XAI 설명 가능한 인공지능 4장 리뷰

작성일시	@2022년 3월 2일 오전 9:41
최종 편집일시	@2022년 3월 3일 오후 10:34
회의 유형	XAI책
작성자	
참가자	
속성	

의사 결정 트리 (Decision Tree)란 질문을 던지고 답을 하는 과정을 연쇄적으로 반복해 집단을 분류(Clalssification) 하는 분석 방법

- 연구자가 분석 과정을 쉽게 이해하고 설명할 수 있다는 장점

의사 결정 트리는 정보 이득 수치를 계산해서 최적 목표를 달성하는 트리를 완성한다.

이 때, 정보 이득은 엔트로피 변화량으로 계산된다.

- $Entropy(A) = -\sum_{k=1}^{10} p_k log_2(p_k)$

{n : number of category , p_k = A 영역에 속하는 레코드 가운데 k category에 속하는 레코드의 비율}

4.1 의사 결정 트리 시각화

의사 결정 트리는 질문 수를 가능한 한 줄이면서도 정확도를 높이기 위해서는 정보 이득량이 큰 방식을 상위 노드(부모 노드)에 배치해야 한다.

의사 결정 트리를 시각화할 수 있다면 사용자가 해당 알고리즘의 계산 방식 , 원리를 이해하지 않아도 도식대로 데이터를 분류하기만 하면 컴퓨터의 의사 결정 방식과 같은 결과 도출 가능

4.2 피처 중요도 구하기

피처 중요도(Feature Importance, 또는 퍼뮤테이션 중요도 [Permutation Importance])는 데이터의 피처가 알고리즘의 정확한 분류에 얼마나 큰 영향을 미치는지 분석하는 기법

- 측정하는 방식으로는 특정 피처의 값을 임의의 값으로 치환했을 때, 원래 데이터보다 예측 에러가 얼마나 커지는가를 측정
  (≃ Ablation study)

피셔, 루딘, 도미니치의 피처 중요도 알고리즘

1. 주어진 모델의 에러를 측정
   - $e^{original} = L(y, f)$
2. X의 피처 k개 (k = 1, ... , p)에 대하여
   a. 피처 매트릭스 $X^{permutation}$ 을 만든다
      - $X^{permutation}$이란, 피처 k를 매트릭스 X에서 임의의 값으로 변경한 모델
   b. $X^{permutation}$으로 모델 에러를 측정한다
      - $e^{permutation} = L(Y, f(X^{permutation}))$
   c. 퍼뮤테이션 피처 중요도를 산정한다.
      - $FI^k = e^{permutation} / e^{original}$ or $FI^k = e^{permutation} - e^{original}$
3. 피처 중요도 FI 를 구한다.

4.3 부분 의존성 플롯 (PDP) 그리기

피처 중요도 - 피처 각각을 변형하는 방식으로 머신러닝 결과를 해석

부분 의존성 플롯 (Partial Dependence Plots) - 피처의 수치를 선형적으로 변형하면서 알고리즘 해석 능력이 얼마나 증가하고 감소하는지 관찰하는 방식

이후에 이 내용에 대해 다시 다룰 예정. 그림 4.8은 혈압 수치의 변화량에 따른 피마 인디언의 당뇨병 진단 모델의 영향도 그래프이다.
당뇨병 진단은 정상 혈압 범위에서 음의 값을 보이다가 고혈압 범위에서 양의 방향으로 상승한다.

- (그림에서는 0~120mmHg의 혈압만을 보여주고 있다.)

부분 의존성 함수 (Partial Dependence Funciton)

- $f_{xs}\hat{(}x_s) = E_{xs}[f_{xs}(\hat{x_s}, x_c)] = \int f_{xs}(\hat{x_s}, x_c)dP(x_c)$

(x_s : 부분 의존성 함수가 시각화해야 할 피처, x_c : 머신러닝 모델 ($\hat{f}$) 이 사용하는 피처들
$\{S \cap C = \emptyset, S \cup C = X\}$, S : 부분 의존성 정도를 알고 싶은 피처들의 집합)

즉, 부분 의존성 함수는 x_c 와 x_s의 조합이 다양할 때, 머신러닝 모델의 결과가 어떤 분포를 보이는지 그 차이를 계산한다.

