Predict deal probability for an online advertisement

by: Itai Yifrach, Stephan Goldberg, Ilai Fallach, Matan Lieber

Table of Contents

Background

Avito.ru is a Russian classified advertisements website with sections devoted to general good for sale, jobs, real estate, personals, cars for sale, and services.

When selling used goods online, a combination of tiny, nuanced details in a product description can make a big difference in drumming up interest. Details like:

Well-Taken, Authentic Photos

Too Glossy

Authentic

Poor Quality

Believable and Informative Description Copy

Description:
AMAZING WATCH FOR SALE!!!!

DON'T MISS THIS DEAL. IT'S THE DEAL OF THE CENTURY!!

Unlikely

Description:
I have an adjustable Chaleur D'Animale Watch for sale.

It's never been worn and still in the original box. Battery included.

Informative

Description:
fancy watch for sale

no low ball offers, cash and carry

Poor Quality

And, even with an optimized product listing, demand for a product may simply not exist–frustrating sellers who may have over-invested in marketing.

Avito, Russia's largest classified advertisements website, 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).

The problem

Predict deal probability 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.

Evaluation

Root Mean Squared Error (RMSE) as evaluated by Avito in the competition.

Dataset Description

- train.csv - Train data (1503424 rows)
    - 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 (508438 rows). Same schema as the train data, minus deal_probability.
- train_jpg.zip - Images from the ads in train.csv.
- test_jpg.zip - Images from the ads in test.csv.

Dataset Analysis

Data Understanding

This are some of the insights we gained from exploring the given datasets:

- We found the most important correlations in the dataset. for example we saw that price and image_top_1 were the most correlative features with the target feature.

  This knowledge mostly helped us in the feature selection stage.
- We found the number of unique values for some of the features.

  We used this information to determine whether a feature should be categorical or numerical (image_top_1, item_seq_number)
- We visualized the distribution of several features.

  It helped us decide on feature transformations (log_price) and it also helped us come up with some additional features(weak/power user, count features).

More details and code can be found in the data_exploration notebook.

Our Solution

Feature Extraction

The feature engineering was composed of 4 main steps:

- Data Imputation
- Text Feature Engineering (#TODO STEPHAN)
- Image Feature Engineering
- Categorical Feature Engineering

One of the major problem was the amount and size of the data - we had to process 1.5mil images with size of 60GB. To overcome this, we used Intel DevCloud - a free cluster for our compute needs.

Text

TODO STEPHAN

Image

The quality of the advertisement image significantly affects the demand volume on an item. We implemented some ideas which can be used to create new features related to images. These features are indicators for the image quality:

- Image Size
- Image Sharpness
- Image Luminance
- Image Colorfulness
- Image Average Color
- Image Dominant Color
- Image Keypoints

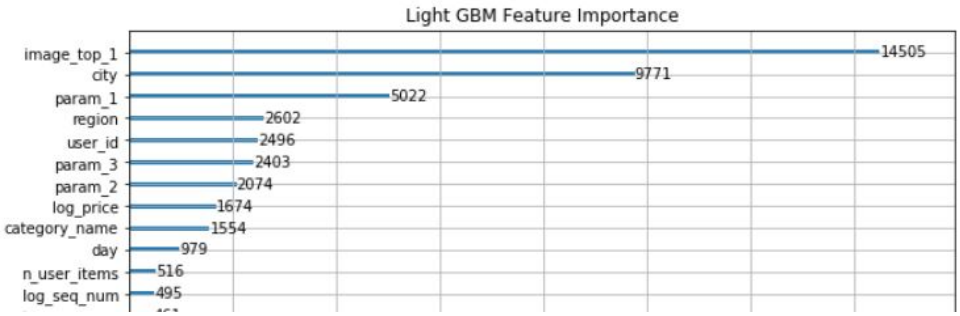
Image Classification with Deep Learning - The assumption is that the image classification accuracy score will reflect how clear it is for a human to identify it, and affect the chance of

buying it. We used Keras, which provides pre-trained deep learning models. Specifically, we used three pre-trained deep learning models - ResNet50, InceptionV3 and Xception. From each model, we extracted the top-3 scores with probabilities, and averaged them to a feature named: “Image Confidence”. more details and code can be found at the relevant notebook.

