Udacity MLND Program

August 11, 2019

1 Machine Learning Engineer Nanodegree

1.1 Project Proposal Author

Adriano Falsarella Monte

1.2 Course Mentor

Fernando Marcos Wittmann

1.3 Capstone Proposal

1.3.1 Domain Background

What is a fair offer to place when selling or buying a home? How to efficiently come to that fair price? The real estate area has been adopting machine learning to analyse historical data in that market and use several criterias to find a well balanced model to help predicting house prices.

Personally, my main motivation for this particular problem is that I've talked to Loft's CTO in a local event at Sao Paulo - Brazil, and I've found out that Loft uses machine learning to predict the house prices in their business, and it is a core value. I already knew about the house prices kaggle getting started competition, but it hadn't took my attention until that moment. I realized it wasn't just a getting started challenge, it is actually a real world problem that real world companies need to solve.

1.3.2 Problem Statement

Given a dataset publicly available at Kaggle of homes in Ames - Iowa, having 79 house features, and containing 1460 entry points for training and 1459 entry points for testing, build the best model to predict house sale prices. A great model will predict the house prices as closest as possible to its actual labelled price. The metric used as evaluation is the root mean squared error between the predicted log price and the actual labelled log price, so we have an error scaling equallity.

1.3.3 Datasets and Inputs

It is reasonable that to predict a house price, a model should be trained with historical data about houses features and the price that they were sold. Despite that the houses in Ames - Iowa are different from the houses here in Brazil, the dataset provided in Kaggle's House Prices competition

would still get the essence of the work that should be done if the data was from a different location since most of the features would also be applicable in different contexts.

The dataset is very feature-detailed, providing several characteristics about the dataset's houses, such as: the sale price in dollars (target variable), the date, type, and condition of the sale, the building class, the zoning class, the lot area in square feet, the street acccess type and road proximity, the property configuration, shape, and slope, neighborhood quality, dwelling type and style, the property's condition and material quality, the construction and remodeling date, infos about the property's roof, foundation, electrical system, exterior, masonry veneer, garage, pool, porch, deck, and basement, the property's heating, fireplace, and air conditioning quality, infos about the rooms like bathrooms, bedroom, and kitchen, and finally how functional the home is overall.

Lots of the features are categorical, some of them are nominal like types, styles, and shapes, and they should be one-hot-encoded, but there are also some categorical values that are ordinal, that may be ordered and transformed into numberical values, like class, grades, quality. One point of attention is that some nominal features may look ordinal, but we shouldn't be biased in this decision, because not always a category that looks better than another would mean that it is in fact better in the datapoints.

There are also lots of numerical values, some of them are continuous like dollars, area, distance, and some of them are discrete like quantity, year, month. They may have their values scaled, or possibly put into categories like 'Close' vs 'Far' or 'Old' vs 'New' and then one-hot-encoded.

1.3.4 Solution Statement

In the conversation I had with Loft's CTO, he told me that with the house's dataset, they make some clusters, and for each cluster they apply a separate linear regression. I loved how it embraces both unsupervised and supervised learning to get a refined result, and I want to try it out, exercising both concepts in this course Capstone proposal.

The idea is to get the transformed data as described in the section above, find the best number of clusters in it, and then for each cluster we train the best linear regression model. With the trained models, to make a prediction, we need to first predict which cluster it is better represented, and then use that cluster's trained linear regression model to predict the house price.

To make all these training and predicting processes replicable, we'll use a fixed training and testing data as provided by Kaggle. Furthermore, in the training, we should always set a constant `random_state` when applicable.

1.3.5 Benchmark Model

- Naive Prediction

A simple Naive Predictor should be built to have a minimum base benchmark to beat with a more sophisticated method. The naive prediction can be calculated by the Lot Area of the predicting house times the dataset's mean Sale Price per Lot Area square feet, which can be calculated dividing the dataset's mean Sale Price, by the dataset's mean Lot Area.

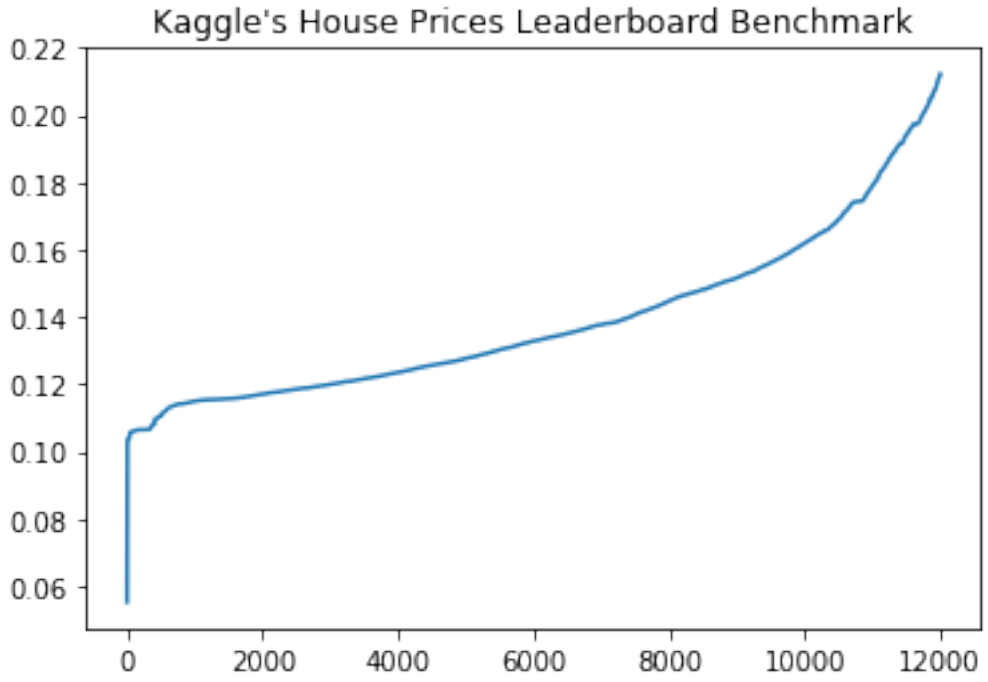
- Kaggle Competition

Analysing the Kaggle's House Prices Leaderboard data, after removing some last mile submission outliers, we have a good overview about scores over existing solution methods resolving this same problem, as we can see below.

