

CJ 고객구매데이터를 활용한 프라임 회원 예측 모델링 고도화

이영석, 이은경, 전진하, 조민서

Contents

01 분석 목적

02 EDA/가설검증

03 전처리

04 모델링

05 활용

06 보완점

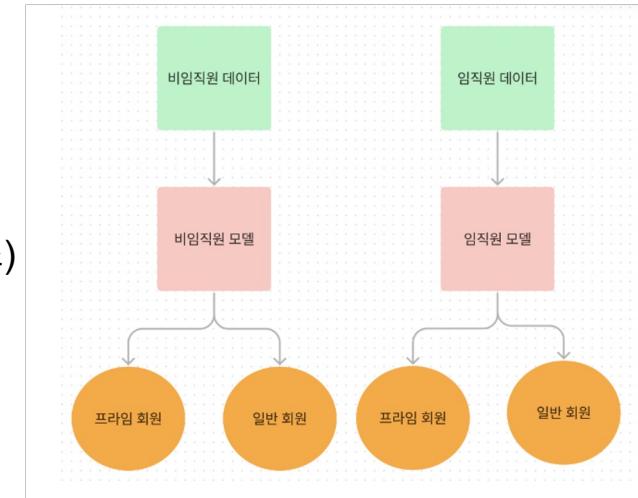
분석 목적

- 주 분석 task

CJ 더 마켓 고객 주문 데이터를 활용하여 임직원과 비임직원으로 나눈 후, 각각 프라임 회원과 일반회원으로 분류하는 모델링을 진행

- 분석의 필요성 = CRM의 중요성

- 고객의 특성에 기초한 마케팅 구상
(프라임회원- 유료 서비스 지속 / 일반회원- 회원 혜택 강조)
- 고객 집단의 특징 일반화 및 설명 가능
- 고객 충성도의 핵심 원인 분석



가설

HYPOTHESIS

1) 임직원은 비임직원에 비해 프라임 서비스를 사용할 확률이 높을 것이다.



직원이 회사의 서비스의 혜택에 대한 접근성이 좋기 때문

증명 방법) 실제 데이터를 통해 임직원 중 프라임서비스를 사용하는 데이터의 개수와 비임직원이 중 프라임서비스를 사용하는 데이터의 개수를 비교

가설

Conclusion

```
[ ] #가설 1
rows_Y = df_Y.shape[0]
rows_N = df_N.shape[0]

if rows_Y > rows_N:
    print("임직원이 더 많이 사용합니다 비 임직원 보다.")
else:
    print("df_N has more rows than df_Y")
```

df_N has more rows than df_Y

비임직원이 더 많이 사용하였다.

가설

HYPOTHESIS

을 찾으니,

이라고 생각

2) 임직원, 비임직원 공통적으로 더 프라임 멤버십 사용 고객의 주요 연령층은 30-40대

주 구매고객의 나이대가 대부분 30-40대로 예상되기에 해당 연령층이 많이 사용할 것

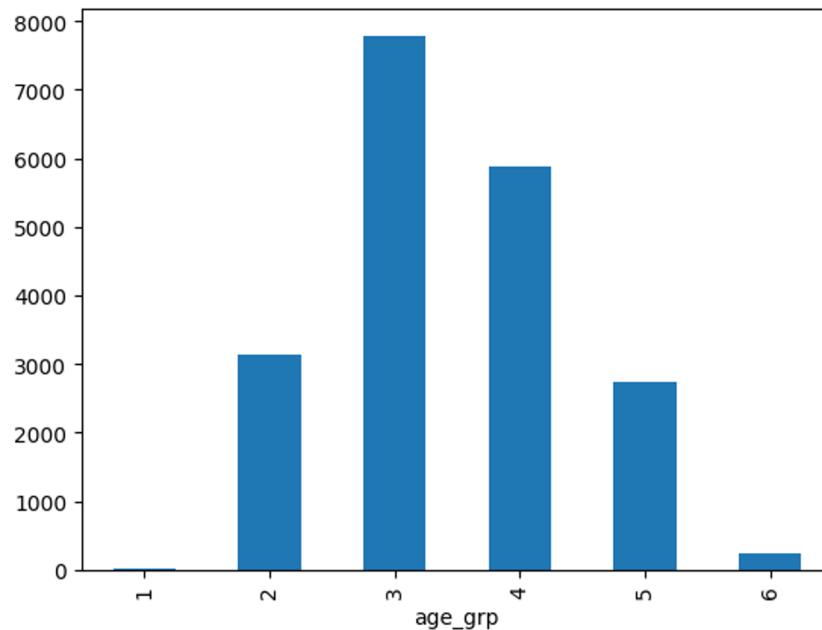
증명 방법) 임직원과 비임직원 각각의 데이터에서 나이대 별로
groupby 하여 프라임회원/일반회원 데이터 개수를 비교

가설

Conclusion

```
df_Y.groupby("age_grp").size()
```

```
age_grp
1      9
2    3140
3   7788
4   5891
5   2730
6    247
dtype: int64
```

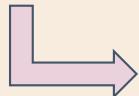


실제로 30,40대 그룹이 더 많이 사용하였다.

가설

HYPOTHESIS

3) 임직원, 비임직원 공통적으로 CJ 더 프라임 회원들이 일반회원보다 특정 상품에 대한 주문 수량이 10개 이상인 경우가 더 많이 나타날 것이다.



