박새롬, 김보현

Machine Learning 보고서

목차

프로젝트 소개

주제

L백화점 고객 거래 데이터를 이용한 성별 예측

모델링 방법

Machine Learning 모델과 Neural Network 모델을 함께 사용해 모델링 진행

데이터 형태

train 데이터 : 3500명의 고객, 원본 shape = (232004,7)
test 데이터 : 2482명의 고객, 원본 shape = (163558,7)
target 비중 : 남성(37.6%), 여성(62.4%)

프로젝트 소개

전략

의미있는 피쳐 생성
: 피쳐의 개수보다는 target 값을 잘 구별하는 의미있는 피쳐 생성에 더 중점을 둠

모델링 할 때 파라미터 튜닝 시간이 많이 걸리더라도 신경쓰기
: tree계열은 파라미터 값에 따라 성능 차이가 많이 나기 때문에 파라미터 튜닝을 열심히 함

프로젝트 임하기 전 마음가짐

꾸준하고 성실하게 차근차근 성능을 올려 1등하기!
1등을 하지 못하더라도 포기하지 않고 끝까지 노력하기!

사용한 피쳐 설명

1 **개인 피쳐**

사전 제작 피쳐 추가
장성민학우 코드 참고해 의
미있는 피쳐 추가

약 600 ~ 700 여개

2 **BOW**

gds_grp_nm으로만
생성해서 따로 학습

143개

3 **W2V**

각자 워드투벡터 적용 피쳐
김세홍학우 피쳐 추가

300개

-> 피쳐 개수가 매우 적었음!

사용한 피쳐 설명

1

개인 피쳐

추가 설명

개인과제 1 때 만들었던 피쳐

EDA를 진행하면서 성별을 잘 구분하는 피쳐를 추가로 생성

장성민님 코드를 참고해서 성별을 잘 구분하는 걸로 보이는 피쳐 추가

피쳐들끼리 상관관계가 높은 피쳐 삭제

범주형 데이터 One-Hot Encoding 진행할 때 각각의 열들을 살피고 성별을 잘 구분하지 못하는 피쳐 삭제

총구매액같이 값의 편차가 매우 큰 피쳐들이 많이 존재해 스케일링과 로그변환 진행

피쳐 증강과 피쳐 셀렉션을 사용

사용한 피쳐 설명

카테고리화

수치형 데이터 중 범주가 큰 값 (ex. 총구매액)

수치형 데이터 중 최소값과 최대값의 차이가 너무 커서 이상치처럼 판별이 되는 의미있는 값들은 데이터를 살피고 각각의 데이터 특성에 따라 직접 범주를 구분해서 데이터를 수정함

로그변환 및 스케일링

값의 범주를 정리

데이터값의 비슷한 형태로 주기 위해 로그변환을 한 후

Min-Max Scaler와 Standard Scaler를 사용해 데이터를 스케일링 해줌

사용한 피쳐 설명

Encoding

One-Hot Encoding 사용

피쳐 셀렉션 사용 전에 처음 사용했는데 데이터가 sparse해져서 성능이 좋지 않았음

Mean Encoding 기법으로 변환

기존 방법보다 성능이 좋아져 계속 Mean Encoding을 사용해 피쳐 변경

피쳐 증강 및 셀렉션

Polynomial로 피쳐 증강

기존 피쳐가 약 600개 정도로 적어서 급하게 피쳐를 늘리기 위해 사용했음

sparse해지고 과하게 많아진 피쳐 셀렉션

shap 패키지를 사용해 모델링 직전에 피쳐 중요도를 보고 차원 축소

사용한 모델 설명

1. Machine Learning

(학습데이터의 30%를 test set으로 사용.)

Kneighbors Classifier

n_neighbors : sp_randint(2,15)
leaf_size : sp_randing(10,30)
weights : ['uniform' ,' distance']

MLP Classifier

batch_size : [32, 64, 128]
learning_rate : ['constant' ,' adaptive']
activation : ['tanh' ,' relu']
solver : ['sgd' ,' adam']

Catboosting Classifier

depth : sp_randint(3,15)
learning_rate : uniform(0.01,0.5)
min_data_in_leaf : sp_randint(1,10)
border_count : sp_randint(5,255)
grow_policy : ['Depthwise' ,' Lossguide']

RandomForest Classifier

max_depth : sp_randint(5,20)
max_features : ['auto' ,' sqrt' ,' log2']
min_samples_leaf : sp_randint(2,15)
min_samples_split : sp_randint(3,15)

GradientBoosting Classifier

n_estimators : sp_randint(10,200)
max_depth : sp_randint(5,12)
learning_rate : uniform(0.05,1)
subsample : uniform(0.1,0)

XGB Classifier

n_estimators : sp_randint(10,200)
learning_rate : uniform(0.01,0.8),
gamma : sp_randint(0,5)
max_depth : sp_randint(5,13)
min_child_weight : sp_randint(2,8)
subsample : uniform(0,1