The main benchmark to achieve are the mean of `0.138870` and then the median of `0.132855` - now we have a mission to accomplish, Yay! Furthermore, I'd consider a great result if we could get under the first quartile of `0.120000`, which would also put us on Top #1000! Finally, if getting under `0.110000` I'll pop a champagne and post the photo!

```
In [1]: import pandas as pd
        import matplotlib.pyplot as plt
        %matplotlib inline

        leaderboard = pd.read_csv('house-prices-leaderboard.csv')

        # sort scores, remove last mile outliers
        ranking = pd.DataFrame(leaderboard['Score'].sort_values()[:12000])

        plt.title("Kaggle's House Prices Leaderboard Benchmark")
        plt.plot(ranking.values)

        ranking.describe()
```

```
Out[1]:               Score
        count  12000.000000
        mean       0.138870
        std        0.023893
        min        0.055330
        25%        0.120000
        50%        0.132855
        75%        0.151622
        max        0.212150
```



1.3.6 Evaluation Metrics

There are many evaluation methods available in scikit learn in the `sklearn.metrics` package. As per the proposed solution, we'll have to take at least one clustering metric and at least one regression metric.

For the clustering part the idea is to use the silhouette score as the metric to find the best number of clusters, since it's the one that we've used most in the course. This evaluation method is provided by scikit learn in `sklearn.metrics.silhouette_score` module.

For the regression part the suggestion is to use the mean squared log error as the metric to find the best regression model, since it's the one that Kaggle will use in the Leaderboard ranking, but mainly because the squared error is aligned with the business objectives (we should predict prices with the least error possible between the prediction and the actual price). This evaluation method is also provided by scikit learn in `sklearn.metrics.mean_squared_log_error` module.

1.3.7 Project Design

Considering all projects we've done in the whole course, and with the help of some references such as a Machine Learning project checklist and the Udacity MLND Capstone Report itself, check out below the complete Capstone Project Design and Workflow.

- Exploratory Data Analysis

Understand each feature type (nominal, ordinal, discrete, continuous, date, time values and corresponding unit), statistics (mean, median, mode, quartiles, standard deviation, distribution, correlation, importance), and abnormalities (missing values, outliers), discussing and visualizing relevant findings.

- Data Processing

Make relevant feature transformations to address each feature type, handle all features abnormalities, make feature aggregation if any looks promising, and possibly make feature selection and/or dimensionality reduction, depending on the previous findings. Every transformantion must also be applied to the test data.

- Naive Predictor

Build the Naive Predictor, so we can have the base benchmark before continuing with more sophisticated approaches. The Naive Predictor should be naïve, yet reasonable, and as previously described, it can be achieved with simple math, as exemplified in the following pseudocode:

```
mean_sale_price = training_dataset['sale_price'].mean()
mean_lot_area = training_dataset['lot_area'].mean()
mean_sale_price_per_loat_area_square_feet = mean_sale_price / mean_lot_area

naive_predictor_price =
    predicting_house['lot_area'] * mean_sale_price_per_loat_area_square_feet
```

This predictor may be submitted to the Kaggle's competition to understand how well it performs by itself.

- Clustering Model Selection

Before clustering, perform a PCA transformation to avoid the curse of dimensionality (we may review this whole section again with different number of components).

Given the transofrmed data, train models with different algorithms (K-Means, GMM, and Random Forest) and evaluate them through visualization and `silhouette_score` to have a feeling on the performance of each algorithm (do that for each number of clusters in a reasonable range of values).

Select the most appropriate clustering algorithm, and find the most reasonable number of clusters through the `silhouette_score`. With the final clustering model, predict the cluster of each house in both train and test data.

- Regression Model Selection

For each cluster found, train models with different algorithms (Logistic Regression, Stochastic Gradient Descent, XGBoost Regressor) and benchmark each of them with K-Fold cross-validation using `mean_squared_log_error` to measure and validate the performance of each algorithm, making sure that they generalize well.

Also perform hyperparameters grid search with both GridSearchCV and RandomizedSearchCV, and if applicable do some quick round of EDA and/or data processing, and rerun the benchmark with the changes in order to find the best model for each cluster. Reprocess everything after having inserted an additional 10% of noisy data to assert the robustness of the solution.

- Results

Visualize the results in the training dataset, comparing the predicted value with the actual value, the error and the squared error of each datapoint.

Check how well the proposed solution is performing. Compare its results with the Naive Predictor, it is expected to be much better. With the promising solution, make the prediction for the competition itself with the test data, submit the results, and compare it with the expectations.

Have we got the results to pop a champagne? Elaborate how well it has been, if possible, review the entire process in order to make some adjustments and possibly get a better result. What worked well and what didn't? What was difficult, what was easier than expected, what was interesting and what was fun? Provide any improvements that could be done and alternatives that could be further explored.

Make the final considerations regarding the capstone, the course, and the nanograduation.