특정 상품을 10개 이상 구매한다는 것은 주기적인 구매가 이루어진 것으로 해석할 수 있습니다

증명 방법) 고객 주문 데이터에 대한 개수가 가설에 유의미한 영향을 주는지
검증해 볼 예정입니다.

가설

Conclusion

```
df_Y.groupby("net_order_qty").size()
```

| | net_order_qty |
|----|---------------|
| 1 | 13855 |
| 2 | 3423 |
| 3 | 961 |
| 4 | 469 |
| 5 | 375 |
| 6 | 205 |
| 7 | 36 |
| 8 | 63 |
| 9 | 29 |
| 10 | 222 |

```
# 전체 그룹들의 크기 계산  
total = grouped_qty.sum()  
  
# 비율 계산  
ratio = larger_than_10 / total  
  
print("비율:", ratio)  
round_ratio = round(ratio*100,2)  
  
print(f"백분위 :{round_ratio}%)
```

비율: 0.008432214087351678
백분위: 0.84%

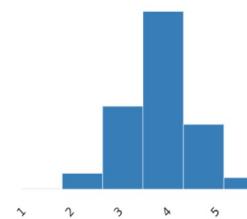
→ 주문 수량이 10이상인 부분은 0.84%

프로파일링

age_grp
Real number (ℝ)

| | |
|--------------|-----------|
| Distinct | 6 |
| Distinct (%) | < 0.1% |
| Missing | 0 |
| Missing (%) | 0.0% |
| Infinite | 0 |
| Infinite (%) | 0.0% |
| Mean | 3.9207518 |
| Minimum | 1 |
| Maximum | 6 |
| Zeros | 0 |
| Zeros (%) | 0.0% |
| Negative | 0 |
| Negative (%) | 0.0% |
| Memory size | 203.8 KiB |

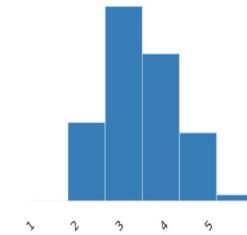
비임직원



ge_grp
Real number (ℝ)

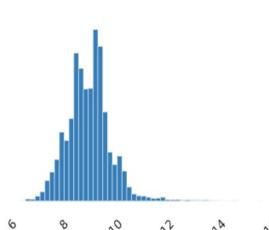
| | |
|--------------|-----------|
| Distinct | 6 |
| Distinct (%) | < 0.1% |
| Missing | 0 |
| Missing (%) | 0.0% |
| Infinite | 0 |
| Infinite (%) | 0.0% |
| Mean | 3.4510982 |
| Minimum | 1 |
| Maximum | 6 |
| Zeros | 0 |
| Zeros (%) | 0.0% |
| Negative | 0 |
| Negative (%) | 0.0% |
| Memory size | 154.9 KiB |

임직원



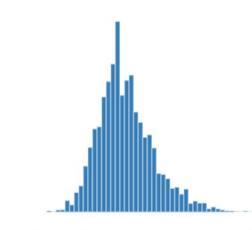
net_order_amt
Real number (ℝ)

| | |
|--------------|-----------|
| Distinct | 6963 |
| Distinct (%) | 26.7% |
| Missing | 0 |
| Missing (%) | 0.0% |
| Infinite | 0 |
| Infinite (%) | 0.0% |
| Mean | 8.959268 |
| Minimum | 6.315358 |
| Maximum | 15.647347 |
| Zeros | 0 |
| Zeros (%) | 0.0% |
| Negative | 0 |
| Negative (%) | 0.0% |
| Memory size | 203.8 KiB |



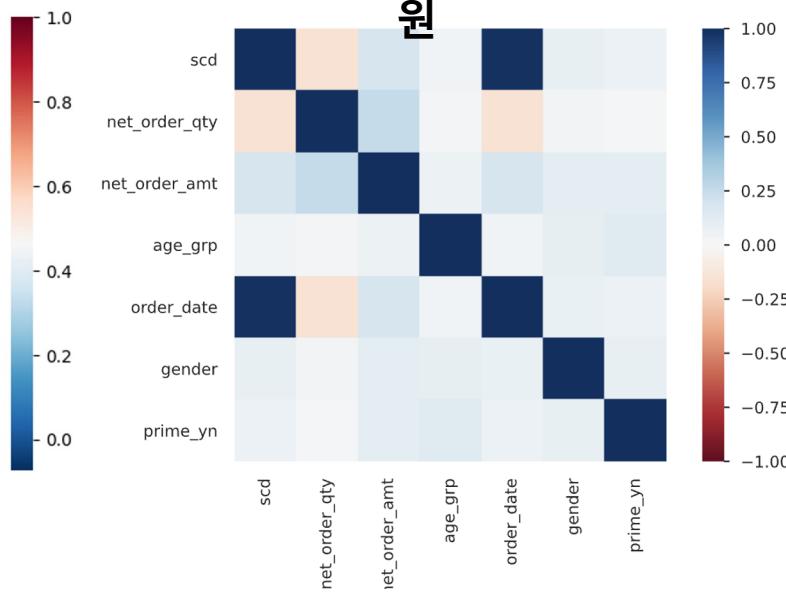
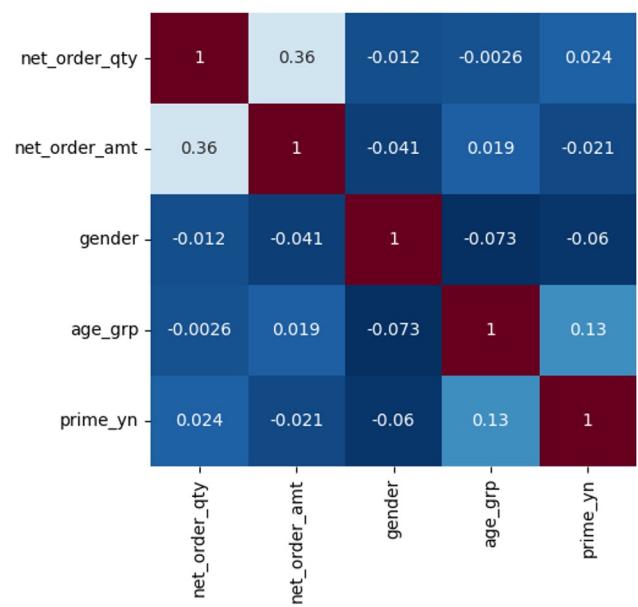
net_order_amt
Real number (ℝ)