집합 S에 대한 PDP를 그리기 위해서는 x_s 피처를 고정하고 x_c 에 대한 모든 값을 모델에 반영해 평균을 취하는 방식을 사용한다.

부분 의존성 함수 $f_{x_s}\hat{(}x_s)$는 몬테카를로 방식을 사용해 근사할 수 있다.

- $f_{x_s}\hat{(}x_s) = 1/n \sum_{k=1}^{10} f(\hat{x_s}, x_c^i)$

(우항에 있는 변수 x_s는 관심 있는 피처, x_c^i 는 관심 없는 피처들의 집합, n은 데어터 셋의 개수이다.)

전체 조합에 대해 x_s가 모델의 판단에 미치는 영향력을 평균함으로써 계산을 한다. 여기서 피처 c는 피처 s와 상관관계가 적다고 가정한다. 이 가정이 위반된다면 PDP는 피처 c에 의존적이게 되기 때문.

4.4 XGBoost 활용하기

4.4.1 XGBoost 의 장점

부스팅 알고리즘은 약한 분류기 (Classifier)를 여러 개 쌓아서 복잡한 분류기를 만드는 알고리즘

장점

1. XGBoost는 병렬 처리를 사용하기에 학습과 분류가 빠르다.
2. 유연성이 좋다. 평가 함수(Evaluation function) 를 포함해 다양한 커스텀 최적화 옵션을 제공한다.
3. Greedy algorithm을 사용해 자동으로 포레스트(Forest)의 가지를 친다. → 과적합이 잘 발생하지 않는다.
4. 다른 알고리즘과 연계 활용성이 좋다. XGBoost 분류기 결론부 아래를 다른 알고리즘과 연결해 앙상블 학습이 가능

4.4.3 기본 원리

XGBoost는 기본적으로 부스팅(Boosting or Additive Training) 이라 불리는 기술을 사용하며, 부스팅은 약한 분류기를 세트로 묶어서 정확도를 예측하는 기법이다.

예를 들어 어떤 학습기 M에 대해 Y를 예측할 확률은

$$Y = M(X) + error_1$$

$error1$에 대해 더 상세히 분류할 수 있는 모델 G가 있다면 ($error_1 > error_2$),

$$error_1 = G(X) + error_2$$

ㅇ이를 더 세밀하게 분리하면 ($error_2 > error_3$) ,

$$error_2 = H(X) + error_3$$

→ 따라서 $Y = M(X) + G(X) + H(X) + error_3$ 로 표현할 수 있다.

여기서 개선을 하자면, 각 분류기 M,G,H 가 성능이 다르기에 다른 비중으로 분류를 진행해야 한다.

이를 적용하면 다음과 같으며 최적의 비중을 찾는다면 더 좋은 성능을 내는 분류기가 될 것이다.

- $Y = \alpha M(X) + \beta G(X) + \gamma H(X) + error_4$

greedy algorithm을 통해 M,G,H를 찾아내고, weight parameter는 분산 처리를 사용해 각 분류기의 최적의 반영 가중치 α, β, γ를 찾아낸다.

이후의 방식은 적기에는 복잡하여 생략한다.

4.4.4 파라미터

XGBoost의 파라미터는 크게 3가지가 있다.

