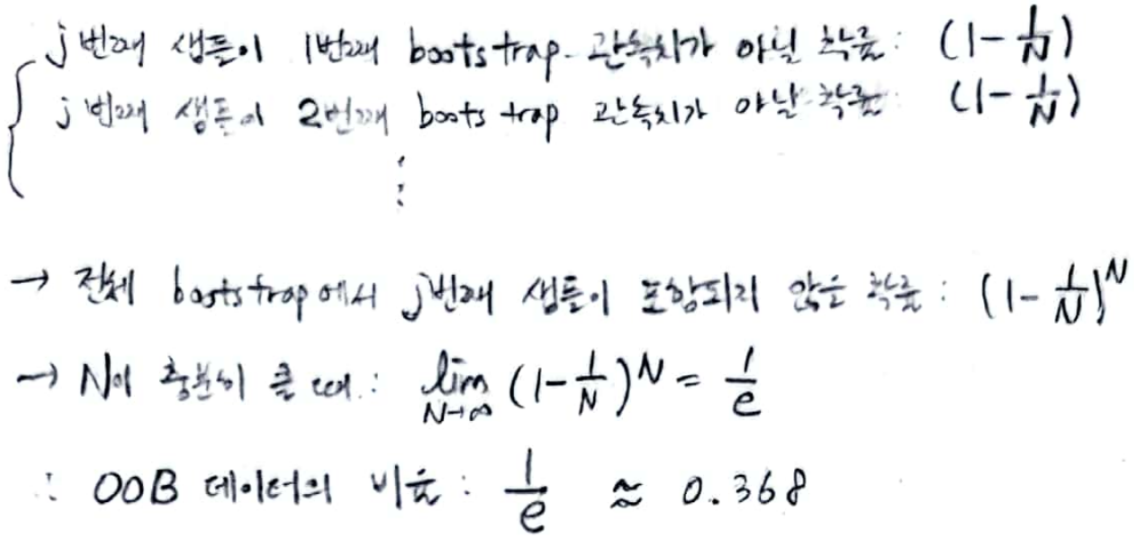
BDA
Machine Learning Study

Assignment #2

7 팀

박지수, 김인서, 박지영, 박혜원, 이주미

1. . N 개의 샘플로 구성된 원 데이터에서 N 개의 부트스트래핑 샘플을 생성한다고 해보자. N 이 충분히 클 때 OOB 데이터의 비율을 구해보시오.



2. 6 장 되새김 문제 2 번

```
    error = 0
    for i in range(len(X_test)):
        row = X_test.iloc[i]
        ind = 1
        node = tree[ind]
        while isinstance(node['struct'], pd.DataFrame):
            if row[node['col']] < node['val']: ind = ind << 1
            else: ind = (ind << 1) + 1
            node = tree[ind]

        y_pred = node['struct']
        error += np.abs(y_pred - y_test.iloc[i])
    print(f'테스트 데이터셋 MAE:{error / len(y_test): .2f}')
```

```
4]  ✓  2.9s
```

```
학습 데이터셋 MAE: 40.32
테스트 데이터셋 MAE: 45.49
```

```
    from sklearn.tree import DecisionTreeRegressor

    reg = DecisionTreeRegressor(random_state=1234,
                                max_depth=4,
                                min_samples_split=4).fit(X_train, y_train)

    y_pred_train = reg.predict(X_train)
    print(f'학습 데이터셋 MAE:{(np.abs(y_pred_train - y_train)).mean(): .2f}')

    y_pred = reg.predict(X_test)
    print(f'테스트 데이터셋 MAE:{(np.abs(y_pred - y_test)).mean(): .2f}')
```

```
5]  ✓  0.0s
```

```
학습 데이터셋 MAE: 39.30
테스트 데이터셋 MAE: 45.67
```

3. 그레이디언트 부스팅 모델은 과적합에 취약하다. 그 이유를 제시하고, 과적합을 방지하는 규제 기법에 대해서 서술해보시오.

현실데이터에서는 ε로 표현되는 noise가 섞여있으나, 그레이디언트 부스팅 모델은 ε까지 학습해버려서 과적합 이슈가 발생한다. ($y = f(x) + \varepsilon$)

규제 기법

1. Subsampling : 각 iteration 마다 전체 데이터에서 일부만 랜덤하게 sampling.
2. Shrinkage: 원래 만들어진 모델의 모든 가중치는 1이지만, 후반부에 만들어진 모델에 대한 가중치를 줄여줌.
3. Early stopping : validation data의 에러가 증가한 것 같으면 학습 중단.

4. XGBoost, LightGBM, Catboost 모델의 특징을 서술하시오.

XGBoost : 학습을 위한 Gradient Boosting 목적식에 규제항을 추가하여 과적합 방지.

LightGBM : XGBoost와는 다르게 Tree의 뿌리를 조금 더 깊게 내릴 수 있음.
뿌리를 leaf 단위로 학습, XGBoost에 비해 속도가 빠르며, GPU 지원.

CatBoost : Unbiased boosting with categorical features
분산과 편향을 모두 줄이는 것을 목표로 만들어진 모델
특히, categorical feature에 강한 모델.
(one-hot encoding 이외엔 수치형으로 변환하는 방법 사용)

5. 7 장 되새김 문제 3 번

Q5

```
from sklearn.datasets import load_diabetes
from sklearn.model_selection import train_test_split

diabetes = load_diabetes(as_frame=True)
X_train, X_val, y_train, y_val = train_test_split(diabetes.data,
                                                  diabetes.target,
                                                  random_state=0)
```

[7] ✓ 0.0s

```
from sklearn.linear_model import Ridge
from sklearn.inspection import permutation_importance


model = Ridge(alpha=1e-2).fit(X_train, y_train)

pi = permutation_importance(model,
                            X_val,
                            y_val,
                            n_repeats=30,
                            random_state=0,
                            scoring='neg_mean_squared_error')

pi_series = pd.Series(pi.importances_mean, index=X_train.columns)
print(f'퍼뮤테이션기반피처별 중요도: {pi_series.sort_values(ascending=False).values[:3]}')
```

[8] ✓ 0.2s

```
퍼뮤테이션기반피처별 중요도: [1013.90265117  872.69427744  438.68103665]
```

6. 8 장 되새김 문제 2 번

```
clf = GradientBoostingClassifier(random_state=1234)
y_pred = clf.fit(X_train, y_train).predict(X_test)

print(f'전체 피처를 사용한 GBT 모델의 정확도: {(y_pred == y_test).mean()*100:.2f}%')

clf = GradientBoostingClassifier(random_state=1234)
selector = RFE(clf, n_features_to_select=20, step=1)
selector = selector.fit(X_train, y_train)
selector.support_[:10]

X_train3 = X_train.iloc[:,selector.support_]
X_test3 = X_test.iloc[:,selector.support_]

clf = GradientBoostingClassifier(random_state=1234)

y_pred = clf.fit(X_train3, y_train).predict(X_test3)

print(f'RFE클래스기반 후진 소거법을 적용한 GBT 모델의 정확도: {(y_pred == y_test).mean()*100:.2f}%')
```

⊗ 0.0s