

4장 분류

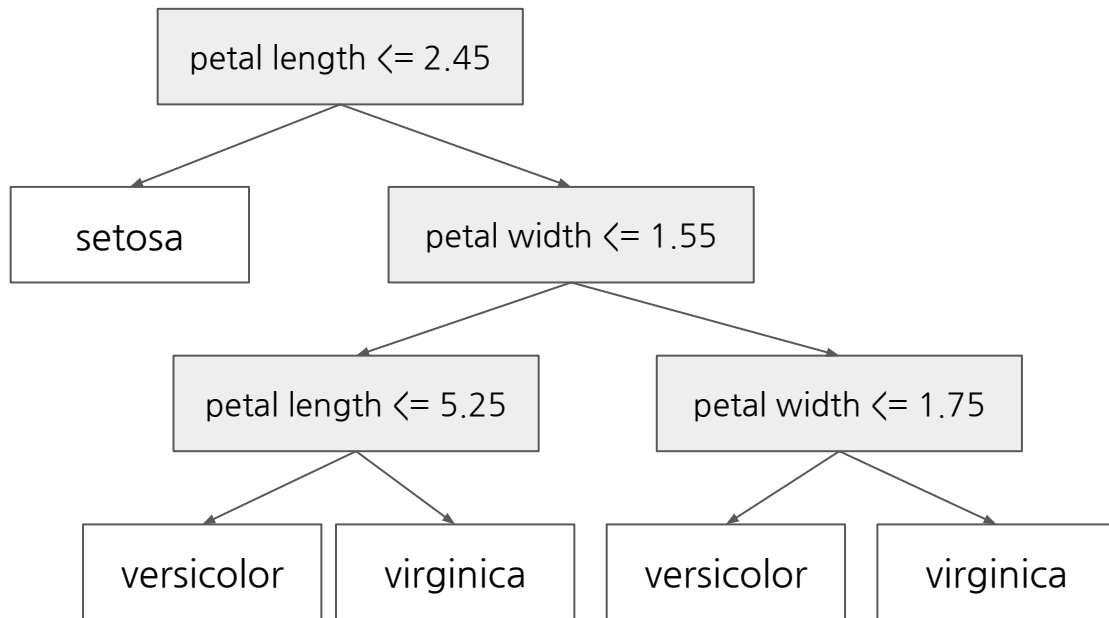
신림프로그래머 최범균

분류(classification)

- 분류
 - 대표적인 지도학습
 - 학습 데이터로 주어진 데이터의 피처와 레이블값(결정 값, 클래스 값)을 학습해 모델 생성
 - 모델에 새로운 데이터 값이 주어졌을 때 레이블 값을 예측
- 책에서는 다양한 분류 머신러닝 알고리즘 중 앙상블 방법 소개

결정 트리(Decision Tree)

- 규칙 노드(Decision Node), 리프 노드(Leaf Node)로 구성
 - 규칙 노드 : 하위 트리를 분할하는 규칙
 - 리프 노드 : 결정 값
- 직관적으로 이해하기 쉬움



결정 트리

- 결정 노드는 최대한 균일한 데이터 세트를 구성할 수 있도록 분할
 - 정보 이득(Information Gain)이나 지니 계수(Gini coefficient) 이용해서 분리 기준 선택
- 과적합 방지 위해 트리 크기 제한 (괄호는 DecisionTreeClassifier의 파라미터)
 - 노드 분할 위한 최소 샘플 데이터 수(min_samples_split) : 작을수록 분할되는 노드가 많아져 과적합 가능성 증가
 - 말단 노드가 되기 위한 최소 샘플 데이터 수(min_samples_leaf) : 작을수록 말단 노드 증가로 과적합 가능성 증가 (단, 레이블 데이터 분포가 비대칭이면 작게 설정 필요)
 - 최적 분할 위해 고려할 최대 피처 개수(max_features)
 - 트리의 최대 깊이(max_depth) : 깊이가 깊어질수록 과적합 가능성 증가
 - 말단 노드의 최대 개수(max_leaf_nodes)

