Credit Risk Management: Classification Models & Hyperparameter Tuning

Andrew Nguyen · Follow

Published in Towards Data Science · 8 min read · Aug 10, 2020

The final part aims to walk you through the process of applying different classification algorithms on our transformed dataset as well as producing the best-performing model using Hyperparameter Tuning.

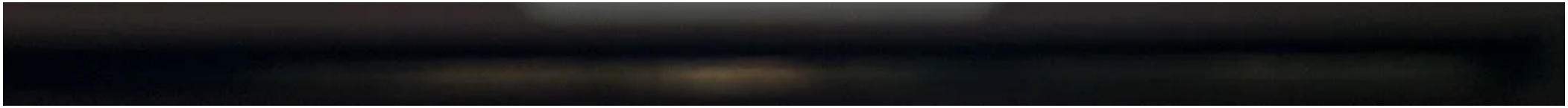


Image credit: https://unsplash.com/photos/w7ZyuGYNpRQ

As a reminder, this end-to-end project aims to solve a classification problem in Data Science, particularly in finance industry and is divided into 3 parts:

1. Explanatory Data Analysis (EDA) & Feature Engineering
2. Feature Scaling and Selection (Bonus: Imbalanced Data Handling)
3. **Machine Learning Modelling (Classification)**

If you have missed the previous two parts, feel free to check them out **here** and **here** before going through the final part which leveraged their output in producing the best classification model.

. . .

A. Classification Models

Which algorithms should be used to build a model that addresses and solves a classification problem?

When it comes to classification, we have quite a handful of different algorithms to use unlike regression. To name some, Logistic Regression, K-Neighbors, SVC, Decision Tree and Random Forest are the top common and widely used algorithms to solve such problems.

Here's a quick recap of what each algorithm does and how it distinguishes itself from the others:

- **Logistic Regression:** this algorithm uses regression to ***predict the continuous probability*** of a data sample (from 0 to 1), then classifies that sample to the more probable target (either 0 or 1). However, it assumes a linear relationship between the the inputs and the target, which might not be a good choice if the dataset does not follow Gaussian Distribution.
- **K-Neighbors:** this algorithm assumes data points which are in close proximity to each other belong to the same class. Particularly, it classifies the target (either 0 or 1) of a data sample by ***a plurality vote of the neighbors*** which are close in distance to it.
- **SVC:** this algorithm makes classifications by ***defining a decision boundary*** and then classify the data sample to the target (either 0 or 1) by seeing which side of the boundary it falls on. Essentially, the

algorithm aims to maximize the distance between the decision boundary and points in each class to decrease the chance of false classification.

- **Decision Tree**: as the name tells, this algorithm ***splits the root of the tree*** (the entire dataset) into decision nodes, and each decision node will be split until no further node is splittable. Then, the algorithm classifies the data sample by sorting them down the tree from the root to the leaf/terminal node and seeing which target node it falls on.
- **Random Forest**: this algorithm is an ensemble technique developed from the Decision Tree, in which it involves many decision tree that work together. Particularly, the random forest gives that data sample to each of the decision trees and ***returns the most popular classification*** to assign the target to that data sample. This algorithm helps avoid overfitting which may occurs to Decision Tree, as it aggregates the classification from multiple trees instead of 1.

Let's see how they work with our dataset compared to one another:

```
from sklearn.linear_model import LogisticRegression
from sklearn.neighbors import KNeighborsClassifier
from sklearn.svm import SVC
```

```
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier

classifiers = {
    "LogisticRegression" : LogisticRegression(),
    "KNeighbors" : KNeighborsClassifier(),
    "SVC" : SVC(),
    "DecisionTree" : DecisionTreeClassifier(),
    "RandomForest" : RandomForestClassifier()
}
```

