IoT·인공지능·빅데이터의 실무응용 연구 1

Machine Learning을 이용한 차주(車主)별 다음 구매 차량 예측 모델

코딩 기술서

기술경영경제정책 석사과정 김민아

2022-1-21

목차

1. 프로젝트 개요

Business Intelligence는 데이터를 분석하여 소비자 Segmentation 및 Targeting을 포함한 마케팅 전략을 비롯하여 경영전략을 수립하는 융합 기술이다. 해당 기술은 사업의 domain에 대한 지식과, business적인 이해도와 함께 computing 스킬 및 통계학적 지식을 두루 갖추었을 때 그 진가를 발휘할 수 있다.

하지만 그 무엇보다 중요한 것은 Data의 확보 자체이다. 전산화로 인해 무수히 많은 Data가 쏟아지고 있으나, 그 중 유의미한 Data를 확보하는 것은 다른 차원이다. 신뢰도가 높은 Data를 다량 확보하고 있더라도 분석 가능한 상태로 정제하는 것은 높은 수준의 지식과 스킬, 그리고 인내력을 요구하는 경우가 많다.

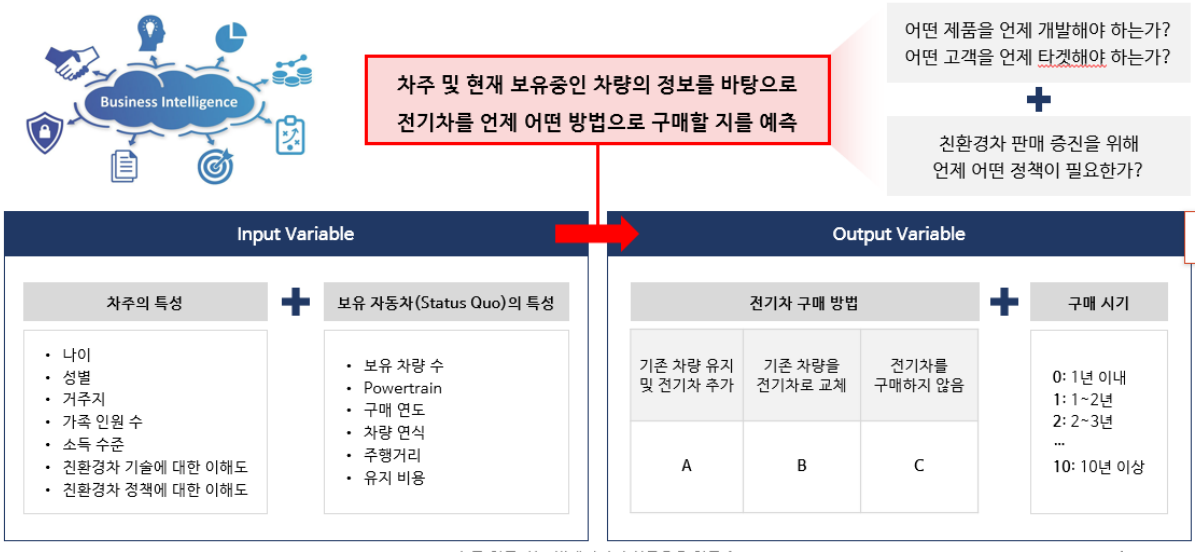
하지만 아예 Data 자체가 부재한 경우도 매우 많다. 무의미한 Data를 대상으로 분석을 진행한다면 얻게 되는 implication 역시 무의미할 뿐이다. Open Source로는 유의미한 분석이 불가능한 질문의 경우, Survey를 통해 Data를 확보하는 것이 일반적이며, 또한 가장 효과적인 방법 중 하나이다.

해당 프로젝트는 한국 국적의 자동차 소유주들 516명을 대상으로 진행한 설문 데이터를 사용한다. 유료 설문조사였던 만큼, 데이터가 잘 정제되어 있으며 모집

