Electricity Stability Prediction

EE 660 Course Project

Project Type: (1) Design a system based on real-world data

Number of student authors: 1

Sunfu yu, sunfuyu@usc.edu

December 2, 2020

1. Abstract

In this project, we need to deal with a local stability electric grid dataset, hosted by UCI Machine learning repository. The dataset has 10000 data points and 13 features. The whole dataset is divided into training, validation and test set. Then we practice models including pruned decision tree, random forest, gradient boosting decision tree and logistic regression to solve a binary classification problem. Our main goal is to compare validation or cross validation performance of candidate models and choose the best model with the highest AUC and good accuracy. Finally, our best model achieves 0.988 AUC and 94.17% accuracy on test set.

2. Introduction

2.1. Problem Type, Statement and Goals

The problem is a binary classification problem to predict "stabf" class, which is the stability label of the system with value "stable" and "unstable". Predicting the stability of the electricity grid can reduce potential hazards, protect personal safety and reduce property losses. But some challenges are as follows:

(1) A physical model that's inherently complicated and hard to abstract.
(2) High dimensionality of feature space.
(3) Nonlinear behaviors.

2.2. Literature Review

"Prediction of Electricity Stability Using Classification Machine Learning Methods" used R language to solve the same problem. http://rstudio-pubs-static.s3.amazonaws.com/496338_f2212c9a6a9f4dc89c2070c897139497.html.

But in this project, we applied machine learning methods based on Python and we chose the final model with highest AUC of ROC instead of accuracy.

2.3. Our Prior and Related Work – None

2.4. Overview of Our Approach

We compared AUC of validation or cross validation set in pruned decision tree, random forest, gradient boosting decision tree and logistic regression. Besides, confusion matrix and ROC were also used to evaluate the performance of different models directly.

3. Implementation

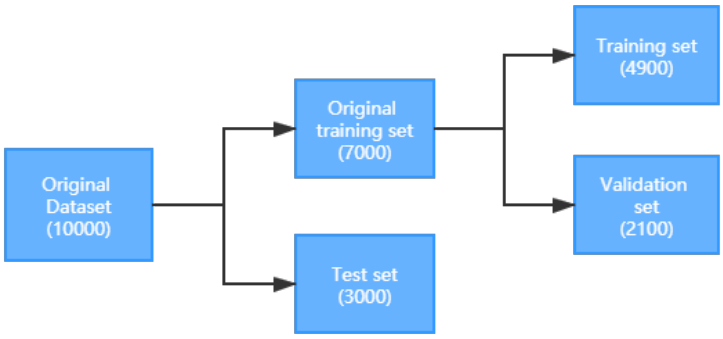
3.1. Data Set

The dataset has 13 real features and 1 binary label.

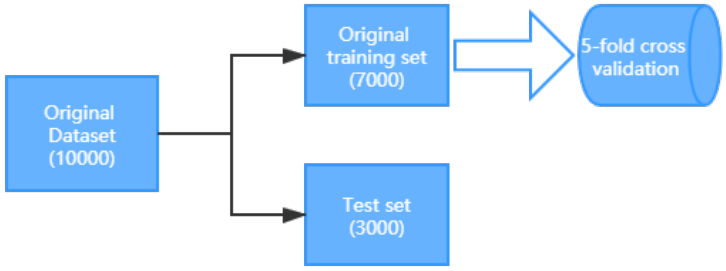
tau1	real	tau1, tau2, tau3, tau4 are electricity produced from node1, node2, node3 and node 4 respectively.
tau2	real	
tau3	real	
tau4	real	
p1	real	p1, p2, p3, p4 are the power consumed on node 1, node2, node3 and node4 respectively.
p2	real	
p3	real	
p4	real	
g1	real	g1, g2, g3, g4 are the price value of electricity on node1, node2, node3 and node 4 at each observation.
g2	real	
g3	real	
g4	real	
stab	real	Stability value of electricity grids. If it is more than 0, the label is stable; otherwise the label is unstable.
stabf	binary	Stability label of electricity grids, stable or unstable.

3.2. Dataset Methodology

The dataset was divided into original training set (7000 data points) and test set (3000 data points). Then the original training set was divided into training set (4900 data points) and validation set (2100 data points). K-fold validation was also used in the original training set with k = 5.



