Predicting demand for an online advertisement

Capstone Project

Ajay Kumar Misra

Machine Learning Engineer Nanodegree

June 22th, 2018

I. Definition

Project Overview

When selling used goods online, a combination of tiny, nuanced details in a product description can make a big difference in drumming up interest. But, even with an optimized product listing, demand for a product may simply not exist–frustrating sellers who may have over-invested in marketing.

https://www.kaggle.com/c/avito-demand-prediction/data

Russia's largest classified advertisements website Avito is deeply familiar with this problem. Sellers on their platform sometimes feel frustrated with both, too little demand (indicating something is wrong with the product or the product listing) or too much demand (indicating a hot item with a good description was underpriced).

Problem Statement

Avito is challenging us to predict demand for an online advertisement based on its full description (title, description, images, etc.), its context (geographically where it was posted, similar ads already posted) and historical demand for similar ads in similar contexts. With this information, Avito can inform sellers on how to best optimize their listing and provide some indication of how much interest they should realistically expect to receive.

The solution outline will be as follows:

1. Preliminary research: Consult the relevant literature to assemble a list of relevant features to use.

2. Data wrangling: Download train and test dataset.

3. Exploratory data analysis: Produce relevant statistics and visualizations to characterize the data.

4. Data preprocessing: Prepare the dataset for application of a machine learning model – clean missing or invalid values, calculate the deal probability.

5. ML model implementation and refinement: Define model metrics, implement relevant code and perform hyperparameter optimization. Once optimized values are found, estimate model performance with cross-validation.

Metrics

Metric used would be root mean squared error. RMSE is defined as:

RMSE=(1/*n*∑*i*=1to*n*(*yi*−*y*^*i*)^2)^1/2

where y hat is the predicted value and y is the original value.

II. Analysis

Data Exploration

To make it easier to download the training images, Avito has added several smaller zip archives that hold the same images as train_jpg.zip. We are free to use either train_jpg.zip or the smaller zip archives.

- **train.csv** - Train data.
  - item_id - Ad id.
  - user_id - User id.
  - region - Ad region.
  - city - Ad city.
  - parent_category_name - Top level ad category as classified by Avito's ad model.
  - category_name - Fine grain ad category as classified by Avito's ad model.
  - param_1 - Optional parameter from Avito's ad model.
  - param_2 - Optional parameter from Avito's ad model.
  - param_3 - Optional parameter from Avito's ad model.
  - title - Ad title.
  - description - Ad description.
  - price - Ad price.
  - item_seq_number - Ad sequential number for user.
  - activation_date- Date ad was placed.
  - user_type - User type.
  - image - Id code of image. Ties to a jpg file in train_jpg. Not every ad has an image.
  - image_top_1 - Avito's classification code for the image.

    - deal_probability - **The target variable**. This is the likelihood that an ad actually sold something. It's not possible to verify every transaction with certainty, so this column's value can be any float from zero to one.
- **test.csv** - Test data. Same schema as the train data, minus deal_probability.
- **train_active.csv** - Supplemental data from ads that were displayed during the same period as train.csv. Same schema as the train data minus deal_probability, image, and image_top_1.
- **test_active.csv** - Supplemental data from ads that were displayed during the same period as test.csv. Same schema as the train data minus deal_probability, image, and image_top_1.
- **periods_train.csv** - Supplemental data showing the dates when the ads from train_active.csv were activated and when they were displayed.
    - item_id - Ad id. Maps to an id in train_active.csv. IDs may show up multiple times in this file if the ad was renewed.
    - activation_date - Date the ad was placed.
    - date_from - First day the ad was displayed.
    - date_to - Last day the ad was displayed.
- **periods_test.csv** - Supplemental data showing the dates when the ads from test_active.csv were activated and when they were displayed. Same schema as periods_train.csv, except that the item ids map to an ad in test_active.csv.
- **train_jpg.zip** - Images from the ads in train.csv.
- **test_jpg.zip** - Images from the ads in test.csv.
- **sample_submission.csv** - A sample submission in the correct format.
- **train_jpg_{0, 1, 2, 3, 4}.zip** - These are the exact same images as you'll find in train_jpg.zip but split into smaller zip archives so the data are easier to download.