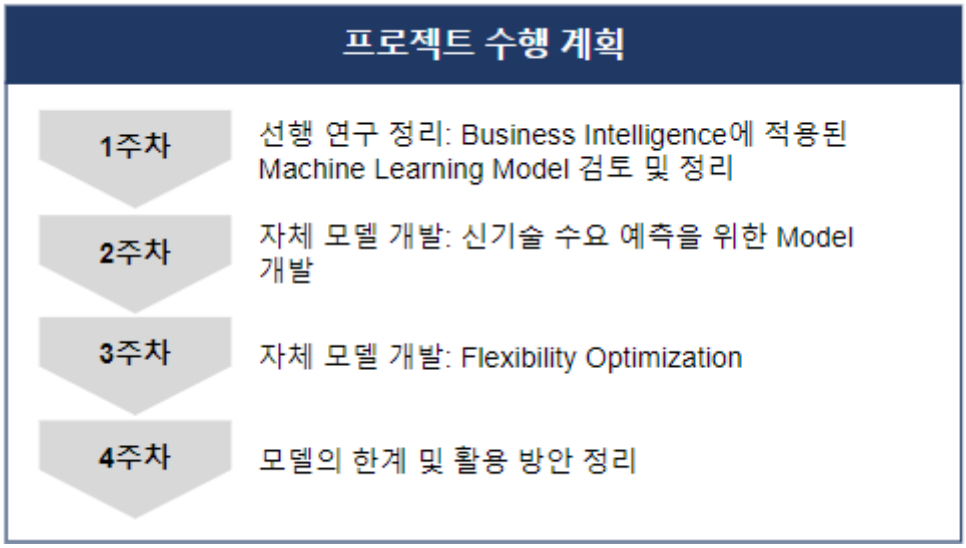
단의 분포도 역시 bias가 낮고 신뢰도가 높다.

설문조사의 본래 의도는 소비자들이 자동차를 선택할 때 중요하게 고려하는 요인들이 어떤 것이 있으며, 그 요인들 사이에 trade-off가 있다면 WTP(Willingness To Pay)가 어떻게 변화하는지 확인하기 위한 Conjoint Analysis의 기초 자를 확보하는 것이었다.

하지만, 해당 데이터의 demographic information과 자동차 선택에 영향을 줄 수 있는 다양한 기초 질문들(수송 부문이 환경 문제와 어떻게 관련되어 있다고 생각하는 지 등)에 대한 응답을 활용하여 차량 소유주에 대한 정보를 부여하면, 다음에 구매할 차량의 종류를 예측하는 자료로도 활용할 수 있을 것이다. 이는 물론 소위 말하는 인공지능, 실제로는 지능이 아니라 통계학적 분석인, Machine Learning을 활용해서 고려해볼 수 있는 방안이다. 본 프로젝트는 이를 실현해보는 데에 그 의의를 두고 있다.



2. 프로젝트 진행 방법



선행 연구들을 통해 Business Intelligence에 주로 적용되는 Machine Learning Model이 Random Forest 모델임을 확인했다. Random Forest 모델은 Testing Accuracy는 높으나, 해석을 통해 implication을 얻는 것이 불가능하므로, 해석을 위해 Decision Tree 모델을 적용했으며, 교수님으로부터 얻은 가이드를 바탕으로 Support Vector Machine 모델도 적용했다.

구체적으로는 Python 3.7.5 버전에서 scikit-learn 오픈 라이브러리를 활용했다.

데이터는 앞서 프로젝트 개요에서 언급한 설문조사를 활용했다. (정확한 문항과 답안은 설문조사 시 계약한 바를 준수하기 위해 공개할 수 없다.)

Ordinal Data는 Numerical로, 그 외에는 Categorical로 처리하여 Input Data로 활용했고, Label(구매 의향이 가장 큰 자동차의 연료유형)은 5가지였다;

1) 휘발유 2) 경유 3) LPG 4) 전기 5) 수소

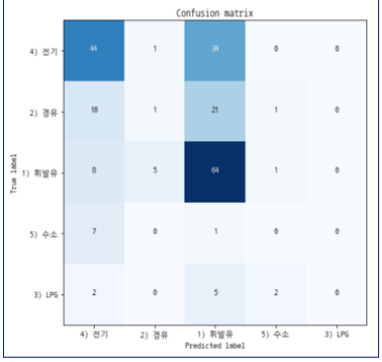
연령, 운전 경력 등 인적 정보와 현재 보유하고 있는 차량의 구매가, 연식, 구매 시기 등 수치로 기입되는 데이터 외에 주관적으로 판단한 "상대적 정도"를 나타내는 데이터 역시 Ordinal 데이터로서 numerical로 처리했다. 또한, 복수 응답이 가능한 Categorical 문항들의 경우, 모든 선택지에 대해 "O/X"로 질문한 것으로 변환했다.