As shown in the flow chart above, the validation set was separated from the original training set randomly with 30% proportion. It was mainly used to tune the parameters based on AUC performance of ROC in the validation set in pruned decision tree, random forest and gradient boosting tree method. After we tuned parameters and obtain our final model, we trained the whole training set again and see the performance on test set.



As shown above, 5-kold cross validation was also used for parameter tuning in random forest, gradient boosting tree and logistic regression.

The test set was used for evaluating the final model and showing confusion matrices and ROC for candidate models. The test set was used 5 times in total. Note that parameters would not be tuned depends on the performance on test set.

3.3. Preprocessing, Feature Extraction, Dimensionality Adjustment

3.3.1. Data Cleaning

Since we want to solve a binary classification problem, the data in “stab” class were dropped. Because if it is handled as feature data, it will be too simple to predict the label by judging whether the value in “stab” is greater than 0 or not. Thus, the number of features used was 12 in total.

3.3.2. Label Encode

Since the label in dataset is binary, we just converted it to 0 and 1. ("stable": 0, "unstable": 1)

3.3.3. Standardization

The standardized data was only used for logistic regression method, because other tree-based methods do not require standardization. We standardized the training set/original training set and apply the means and variances to standardize the validation set/test set.

3.3.4. Polynomial Features Conversion

In nonlinear logistic regression model, we converted data to polynomial features. Original features were converted into 91 features including their squared and crossed combinations.

3.4. Training Process

3.4.1. Pruned Decision Tree

We chose decision tree as a simple model for baseline to see the performance and any difference after pruning.

- Original Model Performance

First, we created a complete tree. The AUC of ROC and accuracy on validation set are as follows:

	Validation AUC	Validation accuracy
Decision Tree	0.808	82.57%

A complete tree always cause overfitting and we need to control the size. Controlling some parameters like max depth of tree could prevent overfitting. Here, we controlled the tree size by pruning based on cost complexity parameter.

- Minimal Cost-Complexity Pruning

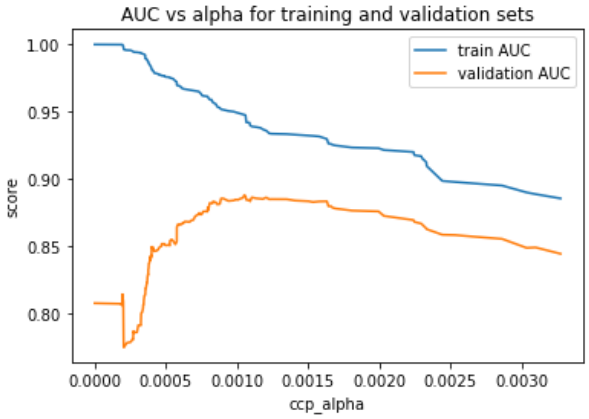
Minimal cost-complexity pruning is an algorithm used to prune a tree to avoid over-fitting. This algorithm is parameterized by known as the complexity parameter. The complexity parameter is used to define the cost-complexity measure, of a given tree :

where is the number of terminal nodes in and is traditionally defined as the total misclassification rate of terminal nodes. Minimal cost-complexity pruning finds the subtree of that minimizes .

The cost complexity measure of a single node is . We define the effective of a node to be the value where they are equal, or . A non-terminal node with the smallest value of is the weakest link and will be pruned. This process stops when the pruned tree's minimal is greater than complexity parameter used for pruning, which is ccp_alpha in DecisionTreeClassifier().

- Parameter Tuning

We chose different complexity parameters (ccp_alpha) values to see the performance of training and validation set. Greater values of complexity parameter increase the number of nodes pruned. Here we only show the effect of complexity parameters on regularizing the trees and how to choose a complexity parameter based on validation scores.



When complexity parameter for pruning is too small, it is obvious to see that the model overfits. However, when it is too large, it causes underfitting with pruning too many nodes.

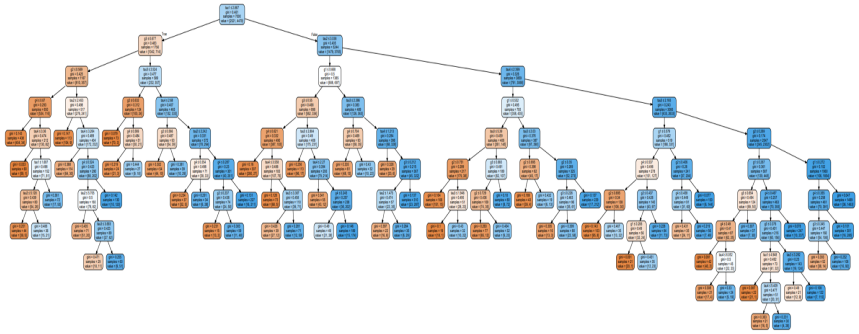
An appropriate complexity parameter prunes a decision tree that generalizes better. In this project, when complexity parameter (ccp_alpha) = 0.00105, maximizes validation AUC with 0.888.