After importing the algorithms from sklearn, I ***created a dictionary which combines all algorithms into one place,*** so that it's easier to apply them on the data at once, without the need to manually iterate each individually.

```
#Compute the training score of each models

train_scores = []
test_scores = []

for key, classifier in classifiers.items():
    classifier.fit(x_a_train_rs_over_pca, y_a_train_over)
    train_score = round(classifier.score(x_a_train_rs_over_pca,
y_a_train_over),2)
    train_scores.append(train_score)
    test_score = round(classifier.score(x_a_test_rs_over_pca,
y_a_test_over),2)
    test_scores.append(test_score)

print(train_scores)
print(test_scores)
```

After applying the algorithms on both train and test sets, it seems that Logistic Regression doesn't work well for the dataset as the scores are relatively low (around 50%, which indicates that the model is not able to classify the target). This is quite understandable and somehow proves that our original dataset is not normally distributed.

```
[0.53, 0.78, 0.7, 0.85, 0.85]
[0.52, 0.55, 0.57, 0.52, 0.56]
```

In contrast, Decision Tree and Random Forest produced a significantly high accuracy scores on the train sets (85%). Yet, it's the otherwise for the test set when the scores are remarkably low (over 50%). Possible reasons that might explain the large gap is (1) overfitting the train set, (2) leaking target to the test set. However, after cross checking, it doesn't seem as the case.

Hence, I decided to look into another scoring metric, **Cross Validation Score,** to see if there's any difference. Basically, this technique splits the training set into n folds (default = 5), then fits the data on n-1 folds and score on the other fold. This process is repeated in n folds from which the average score will be

calculated. Cross validation score ***brings a more objective analysis on how the models works*** as compared to the standard accuracy score.

```
from sklearn.model_selection import cross_val_score

train_cross_scores = []
test_cross_scores = []

for key, classifier in classifiers.items():
    classifier.fit(x_a_train_rs_over_pca, y_a_train_over)
    train_score = cross_val_score(classifier, x_a_train_rs_over_pca,
y_a_train_over, cv=5)
    train_cross_scores.append(round(train_score.mean(),2))
    test_score = cross_val_score(classifier, x_a_test_rs_over_pca,
y_a_test_over, cv=5)
    test_cross_scores.append(round(test_score.mean(),2))

print(train_cross_scores)
print(test_cross_scores)
```

```
[0.53, 0.69, 0.66, 0.72, 0.75]
[0.53, 0.7, 0.68, 0.71, 0.76]
```

As seen, the gap between the train and test scores was significantly bridged!

Since Random Forest model produced the highest cross validation score, we will test it against another score metric named **ROC AUC Score** as well as see how it performs on the **ROC Curve**.

Essentially, **ROC Curve** is a plot of the false positive rate (x-axis) versus the true positive rate (y-axis) within the threshold between 0 and 1 while **AUC** represents the degree or measure of separability (simply, the ability to distinguish the target).

