Published in Towards Data Science

John Ade-Ojo Follow

Jan 16, 2021 · 11 min read ★ · Listen

ADVANCED REGRESSION TECHNIQUES

Predicting House Prices with Machine Learning

An end to end project for Kaggle's advanced regression techniques competition

Intro

I thought I'd try my hand at Kaggle's advanced regression techniques competition taking you along on the journey with me. If you're getting into machine learning and want to see a project end to end, please stick around. I will walk you through the steps I've taken and attempt to deliver a crash course in machine learning at the same time.

Objective & Data

The competition goal is to predict sale prices for homes in Ames, Iowa. You're given a training and testing data set in csv format as well as a data dictionary.

Training: Our training data consists of 1,460 examples of houses with 79 features describing every aspect of the house. We are given sale prices (labels) for each house. The training data is what we will use to "teach" our models.

Testing: The test data set consists of 1,459 examples with the same number of features as the training data. Our test data set excludes the sale price because this is what we are trying to predict. Once our models have been built we will run the best one the test data and submit it to the Kaggle leaderboard.

You can familiarize yourself with the data on the competition page.

Task: Machine learning tasks are usually split into three categories; supervised, unsupervised and reinforcement. For this competition, our task is supervised learning.

> *Supervised learning uses examples and labels to find patterns in data*

It's easy to recognise the type of machine learning task in front of you from the data you have and your objective. We've been given housing data consisting of features and labels, and we're tasked with predicting the labels for houses outside of our training data.

Tools

I used Python and Jupyter notebooks for the competition. Jupyter notebooks are popular among data scientist because they are easy to follow and show your working steps.

Libraries: These are fra y required tasks. I Implore any budding data scientists to familiarise themselves with these libraries:

Pandas — For handling structured data

Scikit Learn — For machine learning

NumPy — For linear algebra and mathematics

Seaborn — For data visualization

Project Pipeline

Generally speaking, machine learning projects follow the same process. Data ingestion, data cleaning, exploratory data analysis, feature engineering and finally machine learning.

The pipeline is not linear and you might find you have to jump back and forth between different stages. It's important I mention this because tutorials often make you believe the process is much cleaner than in reality. So please keep this in mind, your first machine learning project might be a mess.

Before we begin: I'll reiterate, machine learning is an iterative process and It's rarely straightforward! Please do not be discouraged if you find yourself lost in an ML project. Keep reading, keep experimenting, keep asking questions and one day it will click.

The rest of this article will talk through the stages of the project pipeline. Where useful, I will drop in code examples from python. The full end to end project is available to use and play with here. I'll share a link for this at the end of the article.

Data Cleaning

Kaggle does its best to provide users with clean data. However, we mustn't become lazy, there are always surprises in the data.

Warning! Don't skip the data cleaning phase, it's boring but it will save you hours of

Duplicates & NaNs: I st data, checked for missing or NaN (not a number) values. It's important to check for NaNs (and not just because it's socially moral) because these cause errors in the machine learning models.

Categorical Features: There are a variables that are marked as N/A when a feature of the house is nonexistent. For example, when no alley is present. I identified all the cases where this was happening across the training and test data and replaced the N/As with something more descriptive. N/As can cause errors with machine learning later down the line so get rid of them.

Date Features: For this exercise dates would be better used as categories and not integers. After all, it's not so much the magnitude that we care about but rather that the dates represent different years. Solving this problem is simple, just convert the numeric dates to strings.