After pruning decision tree, our validation performance improved a lot. The validation AUC increased 0.080 and the validation accuracy increased 2.57%.

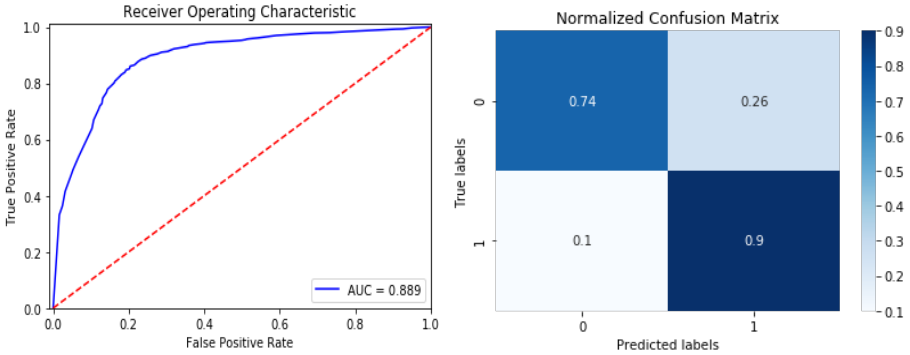
	Validation AUC	Validation accuracy
Pruned Decision Tree	0.888	85.14%

- Final Model Performance

The pruned decision tree visualization is as follows:

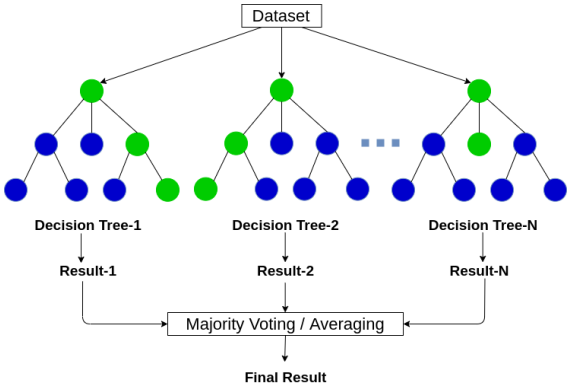


Then, we also used the specific model to train whole training set and see the performance on test set, the ROC and normalized confusion matrix are as follows:



3.4.2. Random Forest

Random forest is a bagged decision tree model that split on a subset of features on each split. And it performs well with high dimensionality data, low bias and moderate variance. We believe it would yield a better performance than a single decision tree.



- Original Model Performance

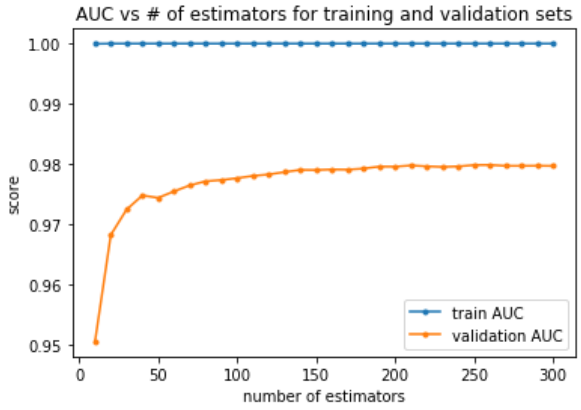
We used default parameters in random forest model to see the performance on cross validation AUC and accuracy.