| | |
|--------------|-----------|
| Distinct | 6089 |
| Distinct (%) | 30.7% |
| Missing | 0 |
| Missing (%) | 0.0% |
| Infinite | 0 |
| Infinite (%) | 0.0% |
| Mean | 9.3696106 |
| Minimum | 6.3965949 |
| Maximum | 14.960234 |
| Zeros | 0 |
| Zeros (%) | 0.0% |
| Negative | 0 |
| Negative (%) | 0.0% |
| Memory size | 154.9 KiB |



EDA

임직원



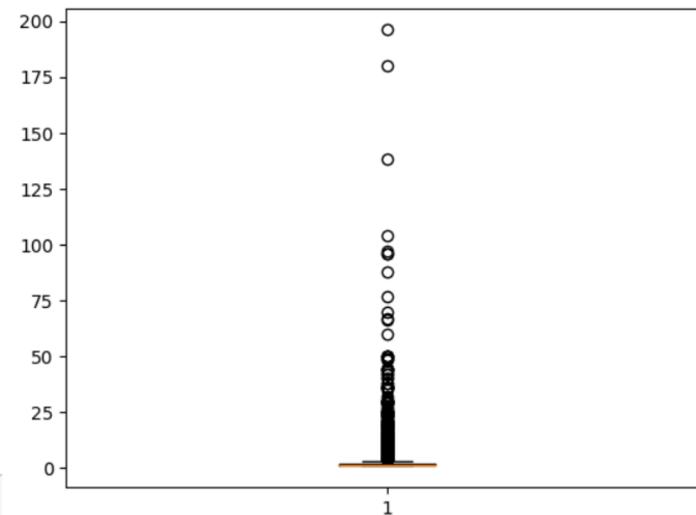
모두 0.4이하로 매우 낮아 별다른 feature selection의 필요성을 느끼지 못함
scd(주문 번호, 고객ID로 해석가능), order_date(주문 날짜)의 경우 데이터에서 고객은 한 날짜
에만 상품을 구매한 이력만 있기 때문에 둘의 상관성이 높게 나온 것으로 해석됨

전처리

net_order_qty(주문 수량) column의 경우,
0~200 범위 중 대부분 1의 값, 96이상: 7개

-> 해당 데이터를 이상치로 판단하여 주문수량이
96이상의 데이터는 96으로 처리

```
df.loc[df.net_order_qty>=96, 'net_order_qty'] = 96
```



| | | | | | | | | |
|---------------|----------------------------|-----|-----------|---|---|---|----------|---|
| I230113090654 | [2023설선물세트]스팸6호 | 96 | 15.613650 | F | 4 | N | 20230113 | N |
| I230113094377 | [2023설선물세트]스팸스위트1호 | 97 | 14.660738 | F | 3 | N | 20230113 | N |
| I230110064825 | [2023설선물세트]스팸고급유4호 | 104 | 14.703951 | M | 5 | N | 20230110 | N |
| I230111071770 | [2023설선물세트대량구매]특별한선택 K-2호 | 180 | 15.224838 | F | 3 | N | 20230111 | N |
| I230110058557 | [2023설사원선물신청] 스팸 8C호 | 138 | 14.960234 | F | 3 | Y | 20230110 | Y |
| I230109046418 | [2023설임직원캠페인대량구매] 특별한선택 Y호 | 196 | 15.647347 | M | 4 | N | 20230109 | N |
| I230101966659 | [2023설선물세트]백설 고소한 참기름 2호 | 96 | 13.846193 | M | 6 | N | 20230101 | Y |