Models

LGBM - Decision tree boosting

We’ve decided to begin with a random forest variation (LGBM) as a starter model. This decision was mainly based on the simplicity of using the model (the data pre-processing required) and the fact that this model is very descriptive in terms of the importance it gives to certain features. Here’s an example of one of the model’s auxilary outputs:



outputs as the one above helped us better understand the significance of each feature we were working with and focusing on the most important features by better pre-process them (better complete missing values, normalize them etc...). For example after witnessing on multiple runs with different hyperparameters the importance of “image_top_1” and “price” we’ve decided to apply deep-learning to complete missing values for those in our train and test sets, i.e writing a neural network classifier that predicts “image_top_1” as a label for records with missing “image_top_1”.

*(Please refer to **main.ipynb** notebook for more reference and actual models) For better understanding them we recommend reading those notebooks after reading the part about neural networks models below.*

Descriptive outputs as above also allowed us picking up the most important features and focus on them while properly feeding the data into our deep learning models.

Using LGBM we've arrived to a model with validation error of 0.219 and 0.2244/0.2281 on site's test data set (31% on public test data / 69% on private test data).

We accomplished it by experimenting with different feature selection and different model parameters. We tried different groups of features and eliminated those who weren't important (like the part of speech text features we've created) or reduced our validation score (features that made the model overfit like 'item_id' / 'day').

Our best model parameters were achieved by gradually decreasing the learning rate (from 0.1 to 0.03) and increasing the number of leaves (from 32 to 512). The final subset of features we used can be found in the LGBM model notebook.

We quickly understood (from experimenting with the LGBM and other decision-tree-boosting models) that without ensembling we will not cross the 0.224 test error barrier. Thus we decided to aim for an ensemble model that will combine the predictions of the best two variations of our LGBM model from above (the best two sets of hyperparameters), and a few neural networks that will differ from each other by their architecture and the way they process the text feature.

Neural Networks

Our exploration of the neural networks models within this project consisted of two phases: the preprocessing of the data towards feeding it into the network (vectorization and encoding) and picking the right architectures of the networks. Here we describe how we've arrived to the networks that our ensemble is consisted of and point out the main conclusions that we've derived during this process.

Data vectorization/encoding towards feeding it into the neural network

This explanation follows the "NN-Model-design-explanation" notebook.

After loading all features we've engineered from the previous part, we choose the ones we'll learn from in this model. This choice is made under the "Vectorize features towards input to an NN" part and based on features' significance as was determined by the LGBM models.

We divide the features into three: text, categorical and numeric.

Obviously, each type gets its own pre processing.

The first task is to vectorize our textual and categorical features that we can feed it as an input for a deep learning model.

Data vectorization/encoding - text features

All our NN models vary mainly in the way they pre-process the text features which is the most important feature. We explored three ways to represent the texts of the ads we learned from:

1. *Binary Count Vectorization* - For each n-gram that appears in the text feature put a “1”, with no importance for the “frequency” of the word in the text. I.e both a word that appears once and a word that appears 17 time are represented as “1”s in the vector that represents the text.
    a. We chose to work with binary representation after experimenting with regular “CountVectorizer”. It’s more problematic to normalize its values/features and working with non-binary vectorization resulted in worse validation scores.
2. *TF-IDF Vectorization* - Vectorize text by TF-IDF (frequency–inverse document frequency). It’s a numerical statistic that is intended to reflect how important a word is to a document in a collection or corpus. For details see wikipedia page.
    a. As can be seen from the relevant notebooks, the TF-IDF vectorization gave much better validation scores than the *Binary Count Vectorization,* but still we kept the binary CountVec models for maintaining high variance within the ensemble’s models.
3. *Using pre-trained word embeddings and feeding them into an LSTM*
    a. We downloaded and used FastText Russian word embeddings (Facebook’s word embeddings learnt by skipgram on Russian Wikipedia).
    b. We encoded each text as a series of word embeddings and fed those sequences to an LSTM. We use the LSTM output as the text representation (Obviously we include the LSTM in the network and alter its weights while learning to better represent the text representation).

c. Those networks gave the best validation scores, but took much longer time (and much more resources - 3 days of monster machine on GCP) to train.