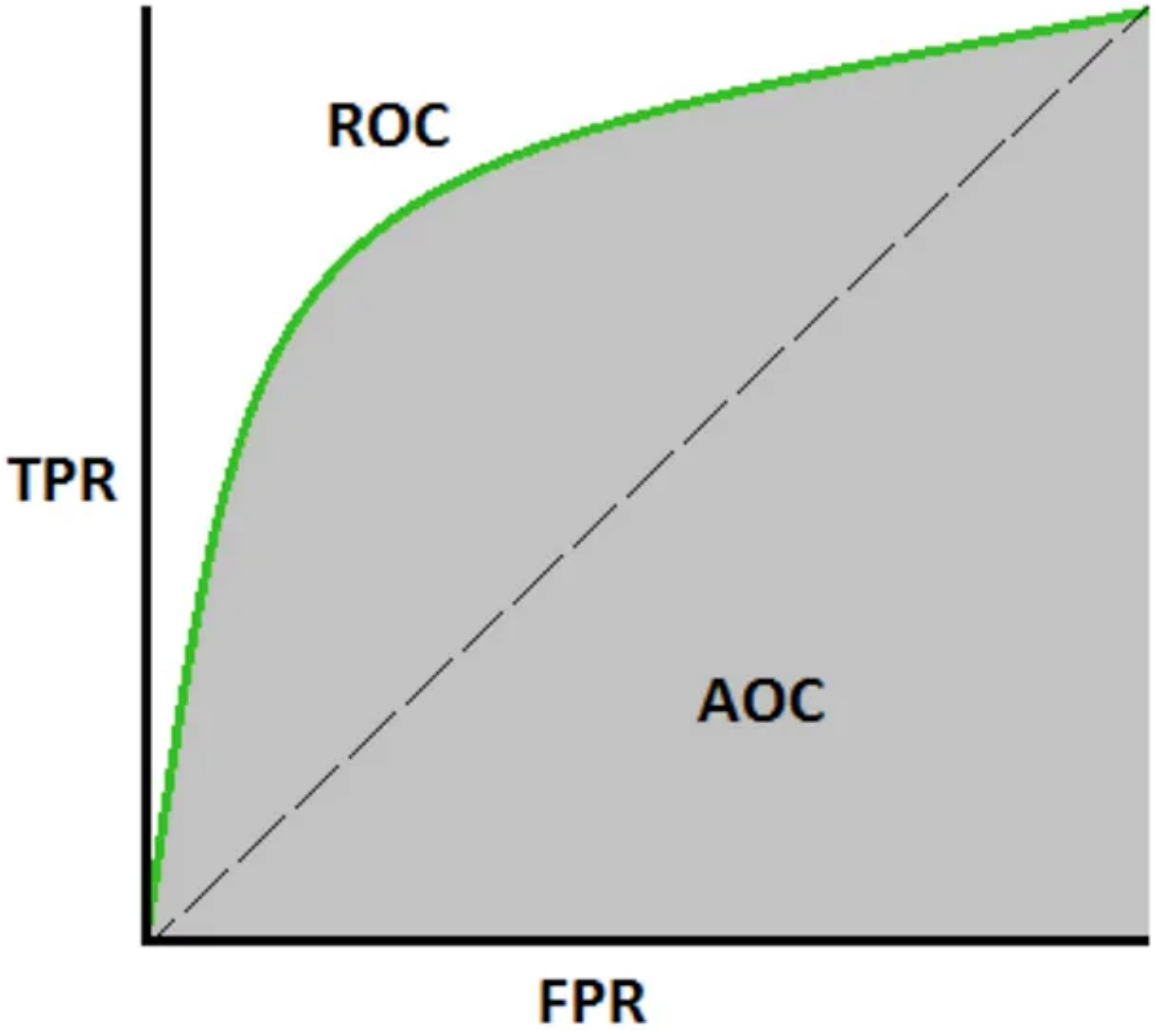


Image credit: https://towardsdatascience.com/understanding-auc-roc-curve-68b2303cc9c5

Below is a quick summary table of how to calculate **FPR** (the inversion of Specificity) and **TPR** (also known as Sensitivity):

