Implementation of Weak Learners

In our program, we first developed 2 weak classifiers `linear_clsfer_fisher` and `tree_clsfer` in `weak_classifiers.py`, which are fisher linear classifier and CART tree classifier with `depth = 2`. In `test.py`, there accuracy on the test set are computed as below:

```
accuracy of linear classifier: 0.5262975778546712
accuracy of tree classifier: 0.611764705882353
```

herein, we set the partition of the training set as 70% and test set is the 30% left.

Note that the linear fisher classifier is a 2-class classifier, in the boosting program, it would cause severe error, we would replace it with the multiclass gaussian classifier `linear_fisher_gaussian_kclass` in our boosting program. The accuracy of the multi-class classifier is

```
accuracy of linear classifier: 0.67058823529411764
```

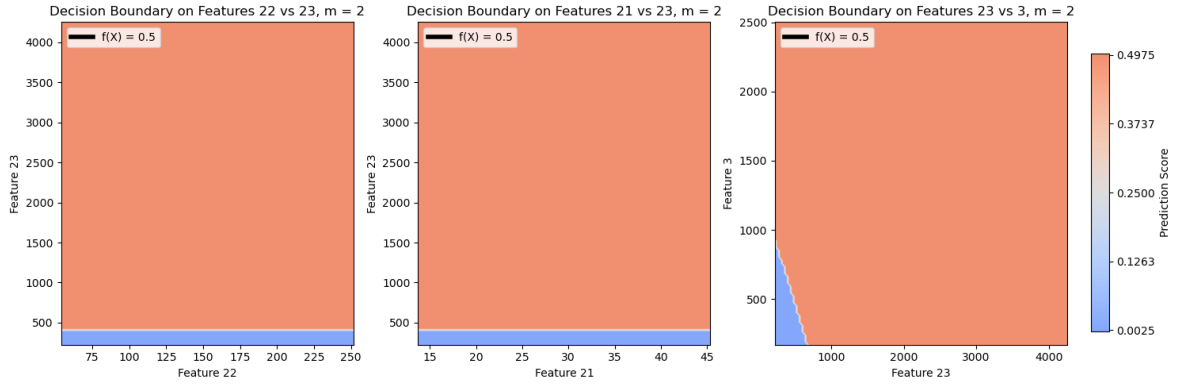
Implement Boosting Algorithms

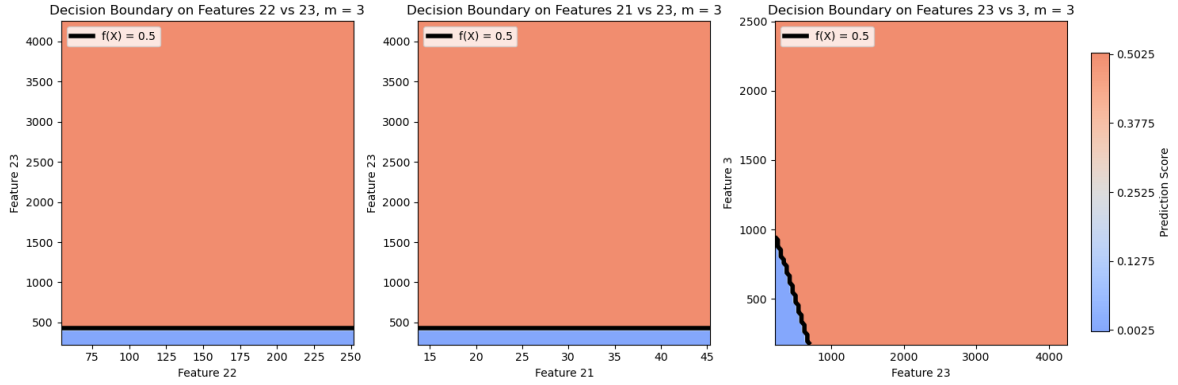
For M in range $(1, 10)$, the accuracy of the boosting model is listed as below:

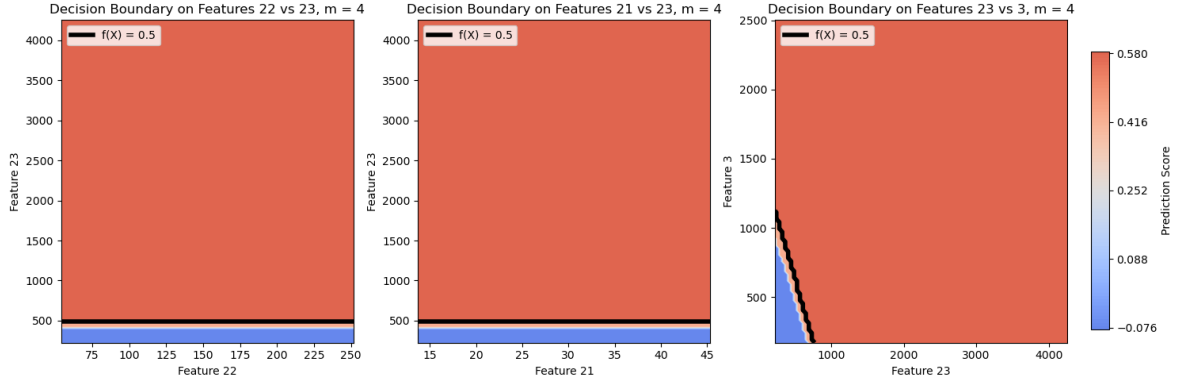
```
m = 1, accuracy = 0.611764705882353
m = 2, accuracy = 0.611764705882353
m = 3, accuracy = 0.67058823529411764
m = 4, accuracy = 0.7823529411764706
m = 5, accuracy = 0.7823529411764706
m = 6, accuracy = 0.8176470588235294
m = 7, accuracy = 0.8176470588235294
m = 8, accuracy = 0.8176470588235294
m = 9, accuracy = 0.8176470588235294
m = 10, accuracy = 0.8176470588235294
```

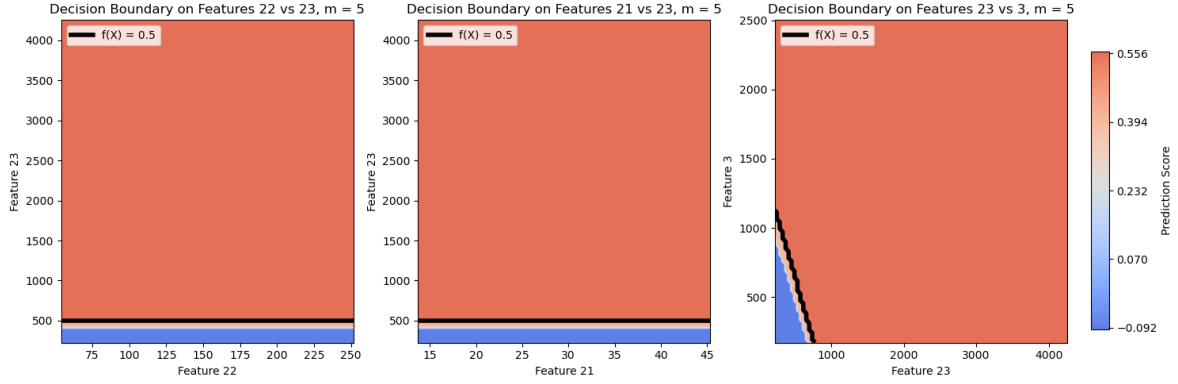
The train-test split is as the first problem (70%-30% split). From the accuracy variation we could know that the estimation ability converges when $M = 6$, with the ultimate accuracy `0.8176470588235294`. Numerical experiment shows that it does not increase any more no matter the number of M.

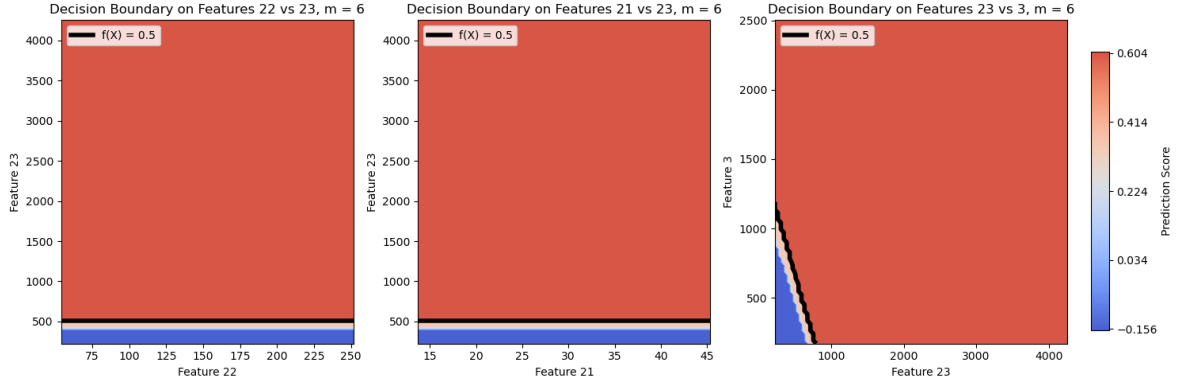
Using function `det_main_feature`, we determined the main features, and plotted the decision boundary on them at different m values as below. Herein the black line is the decision boundary where $F(x) = 0.5$.