1. 일반 파라미터 (General Parameters)
- 도구의 모양 결정 관여
2. 부스터 파라미터 (Booster Parameters)
- 트리마다 가지를 칠 때, 적용하는 옵션을 정의
3. 학습 과정 파라미터 (Learning Task Parameters)
- 최적화 성능 결정 관여

1. 일반 파라미터
- booster : 어떤 부스터 구조를 쓸지 결정한다. (gbtree,gblinear,dart)
- nthread : 몇 개의 스레드를 동시에 처리할 지 결정
- num_feature : 피처의 차원 숫자
2. 부스팅 파라미터
- eta : 학습률
- gamma : 정보 이득 (값이 커지면 트리 깊이가 줄어든다.)
- max_depth : 한 트리의 최대 깊이 (값이 커지면 모델의 복잡도가 커진다.)
- lambda (L2 reg-form) : L2 regularization form에 부여되는 비중 값
- alpha (L1 reg-form) : L1 regularization form에 부여되는 비중 값
3. 학습 과정 파라미터
- objective : 목적함수 (reg :linear , binary:logistic, count:poisson)
- eval_metric : 모델의 성능을 측정하는 방식을 결정
(rmse, logloss(log-likelihood) , map(mean average precision))
4. 커맨드 라인 파라미터
- num_rounds : boosting 시행 횟수를 결정한다.