Statistical overview of the raw data

The above mentioned datasets were downloaded from the https://www.kaggle.com/c/avito-demand-prediction/data. train/test jpg zip files were more than 9 GB, so I did not include them in the submission.
train dataset: 1503424 rows and 18 columns
test dataset: 508438 rows and 17 columns
periods_train dataset: 16687412 rows, 4 columns
periods_test dataset: 13724922 rows, 4 columns
All the columns in the datasets are in Russian language, so we had to use the Yandex Translator to convert the names into English wherever needed. We took help from https://www.kaggle.com/sudalairajkumar/simple-exploration-baseline-notebook-avito kernel.

Outliers and abnormal values

In the train dataset we found the following features with missing values:
param_3, param_2, description, image, image_top_1, price, and param_1
In the periods_train dataset, we found activation_date column with missing values.
Also following features were found to be categorical variables, which need to be converted:
region, city, parent_category_name, category_name, user_type, param_1, param_2, param_3

Exploratory Visualization

Deal Probability is our target variable with float value between 0 and 1. Histogram and Scatter Plot shows that almost 1000000 advertisements sold nothing, while few had a probability of 1, and the rest are in the middle.
From the Pie Chart we can clearly see that out of all the ads which have any chances of selling something, 88% of the ads (deal_probability >= 0.5) have greater than 50% probability, while the rest have less than 50% probability.
Pie Charts and Box plot show tha Kresnodar Krai, Sverdlovsk oblast, and Rostov oblast were the top 3 regions to display the ads.
Displaying parent category with Pie Charts, we found that 46.4% of the ads were for Personal belongings, 11.9% for home and garden and 11.5% for consumer electronics.
Same way we found that top 3 cities for the ads are Krasnodar, Ekaterinburg, and Novosibirsk.
The top 3 categories for the ads are Clothes, shoes, accessories followed by Children clothing and footwear and then goods for children and toys.
Most of the titles have one, two or 3 words in them.

Algorithms and Techniques

My solution would involve inputting the data into XGBoost Model. For each item_id in the test set, we will predict a probability for the deal_probability. These predictions must be in the range [0, 1].

For our problem, we will use a gradient-boosted decision tree ensemble as our classifier. Decision tree ensembles are basically combinations of decision trees. “Gradient boosting” refers to a method by which the optimized tree parameters are learned. Generally speaking, the method begins with a tree of depth 0, and then iteratively tries adding splits using the various features inherent in the data, and selects the best split. During this training process, the tree is also regularized to control for complexity. Specifically, we will use the XGBoost implementation, which is considered highly efficient and is a top-performing classifier in many Kaggle competitions. This classifier has many hyperparameters, and we will manually set the minimal amount required for a preliminary solution, and perform an optimization procedure using a grid search with k-fold cross validation to determine the rest. Finally, we will use k-fold cross validation to analyze the optimized model performance.
Default values: The classifier hyperparameters will initially be set using the default setting, except for the following (which remained constant throughout the optimization process):
• ***objective***: Defines the type of classification. Set to regular:logistic

• ***eta***: 0.3.

• ***seed***: Defines a setting for the random number generator. Need to ensure repeatable results. Set to 99.
• ***tree_method***: “hist”

• ***grow_policy***: “lossguide”

• ***max_leaves***: 1400

• ***max_depth***: 0

• ***subsample***: 0.9

• ***colsample_bytree***: 0.7

• ***colsample_bylevel***: 0.7

• ***min_child_weight***: 0

• ***alpha***: 4

• ***eval_metric***: 'rmse'

• ***random_state***: 99

• ***silent***: True

Benchmark

At this point of time, the random guessing would be the benchmark for this project. Assumption is that deal probability prediction would be either all 0 (no ad sold anything) or all 1 (all ads sold something). Linear Regression would be used as benchmark model.

III. Methodology

Data Preprocessing

The data preparation stage was comprised of the following step:
1. "Data wrangling" – obtaining the relevant data from the database into a pandas dataframe format.

2. Clean up missing/invalid values.

3. Prepare the remaining data for classification:

    a. Scale the numerical features and remove outliers.

    b. Encode categorical features.