LGBM Classifier

min_child_weight : range(0,121,20)
n_estimators : sp_randint(10,200)
max_depth : sp_randint(3,20)
learning_rate : uniform(0.01,1)
subsample : np.arange(0.5,1.0,0.1)

Logistic Regression

C : uniform(0.001, 1.1)
penalty : ['l1' ,' l2']
max_iter : sp_randint(2,100)
tol : uniform(0.001,0.5)

SVC

C : [1,10,15,20,50,100,200]
gamma : [0.001,0.05,0.01,0.1]

ExtraTree Classifier

n_estimators : range(50,200,10)
max_depth : range(2,15)

사용한 모델 설명

1. Machine Learning

Pred_Voting

Kneighbors Classifier와 Catboost Classifier를 제외한 Model들을 사용하여 Soft Voting Classifier 적용 후 test 데이터 예측

Pred_Mean

[0,1,2.56] 중에서 가장 성능이 좋은 p값과 모델의 조합을 찾아내어 test 데이터에 적용

Pred_Stacking

성능이 좋은 모델들로 Stacking 적용.
meta model로 logistic regression 또는 XGBClassifier 사용하여 test 데이터 예측

Pred_Best

사용한 모델 중에서 성능이 가장 높은 모델로 test 데이터 예측

최종 예측 값 :
산술평균
or
기하평균
적용

사용한 모델 설명

2. Deep Learning

(학습데이터의 30%를 validation set으로 사용)

DNN Model A : 하나의 피쳐로 단일모델 개발 후 test 데이터 예측

A

[68]:

```
preds = []
accu = []

for i in tqdm(range(10)):
    SEED = np.random.randint(1, 20000)
    random.seed(SEED)
    np.random.seed(SEED)
    if tf.__version__[0] < '2':
        tf.set_random_seed(SEED)
    else:
        tf.random.set_seed(SEED)

    # Define the NN architecture
    input = Input(shape=(X_train.shape[1],))
    x = Dense(130, activation='relu',kernel_regularizer=l1(0.0001))(input)
    x = Dropout(0.2)(x)
    x = Dense(64,activation='relu',kernel_regularizer=l1(0.0001))(x)
    output = Dense(1, activation='sigmoid')(x)
    model = Model(input, output)

    # Choose the optimizer and the cost function
    model.compile(loss='binary_crossentropy', optimizer=Adamax(0.001), metrics=['acc'])

    # Train the model
    callbacks = [keras.callbacks.EarlyStopping(monitor='val_loss', patience=1000)]
    hist = model.fit(X_train, y_train, validation_split=0.2,batch_size=2048, epochs=500,
                     callbacks=callbacks, shuffle=True, verbose=0)
    print(model.evaluate(X_valid,y_valid))
    print(roc_auc_score(y_valid,model.predict(X_valid)))

    # Make submissions
    pred = model.predict(test).flatten()
    preds.append(pred)
    accu.append(roc_auc_score(y_valid,model.predict(X_valid)))
```

DNN Model A

함수형 API로 DNN 모델 개발

- Seed 값 랜덤으로 설정
- L1 regularizer 적용(0.0001)
- Optimizer로 Adamax 사용(lr = 0.001)

사용한 모델 설명

2. Deep Learning

(학습데이터의 30%를 validation set으로 사용)