Decoded Variables: Some categorical variables had been number encoded. See the example below.

```
MSSubClass: Identifies the type of dwelling involved in the sale.

        20      1-STORY 1946 & NEWER ALL STYLES
        30      1-STORY 1945 & OLDER
        40      1-STORY W/FINISHED ATTIC ALL AGES
        45      1-1/2 STORY - UNFINISHED ALL AGES
        50      1-1/2 STORY FINISHED ALL AGES
        60      2-STORY 1946 & NEWER
        70      2-STORY 1945 & OLDER
        75      2-1/2 STORY ALL AGES
        80      SPLIT OR MULTI-LEVEL
        85      SPLIT FOYER
        90      DUPLEX - ALL STYLES AND AGES
       120      1-STORY PUD (Planned Unit Development) - 1946 & NEWER
       150      1-1/2 STORY PUD - ALL AGES
       160      2-STORY PUD - 1946 & NEWER
       180      PUD - MULTILEVEL - INCL SPLIT LEV/FOYER
       190      2 FAMILY CONVERSION - ALL STYLES AND AGES
```

Image generated by author

The problem here is that the machine learning algorithm could interpret the

Exploratory Data Analysis (EDA)

This is where our data visualisation journey often begins. The purpose of EDA in machine learning is to explore the quality of our data. A question to keep in mind is; are there any strange patterns that leave us scratching our heads?

Labels: I plotted sales price on a histogram. The distribution of sale prices is right skewed, something that is expected. In your neighborhood it might not be unusual to see a few houses that are relatively expensive.

Here I perform my first bit of feature engineering (told you the process was messy). I'll apply a log transform to sales price to compress outliers making the distribution normal.

Outliers can have devastating effects on models that use loss functions minimising squared error. Instead of removing outliers try applying a transformation.