전처리

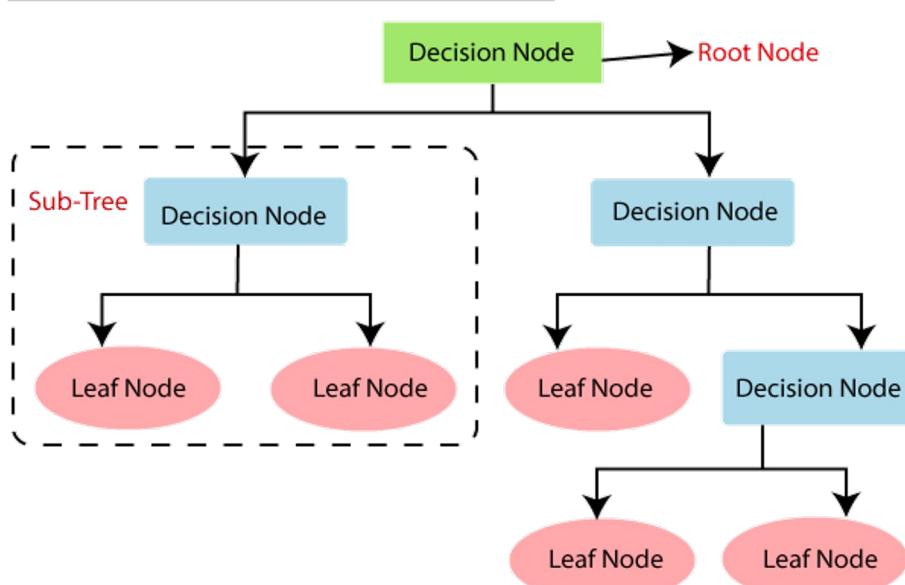
1. 상품구분(묶음/단일)

```
df_test.loc[df_test['product_name'].str.contains('x'), '상품구분'] = '묶음상품'  
df_test.loc[df_test['product_name'].str.contains('*'), '상품구분'] = '묶음상품'  
df_test.loc[df_test['product_name'].str.contains('X'), '상품구분'] = '묶음상품'  
df_test.loc[df_test['product_name'].str.contains('6입'), '상품구분'] = '묶음상품'  
df_test.loc[df_test['product_name'].str.contains('3입'), '상품구분'] = '묶음상품'  
df_test.loc[df_test['product_name'].str.contains('10입'), '상품구분'] = '묶음상품'  
df_test.loc[df_test['product_name'].str.contains('12입'), '상품구분'] = '묶음상품'  
df_test.loc[df_test['product_name'].str.contains('18입'), '상품구분'] = '묶음상품'  
df_test.loc[df_test['product_name'].str.contains('24입'), '상품구분'] = '묶음상품'  
df_test.loc[df_test['product_name'].str.contains('30입'), '상품구분'] = '묶음상품'  
df_test.loc[df_test['product_name'].str.contains('80입'), '상품구분'] = '묶음상품'
```

2.(주말/평일구분)

```
df_Y.loc[df_Y['order_date'] == 20230107, '평일/주말'] = '주말'  
df_Y.loc[df_Y['order_date'] == 20230108, '평일/주말'] = '주말'  
df_Y.loc[df_Y['order_date'] == 20230114, '평일/주말'] = '주말'  
df_Y.loc[df_Y['order_date'] == 20230115, '평일/주말'] = '주말'  
df_Y.loc[df_Y['order_date'] == 20230121, '평일/주말'] = '주말'  
df_Y.loc[df_Y['order_date'] == 20230122, '평일/주말'] = '주말'  
df_Y.loc[df_Y['order_date'] == 20230128, '평일/주말'] = '주말'  
df_Y.loc[df_Y['order_date'] == 20230129, '평일/주말'] = '주말'
```

Modeling



Decision tree

일련의 분류 규칙을 통해 데이터를 분류, 회귀하는 지도 학습 모델 중 하나
특정 기준(질문)에 따라 데이터를 구분

- **Prediction**

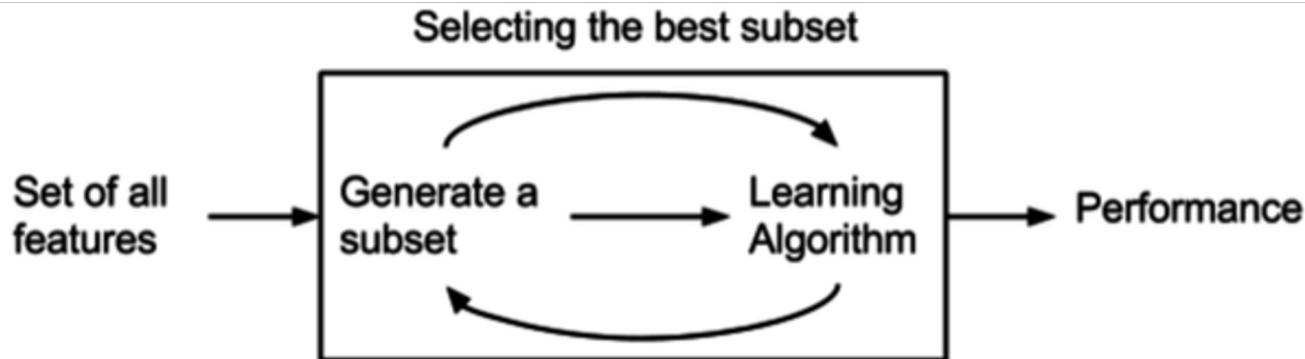
target값인 프라임 회원 여부와 나머지 attribute 간의 관계를 통해 특정 회원에 대한 프라임 회원 여부를 예측

- **Description**

프라임 회원 class에서 두드러지게 나타나는
나이대, 성별, 주문 수량등의 특정 패턴 해석
→프라임 회원 클래스에 대한 경향성 설명 가능

Modeling

Wrapper method (Forward Selection)



