Implement Uplift Decision Tree with DeltaDeltaP split

Expected time: 2 - 4hrs, line count: 150 - 200 lines of code with doc strings

Link to job description: https://www.linkedin.com/jobs/view/3128349142

Interview Process

1. Take home (2-4hrs, 150-200 lines of code)
2. 45 mins Hiring Manager interview
3. 90 mins panel interview (python, SQL, ML theory)
4. 30 mins HR friendly behavioral

We can move fast so all could be done in 5-10 days. Offer is within a few days after the final round.

Take Home

You need to implement uplift-tree with DeltaDeltaP split criterion.

To measure uplift we calculate the difference between average of the target variable in treatment and control groups (***M***). The best split maximizes the difference between uplift measure on the left and right. ***difference(M_left, M_right)***

Hot to grow a tree:

1. Create root with all values
2. Recursively run function Build:

```
Build(node):
   If node is at max depth, stop.

   For feature in all features:
      For value in threshold values:
         make_split(feature, value) -> data_left, data_right
         If number of values in a split is lower than the parameter value
            Continue (do not consider this split)
         Calculate delte_delta_p

   Select the best split
   Create node_left, node_right
   Build(node_left)
   Build(node_right)
```

Please note that model construction and predictions are deterministic.

You will need to implement class UpliftTreeRegressor that must have 3 methods:

```
def __init__(
      self,
      Max_depth: int =3,  # max tree depth
      Min_samples_leaf: int = 1000, # min number of values in leaf
      Min_samples_leaf_treated: int = 300, # min number of treatment values in leaf
      Min_samples_leaf_control: int = 300,  # min number of control values in leaf)
      :
      # do something

def fit(
      self,
      X: np.ndarray, # (n * k) array with features
      Treatment: np.ndarray,  # (n) array with treatment flag
      Y: np.ndarray  # (n) array with the target
) -> None:

      # fit the model
def predict(self, X: np.ndarray) -> Iterable(float):
      # compute predictions
      return predictions
```

IMPORTANT:

Threshold algorithm:

```
# column_values - 1d array with the feature values in current node
# threshold_options - threshold values to go over to find the best one

import numpy as np
unique_values = np.unique(column_values)
If len(unique_values) >10:
      percentiles = np.percentile(column_values, [3, 5, 10, 20, 30, 50, 70, 80, 90,
95, 97])
else:
      percentiles = np.percentile(unique_values, [10, 50, 90])
threshold_options = np.unique(percentiles)
```

If f <= threshold, then values is on the left, on the right otherwise.

We will test if your predictions are within EPS range from ours.

Allowed Imports:

```
import numpy as np
```

```
import pandas as pd
from typing import List, Tuple, Ane  # and other as required
```

Assumptions:

len(x) <= 1000000
Number of feats <= 10
Feat values - floats
Treatment flag values {0, 1}
Target variable values - foat

Files:

Location: https://github.com/sibmike/uplift/tree/main/uplift

Example_x.npy - features X
Example_treatment.npy - treatment flag
Example_y.npy - target variable

```
param_dict = {
        max_depth=3,
        min_samples_leaf = 6000,
        min_samples_leaf_trated=2500,
        min_samples_leaf_control=2500
}
```

Example preds.npy - prediction examples
Example_tree.txt - text description of the tree

Submission:

1. Create submission.py and upload it to your github
2. Reply with the link to your solution **by June 29, 11:59PM Pacific Day Time**
3. Earlier submissions have higher chance of consideration
4. Late submissions are ok, but we can't promise they will be reviewed.
5. Criterias: results accuracy & code quality.