UCI 사용자 행동 데이터 예

- 트리 깊이, 최소 샘플 데이터 수에 따른 정확도 차이
 - 실제로는 다른 파라미터도 조합해서 사용

```
# 학습 데이터로 학습
params = {
    'max_depth': [8, 12, 16, 20],
    'min_samples_split': [16, 24]
}
grid_cv = GridSearchCV(dt_clf, param_grid=params,
                       scoring='accuracy', cv=5)
grid_cv.fit(X_train, y_train)
cv_results_df = pd.DataFrame(grid_cv.cv_results_)
cv_results_df.sort_values(by='rank_test_score',
                          inplace=True)
```

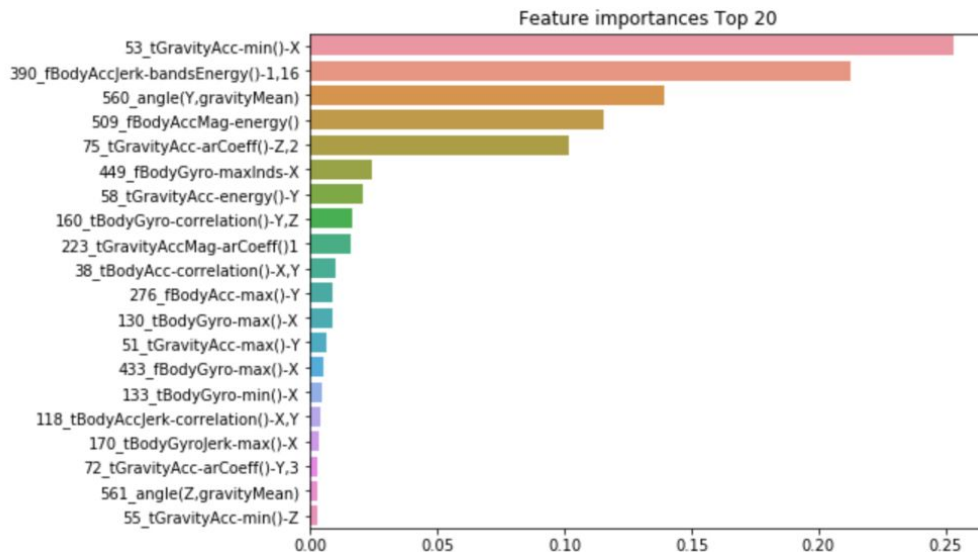
```
# 최적 모델로 예측
best_dt_clf = grid_cv.best_estimator_
pred1 = best_dt_clf.predict(X_test)
accuracy = accuracy_score(y_test, pred1) # 0.8717
```

	param_max_depth	param_min_samples_split	mean_test_score	rank_test_score
0	8	16	0.855005	1
1	8	24	0.851469	2
5	16	24	0.849565	3
7	20	24	0.849293	4
6	20	16	0.848749	5
4	16	16	0.847252	6
3	12	24	0.846300	7
2	12	16	0.845892	8

UCI 사용자 행동 데이터 예

- 561개 피처 중 결정 트리 생성에 중요한 피처 확인

```
ftr_importances_values = best_df_clf.feature_importances_  
ftr_importances = pd.Series(ftr_importances_values, index=X_train.columns)  
ftr_top20 = ftr_importances.sort_values(ascending=False)[:20]  
plt.figure(figsize=(8, 6))  
plt.title('Feature importances Top 20')  
sns.barplot(x=ftr_top20, y=ftr_top20.index)  
plt.show()
```