Plot histogram in Python

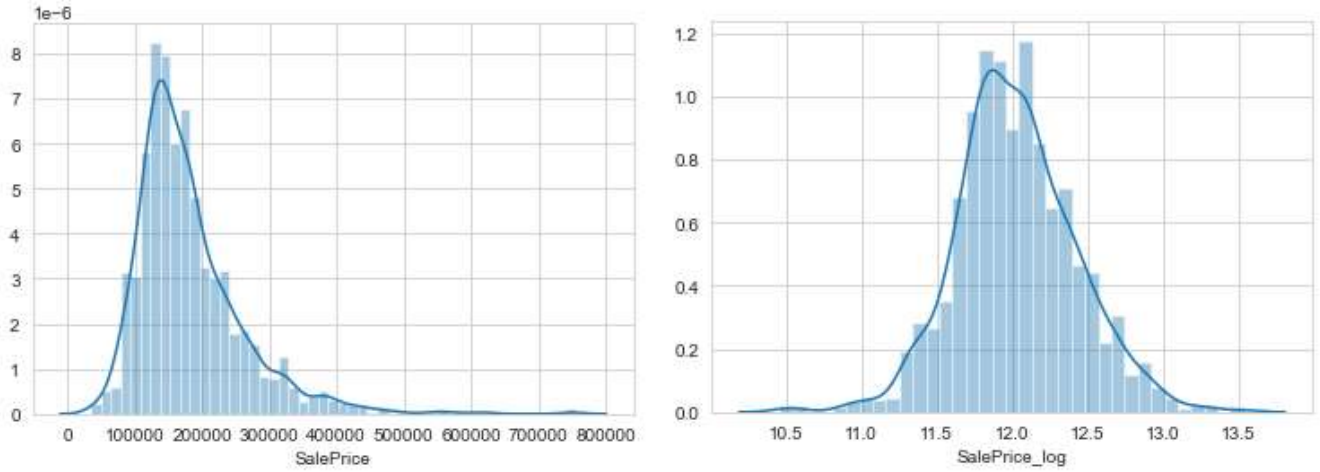
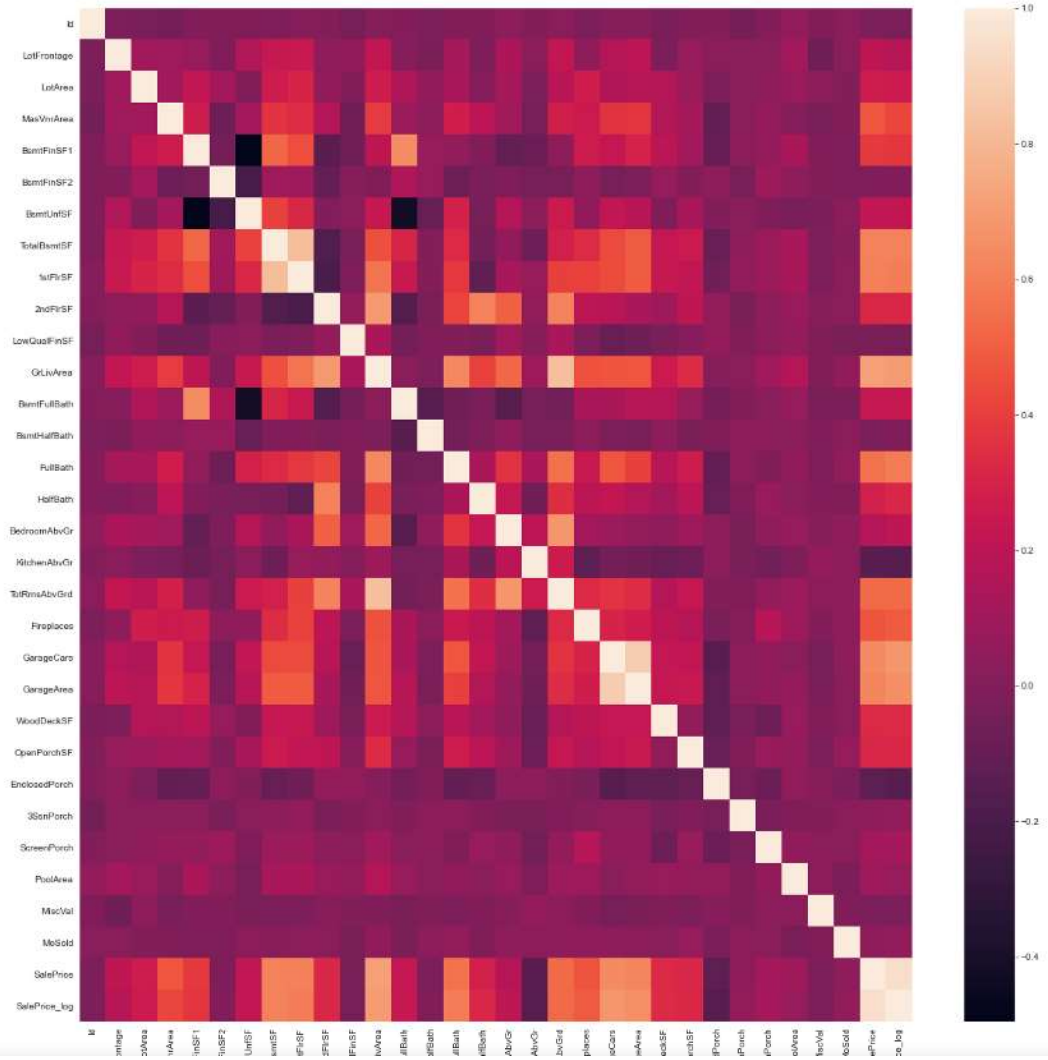


Image generated by author — SalePrice histogram before and after log transform

Correlations: It's often good to plot a correlation matrix to give you an idea of relationships that exist in your data. It can also guide your model building. For example, if you see a lot of your features are correlated with each other you might want to avoid linear regression.

plot Pearson's correlation in Python



The correlation measure ur case the lighter the square the stronger the correlation between two variables.

Features related to space such as lot frontage, garage area, ground living area were all positively correlated with sale price as one might expect. The logic being that larger properties should be more expensive. No correlations look suspicious here.

Categorical Relations: Sales price appears to be approximately normally distributed within each level of each category. No observations appear, untoward. Some categories contain little to no data, whilst other show little to no distinguishing ability between sales class. See full project on GitHub for data visualisation.