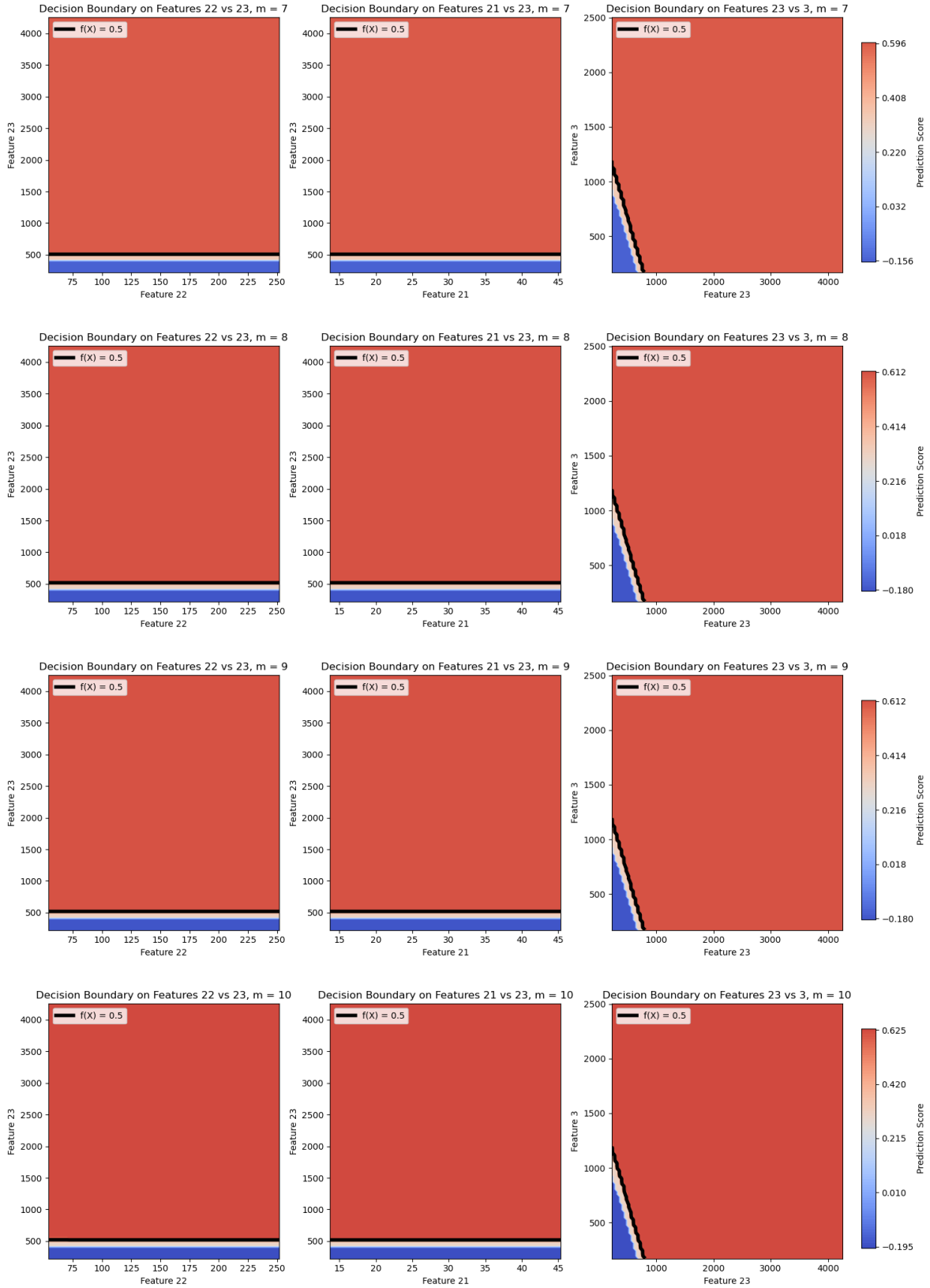

Decision Boundary on Features 22 vs 23, m = 2
Decision Boundary on Features 21 vs 23, m = 2
Decision Boundary on Features 23 vs 3, m = 2
f(X) = 0.5
Feature 22
Feature 21
Feature 23
Feature 23
Feature 3
Prediction Score
0.4975
0.3737
0.2500
0.1263
0.0025


Decision Boundary on Features 22 vs 23, m = 3
Decision Boundary on Features 21 vs 23, m = 3
Decision Boundary on Features 23 vs 3, m = 3
f(X) = 0.5
Feature 22
Feature 21
Feature 23
Feature 23
Feature 3
Prediction Score
0.5025
0.3775
0.2525
0.1275
0.0025


Decision Boundary on Features 22 vs 23, m = 4
Decision Boundary on Features 21 vs 23, m = 4
Decision Boundary on Features 23 vs 3, m = 4
f(X) = 0.5
Feature 22
Feature 21
Feature 23
Feature 23
Feature 3
Prediction Score
0.580
0.416
0.252
0.088
−0.076