The first "data wrangling" step was to read the dataset from the local directory into a pandas dataframe with all the relevant features. Due to size constraints, we read only train, test, periods_train, periods_test datasets, and ignored the train_active, test_active, and train_jpg.zip datasets.

The next step was to clean up the missing and invalid values:
• Did not find any invalid values in the dataset.

• Missing values were found in the param_3, param_2, description, image, image_top_1, price, and param_1 columns, so removed them before passing the data to the model..

Preprocessing the clean dataset:
The **deal_probability** column was used to define the label dataframe. At this point, several features could be removed as they would not be used for the predictive model. These were **item_id, user_id, title, description, activation_date, image, region_en, parent_category_name_en, category_name_en, price_new, deal_probability**.
The remaining features were split into numerical and categorical features. All numerical features were scaled uniformly (mean removed and standard deviation normalized to 1) and outliers were then identified and removed using Tukey's test, i.e. measures that were outside of the range [Q1-1.5*(Q3-Q1), Q3+1.5*(Q3-Q1)] were considered outliers. The categorical features region, city, parent_category_name, category_name, user_type, param_1, param_2, param_3 were transformed into label encoded vectors.

Implementation

Dataset extraction:

The relevant data from train, test, periods_train, periods_test data tables of Avito was imported directly as a pandas dataframe.

Data preprocessing:

All data preprocessing and label calculations described in the preprocessing section were performed using pandas native functionality.

Classifier and metric definition:

I used scikit-learn tools and methodology throughout the classification process. The classifier used is the eXtreme Gradient Boosted decision tree ensemble classifier, which is not included in the scikit-learn library, but rather has an scikit-learn API allowing it to be used almost identically as other scikit-learn classifiers. Hyperparameter optimization and metrics were entirely implemented through scikit-learn. Hyperparameters were explored using grid search with 5-fold cross-validation (GridSearchCV).

Refinement

Hyperparameter optimization for our classifier was performed by a multi-step grid search procedure with 5-fold cross validation at each step (using the scikit-learn implementation GridSearchCV) across the hyperparameter space. Briefly, a parameter range was set for each step and the classifier performance (averaged over a cross-validation split) was measured at

each point in the range. The top performing combination of hyperparameters was then used to inform the next step, either by evaluating a narrower range around the same hyperparameters, or used as-is for a new set of hyperparameters. The order of hyperparameters to be optimized (and their definitions) is as follows:

1. *n_estimators* - The number of trees used.
2. *max_depth* - Maximum tree depth for base learners.
3. *min_child_weight* - Minimum sum of instance weight(hessian) needed in a child.
4. *gamma* - Minimum loss reduction required to make a further partition on a leaf node of the tree.
5. *colsample_bytree* - Subsample ratio of columns when constructing each tree.
6. *Subsample* - Subsample ratio of the training instance.
7. *Alpha* - L1 regularization term on weights.

IV. Results

Model Evaluation and Validation

After hyperparameter optimization, the classifier was run over the entire cleaned dataset in a 5-fold stratified cross-validation process, and set up to provide prediction probabilities. The cross-validation process resulted in relatively consistent performance overall, and the model can be said to generalize quite well to unseen data.

Justification

To compare our model's performance to our predefined benchmark, it is safe to say that our classifier significantly outperforms the benchmark at predicting deal probability.

V. Conclusion

Free-Form Visualization

Last figure reveals some interesting insights about the features used in our model. Figure shows the feature importance, calculated by "weight", i.e. the number of times each feature is used in a decision tree. The more times a feature is examined within a decision tree, the more important it is deemed. The top features in terms of importance are the price and city. Figure shows all the features in order of their importance.

Reflection

This project entailed exploring a large database with many different types of values (static numerical and categorical features, date-time objects, images), and wrangling the relevant features into a easily digestible pandas dataframe before applying fairly standard machine-learning methodology to obtain and optimize a predictive model. For me picking up the benchmark and metrics presented the most significant challenge.
The nature of the data and the required preprocessing steps resulted in a significant reduction in size of the relevant data in the final dataset. Nevertheless, the resulting model performed better than I expected. This is an exciting result, as real-world applications of such a model could have significant financial consequences.

Improvement

If the time permits, we will try to improve the model by:

- Adding more new features
- Tuning the parameters as these are not tuned
- Blending with other models.