앙상블(ensemble) 학습

- 여러 개의 분류기를 생성하고 그 예측을 결합해서 예측 도출
 - 보통 단일 분류기보다 성능이 좋음
- 유형
 - 보팅(Voting)
 - 일반적으로 서로 다른 다른 알고리즘을 가진 분류기를 결합
 - 배깅(Bagging)
 - 같은 알고리즘 기반 분류기를 사용
 - 분류기마다 데이터 샘플링을 서로 다르게 가져가 학습
 - 랜덤 포레스트가 대표적
 - 부스팅(Boosting)
 - 여러 분류기가 순차적으로 학습 수행
 - 앞서 학습한 분류기가 예측이 틀린 데이터에 대해 다음 분류기는 올바르게 예측할 수 있게 가중치를 부여
 - GBM, XgBoost, LightGBM 등

앙상블 학습 : 투표 유형

- 하드 보팅
 - 다수결 원칙과 비슷
 - 다수 분류기가 결정한 예측값을 최종 결괏값으로 선정
- 소프트 보팅
 - 각 분류기의 레이블 값 결정 확률을 평균해서 확률이 가장 높은 레이블 값을 최종 결괏값으로 선정
 - 일반적으로 소프트 보팅이 성능은 좋다고 함

랜덤 포레스트(Random Forest)

- 기반 알고리즘은 결정 트리
- 배깅의 대표적인 알고리즘으로 비교적 수행 속도가 빠름
- 전체 데이터에서 일부가 중첩되게 샘플링된 데이터 세트를 개별 트리에 적용
 - 부트스트래핑(bootstrapping)
- 주요 하이퍼 파라미터
 - n_estimators : 결정 트리 개수
 - 결정 트리와 동일 파라미터 (max_features의 값은 'auto' 즉 'sqrt' 사용)
 - 결정 트리의 max_features 기본 값은 None

GBM(Gradient Boosting Machine)

- 부스팅 알고리즘
- 경사 하강법(Gradient Descent)을 이용해서 가중치 업데이트
- 단점 : 수행 시간이 오래 걸리고 하이퍼 파라미터 튜닝 노력이 더 필요
- 하이퍼 파라미터 (결정 트리 파라미터 포함)
 - loss : 경사 하강법에서 사용할 비용 함수. 기본값은 'deviance'
 - learning_rate : 오류 값을 보정할 때 적용할 계수(학습률). 기본값은 0.1
 - n_estimators : weak learner의 개수
 - subsample : 학습에 사용하는 데이터 샘플링 비율로 기본값은 1. 과적합이 염려되는 경우 1보다 작은 값으로 설정

XGBoost(eXtra Gradient Boost)

- GBM에 기반하지만 GBM의 단점을 해결
 - 병렬 CPU 환경에서 병렬 학습이 가능
- 주요 장점
 - 예측 성능이 뛰어남
 - GBM 대비 빠른 수행 시간(다른 알고리즘보다 빠르다는 것은 아님)
 - 과적합 규제(regularization) : 과적합에 좀 더 강한 내구성
 - 나무 가지치기(Tree pruning) : 이득이 없는 분할을 가지치기 해서 분할 수를 줄임
 - 자체 내장 교차 검증 : 반복 수행 시마다 내부적으로 교차 검증 수행
 - 조기 중단 : 평가 값이 최적화 되면 반복을 중간에 멈춤
 - 결손값 자체 처리

LightGBM

- 리프 중심(Leaf Wise) 트리 분할 방식 사용
 - 최대 손실 값(max delta loss)을 가지는 리프 노드를 지속 분할
- XGBoost 대비 장점
 - 더 빠른 학습과 예측 수행 시간
 - 더 작은 메모리 사용량
 - 카테고리형 피처의 자동 변환과 최적 분할