Feature Engineering

Machine learning models can't understand categorical data. Therefore we will need to apply transformations to convert the categories into numbers. The best practice for doing this is via one hot encoding.

Pro tip! — Make sure you use Sci Kit Learn's OneHotEncoder and **AVOID** panda's get_dummies. You will run into a whole host of problems if the levels of categories in your training and test data set differ when using get_dummies.

OneHotEncoder solves this problem with the options that can be set for categories and handling unknowns. It's slightly harder to use but definitely necessary for machine learning.

Here's a brilliant article with python examples for the OneHotEncoder.

Machine learning

I follow a standard development cycle for machine learning. As a beginner or even a pro, you'll likely have to go through many iterations of the cycle before you are able to

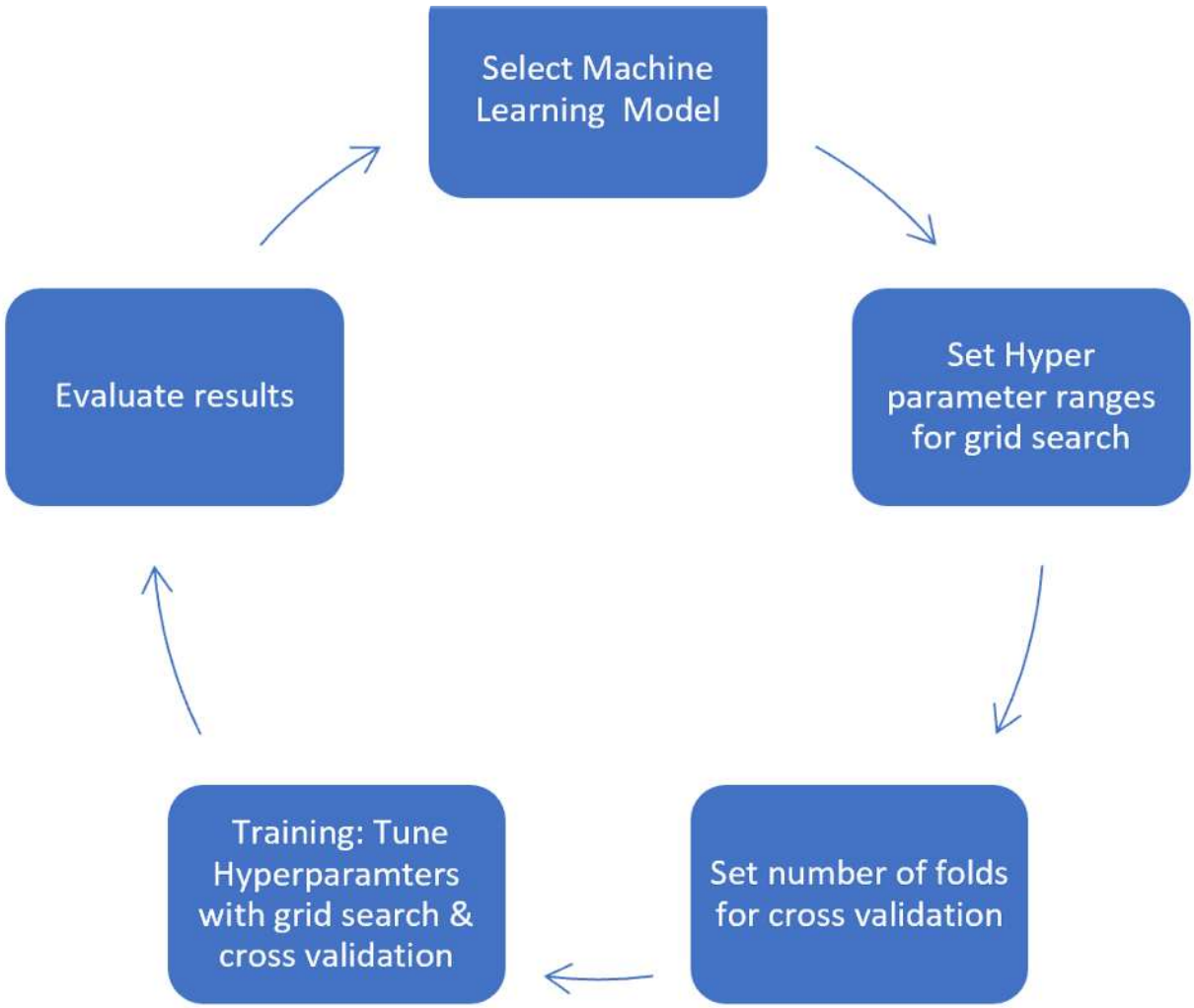


Image by author: Machine learning model development cycle

Model Selection

As mentioned at the start of the article the task is supervised machine learning. We know it's a regression task because we are being asked to predict a numerical outcome (sale price).

Therefore, I approached this problem with three machine learning models. Decision tree, random forest and gradient boosting machines. I used the decision tree as my baseline model then built on this experience to tune my candidate models. This approach saves a lot of time as decision trees are quick to train and can give you an idea of how to tune the hyperparameters for my candidate models.

Model mechanics: I will not go into too much detail about how each model works

> *Decision Tree — A tree al* *l patterns in data by learning decision rules.*
>
> *Random Forest — A type of bagging method that plays on 'the wisdom of crowds' effect. It uses multiple independent decision trees in parallel to learn from data and aggregates their predictions for an outcome.*
>
> *Gradient Boosting Machines — A type of boosting method that uses a combination of decision tree in series. Each tree is used to predict and correct the errors by the preceding tree additively.*

Random forests and gradient boosting can turn individually weak decision trees into strong predictive models. They're great algorithms to use if you have small training data sets like the one we have.

Training