	CV AUC	CV accuracy
Random Forest	0.977	91.54%

- Parameters Tuning

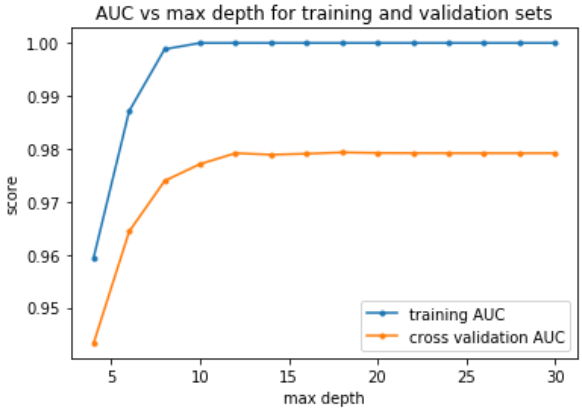
There are two most important parameters to be tuned: the number of trees (n_estimators) and max depth of trees (max_depth).

1. As the most important parameter, the number of trees was tuned first to evaluate the performance on validation set.



We can conclude that when the number of trees is 260, the highest validation AUC is 0.980. Even though the larger number of trees might obtain better performance, but it would also be more time consuming.

2. Then, we tried different max depths of trees in the model. We used 5-kold cross validation and compared the average cross validation AUC.



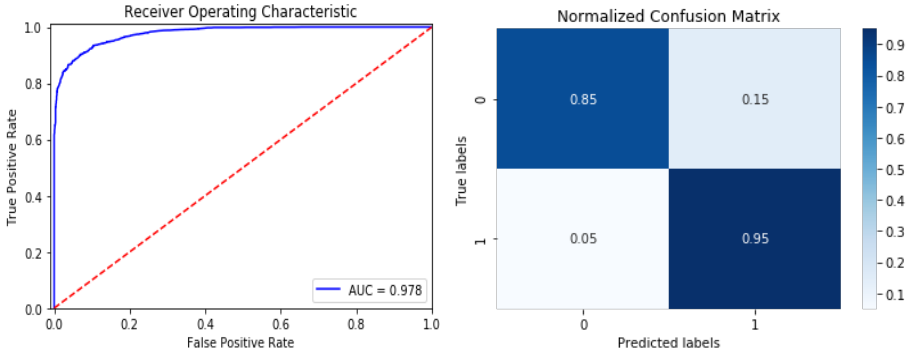
We can see that when max depth is 18, the highest cross validation AUC is 0.979. Controlling the size of trees could efficiently prevent underfitting and overfitting.

After choosing the number of trees and max depth, our cross validation AUC increased 0.002 and cross validation accuracy increased 0.30%.

	CV AUC	CV accuracy
Random Forest	0.979	91.84%

- Final Model Performance

The ROC and normalized confusion matrix on test set are as follows:



3.4.3. Gradient Boosting Decision Tree

Gradient boosting decision tree is a good model due to its efficiency and accuracy. We believe it would also yield a good performance.

GBDT algorithm includes:

1. Each iteration of the algorithm generates a new decision tree.
2. Calculate the first-order derivative and second-order derivative of the loss function for each sample.
3. Generate a new decision tree through the greedy strategy, and calculate the weight w of each leaf node at the same time.
4. Add the newly generated decision tree f(x) to the model.

$$\hat{y}_i^t = \hat{y}_i^{t-1} + \epsilon f_t(x_i)$$

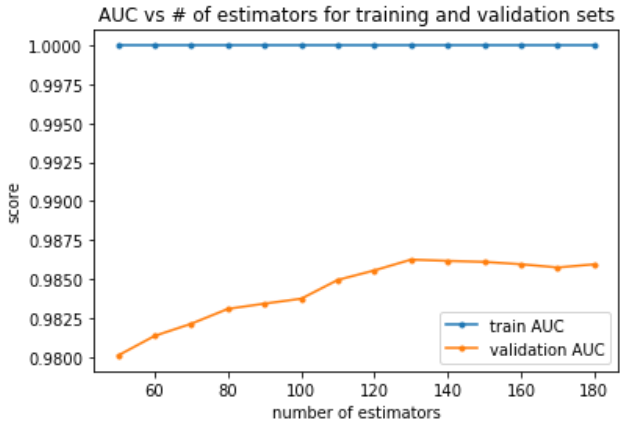
- Original Model Performance

First, we train the GBDT model with default parameters, the average cross validation AUC and accuracy are as follows:

	CV AUC	CV accuracy
GBDT	0.977	91.91%

- Parameters Tuning

1. Choose a relatively high learning rate (0.5) and determine the optimal number of trees for this learning rate. We evaluated the performance on validation AUC.



When the number of trees is 130, the highest validation AUC is 0.986. Even though when number of trees goes larger, the performance would be better. However, we would tune this parameter in step 4, we just chose a suitable value here.