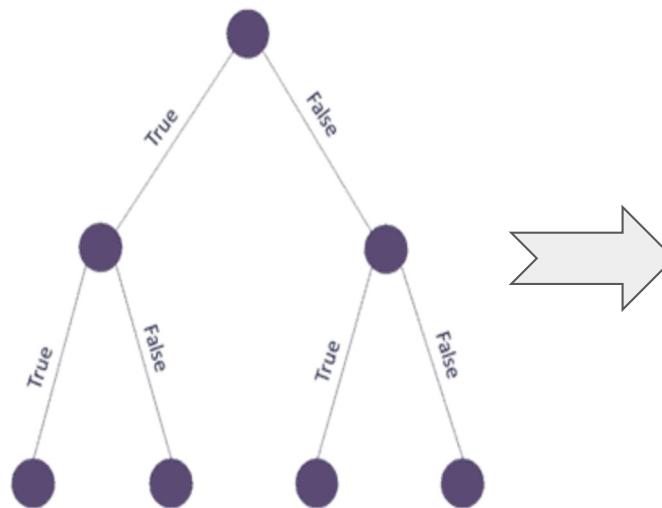
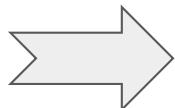
- F1-score 측면에서 가장 좋은 성능을 보이는 Feature subset을 뽑아내는 방법
- 반복할 때마다 가장 중요한 변수를 추가하여 더 이상 성능의 향상이 없을 때까지 변수를 추가
- 최종적으로 Best Feature Subset을 찾기 때문에, 모델의 성능을 위해서는 매우 바람직한 방법
- 유의미한 피처의 개수가 6개로 많지 않기 때문에 이 방법을 사용

Modeling

“net_order_qty(주문 수량)”

“net_order_amt(주문 금액)”

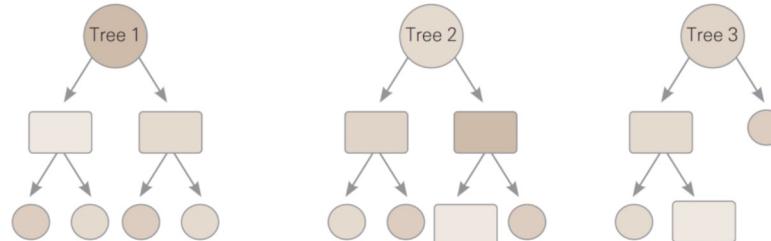
“상품 구분(단일/묶음)”



F1-score
임직원 0.90
비임직원 0.87

Modeling

- Decision tree 외의 분류 알고리즘 성능도 확인



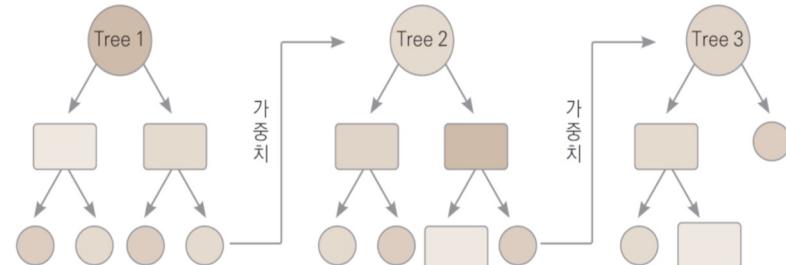
RandomForest

```
1 from sklearn.ensemble import RandomForestClassifier
2 from sklearn.metrics import f1_score
3
4 R_clf = RandomForestClassifier(n_estimators=300, criterion = 'entropy')
5 R_clf.fit(X_train,y_train)
6
7 predict = R_clf.predict(X_test)
8
9 f1 = f1_score(y_test, predict)
10 print('랜덤포레스트 f1-score :',f1)
```

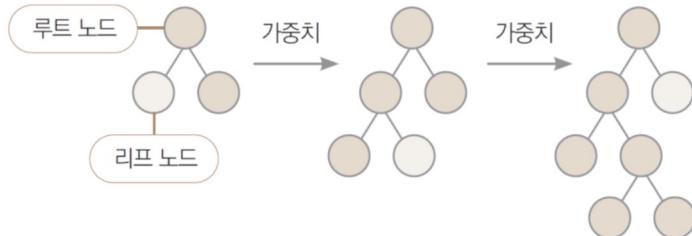
랜덤포레스트 f1-score : 0.8677098150782362

```
1 # RandomizedSearchCV를 사용하여 최적의 하이퍼파라미터 탐색
2 rs_cv = RandomizedSearchCV(estimator=xgb, param_distributions=param_dist,
3                             scoring='accuracy', cv=5, n_iter=10, random_state=42)
4
5 # 데이터에 대해 RandomizedSearchCV 수행
6 rs_cv.fit(X_train, y_train)
7
8 # 최적의 하이퍼파라미터와 최고 정확도 출력
9 print("Best Hyperparameters:", rs_cv.best_params_)
10 print("Best Accuracy:", rs_cv.best_score_)
11
12 # 최적의 하이퍼파라미터로 학습된 모델을 사용하여 예측
13 xgb_best_model = rs_cv.best_estimator_
14 pred = xgb_best_model.predict(X_test)
15
16 # f1-score 평가
17 f1 = f1_score(y_test, pred)
18 print('XGBoost f1-score:', f1)

Best Hyperparameters:
{'colsample_bytree': 0.863635997928104, 'learning_rate': 0.07530815376116708, 'max_depth': 8, 'n_estimators': 826, 'reg_alpha': 0
Best Accuracy: 0.8306485588640738
XGBoost f1-score: 0.8196242171189979
```