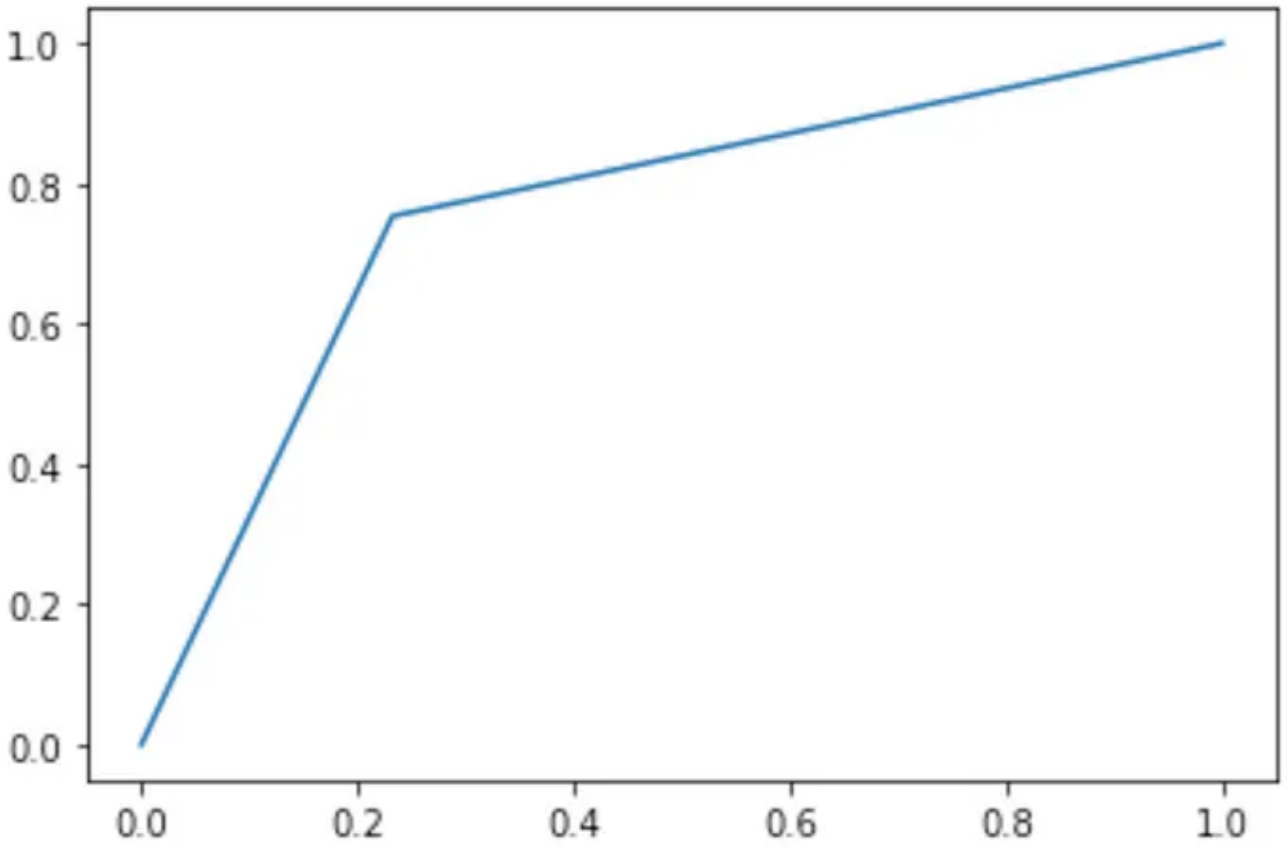
Metric Name	Formula from Confusion Matrix
Accuracy	$\frac{TP + TN}{TP + TN + FP + FN}$
Precision	$\frac{TP}{TP + FP}$
Recall, Sensitivity, TPR	$\frac{TP}{TP + FN}$
Specificity, 1-FPR	$\frac{TN}{TN + FP}$
F1	$\frac{2 * precision * recall}{precision + recall}$

Image credit: https://towardsdatascience.com/hackcvilleds-4636c6c1ba53

```
from sklearn.model_selection import cross_val_predict
from sklearn.metrics import roc_curve
from sklearn.metrics import roc_auc_score

rf = RandomForestClassifier()
rf.fit(x_a_train_rs_over_pca, y_a_train_over)
rf_pred = cross_val_predict(rf, x_a_test_rs_over_pca, y_a_test_over,
cv=5)
print(roc_auc_score(y_a_test_over, rf_pred))
```

```
#Plot the ROC Curve
fpr, tpr, _ = roc_curve(y_a_test_over, rf_pred)
plt.plot(fpr, tpr)
plt.show()
```



ROC Curve with ROC AUC Score = 76%

As I had proved that cross validation worked on this dataset, I then applied another cross validation technique called "**cross_val_predict**", which follows similar methodology of splitting n-folds and predicting the value accordingly.

. . .

B. Hyperparameter Tuning

What is hyperparameter tuning and how does it help to improve the accuracy of the model?

After computing the model from the default estimators of each algorithm, I was hoping to see if further improvement could be made, which comes down to Hyperparameter Tuning. Essentially, this technique ***chooses a set of optimal estimators*** from each algorithm that (might) produces the highest accuracy score on the given dataset.