3. 프로젝트 결과 및 결론

Machine Learning 모델 별 Testing Accuracy

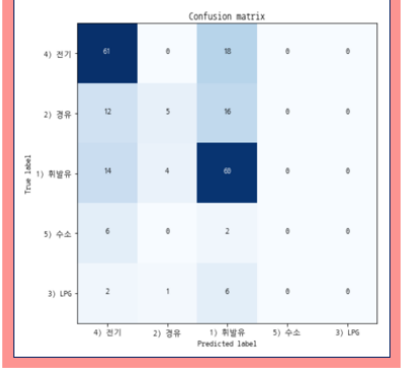
Decision Tree

	Precision	Recall	F1-score	support
휘발유	0.51	0.82	0.63	78
경유	0.14	0.03	0.05	33
LPG	0	0	0	9
전기	0.62	0.56	0.59	79
수소	0	0	0	8
Accuracy	0.53			



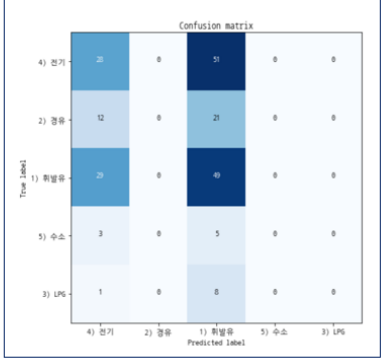
Random Forest

	Precision	Recall	F1-score	support
휘발유	0.59	0.77	0.67	78
경유	0.5	0.15	0.23	33
LPG	0	0	0	9
전기	0.64	0.77	0.7	79
수소	0	0	0	8
Accuracy	0.61			



Support Vector Machines

	Precision	Recall	F1-score	support
휘발유	0.37	0.63	0.46	78
경유	0	0	0	33
LPG	0	0	0	9
전기	0.38	0.35	0.37	79
수소	0	0	0	8
Accuracy	0.37			



Random Forest 모형의 Feature Importance

Numerical Input Data		Categorical Input Data	
Feature Importance	Feature	Feature Importance	Feature
0.035	지불의향(만원)	0.04	가장 유력한 신차의 차종
0.03	보유차량의 평균 구매가	0.03	휘발유 보유 여부
0.022	보유차량 중 가장 오래된 연식	0.029	[컨조인트1]
0.022	보유차량의 평균 유지비	0.028	거주지
0.021	운전 연차	0.025	[컨조인트2]
0.021	연평균 총주행거리	0.02	[컨조인트5]
0.018	월평균 개인 소득	0.019	[컨조인트3]
0.018	연령	0.018	직업군
0.016	환경에 도움이 된다면, 전기차를 사용할 의사가 있는 정도	0.018	[컨조인트4]
0.016	보유차량 중 가장 오래 전에 구입	0.016	[컨조인트6]
0.015	월평균 가구 소득	0.016	[컨조인트7]
0.015	정부의 친환경 차량 구매 지원 및 세제 혜택에 대해 알고 있는 정도	0.016	[컨조인트8]

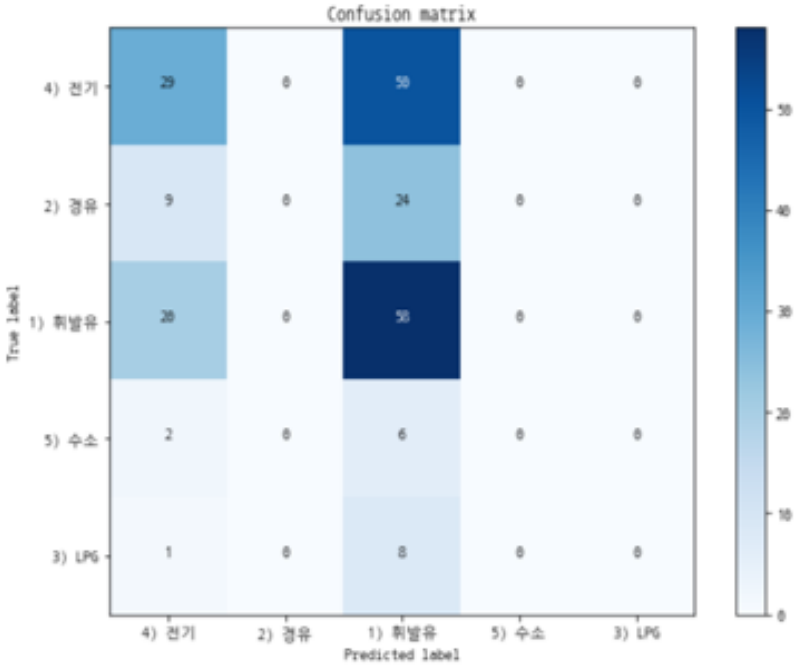
분석 결과 Machine Learning 모델들이 MNL에 비해 더 높은 Testing Accuracy를 보였으며, 특히 Random Forest가 가장 높은 Accuracy를 보였다. Prediction Accuracy가 0.85에 미치지 못 하는 0.6~0.7 수준인 것은 해당 Input Data만으로 차주의 다음 구매 차량을 정확히 예측하는 것에 한계가 있음을 드러낸다.