Modeling



LightGBM

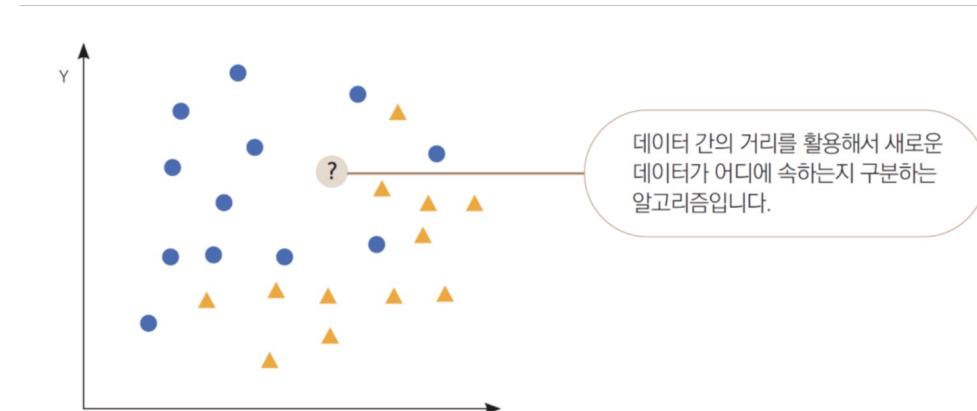
```
1 # RandomizedSearchCV를 사용하여 모델 훈련 및 최적의 하이퍼파라미터 탐색
2 random_search = RandomizedSearchCV(lgbm, param_distributions=param_dist,
3                                     n_iter=10, cv=5, scoring='f1', random_state=42)
4 random_search.fit(X_train, y_train)
5
6 # 최적의 하이퍼파라미터 출력
7 print("Best Parameters:\n", random_search.best_params_)
8
9 # 최적의 모델로 예측 수행
10 lgb_best_model = random_search.best_estimator_
11 pred = lgb_best_model.predict(X_test)
12
13 # f1-score 평가
14 f1 = f1_score(y_test, pred)
15 print("Light GBM f1-score:", f1)

Best Parameters:
{'colsample_bytree': 1.0, 'learning_rate': 0.05, 'max_depth': 9, 'n_estimators': 591, 'num_leaves': 49, 'reg_alpha': 0.1,
Light GBM f1-score: 0.6801248699271593
```

KNN

```
1 from sklearn.neighbors import KNeighborsClassifier
2
3 # KNN 모델 객체 생성
4 knn = KNeighborsClassifier(n_neighbors=1)
5
6 # 모델 학습
7 knn.fit(X_train, y_train)
8
9 # 예측
10 y_pred = knn.predict(X_test)
11
12 # 정확도 평가
13 f1 = f1_score(y_test, pred)
14 print('KNN f1-score:', f1)

KNN f1-score: 0.6801248699271593
```



Modeling

앙상블(Decision tree, Xgboost)

```
1 from sklearn.ensemble import VotingClassifier
2
3 # Decision Tree 객체 생성
4 dt_clf= DecisionTreeClassifier(criterion='entropy')
5
6 # VotingClassifier를 사용하여 앙상블 학습 모델 생성
7 ensemble_clf = VotingClassifier(estimators=[('dt', dt_clf),
8 ('xgb', xgb_best_model)], voting='hard')
9
10 # 앙상블 학습 모델 학습
11 ensemble_clf.fit(X_train, y_train)
12
13 # 앙상블 학습 모델 예측
14 pred = ensemble_clf.predict(X_test)
15
16 # f1-score
17 f1 = f1_score(y_test,pred)
18
19 #교차검증
20 scores = cross_val_score(ensemble_clf, X, y, scoring='f1', cv=5)
21 print('Ensemble f1-score :', np.round(f1,4))

Ensemble f1-score : 0.8307
```

스태킹(Decision tree, Xgboost, LightGBM)

```
1 from sklearn.ensemble import StackingClassifier
2
3 # 기본 분류기 생성
4 estimators = [
5     ('decision_tree', DecisionTreeClassifier()),
6     ('xgboost', xgb_best_model),
7     ('lightgbm', lgb_best_model)
8 ]
9
10 # 최종 모델 생성
11 final_model = DecisionTreeClassifier()
12
13 # 스태킹 모델 생성
14 stacking_model = StackingClassifier(
15     estimators=estimators,
16     final_estimator=final_model,
17     cv=5 # 교차 검증 폴드 수
18 )
19 # 스태킹 모델 학습
20 stacking_model.fit(X_train, y_train)
21 # 스태킹 모델 예측
22 y_pred = stacking_model.predict(X_test)
23 # f1-score 평가
24 f1 = f1_score(y_test, y_pred)
25 print("스태킹 앙상블 f1-score:", f1)
```

스태킹 앙상블 f1-score: 0.7744645107097857

Modeling - 비임직원

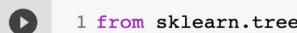
RandomGridSearchCV 최적화

```
1 #랜덤 서치
2 from sklearn.tree import DecisionTreeClassifier
3 from sklearn.model_selection import RandomizedSearchCV
4 from scipy.stats import randint
5
6 # Decision Tree 모델 생성
7 dt_clf = DecisionTreeClassifier()
8
9 # 튜닝할 하이퍼파라미터와 범위 설정
10 param_dist = {
11     'max_depth': [None, 5, 10, 15], # 최대 트리 깊이
12     'min_samples_split': randint(2, 20), # 최소 분할 샘플 수
13     'min_samples_leaf': randint(1, 20), # 리프 노드의 최소 샘플 수
14     'max_features': ['sqrt', 'log2', None], # 최대 특성 수
15     'criterion' : ['gini', 'entropy'] #노드 분할 지표
16 }
17
18 # RandomizedSearchCV를 사용하여 투닝 수행
19 random_search = RandomizedSearchCV(dt_clf, param_distributions=param_dist, scoring='f1', n_iter=10, cv=5)
20 random_search.fit(X_train, y_train)
21
22 # 최적의 하이퍼파라미터와 모델 성능 출력
23 print("Best Hyperparameters:\n", random_search.best_params_)
24 print("Best f1-score: ", random_search.best_score_)
```

```
Best Hyperparameters:
{'criterion': 'gini', 'max_depth': None, 'max_features': 'log2', 'min_samples_leaf': 1, 'min_samples_split': 11}
Best f1-score:  0.8220663461298299
```