The reason why I put (might) in the definition is that for some cases, little to none improvement is seen depends on the dataset as well as the preparation done initially (plus it takes like forever to run). However, Hyperparameter

Tuning should be taken into consideration with the hope of finding the best performing model.

```
#Use GridSearchCV to find the best parameters

from sklearn.model_selection import GridSearchCV

#Logistic Regression
lr = LogisticRegression()
lr_params = {"penalty": ['l1', 'l2'], "C": [0.001, 0.01, 0.1, 1, 10,
100, 1000], "solver": ['newton-cg', 'lbfgs', 'liblinear', 'sag',
'saga']}
grid_logistic = GridSearchCV(lr, lr_params)
grid_logistic.fit(x_a_train_rs_over_pca, y_a_train_over)
lr_best = grid_logistic.best_estimator_

#KNearest Neighbors
knear = KNeighborsClassifier()
knear_params = {"n_neighbors": list(range(2,7,1)), "algorithm":
['auto', 'ball_tree', 'kd_tree', 'brutle']}
grid_knear = GridSearchCV(knear, knear_params)
grid_knear.fit(x_a_train_rs_over_pca, y_a_train_over)
knear_best = grid_knear.best_estimator_

#SVC

svc = SVC()
svc_params = {"C": [0.5, 0.7, 0.9, 1], "kernel":['rbf', 'poly',
'sigmoid', 'linear']}
grid_svc = GridSearchCV(svc, svc_params)
grid_svc.fit(x_a_train_rs_over_pca, y_a_train_over)
svc_best = grid_svc.best_estimator_

#Decision Tree
```

```
tree = DecisionTreeClassifier()
tree_params = {"criterion": ['gini', 'entropy'],
"max_depth":list(range(2,5,1)),
"min_samples_leaf":list(range(5,7,1))}
grid_tree = GridSearchCV(tree, tree_params)
grid_tree.fit(x_a_train_rs_over_pca, y_a_train_over)
tree_best = grid_tree.best_estimator_
```

GridSearchCV is the key to finding the set of optimal estimators in each algorithm, as it scrutinizes and combines different estimators to fit the dataset, then returns the best set among all.

One thing of note is that we have to remember by heart all available estimators of each algorithm to be able to use. For example, with Logistic Regression, we have a set of “penalty”, “C”, and “solver” which do not belong to other algorithms.