4.5.1 학습하기

```
from numpy import loadtxt
from sklearn.metrics import accuracy_score
from sklearn.model_selection import train_test_split
from xgboost import XGBClassifier
import pandas as pd

# 예제 4.1 당뇨병 진단 모델을 XGBoost 알고리즘을 사용해서 코드로 구현

# load data
dataset = loadtxt('https://gist.githubusercontent.com/ktisha/c21e73a1bd1700294ef790c56c8aec1f/raw/819b69b5736821ccee93d05b51de0510bea0
# split data into X and Y
X = dataset[:, 0:8]
y = dataset[:, 8]
x_train, x_test, y_train, y_test = train_test_split(X, y, test_size=0.20, random_state=7)
x_train = pd.DataFrame(x_train)

x_train.columns = ['Pregnancies',
    'Glucose',
    'BloodPressure',
    'SkinThickness',
    'Insulin',
    'BMI',
    'DiabetesPedigreeFunction',
    'Age']
y_train = pd.DataFrame(y_train)
x_test = pd.DataFrame(x_test)
x_test.columns = ['Pregnancies',
    'Glucose',
    'BloodPressure',
    'SkinThickness',
    'Insulin',
    'BMI',
    'DiabetesPedigreeFunction',
    'Age']
y_test = pd.DataFrame(y_test)
# fit model into training-data
model = XGBClassifier()
model.fit(x_train, y_train)
# make predictions
y_pred = model.predict(x_test)
predictions = [round(value) for value in y_pred]
# evaluations
accuracy = accuracy_score(y_test, predictions)
print('Accuracy: %.2f%%' % (accuracy * 100.0))
```

```
# _# 예제 4.2 학습된 모델로 특정 환자에 대해 당뇨병을 진단하는 코드
# import numpy as np
# patient = {
#     'Pregnancies': [1],
#     'Glucose': [161],
#     'BloodPressure': [72],
#     'SkinThickness': [35],
#     'Insulin': [0],
#     'BMI': [28.1],
#     'DiabetesPedigreeFunction': [0.527],
#     'Age': [20]
# }

# value = [1, 161, 72, 35, 0, 28.1, 0.527, 20]
# print(model.get_booster().feature_names)
# l = model.predict_proba(value)
# print('No diabetes: {:.2%}\n Yes diabetes: {:.2%}'.format(l[0][0], l[0][1]))
#l = model.predict_proba(value)
#print('No diabetes : {:.2%}\n Yes diabetes : {:.2%}'.format(l[0][0],l[0][1])
```

```
# 4.3  Graphviz 패키지를 이용해 모델의 의사 결정 나무를 시각화하는 코드

%matplotlib inline

import os
# set graphviz path
from xgboost import plot_tree
import matplotlib.pyplot as plt
from matplotlib.pylab import rcParams

# set up the parameters
```

```
rcParams['figure.figsize'] = 30,60
# show decision tree
plot_tree(model)
plt.show()
```

```
#  예제 4.7 XGBClassifier의 max_depth 옵션을 조절해 의사 결정 나무를 가지 치는 코드

model = XGBClassifier(max_depth=2)
model.fit(x_train, y_train)

plot_tree(model)
plt.show()

model = XGBClassifier(max_depth=4)
model.fit(x_train, y_train)

plot_tree(model)
plt.show()
```

```
# 예제 4.8 당뇨병 진단 모델의 피처 중요도를 계산하고 시각화하는 코드

from xgboost import plot_importance
# figure size change
rcParams['figure.figsize'] = 10, 10
plot_importance(model)
plt.yticks(fontsize=15)
plt.show()
```

```
# 예제 4.9 pdpbox를 사용해 GTT 피처에 대한 목표 플롯(Target plots)을 그리는 코드
# !pip install pdpbox
import pandas as pd
dataset = loadtxt('https://gist.githubusercontent.com/ktisha/c21e73a1bd1700294ef790c56c8aec1f/raw/819b69b5736821ccee93d05b51de0510bea0
df_dataset = pd.DataFrame(dataset[:,:9])
print(df_dataset)
df_dataset.columns = ['Pregnancies',
    'Glucose',
    'BloodPressure',
    'SkinThickness',
    'Insulin',
    'BMI',
    'DiabetesPedigreeFunction',
    'Age',
    'label']
from pdpbox import pdp, info_plots
pima_data = df_dataset
pima_features = df_dataset.columns[:8]
pima_target = df_dataset.columns[8]

fig, axes, summary_df = info_plots.actual_plot(
    model=model,
    X=pima_data[pima_features],
    feature='Glucose',
    feature_name='Glucose',
    predict_kwds={}
)
fig, axes, summary_df = info_plots.target_plot( df=pima_data, feature='Glucose', feature_name='Glucose', target=pima_target )
```

```
summary_df
```

```
# 예제 4.10 GTT 데이터에 대한 모델의 실제 예측 분포 플롯을 그리는 코드

fig, axes, summary_df = info_plots.actual_plot(
    model=model,
```

```
    X=pima_data[pima_features],
    feature='Glucose',
    feature_name='Glucose',
    predict_kwds={}
)
```

```
# 예제 4.11 GTT 테스트 피처에 대해 부분 의존성을 계산하고 플롯을 그리는 코드

# pdp isolation plot
pdp_gc = pdp.pdp_isolate(
    model=model,
    dataset=pima_data,
    model_features=pima_features,
    feature='Glucose'
)

# plot information
fig, axes = pdp.pdp_plot(
    pdp_gc,
    'Glucose',
    plot_lines=True, #False
    frac_to_plot=0.5,
    plot_pts_dist=True
)
```

```
# 예제 4.12 혈압과 GTT 테스트 데이터 두 피처에 대해 목표 플롯을 그리는 코드
fig, axes, summary_df = info_plots.target_plot_interact(