In machine learning training refers to the process of teaching your model using examples from your training data set. In the training stage, you'll tune your model hyperparameters.

Before we get into further detail, I wish to briefly introduce the bias-variance trade-off.

> *Model Bias — Models that underfit the training data leading to poor predictive capacity on unseen data. Generally, the simpler the model the higher the bias.*
>
> *Model Variance — Models that overfit the training data leading to poor predictive capacity on unseen data. Generally, the more complexity in the model the higher the variance.*

Complexity can be thought of as the number of features in the model. Model variance and model bias have an inverse relationship leading to a trade-off. There is an optimal point for model complexity that minimizes the error. We seek to establish that by tuning our hyper parameters.

Here's a good article to help you explore this stuff in more detail.

max_depth — The maxi ion tree.

max_features — The size of the subset of features to consider for splitting at a node.

n_estimators — The number of trees used for boosting or aggregation. This hyperparameter only applies to the random forest and gradient boosting machines.

learning_rate — The learning rate acts to reduce the contribution of each tree. This only applies for gradient boosting machines.

> *Decision Tree — Hyperparameters tuned are the max_depth and the max_features*
>
> *Random Forest — The most important hyperparameters to tune are n_estimators and max_features [1].*
>
> *Gradient boosting machines — The most important hyperparameters to tune are n_estimators, max_depth and learning_rate [1].*

Grid search: Choosing the range of your hyperparameters is an iterative process. With more experience you'll begin to get a feel for what ranges to set. The good news is once you've chosen your possible hyperparameter ranges, grid search allows you to test the model at every combination of those ranges. I'll talk more about this in the next section.

Cross validation: Models are trained with a 5-fold cross validation. A technique that takes the entirety of your training data, randomly splits it into train and validation data sets over 5 iterations.

You end up with 5 different training and validation data sets to build and test your models. It's a good way to counter overfitting.

More generally, cross validation of this kind is known as k-fold cross validation. More on k-fold cross validation here.