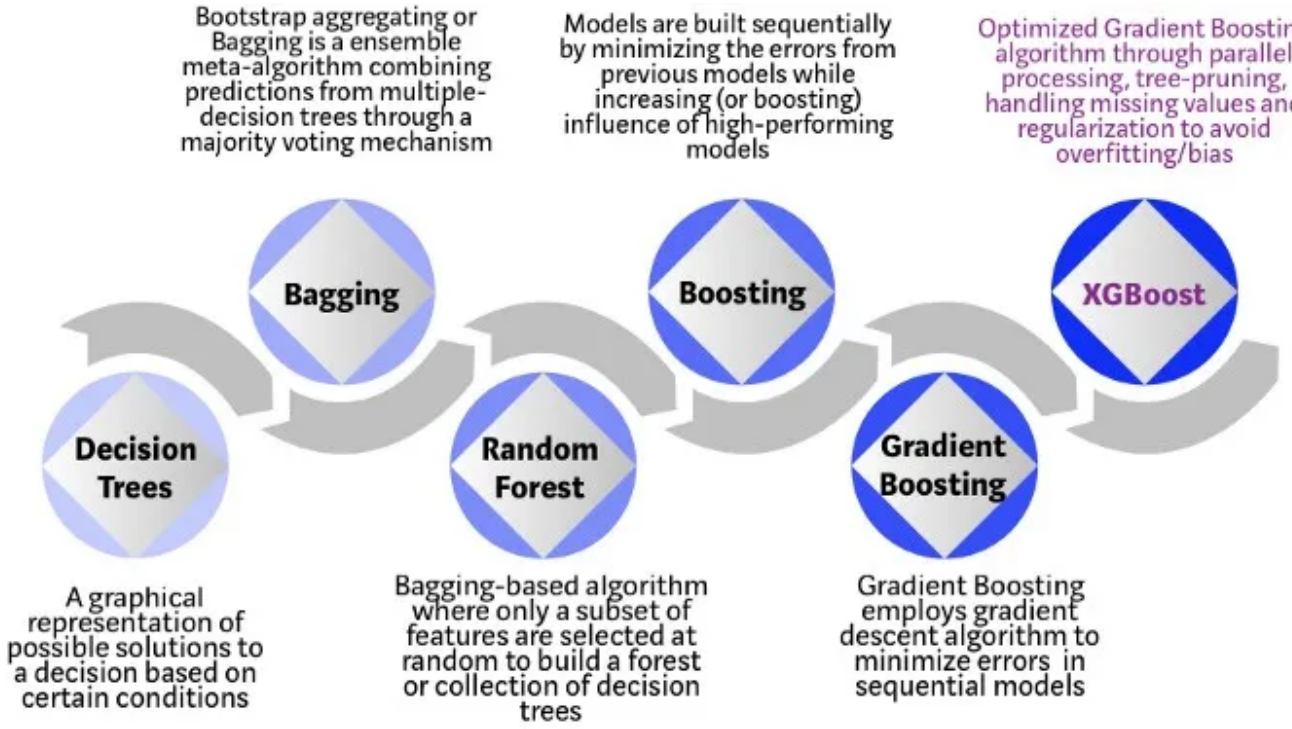
After finding the **.best_estimator_** of each algorithm, fit and predict the data using each algorithm with its best set. However, we need to compare the new scores against the original to determine if any improvement is seen or to continue fine-tuning the estimators again.

. . .

Bonus: XGBoost and LightGBM

What are XGBoost and LightGBM and how significantly better do these algorithms do compared to the traditional?

Apart from the common classification algorithms I've heard of, I also have known a couple of advanced algorithms which rooted from the traditional. In this case, **XGBoost and LightGBM** can be considered as *the successor of Decision and Random Forest.* Look at the below timeline for a better understanding of how these algorithms were developed:


Bootstrap aggregating or Bagging is a ensemble meta-algorithm combining predictions from multiple-decision trees through a majority voting mechanism
Models are built sequentially by minimizing the errors from previous models while increasing (or boosting) influence of high-performing models
Optimized Gradient Boosting algorithm through parallel processing, tree-pruning, handling missing values and regularization to avoid overfitting/bias
Bagging
Boosting
XGBoost
Decision Trees
Random Forest
Gradient Boosting
A graphical representation of possible solutions to a decision based on certain conditions
Bagging-based algorithm where only a subset of features are selected at random to build a forest or collection of decision trees
Gradient Boosting employs gradient descent algorithm to minimize errors in sequential models

