XGBoost hyperparameters

The overall parameters have been divided into 3 categories by XGBoost authors:

- **General Parameters:** Guide the overall functioning
- **Booster Parameters:** Guide the individual booster (tree/regression) at each step
- **Learning Task Parameters:** Guide the optimization performed

General Parameters

- booster [default=gbtree]
- silent [default=0]
- Nthread [default to maximum number of threads available if not set]

Booster Parameters

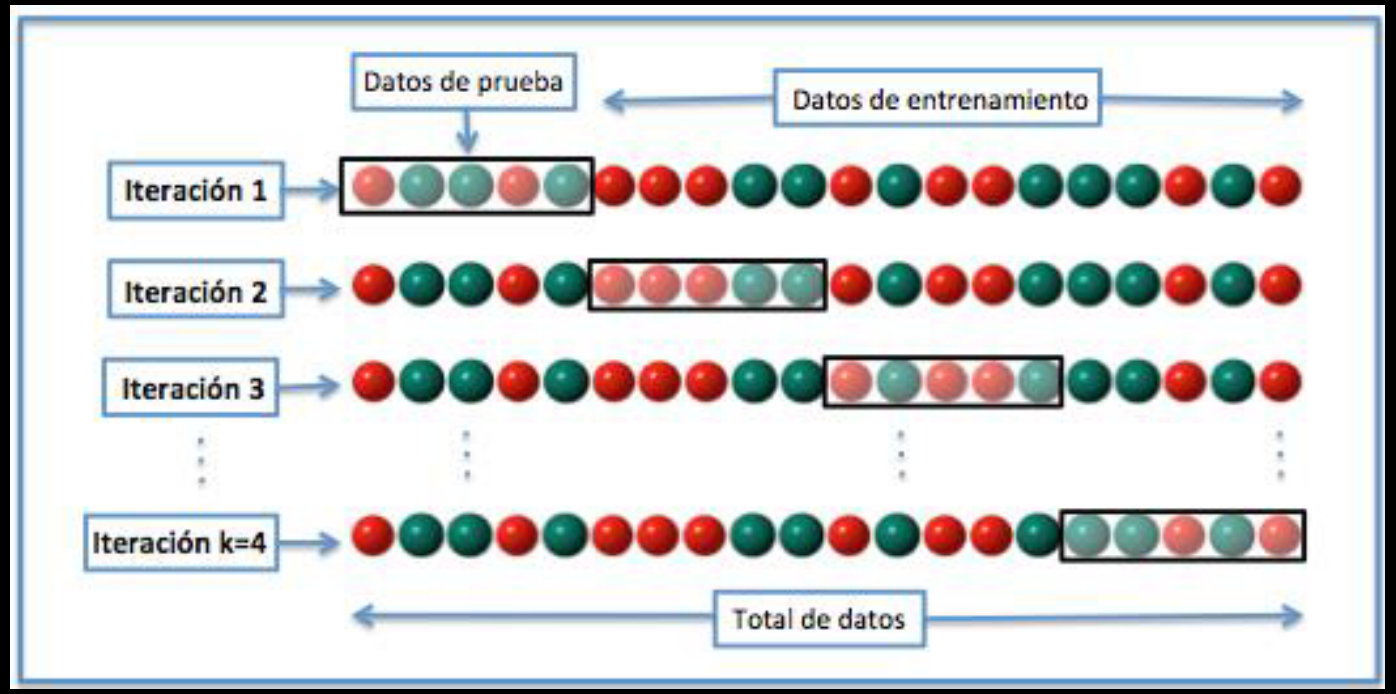
- learning_rate [0.1, 0.2, 0.3] (5.325)
- max_depth
- min_child_weight
- gamma [default=0]
- reg_lambda
- reg_alpha
- subsample
- colsample_bytree

Learning Task Parameters

- objective='binary:logistic' (returns predicted probability)
- eval_metric [default acording to objective]
- seed

Cross-validation

- Cross-validation is any of various similar model validation techniques for assessing how the results of a statistical analysis will generalize to an independent data set.



Initial values

We set some default values for the hyperparameters and introduce a new variable: n_estimators.

