Ensemble methods

TP 4 – Gradient Boosting Machines

1. We are using DecisionTreeRegressor from scikit-learn to build trees which helps us just focus on the gradient boosting algorithm itself instead of the tree algorithm. We are imitating scikit-learn style implementation where you train the model with fit method and make predictions with predict method.

```
class CustomGradientBoostingRegressor:

    def __init__(self, learning_rate, n_estimators, max_depth=1):
        self.learning_rate = learning_rate
        self.n_estimators = n_estimators
        self.max_depth = max_depth
        self.trees = []

    def fit(self, X, y):

        #initialize to mean values in the 1st iteration
        self.F0 = y.mean()
        Fm = self.F0

        for _ in range(self.n_estimators):

            #residu calculation
            r = y - Fm

            #Fit decision tree on the residu
            tree = DecisionTreeRegressor(max_depth=self.max_depth, random_state=0)
            tree.fit(X, r)

            #Gamma calculation, to update F
            gamma = tree.predict(X)
            Fm += self.learning_rate * gamma
            self.trees.append(tree)

    def predict(self, X):

        Fm = self.F0

        for i in range(self.n_estimators):
            Fm += self.learning_rate * self.trees[i].predict(X)

        return Fm
```

2. Compare with scikit learn implementation : we are checking if our CustomGradientBoostingRegressor performs as the same as GradientBoostingRegressor from scikit-learn by looking at their RMSE on our data.

```
from sklearn.ensemble import GradientBoostingRegressor
from sklearn.metrics import mean_squared_error
from sklearn.tree import DecisionTreeRegressor
import pandas as pd
```

3. Prepare titanic data for the comparison

```
#reading titanic data again
df = pd.read_csv('./data/titanic.csv')

#preparing dataset for training
string_list = [each_string.lower() for each_string in df.columns]
df.columns = string_list
df.dropna(inplace=True)

#target variable
y = df['survived']

#binarize columns
X = df[['pclass', 'sex', 'age','embarked']].copy()

#using sklearn to binarize
from sklearn import preprocessing
lb = preprocessing.LabelBinarizer()

#transforming sex
X['sex'] = lb.fit_transform(X['sex'])
X['embarked'] = lb.fit_transform(X['embarked'])
```

4. Running both models and comparing, with a pre defined hyperparameter values

```
# define hyper parameters:
n_estimators = 20
learning_rate = 0.1
max_depth = 1


#call to our custom made function
custom_gbm = CustomGradientBoostingRegressor(
    n_estimators=n_estimators,
    learning_rate=learning_rate,
    max_depth=max_depth
)
custom_gbm.fit(X, y)
custom_gbm_rmse = mean_squared_error(y, custom_gbm.predict(X), squared=False)
print(f"Custom GBM RMSE:{custom_gbm_rmse:.15f}")

#call to scikit learn model
sklearn_gbm = GradientBoostingRegressor(
    n_estimators=n_estimators,
    learning_rate=learning_rate,
    max_depth=max_depth
)
sklearn_gbm.fit(X, y)

#comparing both results
sklearn_gbm_rmse = mean_squared_error(y, sklearn_gbm.predict(X), squared=False)
print(f"Scikit-learn GBM RMSE:{sklearn_gbm_rmse:.15f}")
```

- Try different combinasion of hyperparameters and retrain RMSE,
  N_estimators = [200, 2000]
  learning_rate = [0.01 , 1]
  max_depth = [3, 5]
- What happen when increasing/decreasing one or multiple hyperparaters ?