Decision Boundary on Features 22 vs 23, m = 5
Decision Boundary on Features 21 vs 23, m = 5
Decision Boundary on Features 23 vs 3, m = 5
f(X) = 0.5
Feature 22
Feature 21
Feature 23
Feature 23
Feature 3
Prediction Score
0.556
0.394
0.232
0.070
−0.092

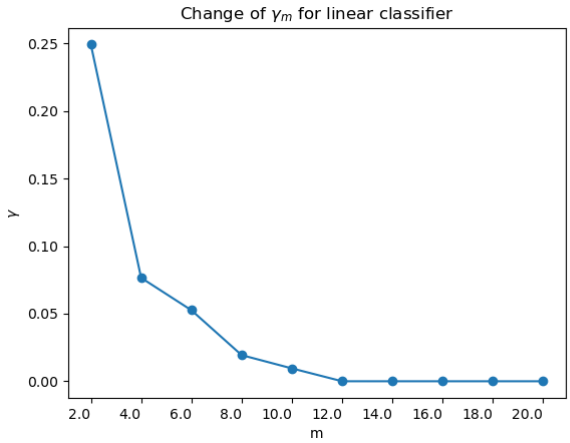

Decision Boundary on Features 22 vs 23, m = 6
Decision Boundary on Features 21 vs 23, m = 6
Decision Boundary on Features 23 vs 3, m = 6
f(X) = 0.5
Feature 22
Feature 21
Feature 23
Feature 23
Feature 3
Prediction Score
0.604
0.414
0.224
0.034
−0.156

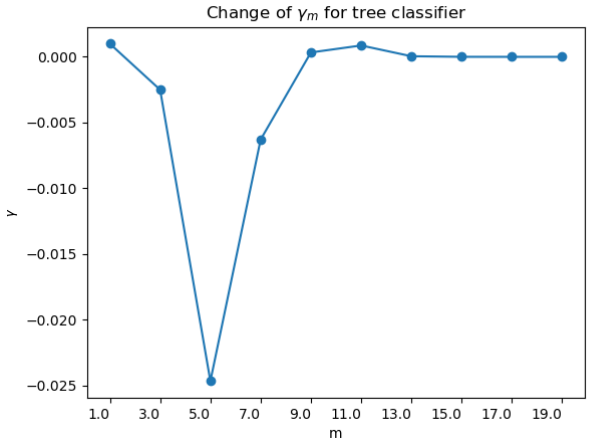


We could see from the results that the decision boundary begins saturating at $m = 4$, and when $m = 6$, the saturation nearly ended. In plotting, we set the values of the features that are not in consideration as the average value over the dataset.

