.23 방의 개수와 집값 간의 관계를 산점도로 그리는 코드

# drawing scatter plot
import matplotlib.pylab as plt
import matplotlib

%matplotlib inline

matplotlib.style.use('ggplot')

fig, ax1 = plt.subplots(1,1, figsize = (12,6))

ax1.scatter(X['RM'], y, color='black', alpha=0.6)
ax1.set_title('Relation # of Rooms with MEDV')
ax1.set_xlim(2.5, 9)
ax1.set_xlabel('RM')
ax1.set_ylim(0, 55)
ax1.set_ylabel('MEDV \n Price $1,000')
```

```
# 예제 5.38 선형 모델을 이용해서 방 개수와 주택 가격 간의 관계를 구하는 코드

from sklearn import linear_model
import pandas as pd

linear_regression = linear_model.LinearRegression()
linear_regression.fit(X=pd.DataFrame(X_train['RM']), y=y_train)
prediction = linear_regression.predict(X=pd.DataFrame(X_test['RM']))

print('a value: ', linear_regression.intercept_)
print('b value: ', linear_regression.coef_)
print('MEDV = {:.2f} * RM {:.2f}'.format(linear_regression.coef_[0], linear_regression.intercept_))
```

```
# 예제 5.39 방의 개수가 달라질 때 주택 매매 가격을 예측하는 그래프와 데이터를 한꺼번에 플롯으로 그리는 코드

# scatter Train, Test data with Linear Regression Prediction
fig, ax1 = plt.subplots(1,1, figsize = (12,6))
ax1.scatter(X_train['RM'], y_train, color='black', alpha=0.4, label='data')
ax1.scatter(X_test['RM'], y_test, color='#993299', alpha=0.6, label='data')
ax1.set_title('Relation # of Rooms with MEDV')
ax1.set_xlim(2.5, 9)
ax1.set_xlabel('RM')
ax1.set_ylim(0, 55)
ax1.set_ylabel('MEDV \n Price $1,000')
ax1.plot(X_test['RM'], prediction, color='purple', alpha=1, linestyle='--', label='linear regression line')
ax1.legend()
```

```
# 예제 5.40 모델 예측치와 실제 집값 간의 RMSE를 구하는 코드

from sklearn.metrics import mean_squared_error
rmse = np.sqrt(mean_squared_error(y_test, prediction))
print("RMSE: %f" % (rmse))
```

```
# 예제 5.41 xgboost의 선형 회귀 모델로 주택 매매 가격을 예측하는 모델을 만들고 학습하는 코드

import xgboost

# train XGBoost model
model = xgboost.XGBRegressor(objective ='reg:linear')
model.fit(X_train, y_train)
preds = model.predict(X_test)
```

```
# 예제 5.42 전체 피처를 사용해서 학습시킨 모델의 RMSE를 구하는 코드

from sklearn.metrics import mean_squared_error
rmse = np.sqrt(mean_squared_error(y_test, preds))
print("RMSE: %f" % (rmse))
```

```
# 예제 5.43 SHAP의 설명체를 정의하고 섀플리 값을 계산하는 로직

# load JS visualization code to notebook
shap.initjs()

# explain the model's predictions using SHAP values
# (same syntax works for LightGBM, CatBoost, and scikit-learn models)
explainer = shap.TreeExplainer(model)
shap_values = explainer.shap_values(X_train)

# visualize the first prediction's explanation
# (use matplotlib=True to avoid Javascript)
shap.force_plot(explainer.expected_value, shap_values[0,:], X_train.iloc[0,:])
```

```
# 예제 5.44 259번 데이터에 대해서 방의 개수(RM)와 집 가격(MEDV)이 어떤 관계가 있는지 플롯으로 그리는 코드

fig, ax1 = plt.subplots(1,1, figsize = (12,6))

idx = 259
ax1.scatter(X['RM'], y, color='black', alpha=0.6)
```

```
ax1.scatter(X_train['RM'].iloc[idx], y_train[idx], c='red', s=150)
ax1.set_title('Relation # of Rooms with MEDV')
ax1.set_xlim(2.5, 9)
ax1.set_xlabel('RM')
ax1.set_ylim(0, 55)
ax1.set_ylabel('MEDV \n Price $1,000')
```

```
# 예제 5.45 데이터 259번에 대한 섀플리 영향도를 그리는 코드

# load JS visualization code to notebook
shap.initjs()

shap.force_plot(explainer.expected_value, shap_values[259,:], X_train.iloc[259,:])
```

```
# 예제 5.46 전체 데이터에 대한 섀플리 값을 플롯으로 그리는 코드

# load JS visualization code to notebook
shap.initjs()

# visualize the training set predictions
shap.force_plot(explainer.expected_value, shap_values, X_train)
```

```
# 예제 5.47 방 개수 피처가 집값에 미치는 섀플리 영향도를 시각화하는 플롯

# create a SHAP dependence plot to show the effect
# of a single feature across the whole dataset
shap.dependence_plot("RM", shap_values, X_train)
```

```
# 예제 5.48 전체 피처들이 섀플리 값 결정에 어떻게 관여하는지 시각화하는 코드

# summarize the effects of all the features

shap.summary_plot(shap_values, X_train)
```

```
# 예제 5.49 피처별 섀플리 값을 막대 타입으로 비교하는 코드

shap.summary_plot(shap_values, X_train, plot_type="bar")
```

```
# 예제 5.50 xgboost의 피처 중요도를 호출하는 코드

xgboost.plot_importance(model)
```