2. Tune tree-specific parameters for decided learning rate and number of trees. The range of parameters are as follows:

Parameter name	Range
max_features	[1, 2, 3, … 11, 12]
min_samples_split	[10, 15, 20, … 55, 60]
min_samples_leaf	[6, 8, 10, … 28, 30]
max_depth	[5, 6, 7, … 29, 30]

Using 5-fold cross validation to compare the cross validation AUC. Since the combinations of numbers are large, we used RandomizedSearchCV() in our code to obtain a result similar to the optimal value. The parameters we chose are as follows:

Parameter name	Value
max_features	12
min_samples_split	50
min_samples_leaf	28
max_depth	13

3. Choose the fraction of samples to be used for fitting the individual base learners.

Parameter name	Range
subsample	[0.5, 0.55, 0.6, … 0.95, 1]

Also using 5-fold cross validation but we used GridSearchCV() here because it was not time consuming. And we chose the best parameter of highest cross validation AUC as follows:

Parameter name	Value
subsample	0.8

4. Lower the learning rate and increase the estimators proportionally to get more robust models.

The current learning rate is 0.5 and number of trees is 130. We decreased the learning rate to half (0.25) with twice the number of trees (260). The average cross validation AUC was 0.987.

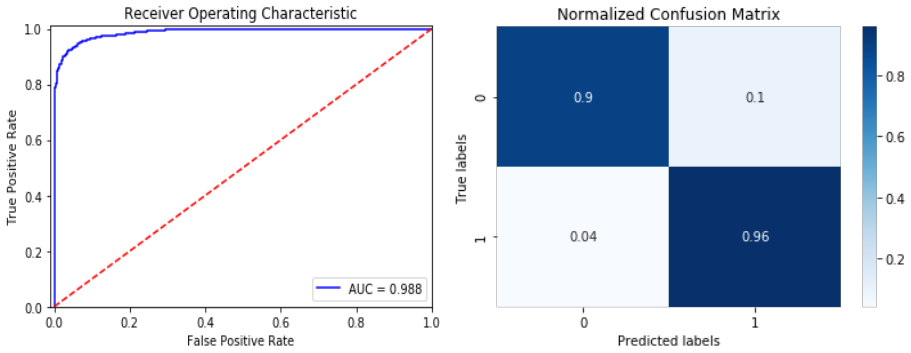
Then, reduce to one-tenth of the original learning rate (0.05) and increase the number of trees tenfold (1300). The average cross validation AUC was improved into 0.988. We just stopped here because it would be very time consuming to keep increasing the number of trees.

After pruning all parameters, the average cross validation AUC increased 0.011 and accuracy increased 2.15%.

	CV AUC	CV accuracy
GBDT	0.988	94.06%

- Final Model Performance

Finally, we trained the original training set and evaluated the test performance. The ROC and normalized confusion matrix are as follows:



3.4.4. Logistic Regression

Logistic Regression is a commonly used algorithm for binary classification tasks. At the same time, it is also a basic method and we want to see its performance with different regularizations and standardized data and how to use it to solve nonlinear classification problem.

- No Regularization

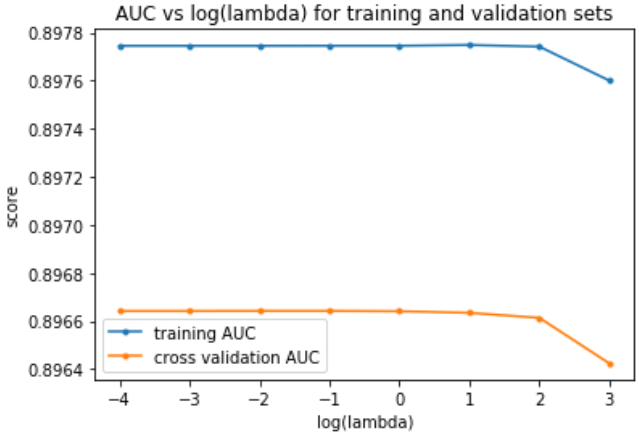
First, we used the logistic regression model without regularizations. In this way, we assumed that it would cause overfitting and we could try some regularizations later.

	CV AUC	CV accuracy
Logistic Regression	0.897	82.09%

- L2 Regularization

It is also called Ridge Regression. It adds "squared magnitude" of coefficient as penalty term to the loss function.