Implementation: SciKit Learn helps us bring together hyperparameter tuning and cross validation with ease in using GridSearchCv. It gives you options to view the results of each of your training runs


Evaluation

This is the last step in the process. Here's where we either jump with joy or pull our hair with frustration (just kidding, we don't do that…ever). We can use data visualisation to see the results of each of our candidate models. If we are not happy with our results, we might have to revisit our process at any of the stages from data cleaning to machine learning.

Predictably this was our cision tree score -0.205 NRMSE. Tuning the hyperparameters didn't appear to make much of a difference to the model's however it trained in under 2 seconds. There is definitely some scope to assess a wider range of hyperparameters.

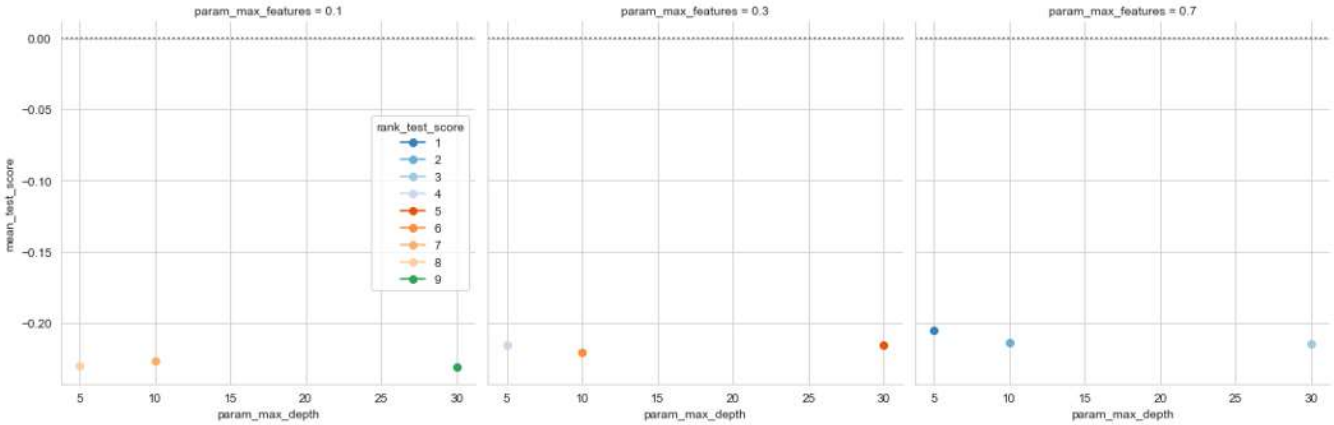


Image by author: NRMSE plot for decision tree

Random Forest

Our random forest model was a marked improvement on our decision tree with a NRMSE of -0.144. The model took around 75 seconds to train.