In [38]:
```
model=xgb.XGBClassifier(objective='binary:logistic',
                        learning_rate=0.2,
                        max_depth=6,
                        reg_lambda=1010,
                        n_estimators=500)

model.fit(train_x, train_y)
```

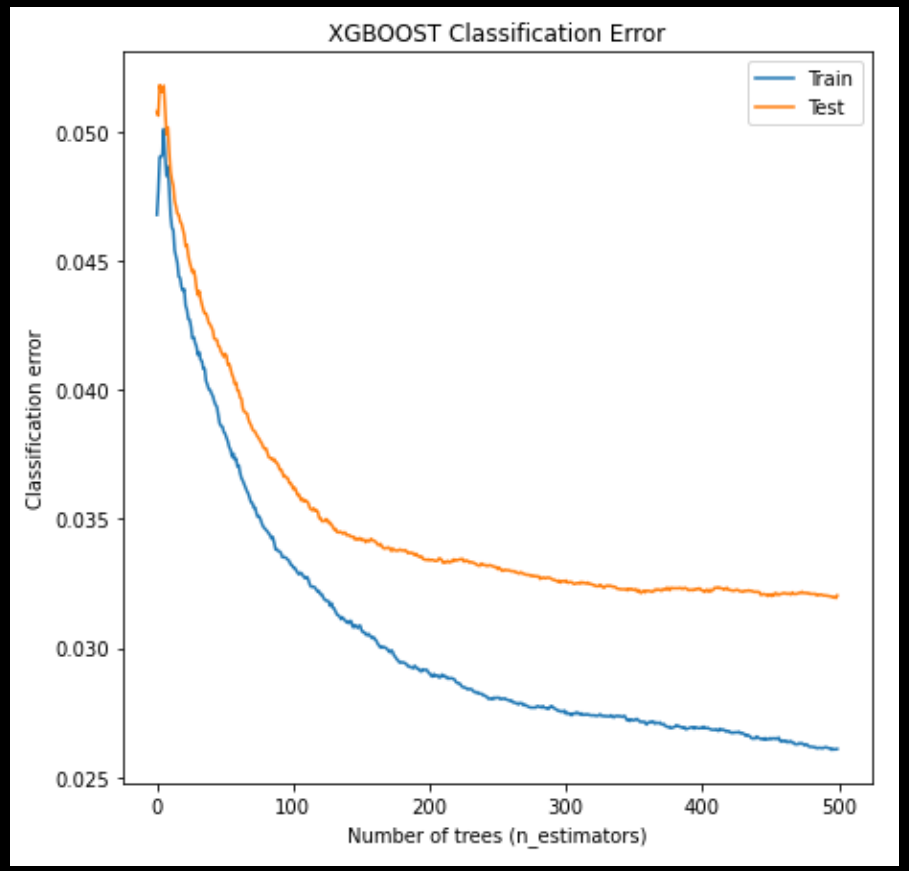
In [39]:
```
predict_signal = model.predict(test_x)
print(np.round(accuracy_score(test_y,predict_signal)*100, 2),'%') #96.37%
```