We compared 5-fold cross validation AUC on different regularization strengths.



When $\lambda = 0.01$, the best cross validation AUC of ROC is 0.897.

	CV AUC	CV accuracy
LR (l2 regularization)	0.897	82.09%

The result was surprising, since it was almost the same result without regularization. Regularization always could prevent overfitting in logistic regression and improve the performance. However, it showed that the model without regularization might underfit so that regularization did not help a lot. We would try to use nonlinear logistic regression later to confirm our assumption.

- Nonlinear Logistic Regression

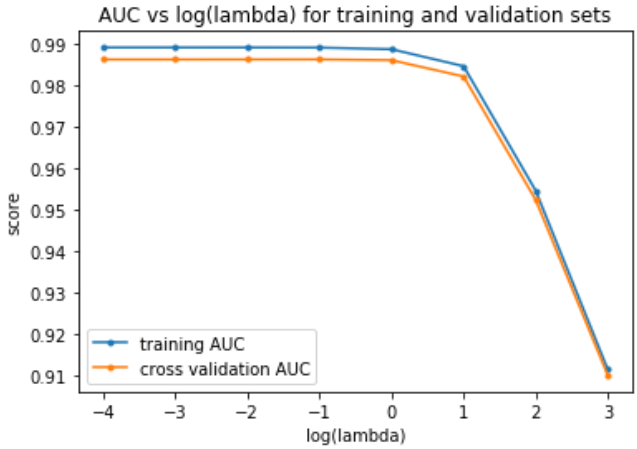
We used polynomial features to implement nonlinear logistic regression model. The average cross validation AUC and accuracy are as follows:

	CV AUC	CV accuracy
Nonlinear LR	0.986	94.14%

Obviously, nonlinear model performed better on our data. Both of them increased a lot, with cross validation AUC increased 0.089 and accuracy increased 12.05%.

- Nonlinear Logistic Regression with L2 Regularization

To prevent overfitting, we tried regularization on nonlinear logistic model and compared different regularization strengths.



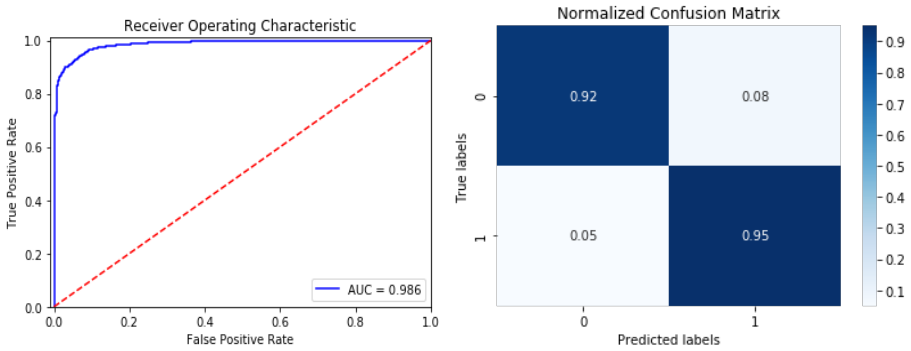
When λ = 0.1, the best cross validation AUC of ROC is 0.986 (increased 2E-05 actually compared with no regularization). We just chose this specific λ and the average cross validation AUC and accuracy are as follows:

	CV AUC	CV accuracy
Nonlinear LR (L2)	0.986	94.17%

After tuning regularization strength, our model performance increased slightly. That is to say, the nonlinear model without any regularization did not overfit too much.

- Final Model Performance

The best model among logistic regression models was nonlinear logistic regression with L2 regularization and ROC and normalized confusion matrix on test set are as follows:

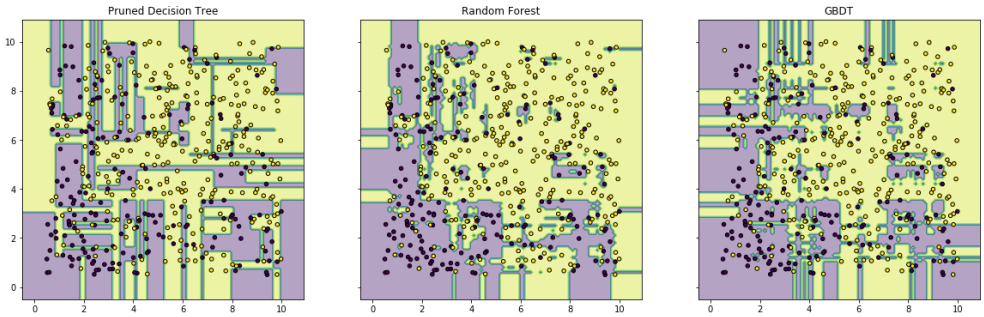


3.5. Model Selection and Comparison of Results

Here is a table of our model performance. Finally, we chose gradient boosting decision tree as our optimal model with 0.988 average cross validation AUC.