Analysis and Ablation

Taking the number of classifiers $M = 20$, where the boosting model has been converged, we plotted the γ_m for linear and tree classifiers below respectively:





We can know from the figures that the γ_m for linear classifier monotonically goes down as the iteration number increases, and finally converged to 0. However, the γ_m for tree classifier goes down and then up sharply, but tends to 0 as the iteration number is large enough as well.

The method I used for determining the γ_m is gradient descent. The expression of γ_m is that

$$\gamma_m = \arg\min_{\gamma} \sum_{i=1}^{N} L(y_i, F_{m-1} + \gamma h_m(x_i)) \tag{1}$$

In our program, the loss function L was set as L2 loss, and the gradient w.r.t. γ can be expressed as

$$\nabla_\gamma L(x) = \sum_{i=1}^{N} \frac{\partial L(y_i, F(x_i))}{\partial F(x_i)} \cdot h_m(x_i) \tag{2}$$

And for each iteration `j`, the value of γ_m is updated by

$$(\gamma_m)_{j+1} = (\gamma_m)_j - \alpha \nabla_\gamma L(x) \tag{3}$$

In the function used to determine γ_m `min_gamma`, the default value of learning rate `alpha` was set as `0.05`, and the default value of maximum iteration was set as `50`. By the means above, we obtained the γ_m values for gradient boosting.

The intuition of step 7 is that in raw gradient boosting model, the classifiers are trained to satisfy the following expression

$$\gamma_m, h_m = \arg\min_{(\gamma_m, h_m)} \sum_{i=1}^{N} L(y_i, F_{m-1}(x_i) + \gamma_m h_m(x_i)) \tag{4}$$

According to gradient descent, h_m should fit the minus gradient of the expression, i.e.

$$h_m(x_i) = -\frac{\partial L(y_i, F_{m-1}(x_i))}{\partial F_{m-1}(x_i)} \tag{5}$$

and for L2 loss, the relation could be

$$h_m(x_i) = -\left[\frac{\partial L(y_i, F(x_i))}{\partial F(x_i)}\right]_{F=F_{m-1}} = 2(y_i - F_{m-1}(x_i)) = 2r_{im} \tag{6}$$

As a generalization, for any loss function the pseudo-residual can be defined as

$$r_{im} = -\left[\frac{\partial L(y_i, F(x_i))}{\partial F(x_i)}\right]_{F=F_{m-1}} \tag{7}$$

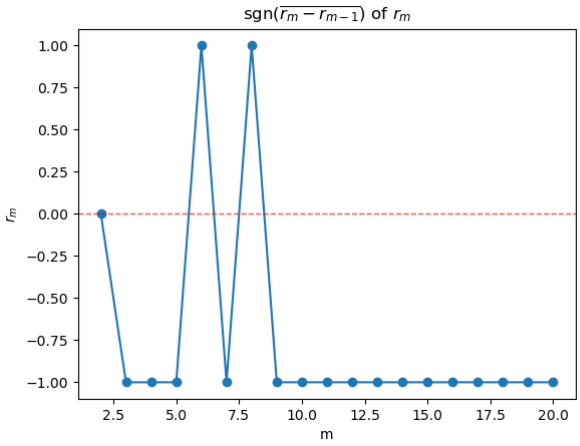
where the constant coefficient was neglected, for it does not influence the direction.

The procedure can be explained by such an intuition: by adding the new classifier trained by the residual, the return value of the total classifier would approach the ground-truth value better and better, i.e. the residual given by the total classifier would go lower.

To justify whether all the $r_{i,m}$ monotonically changes over m, we used function `judge_r_var` in `main.py`, and obtained the monotonical rate of the $r_{i,m}$ values:

```
monotonical rate: 0.5789473684210527
```

The monotonical rate means that $r_{i,m}$ values of 57.89% features (corresponding with i) varies monotonically, while others are not. For further justification, another plot should be shown:



The plot shows the variation of $\mathrm{sgn}(\overline{r_m - r_{m-1}})$ w.r.t. m, which represents the changing trend of the total residual r_m. It's clear that as m goes up, the r_m does not increase any more. A reason of such phenomena might be that the model was not converged in the beginning, when some of the main features dominated the training of the new classifiers, for which the monotonicity corresponding to the other features could not maintain the same monotonicity with the main ones. After the model becomes stable, the modification of the new classifier now becomes moderate, and the variation of $r_{i,m}$ s could be the same-and also moderate.