For each one of the first two methods we built four models. Two that concatenate all text features (Title, Description, Parameters) and vectorize it (each by its method from those above). And the other two represent each feature as a separate vector.

Each pair contains one model that encode the text by unigrams (“bag of words” encoding) and the other encode bigrams. To sum up those, in total we’ve got 8 models for the first two encoding methods:

	Binary Count Vectorization	**TF-IDF**
Merged texts	Unigrams Bigrams	Unigrams Bigrams
Separated title and description	Unigrams Bigrams	Unigrams Bigrams

We also created four LSTM-based models. Two learn the probabilities solely from the concatenated text features, while the other two combine the first with all the other features. In each pair one model uses Dropout layers within the LSTM (masking out part of the neurons in the recurrent layer) to help generalization and the other does not use Dropout.

Keypoints parameter tuning for text encodings:

1. For all of the above we filter out Russian stopwords (downloaded from from a predefined list.
2. **We do not stemm** the words because stemming showed dramatic increase in error rates. We used NLTK’s russian stemmer. It just doesn’t work well in Russian probably because this process losses too much descriptive information that the original words hold.
3. We tried vectorizing texts (within the TF-IDF and CountVec methods) to various vector sizes. We tried a grid search between 1000 and 200,000. Our vocabulary size is 900,000

(unstemmed, meaning many suffixes for each word). Surprisingly we've got the best result for smaller vector sizes (~7,500).

For vectors of size > 20,000 the networks overfitted after one epoch and resulted with terrible validation scores.

4. We used a max_df=0.5 filter for both TF-IDF and CountVec methods, meaning a word that appears in more than half of the records is filtered out (kind of a "corpus specific stopword"). It show a nice increase in validation score and learning rate.
5. We do not use min_df as some words are pretty rare but once they appear they are very indicative (e.g some kind of item name\model).
6. Problems with LSTM (Fast text etc) - stemming vs dimensionality.
7. Just LSTM - what made us do it?

You can see all of the above exact implementation in the notebooks for the models that are named accordingly to the way they pre-process text features.

Data vectorization/encoding - Categorical features

We used one hot encodings, first by using label encoder and applying the one hot encoder on the labeled categorical data.

Data vectorization/encoding - numeric features

We log scaled each feature with high skewness and/or variance.

After log scaling we clamped the 1% of the top outliers in high-skewed features (such as "price" that had "max int" values).

After scaling and clamping we normalized (L2) each numeric feature in order to allow the network to efficiently process this data (without "exploding gradients").

Neural Networks' Architecture

For the TF-IDF and CountVec networks we've used

Main Architechture - Number of hidden layers.

Why do we multiply batch size by 2?

Why do we make val_set smaller?

callbacks and saving weights.

Appendix - deep feature completion/cleaning

completed image_top_1 and price with those as well. Train the above again to get much better validation score.

Trained again the LGBM with better params.

The ensemble

Got ~12 models → ensemble.

What kind of ensembling? Lasso, Meta NN.

Insights and Applications

According to our models, the most important features are image_top_1 (which is image classification to categories), price and some of the extracted features from the text and images.

We can combine all the information to advice the user, which is applying a new ad, which parameters to improve in order to increase the probability of the ad to be sold. For example, we can tell the user online if his image is quality enough, or if his text is informative/related to his ad.

References

1. Kernels: https://www.kaggle.com/bminixhofer/aggregated-features-lightgbm
2. Text:
3. Images:
    a. how to calculate image luminance? https://stackoverflow.com/questions/596216/formula-to-determine-brightness-of-rgb-color

4. how to calculate image colorfulness?
   https://www.pyimagesearch.com/2017/06/05/computing-image-colorfulness-with-opencv-and-python/
5. how to calculate image sharpness?
   https://stackoverflow.com/questions/28717054/calculating-sharpness-of-an-image
6. how to calculate average color and dominant color?
   https://zeevgilovitz.com/detecting-dominant-colours-in-python
7. how to find image keypoints?
   https://stackoverflow.com/questions/29133085/what-are-keypoints-in-image-processing