산탄데르 은행 고객 만족 데이터 탐색/가공 예

- 타겟이 한 쪽으로(만족에) 치우침 → 정확도보다는 ROC-AUC로 성능 평가

```
# 데이터 구성  
cust_df.info()
```

```
<class 'pandas.core.frame.DataFrame'>  
RangeIndex: 76020 entries, 0 to 76019  
Columns: 371 entries, ID to TARGET  
dtypes: float64(111), int64(260)  
memory usage: 215.2 MB
```

```
# 만족 불만족 비율  
print(cust_df['TARGET'].value_counts())  
unsatisfied_cnt = cust_df[cust_df['TARGET'] == 1].loc[:, 'TARGET'].count()  
total_cnt = cust_df.TARGET.count()  
print('불만족 비율: {0:.2f}'.format((unsatisfied_cnt / total_cnt)))
```

```
0    73012  
1     3008  
Name: TARGET, dtype: int64  
불만족 비율: 0.04
```

산탄데르 은행 고객 만족 데이터 탐색/가공 예

- 데이터 분포 분석
 - `cust_df.describe()`

	ID	var3	var15	imp_ent_var16_ult1	imp_op_var39_comer_ult1	imp_op_var39_comer_ult3
count	76020.000000	76020.000000	76020.000000	76020.000000	76020.000000	76020.000000
mean	75964.050723	-1523.199277	33.212865	86.208265	72.363067	119.529632
std	43781.947379	39033.462364	12.956486	1614.757313	339.315831	546.266294
min	1.000000	-999999.000000	5.000000	0.000000	0.000000	0.000000
25%	38104.750000	2.000000	23.000000	0.000000	0.000000	0.000000
50%	76043.000000	2.000000	28.000000	0.000000	0.000000	0.000000
75%	113748.750000	2.000000	40.000000	0.000000	0.000000	0.000000
max	151838.000000	238.000000	105.000000	210000.000000	12888.030000	21024.810000

-999999를 가장 많은 값인 2로 변경

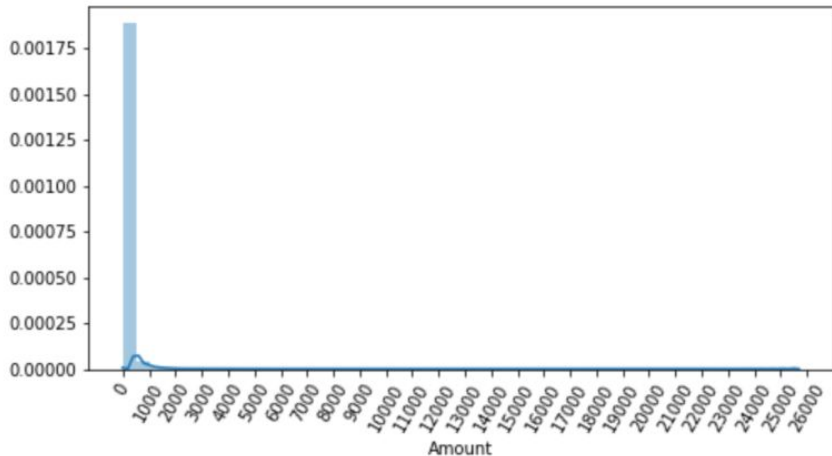
→ `cust_df['var3'].replace(-999999, 2, inplace=True)`

캐글 신용카드 사기 검출 데이터 탐색/가공 예

- 왜곡된 데이터 분포
 - Amount가 1K불 이하에 몰려 있음
- 로그 변환으로 데이터 처리
 - 왜곡 감소

```
import seaborn as sns
plt.figure(figsize=(8,4))
plt.xticks(range(0, 30000, 1000), rotation=60)
sns.distplot(card_df['Amount'])
```