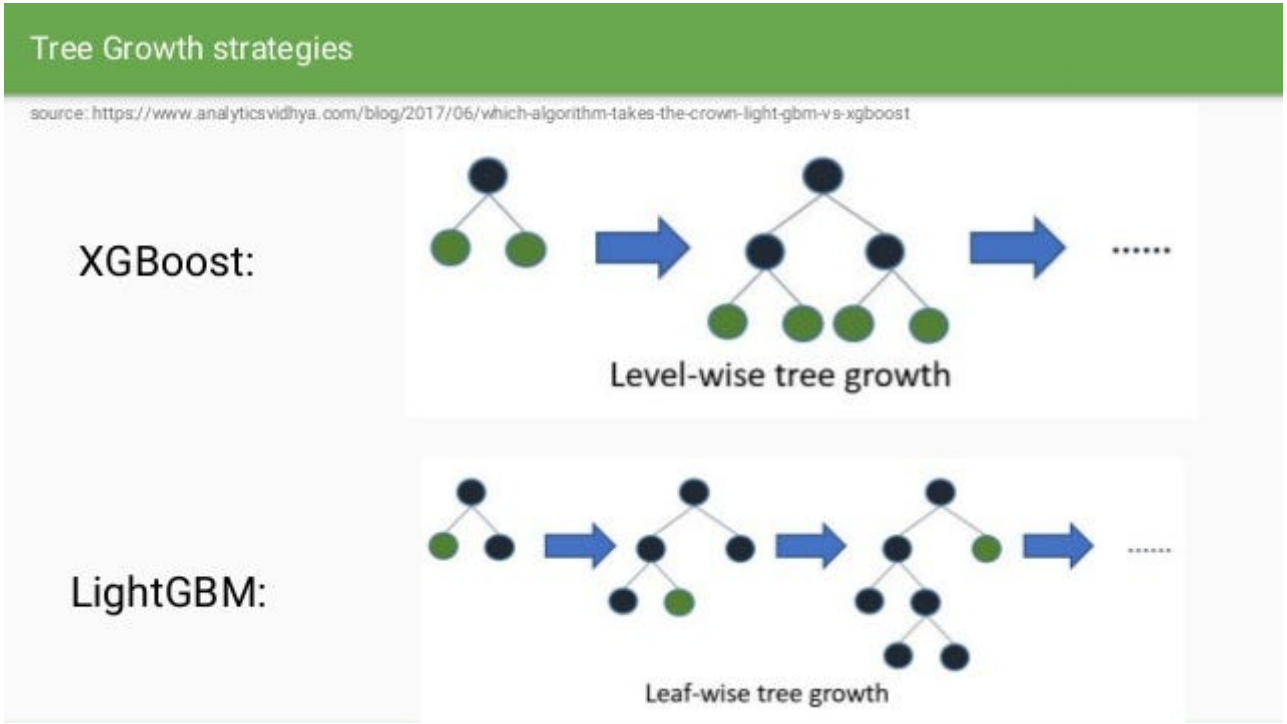


Image credit: https://www.slideshare.net/GabrielCyprianoSaca/xgboost-lightgbm

I'm not going to go into details of how these algorithms differ mathematically, but in general, they are able to ***prune the decision trees better while handling missing values + avoid overfitting at the same time.***

```
#XGBoost
import xgboost as xgb

xgb_model = xgb.XGBClassifier()
xgb_model.fit(x_a_train_rs_over_pca, y_a_train_over)
xgb_train_score = cross_val_score(xgb_model, x_a_train_rs_over_pca,
```

```
y_a_train_over, cv=5)
xgb_test_score = cross_val_score(xgb_model, x_a_test_rs_over_pca,
y_a_test_over, cv=5)

print(round(xgb_train_score.mean(),2))
print(round(xgb_test_score.mean(),2))

#LightGBM
import lightgbm as lgb

lgb_model = lgb.LGBMClassifier()
lgb_model.fit(x_a_train_rs_over_pca, y_a_train_over)
lgb_train_score = cross_val_score(lgb_model, x_a_train_rs_over_pca,
y_a_train_over, cv=5)
lgb_test_score = cross_val_score(lgb_model, x_a_test_rs_over_pca,
y_a_test_over, cv=5)

print(round(lgb_train_score.mean(),2))
print(round(lgb_test_score.mean(),2))
```

After computing, the train and set scores of each model are 72% & 73% (XGBoost) and 69% & 72% (LightGBM), which is relatively the same as Random Forest model computed above. However, we are still able to make further optimisations via Hyperparameter Tuning for these advanced models, but beware that it might take forever since XGBoost and LightGBM have longer runtime due to the complexity of their algorithm.

. . .

Voila! That's the wrap for this end-to-end project with regards to Classification! If you are keen to explore the entire code, feel free to check out my Github below:

Repository: https://github.com/andrewnguyen07/credit-risk-management
LinkedIn: www.linkedin.com/in/andrewnguyen07

Follow my Medium to keep posted on future projects coming up soon!