이는 자동차 모델의 선택이 "취향"을 반영하기 때문이다. 다시 말해 동일한 성별, 연령, 소득수준, 가족 구성원, 거주지, 직업군, 학력 등 어떤 사람을 설명하는 지표들의 동일한 사람들일지라도, 전혀 다른 자동차를 선호할 수 있다는 것이다. 완성차 업체가 특정 소비자군을 타겟하는 차량을 출시할 때 다양한 색상과 옵션을 준비하는 것 역시 자동차라는 상품이 소비자의 취향에 따라 결정되는 재화임을 증명한다.

해당 프로젝트는 아주 높은 수준의 Prediction Accuracy를 바탕으로 차주에 따라 다음에 구매할 차량의 연료유형을 예측하지는 못 했으나, 기존의 MNL (Multinominal Logit)모델에 비해 높은 Accuracy를 보임으로써 프로젝트의 목표인 Business Intelligence의 필요성을 드러냈다.

Multinomial Logit 모형으로는 다음에 어떤 연료유형의
자동차를 선택하는 지에 대한 Prediction Accuracy가
낮게(0.42) 도출되며, 5개 중 2개의 연료유형(휘발유/전기)만
Predict하는 편향성을 보임

accuracy			0.42	207
macro avg	0.17	0.22	0.19	207
weighted avg	0.33	0.42	0.35	207



4. 코드

```
from collections import Counter
import warnings
warnings.filterwarnings(action='ignore')

import numpy as np
import pandas as pd
from pandas.api.types import is_numeric_dtype
import matplotlib.pyplot as plt
from sklearn.linear_model import LogisticRegression
from sklearn.discriminant_analysis import LinearDiscriminantAnalysis,
QuadraticDiscriminantAnalysis
from sklearn.tree import DecisionTreeClassifier, plot_tree
from sklearn.naive_bayes import GaussianNB
from sklearn.svm import SVC
from sklearn.ensemble import RandomForestClassifier, AdaBoostClassifier,
GradientBoostingClassifier
from sklearn.model_selection import train_test_split
from sklearn.metrics import classification_report
from sklearn.preprocessing import OneHotEncoder
%matplotlib inline

plt.rcParams["figure.figsize"] = (10, 10)
plt.rcParams["font.family"] = 'NanumGothicCoding'

import matplotlib.pyplot as plt
import numpy as np
import itertools
from sklearn.metrics import confusion_matrix

# Ref https://www.kaggle.com/grfiv4/plot-a-confusion-matrix
def plot_confusion_matrix(y_true, y_pred,
                          target_names=None, title='Confusion matrix',
cmap=plt.get_cmap('Blues'), normalize=False, figsize=(8,6)):
    '''
    given a sklearn confusion matrix (cm), make a nice plot

    Arguments
    ---------
    cm:           confusion matrix from sklearn.metrics.confusion_matrix

    target_names: given classification classes such as [0, 1, 2]
                  the class names, for example: ['high', 'medium', 'low']

    title:        the text to display at the top of the matrix
```

```
    cmap:         the gradient of the values displayed from
matplotlib.pyplot.cm
                  see
http://matplotlib.org/examples/color/colormaps_reference.html
                  plt.get_cmap('jet') or plt.cm.Blues

    normalize:    If False, plot the raw numbers
                  If True, plot the proportions

    Usage
    -----
    plot_confusion_matrix(cm           = cm,                  # confusion
matrix created by
                                                             #
sklearn.metrics.confusion_matrix
                          normalize    = True,                # show proportions
                          target_names = y_labels_vals,       # list of names
of the classes