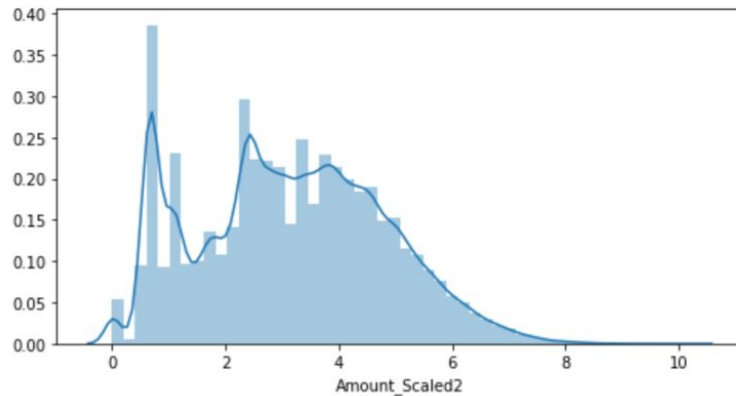
<matplotlib.axes._subplots.AxesSubplot at 0x191c2d0e188>



```
amount_n = np.log1p(card_df['Amount'])
card_df['Amount_Scaled2'] = amount_n
```

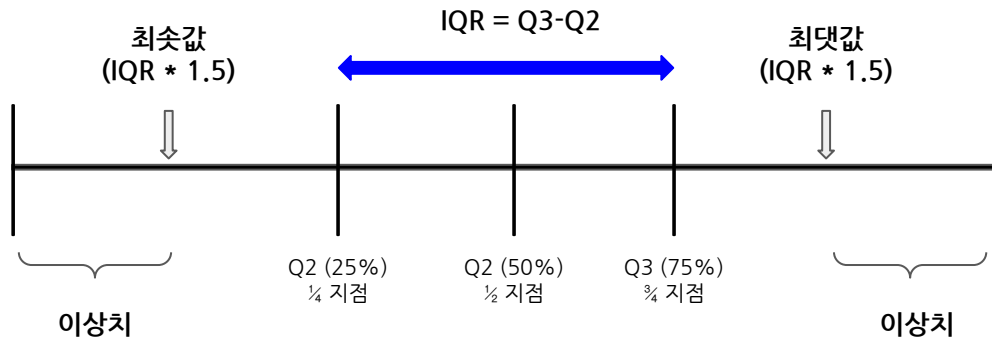
```
plt.figure(figsize=(8,4))
# plt.xticks(range(-2, 2, 1))
sns.distplot(card_df['Amount_Scaled2'])
```

<matplotlib.axes._subplots.AxesSubplot at 0x256dfbf0608>



캐글 신용카드 사기 검출 데이터 탐색/가공 예


- 이상치(outlier) : 전체 데이터 패턴에서 벗어난 이상 값을 가진 데이터
 - 모델 성능에 영향
- 이상치 제거로 성능 향상
 - 결정값과 상관성이 높은 피처에 대해 이상치 제거
- IQR(Inter Quantile Range) 방식 : 사분위 값의 변차를 이용하는 기법
 - IQR에 1.5를 곱해 생성된 범위를 이용해 최댓값과 최솟값을 결정
 - 최댓값을 초과하거나 최솟값에 미달하는 데이터를 이상치로 간주



캐글 신용카드 사기 검출 데이터 탐색/가공 예

- 극도로 불균형한 레이블 값 분포
 - 비사기(0)에 극단적으로 몰려 있음

```
print(card_df['Class'].value_counts())
```



```
0    284315  
1      492  
Name: Class, dtype: int64
```

- 레이블 값 분포로 인한 문제점을 해결하기 위해 적절한 학습 데이터를 확보하는 방안 필요

- 오버샘플링(Oversampling)
 - 적은 데이터 세트를 증식
 - 원본 데이터의 피처를 약간 변경하여 증식해서 과적합 방지
- 언더샘플링(Undersampling)
 - 많은 데이터를 적은 데이터 세트 수준으로 감소
 - 많은 정상 레이블 데이터를 감소시켜 정상 레이블의 경우 오히려 학습이 제대로 되지 않는 단점
- 책에서는 SMOTE(Synthetic Minority Over-sampling Technique) 기법 사용한 예시 제공