<Best Hyperparameters>
criterion : gini
max_depth: None
max_features: log2
min_samples_leaf: 1
min_samples_split: 11

Modeling - 비임직원



```
1 from sklearn.tree import DecisionTreeClassifier
2 from sklearn.metrics import f1_score
3
4 #Decision Tree 모델 생성
5 dt_clf = DecisionTreeClassifier(max_features ='log2',min_samples_leaf=1,
6                                 min_samples_split=11)
7
8 dt_clf.fit(X_train, y_train) #모델 학습
9 y_pred = dt_clf.predict(X_test) #모델 예측
10
11 f1 = f1_score(y_test, y_pred) #f1-score
12
13 print('Decision Tree f1-score :',f1)

Decision Tree f1-score : 0.8395721925133689
```



```
1 from sklearn.tree import DecisionTreeClassifier
2 from sklearn.metrics import f1_score
3
4 #Decision Tree 모델 생성
5 dt_clf = DecisionTreeClassifier()
6
7 dt_clf.fit(X_train, y_train) #모델 학습
8 y_pred = dt_clf.predict(X_test) #모델 예측
9
10 f1 = f1_score(y_test, y_pred) #f1-score
11
12 print('Decision Tree f1-score :',f1)

Decision Tree f1-score : 0.8698140200286123
```

Best Hyperparameters 훈련 결과
F1-score : 0.84

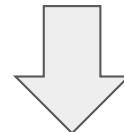
Default parameters 훈련 결과
F1-score : 0.87

Modeling - 비임직 원

```
▶ 1 from sklearn.tree import DecisionTreeClassifier
  2 from sklearn.metrics import f1_score
  3
  4 #Decision Tree 모델 생성
  5 dt_clf = DecisionTreeClassifier(criterion='entropy')
  6
  7 dt_clf.fit(X_train, y_train) #모델 학습
  8 y_pred = dt_clf.predict(X_test) #모델 예측
  9
 10 f1 = f1_score(y_test, y_pred) #f1-score
 11
 12 print('Decision Tree f1-score :',f1)
```

Decision Tree f1-score : 0.8715427166564229

노드 분할 지표 ‘entropy’ 설정 후
F1-score : **0.871**



최종 모델 선정

Modeling-임직

원

RandomGridSearchCV 최적화

```
1 #랜덤 서치
2 from sklearn.tree import DecisionTreeClassifier
3 from sklearn.model_selection import RandomizedSearchCV
4 from scipy.stats import randint
5
6 # Decision Tree 모델 생성
7 dt_clf = DecisionTreeClassifier()
8
9 # 튜닝할 하이퍼파라미터와 범위 설정
10 param_dist = {
11     'max_depth': [None, 5, 10, 15], # 최대 트리 깊이
12     'min_samples_split': randint(2, 20), # 최소 분할 샘플 수
13     'min_samples_leaf': randint(1, 20), # 리프 노드의 최소 샘플 수
14     'max_features': ['sqrt', 'log2', None], # 최대 특성 수
15     'criterion' : ['gini', 'entropy'] #노드 분할 지표
16 }
17
18 # RandomizedSearchCV를 사용하여 훈련 수행
19 random_search = RandomizedSearchCV(dt_clf, param_distributions=param_dist, scoring='f1', n_iter=10, cv=5)
20 random_search.fit(x_train, y_train)
21
22 # 최적의 하이퍼파라미터와 모델 성능 출력
23 print("Best Hyperparameters:\n", random_search.best_params_)
24 print("Best f1-score: ", random_search.best_score_)

Best Hyperparameters:
{'criterion': 'gini', 'max_depth': None, 'max_features': 'sqrt', 'min_samples_leaf': 2, 'min_samples_split': 12}
Best f1-score:  0.8527223782778508
```

<Best Hyperparameters>

criterion : gini

max_depth: None

max_features: sqrt

min_samples_leaf: 2

min_samples_split: 12

Modeling-임직

우

```
[28] 1 from sklearn.tree import DecisionTreeClassifier  
2 from sklearn.metrics import f1_score  
3  
4 #Decision Tree 모델 생성  
5 dt_clf = DecisionTreeClassifier(max_features = 'sqrt', min_samples_leaf=2, min_samples_split=12)  
6  
7 dt_clf.fit(x_train, y_train) #모델 학습  
8 y_pred = dt_clf.predict(x_test) #모델 예측  
9  
10 f1 = f1_score(y_test, y_pred) #f1-score  
11  
12 print('Decision Tree f1-score :',f1)
```

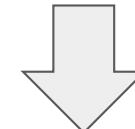
Decision Tree f1-score : 0.8704788846231561

```
[25] 1 from sklearn.tree import DecisionTreeClassifier  
2 from sklearn.metrics import f1_score  
3  
4 #Decision Tree 모델 생성  
5 dt_clf = DecisionTreeClassifier()  
6  
7 dt_clf.fit(x_train, y_train) #모델 학습  
8 y_pred = dt_clf.predict(x_test) #모델 예측  
9  
10 f1 = f1_score(y_test, y_pred) #f1-score  
11  
12 print('Decision Tree f1-score :',f1)
```

Decision Tree f1-score : 0.9054766734279918

Best Hyperparameters 훈련 결과
F1-score : 0.87

Default Parameters 훈련 결과
F1-score : 0.91



최종 모델 선정

적용 방안-타겟 마케팅

- 프라임회원 클래스에서 나타나는 패턴을 분석하여 타겟 마케팅을 제안

Persona
연령, 성격, 관심사, 심리...