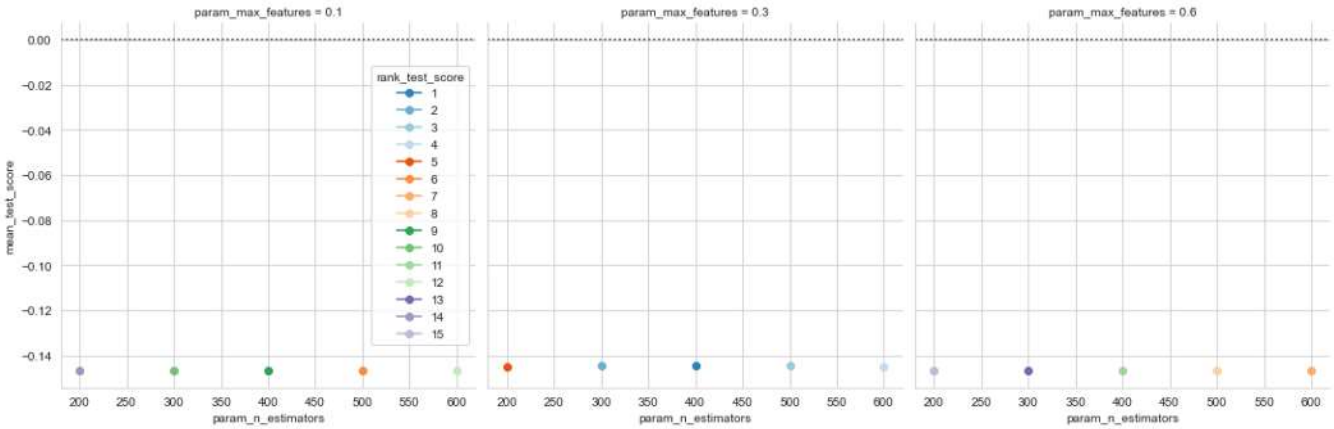


Image by author: NRMSE plot for random forest

Gradient Boosting Machine

This was our best performer by a significant amount with a NRMSE of -0.126. The hyperparameters significantly impact the results illustrating that we must be incredibly careful in how we tune these more complex models. The model took around 196

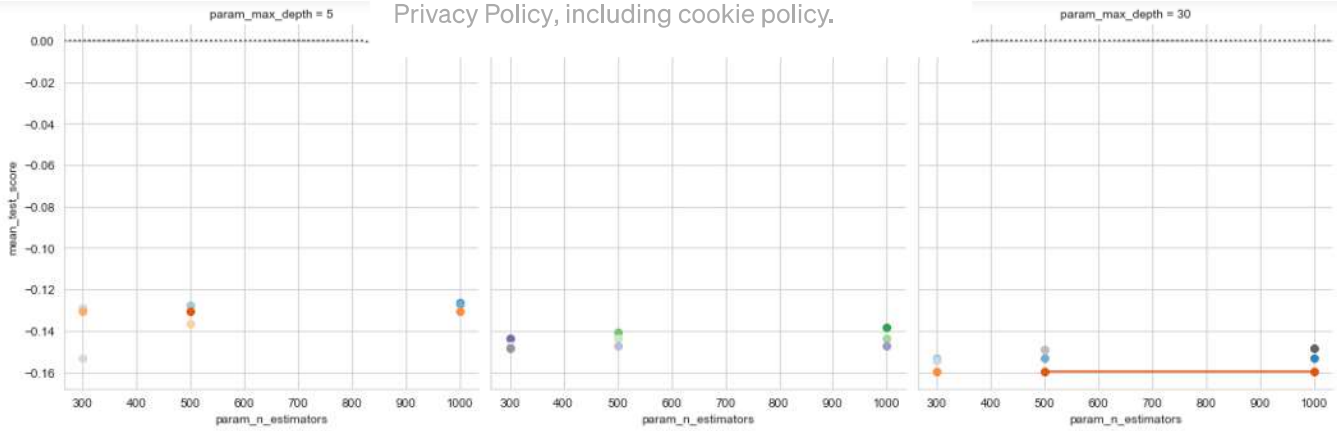


Image by author: NRMSE plot for gradient boosting machine full version available in GitHub

Competition Result

I tested the best performing model on the Kaggle test data. My model put me in the top 39% of entrants (at the time of writing). This isn't a bad result, but it can definitely be improved. Here are some ways we could do that:

Categorical variables: Some of our categorical features in the data have a high cardinality. Tree models can be biased to these features because of this. We might be able to improve model performance by recategorizing these high dimensional features into lower dimensions.

Hyperparameter tuning: We can look to widen our solutions space for our hyperparameters in the hope of finding a more optimal position. Be warned, this will require heavy computation power if you're just working on your laptop.

I hope this article has helped you understand more about machine learning. As always, please do not be discouraged everything is practice and patience.

🚀 Here's a link to the Jupyter Notebook containing the code. You can try it for yourself

References

[1] Burkov, A (2019). The Hundred Page Machine Learning Book, pp.84–85