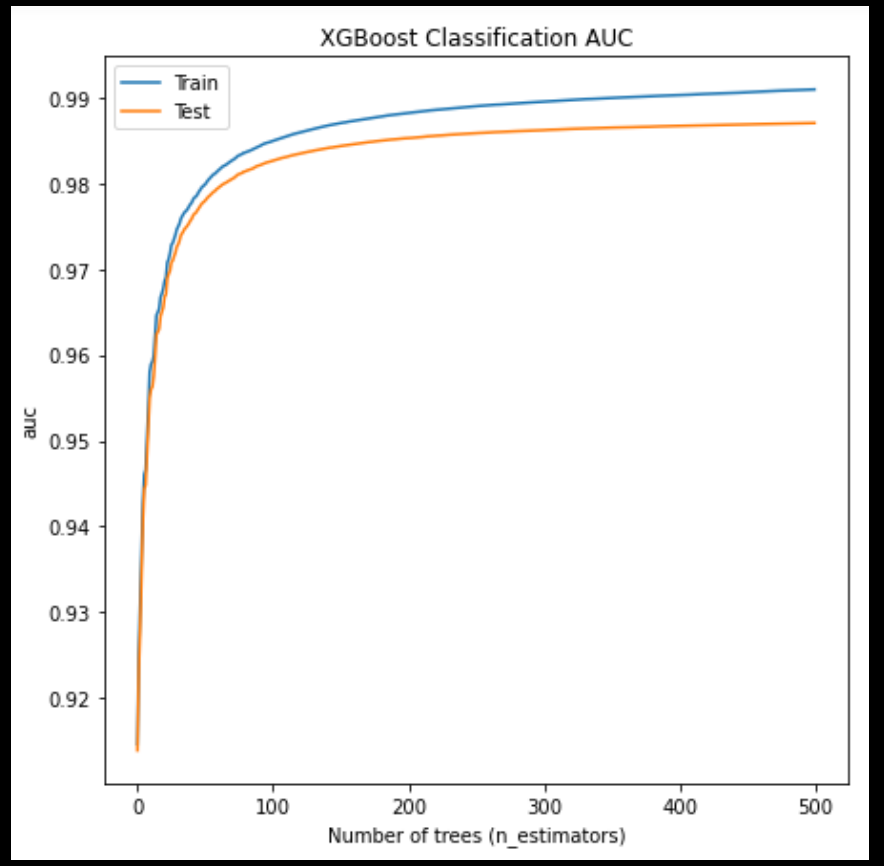
```
96.8 %
```

Validation of ‘n_estimators’

```
In [40]: eval_set = [(train_x, train_y), (test_x, test_y)]
         eval_metric = ["auc","error"]
         %time model.fit(train_x, train_y, eval_metric=eval_metric, eval_set=eval_set, verbose=True)

         [0]    validation_0-auc:0.91463    validation_0-error:0.04678    validation_1-auc:0.91391    validation_1-error:0.
```





Accuracy with a new number of trees

We proceed to calculate the accuracy of the model with the new value for n_estimators.

```
In [43]: predict_signal = model.predict(test_x)
         print(np.round(accuracy_score(test_y,predict_signal)*100, 2),'%') #96.37%

         96.56 %
```

Parameter tuning

Now, we proceed to make a grid search in order to find the optimal values for the parameters min_child_weight and max_depth. We also calculate the accuracy of the model with the new values.

In [46]:
```
param_test2 = {
 'max_depth':[5,6,7],
 'min_child_weight':[4,5,6]
}
```

```
{'max_depth': 7, 'min_child_weight': 5},
0.9833342326749174)
```

In [48]:
```
predict_signal = model.predict(test_x)
print(np.round(accuracy_score(test_y,predict_signal)*100, 2),'%') #96.37%
```

```
96.58 %
```

Gamma

Finally, we test gamma.

```
In [55]: param_test3 = {
          'gamma':[0.15, 0.1, 0.05, 0]
         }
```

```
 {'gamma': 0},
 0.9833401299059255)
```

Subsample and colsample_bytree

```
In [6]: param_test4 = {
 'subsample':[0.825, 0.85, 0.875, 0.9],
 'colsample_bytree':[0.85, 0.875, 0.9]
}
```

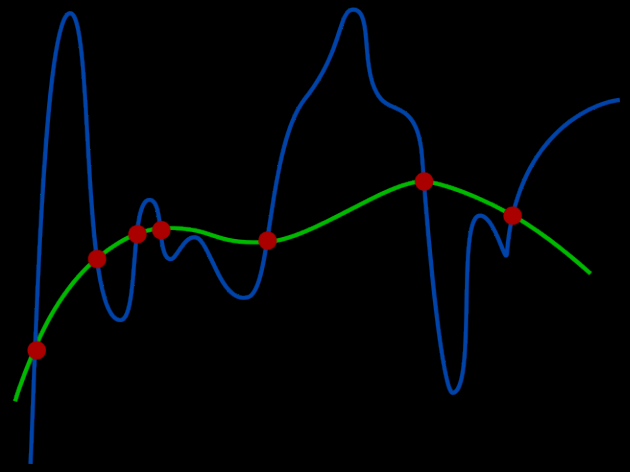
```
{'colsample_bytree': 0.85, 'subsample': 0.9},
0.9828006264617647)
```

At the end we decided to use the default values for this hyperparameters. (i.e. colsample_bytree =1 and subsample=1)

Regularization parameters

Reg_alpha [default=0] L1 regularization term on weights.
Reg_lambda [default=1] L2 regularization term on weights.

A higher value of this hyperparameters will make a model more conservative.