DNN Model B : 4개의 피쳐를 concat하여 만든 모델로 test 데이터 예측

B

```
[ 25]:

in_data1 = Input(shape=(train_X1.shape[1],), name='data1')
in_data2 = Input(shape=(train_X_w2v1.shape[1],), name='data2')
in_data3 = Input(shape=(train_X_w2v2.shape[1],), name='data3')
in_data4 = Input(shape=(train_X2.shape[1],), name='data4')

mid_data1 = Dense(128, activation = 'tanh',kernel_initializer='random_uniform', kernel_regularizer=l2(0.01))(in_data1)
mid_data2 = Dense(512, activation = 'tanh',kernel_initializer='random_uniform', kernel_regularizer=l2(0.01))(in_data2)
mid_data3 = Dense(512, activation = 'tanh',kernel_initializer='random_uniform', kernel_regularizer=l2(0.01))(in_data3)
mid_data4 = Dense(128, activation = 'tanh',kernel_initializer='random_uniform', kernel_regularizer=l2(0.01))(in_data4)

mid_data1 = Dense(64, activation = 'tanh', kernel_regularizer=l2(0.001))(mid_data1)
mid_data2 = Dense(256, activation = 'tanh', kernel_regularizer=l2(0.001))(mid_data2)
mid_data3 = Dense(256, activation = 'tanh', kernel_regularizer=l2(0.001))(mid_data3)
mid_data4 = Dense(128, activation = 'tanh', kernel_regularizer=l2(0.001))(mid_data4)

mid_data1 = Lambda(lambda y: y**2)(mid_data1)
mid_data2 = Lambda(lambda y: y**2)(mid_data2)
mid_data3 = Lambda(lambda y: y**2)(mid_data3)
mid_data4 = Lambda(lambda y: y**2)(mid_data4)

mid_data1 = Dense(32, activation = 'tanh', kernel_regularizer=l2(0.005))(mid_data1)
mid_data2 = Dense(64, activation = 'tanh', kernel_regularizer=l2(0.005))(mid_data2)
mid_data3 = Dense(64, activation = 'tanh', kernel_regularizer=l2(0.005))(mid_data3)
mid_data4 = Dense(64, activation = 'tanh', kernel_regularizer=l2(0.005))(mid_data4)

mid_data1 = Dropout(0.3)(mid_data1)
mid_data2 = Dropout(0.2)(mid_data2)
mid_data3 = Dropout(0.4)(mid_data3)
mid_data4 = Dropout(0.5)(mid_data4)

end_data1 = Dense(32, activation = 'tanh', kernel_regularizer=l2(0.001))(mid_data1)
end_data2 = Dense(32, activation = 'tanh', kernel_regularizer=l2(0.001))(mid_data2)
end_data3 = Dense(32, activation = 'tanh', kernel_regularizer=l2(0.001))(mid_data3)
end_data4 = Dense(32, activation = 'tanh', kernel_regularizer=l2(0.001))(mid_data4)

merge = concatenate([end_data1, end_data2, end_data3, end_data4])

x = Dense(64, activation = 'relu')(merge)
x = Lambda(lambda y: y**2)(x)
x = Dropout(0.5)(x)
x = Dense(256,activation = 'relu')(x)
x = Dense(128,activation = 'relu')(x)
x = Dropout(0.2)(x)
x = Dense(64, activation='relu')(x)
x = Dense(128, activation='relu')(x)
x = Dropout(0.1)(x)
x = Dense(64, activation='relu')(x)
x = Dense(32, activation='relu')(x)
output = Dense(1, activation='sigmoid')(x)

model = Model(inputs=[in_data1, in_data2, in_data3, in_data4], outputs=output)
model.summary()
model.compile(optimizer='rmsprop',loss='binary_crossentropy',metrics=['accuracy'])
callbacks = [keras.callbacks.EarlyStopping(monitor='val_loss', patience=5),
             keras.callbacks.ModelCheckpoint(filepath='model_0615-cat.h5', monitor='val_loss', save_best_only=True)]
```

DNN Model B

함수형 API로 DNN 모델 개발

- Activation 함수를 'tanh'와 'relu' 그리고 'sigmoid' 사용
- L2 regularizer(0.01)와 initializer, lambda 적용
- 데이터 4개를 입력받아 concatenate 해주는 방식
- Optimizer로 RMSprop 사용

앙상블

주로 캐글에서 사용

79.964%

ML + W2V Best

Gmean 적용

그룹과제-1 Best (0.790)
+
김세홍 W2V(0.793)
+
오주영 W2V(0.788)

80.728%

ML + DL + W2V

Gmean 적용

ML + W2V Best(0.799)
+
황윤재 DL
+
홍재성 DL

80.896%

ML + DL Best

Power Mean 적용

새로운 피쳐에 머신러닝 적용
+
황윤재 DL(0.806)
+
홍재성 DL(0.804)
+
박혜지 DNN(0.798)
+
유덕상 CNN(0.797)

80.993%

DL Best

Power Mean(p=4.5), Gmean 적용

황윤재 DL(0.806)
+
홍재성 DL(0.804)
+
유덕상 CNN(0.797)
+
유덕상 DNN(0.797)

한계점 및 보완점

피쳐

피쳐 증강을 하기 전에는 600개 정도로 충분한 학습을 하기에는 양이 적었다. 초반에 성능이 오르지 않다보니, 피쳐 개발에 크게 집중하지 못하고 바로 모델링으로 넘어가 파라미터 튜닝에만 과하게 집중했었다. 다양한 방법을 시도해보고 다른 팀의 코드도 많이 참고해서 피쳐를 많이 생성하지 못한 것에 대한 아쉬움이 존재한다.

파라미터 튜닝에는 한계가 있기 때문에, 초반부터 피쳐 개발에 집중을 했었다면 더 좋은 성과를 내지 않았을까 하는 생각이 든다.

한계점 및 보완점

모델링

모델링을 할 때 독립적인 각각의 피쳐그룹을 사용해서 모델링을 따로 한 후 앙상블을 하는게 좋은 방법인데, 피쳐가 적다보니 피쳐를 나누지 못했다. 처음에 score가 베이스라인을 넘지 못하고 있어서 점수 올리기에 급급해 너무 1차원적으로만 접근을 했던 것 같다.

또한, Conv1d, LSTM 등과 같은 다양한 Deep Learning 모델들을 사용해봤지만, 성능이 더욱 안좋아지는 것을 알 수 있었다. 가장 기본인 Dense층을 사용한 DNN으로 모델이 훨씬 성능이 좋았다.

앙상블

앙상블 할 때 성능이 좋은 것만 앙상블 할 게 아니라 상관관계도 고려해서 독립적인 모델들끼리 앙상블을 해야하는데, 이미 시간이 많이 지난 후에 그 부분을 파악해서 제대로 앙상블을 못 한 것 같다.

감사합니다

Thank You