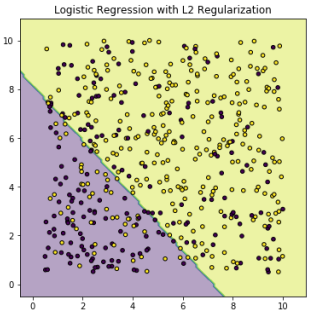
Model	Method	Validation AUC
Pruned Decision Tree	validation	0.888
Random Forest	cross validation	0.979
GBDT	cross validation	0.988
Logistic Regression	cross validation	0.986

From the feature importance of GBDT method, we just chose tau1 and tau2 as features and plotted decision regions of tree-based models on 500 data points as follows:



Random forest consists of multiple single trees each based on a random sample of the training data. It is typically more accurate than a single decision tree. As expected, random forest was more accurate on our prediction.

Like random forest, gradient boosting decision tree is a set of decision trees. The two main differences are: random forest builds each tree independently while gradient boosting builds one tree at a time; random forest combines results at the end of the process while gradient boosting combines results along the way. If we carefully tune parameters, GBDT can typically result in better performance than random forests. As what we expected, GBDT performed a little better than random forest.



As shown above, as a linear model, it is typically hard for logistic regression to totally fit our data. That is why our nonlinear logistic regression model resulted in much better result.

4. Final Results and Interpretation

- The Final System and Performance

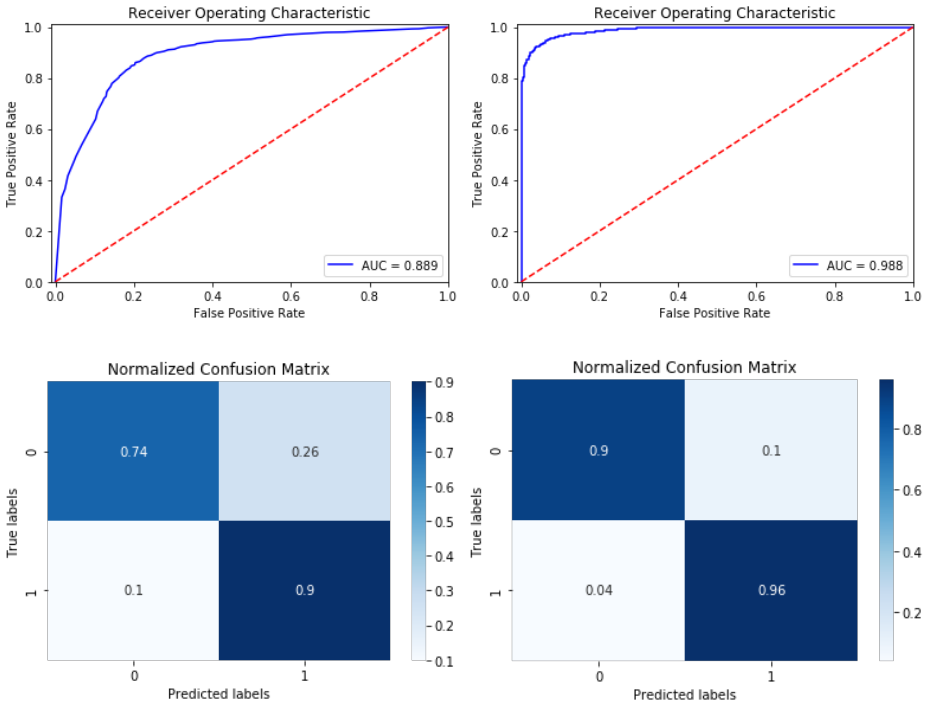
The final model we chosen was gradient boosting decision tree, and the decided parameters are shown below:

	Parameter name	Value
GBDT	n_estimators	1300
	learning_rate	0.05
	min_samples_split	50
	min_samples_leaf	28
	max_features	12
	max_depth	13
	subsample	0.8

Finally, we trained the whole training set with 7000 data points and evaluated the final performance on unseen test set, with 0.988 AUC of ROC and 94.17% accuracy. The baseline model chosen was pruned decision tree and also evaluated it on test set. The comparison is shown below:

	Test AUC	Test accuracy
Pruned Decision Tree (Baseline)	0.889	84.17%
GBDT	0.988	94.17%

ROC and confusion matrix are also compared below (the left is baseline model and the right is our final model):



- Out of Sample Error

Assume the tolerance is 0.01 and use the formula below:

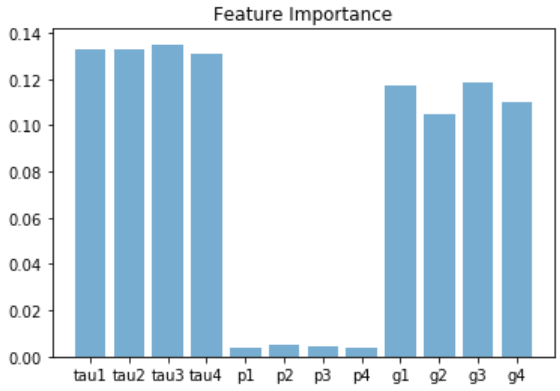
$$\mathrm{Eout(hg)} \leq \mathrm{Etest\ (hg)} + \sqrt{\frac{1}{2N} ln \frac{2M}{\delta}} \quad \text{with probability} \geq 1\text{-}\ \delta$$

With M = 1, N = 3000 and $^{\mathrm{Etest\ (hg)}}$ = 5.83%, $^{\mathrm{Eout(hg)} \leq}$ 0.088.

- Interpretation

1. Random forest is typically more accurate than a single decision tree. Because decision trees are prone to overfitting, especially when a tree is particularly deep. The randomness in random forest significantly solves the overfitting problem.
2. GBDT generally performs better than random forest. Random forest build trees in parallel, while in boosting, trees are built sequentially. That is to say, each tree is grown using information from previously grown trees unlike in bagging where we create multiple copies of original training data and fit separate decision tree on each.
3. A single decision tree is a weak predictor, but is relatively fast to build and easy to read. Random forest build trees in parallel and thus are fast and also efficient, while GBDT is much slower.
4. The decision tree model gives high importance to a particular set of features. But the random forest chooses features randomly during the training process. Thus, it does not depend highly on any specific set of features. This is a characteristic of random forest.
5. One drawback of gradient boosted trees is that they have a lot of parameters to tune, while random forest has much fewer parameters (or only one parameter: number of trees or number of features to randomly select from set of features, building unpruned trees in random forest is not prone to overfitting).
6. For underfitting in logistic regression, we could add new features and increase the complexity of model.
7. For our final GBDT model, we could improve it further to increase the number of trees and reduce the learning rate at the expense of efficiency.

5. Summary and conclusions



From feature importance of our final GBDT model, we could conclude that tau[i] (electricity produced from nodes) plays an important role on predicting the electricity stability. While p[i] (power consumed on nodes) has a negligible impact on the prediction.

6. References

[1] "Decision Tree vs Random Forest vs Gradient Boosting Machines: Explained Simply," 28 July 2019. [Online]. Available: https://www.datasciencecentral.com/profiles/blogs/decision-tree-vs-random-forest-vs-boosted-trees-explained.

[2] "Gradient boosted trees: Better than random forest," 23 February 2018. [Online]. Available: https://kharshit.github.io/blog/2018/02/23/gradient-boosted-trees-better-than-random-forest.

[3] "Complete Machine Learning Guide to Parameter Tuning in Gradient Boosting (GBM) in Python," 21 February 2016. [Online]. Available: https://www.analyticsvidhya.com/blog/2016/02/complete-guide-parameter-tuning-gradient-boosting-gbm-python/.

[4] "Visualize a Decision Tree in 4 Ways with Scikit-Learn and Python," 22 June 2020. [Online]. Available: https://mljar.com/blog/visualize-decision-tree/.

[5] "In Depth: Parameter tuning for Random Forest," 21 December 2017. [Online]. Available: https://medium.com/all-things-ai/in-depth-parameter-tuning-for-random-forest-d67bb7e920d.