This two are used to prevent overfitting/overtraining in a set of data.

```
In [7]: param_test6 = {
         'reg_alpha':[1e-5, 1e-2, 0.1, 1, 100],
         'reg_lambda':[800, 900, 1000, 1100]
        }
```

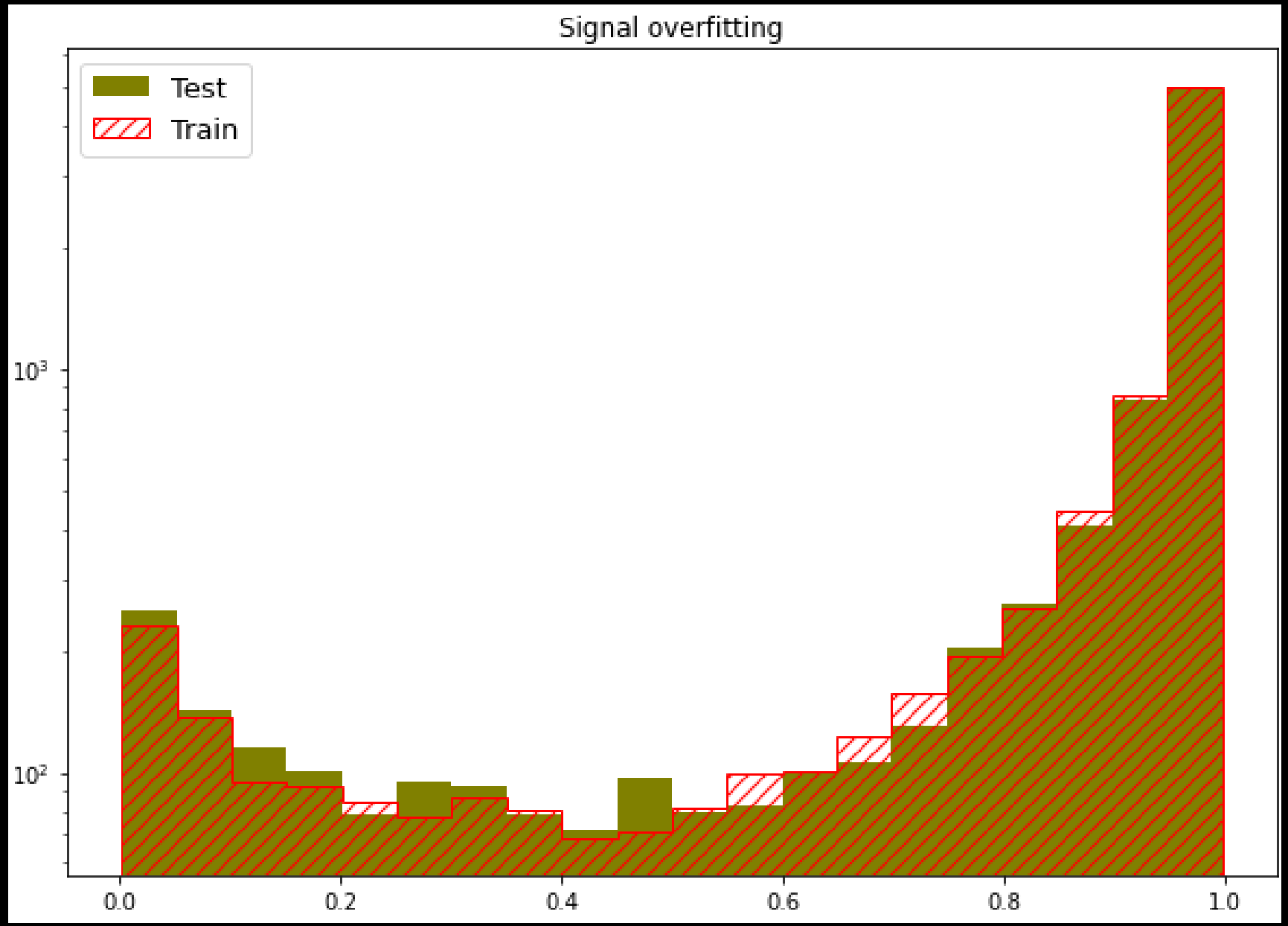
```
{'reg_alpha': 0.01, 'reg_lambda': 800},
0.9839582607115924)
```