    df=pima_data,
    features=['BloodPressure', 'Glucose'],
    feature_names=['BloodPressure', 'Glucose'],
    target=pima_target
)
```

```
# 예제 4.13 혈압과 GTT 테스트 데이터로 모델에 대한 부분 의존성 플롯을 그리는 코드

pdp_interaction = pdp.pdp_interact(
    model=model,
    dataset=pima_data,
    model_features=pima_features,
    features=['BloodPressure', 'Glucose']
)

fig, axes = pdp.pdp_interact_plot(
    pdp_interact_out=pdp_interaction,
    feature_names=['BloodPressure', 'Glucose'],
    plot_type='grid', #contour
    x_quantile=True,
    plot_pdp=True
)
```

```
# 예제 4.14 혈압 피처에 대해 부분 의존성 플롯을 그리는 코드

# calculate model with BloodPressure to express pdp
pdp_bp = pdp.pdp_isolate(
    model=model,
    dataset=pima_data,
    model_features=pima_features,
    feature='BloodPressure'
)

# plot PDP on BloodPressure
fig, axes = pdp.pdp_plot(
    pdp_bp,
    'BloodPressure',
    plot_lines=False,
    frac_to_plot=0.5,
    plot_pts_dist=True
)
```

```
# 예제 4.15 GridSearchCV를 통해 당뇨병 진단 모델의 최적 하이퍼파라미터를 찾는 코드
import numpy as np
from sklearn.model_selection import GridSearchCV

cv_params = {
    'max_depth': np.arange(1, 6, 1),
}
fix_params = {
    'booster': 'gbtree', 'objective': 'binary:logistic',
}

csv = GridSearchCV(
    XGBClassifier(**fix_params),
    cv_params,
    scoring = 'precision',
    cv = 5,
    n_jobs=5
)

csv.fit(x_train, y_train)
print(csv.best_params_)

# make predictions for test data
y_pred = csv.predict(x_test)
predictions = [round(value) for value in y_pred]

# evaluate predictions
accuracy = accuracy_score(y_test, predictions)
print("Accuracy: %.2f%%" % (accuracy * 100.0))
```

```
# 예제 4.17 GridSearchCV를 사용해 max_depth, learning_rate, n_estimators 파라미터값 변화에 대한 최적의 모델을 찾는 코드

import numpy as np
from sklearn.model_selection import GridSearchCV
cv_params = {
    'max_depth': np.arange(1, 6, 1),
    'learning_rate': np.arange(0.05, 0.6, 0.05),
    'n_estimators': np.arange(50, 300, 50),
}
fix_params = {
    'booster': 'gbtree',
    'objective': 'binary:logistic',
}
csv = GridSearchCV(
    XGBClassifier( **fix_params),
    cv_params,
    scoring = 'precision',
    cv = 5,
    n_jobs=5
)
csv.fit(x_train, y_train)
# show best parameter score
print(csv.best_params_)
# make predictions for test data

y_pred = csv.predict(x_test)
predictions = [round(value) for value in y_pred]

# evaluate predictions
accuracy = accuracy_score(y_test, predictions)
print("Accuracy: %.2f%%" % (accuracy * 100.0))

# show every combination parameter in Grid Search

for parameter in csv.cv_results_["params"]:
    print(parameter)
```

```
# 예제 4.19 GridSearchCV로 찾아낸 당뇨병 진단 모델의 최적 파라미터

model = XGBClassifier(
    booster = 'gbtree',
    objective = 'binary:logistic',
```

```
    learning_rate = 0.1,
    n_estimators =  50,
    reg_alpha = 0.15,
    reg_lambda = 0.7,
    max_depth = 4
)
```

```
# 예제 4.20 sklearn 패키지로 컨퓨전 행렬을 계산하는 코드

from sklearn.metrics import confusion_matrix
cm = confusion_matrix(y_test, y_pred)
print(cm)
```

```
# 예제 4.22 sklearn 패키지로 계산한 컨퓨전 행렬을 matplotlib을 통해 시각화하는 코드

import itertools

def plot_confusion_matrix(cm, classes, normalize=False, title='Confusion matrix', cmap=plt.cm.Blues):
    plt.imshow(cm, interpolation='nearest', cmap=cmap)
    plt.title(title)
    plt.colorbar()
    tick_marks = np.arange(len(classes))
    plt.xticks(tick_marks, classes, rotation=45)
    plt.yticks(tick_marks, classes)
    if normalize:
        cm = cm.astype('float') / cm.sum(axis=1)[:, np.newaxis]

    thresh = cm.max() / 2.

    for i, j in itertools.product(range(cm.shape[0]), range(cm.shape[1])):
        plt.text(
            j,
            i,
            cm[i, j],
            horizontalalignment="center",
            color="white" if cm[i, j] > thresh else "black")
    plt.tight_layout()
    plt.ylabel('True label')
    plt.xlabel('Predicted label')
    plt.show()

def show_data(cm, print_res = 0):
    tp = cm[1,1]
    fn = cm[1,0]
    fp = cm[0,1]
    tn = cm[0,0]
    if print_res == 1:
        print('Precision = {:.3f}'.format(tp/(tp+fp)))
    print('Recall (TPR) = {:.3f}'.format(tp/(tp+fn)))
    print('Fallout (FPR) = {:.3e}'.format(fp/(fp+tn)))

    return tp/(tp+fp), tp/(tp+fn), fp/(fp+tn)

plot_confusion_matrix(cm, ['0', '1'], )
show_data(cm, print_res=1)
```