                          title        = best_estimator_name) # title of graph

    Citiation
    ---------
    http://scikit-
learn.org/stable/auto_examples/model_selection/plot_confusion_matrix.html

    '''

    # Compute confusion matrix
    cm = confusion_matrix(y_true, y_pred)

    plt.figure(figsize=figsize)
    plt.imshow(cm, interpolation='nearest', cmap=cmap)
    plt.title(title)
    plt.colorbar()

    if target_names is not None:
        tick_marks = np.arange(len(target_names))
        plt.xticks(tick_marks, target_names)
        plt.yticks(tick_marks, target_names)

    if normalize:
        cm = cm.astype('float') / cm.sum(axis=1)[:, np.newaxis]

    thresh = cm.max() / 1.5 if normalize else cm.max() / 2
    for i, j in itertools.product(range(cm.shape[0]), range(cm.shape[1])):
        if normalize:
            plt.text(j, i, "{:0.4f}".format(cm[i, j]),
                     horizontalalignment="center",
```

```
                    color="white" if cm[i, j] > thresh else "black")
        else:
            plt.text(j, i, "{:,}".format(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()



from visualize import plot_confusion_matrix

DATA_PATH = 'data_modified_v3.xlsx'
df = pd.read_excel(DATA_PATH)
df = df.drop(['응답자 번호'], axis='columns')
print(df.shape)
df.head()

NUMERICAL_COLUMNS = [
    '연령',
    '운전 연차',
    '정치적 성향이 보수적인 정도',
    '만 60세 이상 가족 수',
    '초중고생 가족 수',
    '미취학 아동 가족 수',
    '보유 대수',
    '연평균 총주행거리',
    '구매까지 남은 시간(년)',
    '지불의향(만원)',
    '온실가스 배출로 인한 기후변화 문제에 관심 많은 정도',
    '미세먼지 배출로 인한 대기오염 문제에 관심 많은 정도',
    '정부의 친환경 차량 구매 지원 및 세제 혜택에 대해 알고 있는 정도',
    '환경 보호를 위해 노력하는 정도',
    '환경에 도움이 된다면, 전기차를 사용할 의사가 있는 정도',
    '환경에 도움이 된다면, 수소차를 사용할 의사가 있는 정도',
    '수송분야의 미세먼지로 인한 환경 문제는 심각하다고 생각하는 정도',
    '수송분야의 온실가스로 인한 환경 문제는 심각하다고 생각하는 정도',
    '전기차를 사용하면 대기오염 문제를 줄일 수 있을 것이라고 생각하는 정도',
    '친환경차 보급의 핵심이 수소차 보다는 전기차가 되어야 한다고 생각하는 정도',
    '수소차와 전기차의 기술적 차이 등에 대해 잘 알고 있는 정도',
    '보유차량 중 가장 오래 전에 구입',
    '보유차량 중 가장 오래된 연식',
    '자율주행차를 구입할 의향이 높은 정도',
    '자율주행차에 대해 알고 있는 정도',
    '보유차량의 평균 구매가',
```

```
    '보유차량의 평균 유지비',
    '보유 예정 차량 수',
    '동거 가족 수',
    '최종학력',
    '월평균 가구 소득',
    '월평균 개인 소득',
    '현 정부인 문재인 정부를 지지하는 정도',
]

CATEGORICAL_COLUMNS= [
    '성별',
    '거주지',
    '직업군',
    '자율주행차가 보행자를 치도록 세팅해야 되는가',
    '휘발유 운전경험',
    '경유 운전경험',
    'LPG 운전경험',
    '전기 운전경험',
    '수소 운전경험',
    '구매이유: 노후',
    '구매이유: 연비',
    '구매이유: 차량유형 변경',
    '구매이유: 친환경차 희망',
    '구매이유: 소득 개선',
    '구매계획 없음',
    '가장 유력한 신차의 차종',
    '휘발유 보유 여부',
    '경유 보유 여부',
    'LPG 보유 여부',
    '전기 보유 여부',
    '자율주행모드 중 사고의 책임자',
    '경차 보유 여부',
    '준중형 및 중형 세단 보유 여부',
    '준대형 및 대형 세단 보유 여부',
    '소형 SUV 보유 여부',
    '중형 및 대형 SUV 보유 여부',
    '승합차(RV) 보유 여부',
    '지지하는 정당',
    # '[컨조인트 1]',
    # '[컨조인트 2]',
    # '[컨조인트 3]',
    # '[컨조인트 4]',
    # '[컨조인트 5]',
    # '[컨조인트 6]',
    # '[컨조인트 7]',
    # '[컨조인트 8]',
]
TARGET_COLUMNS = [
```

```
    '전기차 구매방법 v.1',
    '다음차량 연료유형 및 구매방법',
    '전기차 구매방법 v.2',
    '가장 유력한 신차의 연료유형',