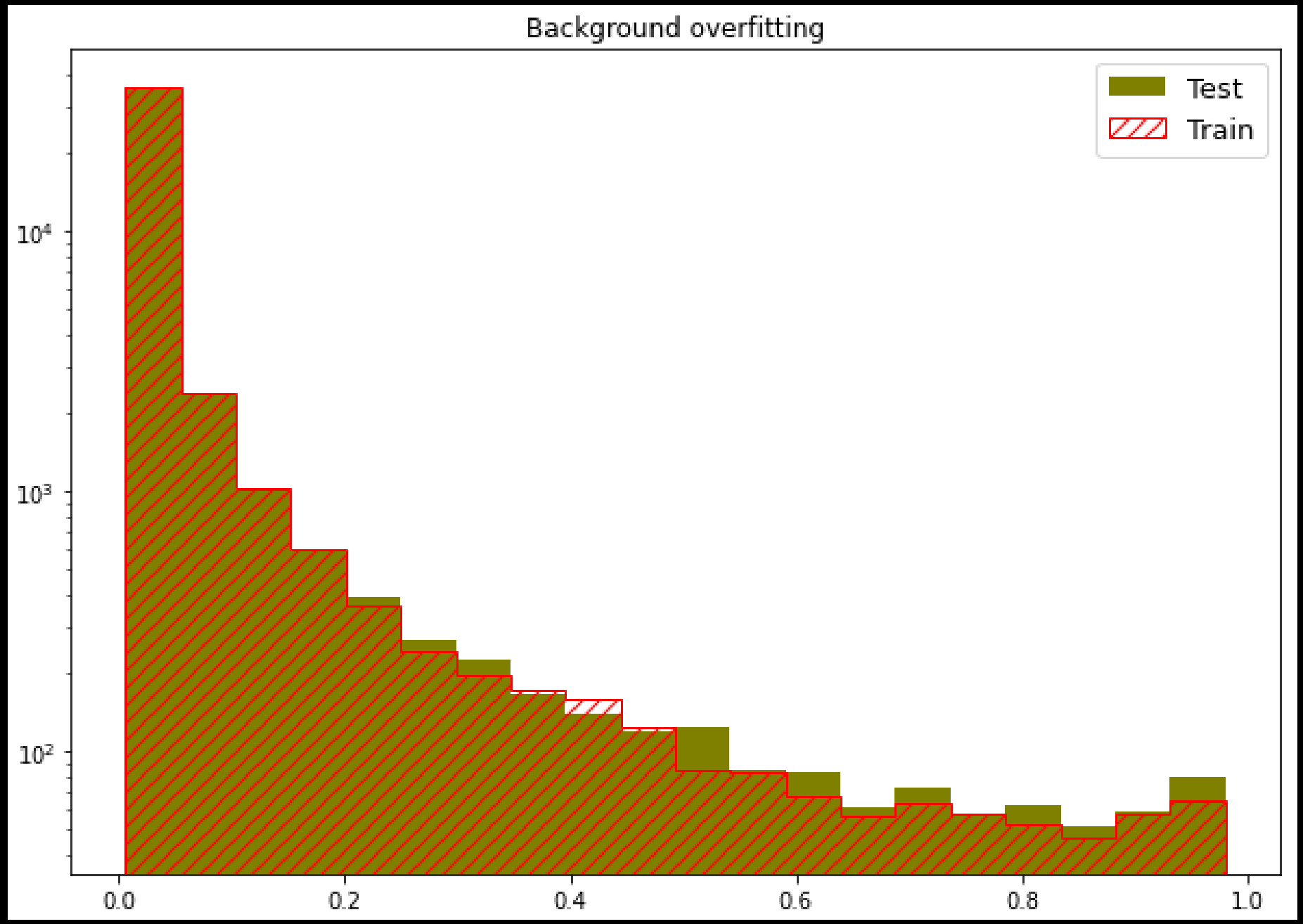
Final Model

```
In [8]: model=xgb.XGBClassifier(objective='binary:logistic',
                                learning_rate=0.2,
                                max_depth=7,
                                reg_lambda=800,
                                n_estimators=150,
                                reg_alpha=0.01, #001
                                min_child_weight=5)

        model.fit(train_x, train_y)
```

Finally, we proceed to evaluate the results in terms of the overfitting.


Signal overfitting
Test
Train
10^3
10^2
0.0
0.2
0.4
0.6
0.8
1.0


Background overfitting
Test
Train
10^4
10^3
10^2
0.0
0.2
0.4
0.6
0.8
1.0

```
Test :  0.9850992691477016
Train :  0.9880027140796667
```