Data Science | Machine Learning | Hyperparameter Tuning | Crossvalidation

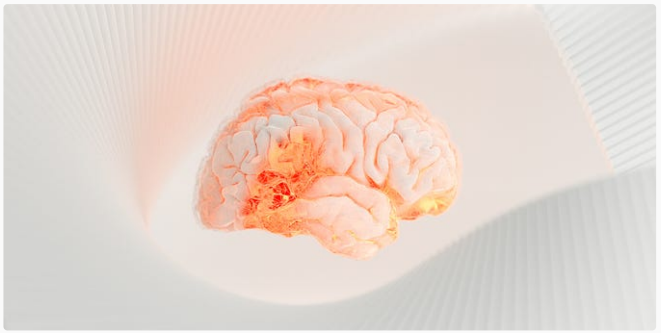
Written by Andrew Nguyen

Follow

116 Followers · Writer for Towards Data Science

A data analyst, without a higher degree, aspires to master data science skillsets by on-the-side projects.

More from Andrew Nguyen and Towards Data Science



Andrew Nguyen in Towards Data Science

Exchange Rate Prediction: Machine Learning with 5 Regression Models

Beatriz Stollnitz in Towards Data Science

How GPT Models Work

Learn the core concepts behind OpenAI's GPT models

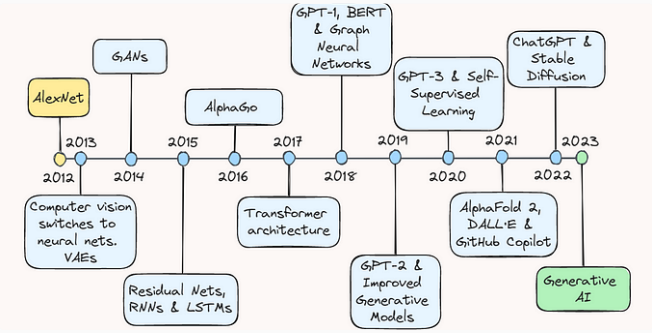
Today, I'll bring you through the 2nd part which deploys Machine Learning with the ai...

10 min read · May 29, 2020

59 2

14 min read · May 20

1.1K 13



Thomas A Dorfer in Towards Data Science

Ten Years of AI in Review

From image classification to chatbot therapy

✦ · 15 min read · May 23

1.3K 12

Andrew Nguyen in Towards Data Science

Credit Risk Management: EDA & Feature Engineering

How to clean and pre-process data using EDA and Feature Engineering techniques for...

10 min read · Aug 2, 2020

73

See all from Andrew Nguyen

See all from Towards Data Science

Recommended from Medium

Tracyrenee in MLearning.ai

How I used XGBoost to make predictions on a Paris House Pric...

This morning I went onto the Kaggle website and saw that the another playground...

✦ · 5 min read · Feb 7

11

Alexander Nguyen in Level Up Coding

Why I Keep Failing Candidates During Google Interviews...

They don't meet the bar.

✦ · 4 min read · Apr 13

4.5K 136

Lists

What is ChatGPT?

9 stories · 104 saves

Stories to Help You Level-Up at Work

19 stories · 100 saves

Staff Picks

348 stories · 111 saves

Paul Iusztin in Towards Data Science

A Framework for Building a Production-Ready Feature...

Lesson 1: Batch Serving. Feature Stores. Feature Engineering Pipelines.

✦ · 13 min read · Apr 28

595 8

Zach Qui... in Pipeline: Your Data Engineering Res...

3 Data Science Projects That Got Me 12 Interviews. And 1 That Got...

3 work samples that got my foot in the door, and 1 that almost got me tossed out.

✦ · 7 min read · Aug 30, 2022

3.2K 46

Arthur Mello in Level Up Coding

Exploratory Data Analysis: The Ultimate Workflow

Explore the true potential of your data with Python

· 16 min read · Apr 20

746 6

Matt Chapman in Towards Data Science

How I Stay Up to Date With the Latest AI Trends as a Full-Time...

No, I don't just ask ChatGPT to tell me

· 8 min read · May 1

1.92K 25

See more recommendations