Keyword →

30~40대

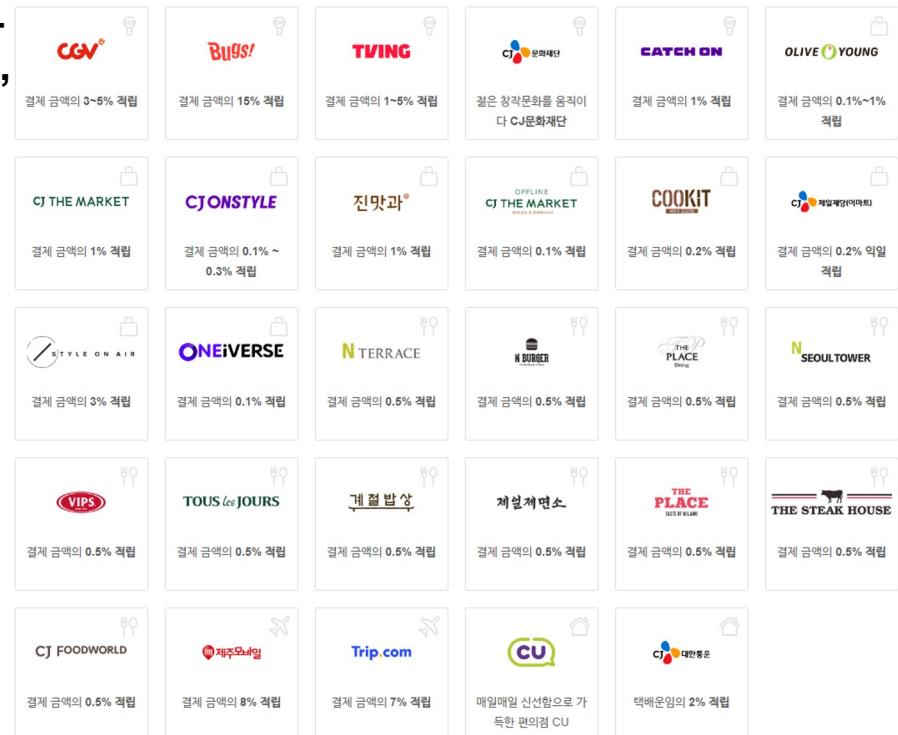
가정

생활품

기존 Cj를 이용하는 고객

적용 방안-타겟 마케팅

2. 기존 CJ one 포인트 제휴 브랜드를 이용하는 고객 중 CJ더마켓과 고객층이 유사한 브랜드를 공략, 홍보

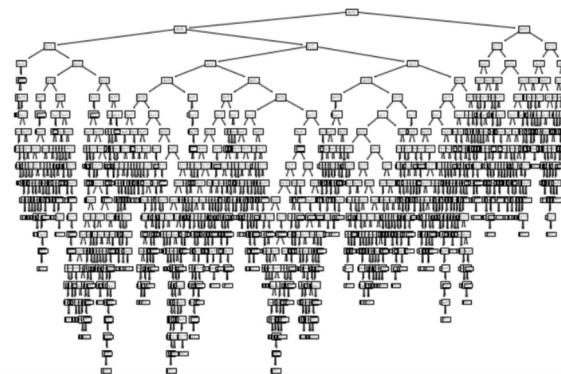
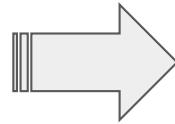


▲CJ one 포인트 제휴 브랜드

보완점

1. 과적합 방지

- Feature의 개수가 3개 밖에 되지 않는 것에 비해 depth가 74
- 전처리 과정 부족
- 피처 선택, 피처 엔지니어링의 부족



과적합 발생

<해결 방안>

- 도메인 지식 활용: 해당 문제 도메인의 전문가들과 협력하여 중요한 피처를 선택하기
- 이상치 탐지
- 피처 중요도 평가: 모델링 알고리즘을 사용하여 각 피처의 중요도를 평가하여 feature selection 진행

보완점

2. 임계값 설정

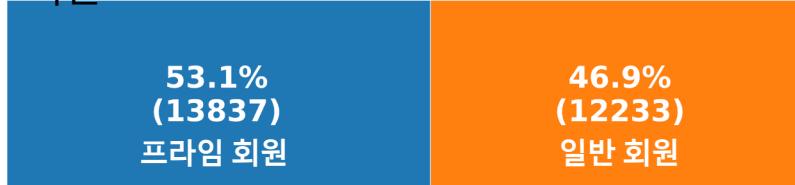
임직원 데이터 셋의 레이블 분포 비율



레이블이 프라임 회원 쪽으로 치우친
데이터 불균형 현상

→ 모델이 프라임 회원쪽으로 예측하는 경향성을 띠게 됨

비임직원 데이터 셋의 레이블 분포 비율



<해결방안>

높은 F1-score를 내는 방향으로 모델의 **임계값을 조정하여**

데이터 불균형 현상 해결,
정밀도와 재현율 사이의 균형을 맞춰주는 작업이 필요함

Thank You