    '[컨조인트 1]',
    '[컨조인트 2]',
    '[컨조인트 3]',
    '[컨조인트 4]',
    '[컨조인트 5]',
    '[컨조인트 6]',
    '[컨조인트 7]',
    '[컨조인트 8]',
]

#BINARY = False
TARGET_COLUMN = TARGET_COLUMNS[2]

FEATURE_NAMES = NUMERICAL_COLUMNS[:]
X_numerical = df[NUMERICAL_COLUMNS].values

num_categories = []
X_categorical = []
for column in CATEGORICAL_COLUMNS:
    df_column = df[column]
    X_column = df_column.map({value:i for i, value in
enumerate(sorted(list(set(df_column))))}).values
    X_categorical.append(X_column.reshape(-1, 1))
    num_categories.append(len(set(X_column)))
    FEATURE_NAMES += [column + '_' + value for value in
sorted(list(set(df_column)))]
X_categorical = np.hstack(X_categorical)

one_hot_encoder = OneHotEncoder()
X_categorical = one_hot_encoder.fit_transform(
    X_categorical
).toarray()
assert sum(num_categories) == X_categorical.shape[1]

X = np.hstack([X_numerical, X_categorical])
assert X.shape[1] == len(FEATURE_NAMES)

# if BINARY:
#     y = (df[TARGET_COLUMN].values == 4).astype(np.int32)
# else:
#     y = df[TARGET_COLUMN].values

y = df[TARGET_COLUMN]
y, target_names = y.factorize()
```

```
print(X.shape, y.shape)

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.4,
random_state=0)

Counter(y_test)

target_names

clf = DecisionTreeClassifier(max_depth=5)
y_pred = clf.fit(X_train, y_train).predict(X_test)

print(classification_report(y_test, y_pred, target_names=target_names))
plot_confusion_matrix(y_test, y_pred, target_names=target_names)

fig, ax = plt.subplots(figsize=(50, 20))
plot_tree(clf, feature_names=FEATURE_NAMES, fontsize=10, ax=ax)
fig.savefig('tree.png')

clf = RandomForestClassifier()
y_pred = clf.fit(X_train, y_train).predict(X_test)

print(classification_report(y_test, y_pred, target_names=target_names))
plot_confusion_matrix(y_test, y_pred, target_names=target_names)

cumsum_num_categories = [0] + list(np.cumsum(num_categories))

importances = clf.feature_importances_
numerical_importances = importances[:len(NUMERICAL_COLUMNS)]
categorical_importances = []
for start_index, end_index in zip(cumsum_num_categories[:-1],
cumsum_num_categories[1:]):
    categorical_importances.append(sum(importances[len(NUMERICAL_COLUMNS):][st
art_index:end_index]))

sum(numerical_importances) + sum(categorical_importances)

print('Numerical')
for i in np.argsort(numerical_importances)[::-1]:
    print(f'{numerical_importances[i]:.3f}', NUMERICAL_COLUMNS[i])

    print('Categorical')
for i in np.argsort(categorical_importances)[::-1]:
    print(f'{categorical_importances[i]:.3f}', f'{num_categories[i]:3d}',
CATEGORICAL_COLUMNS[i])

    clf = LogisticRegression()
```

```
y_pred = clf.fit(X_train, y_train).predict(X_test)

print(classification_report(y_test, y_pred,target_names=target_names))
plot_confusion_matrix(y_test, y_pred,target_names=target_names)

import statsmodels.api as sm

X2 = sm.add_constant(X_train)
est = sm.OLS(y_train, X2)
est2 = est.fit()
print(est2.summary())
```