Avito Duplicate Ad Detection

-Sanny Kumar

What is Avito

- An online classified where sellers can post their Ads

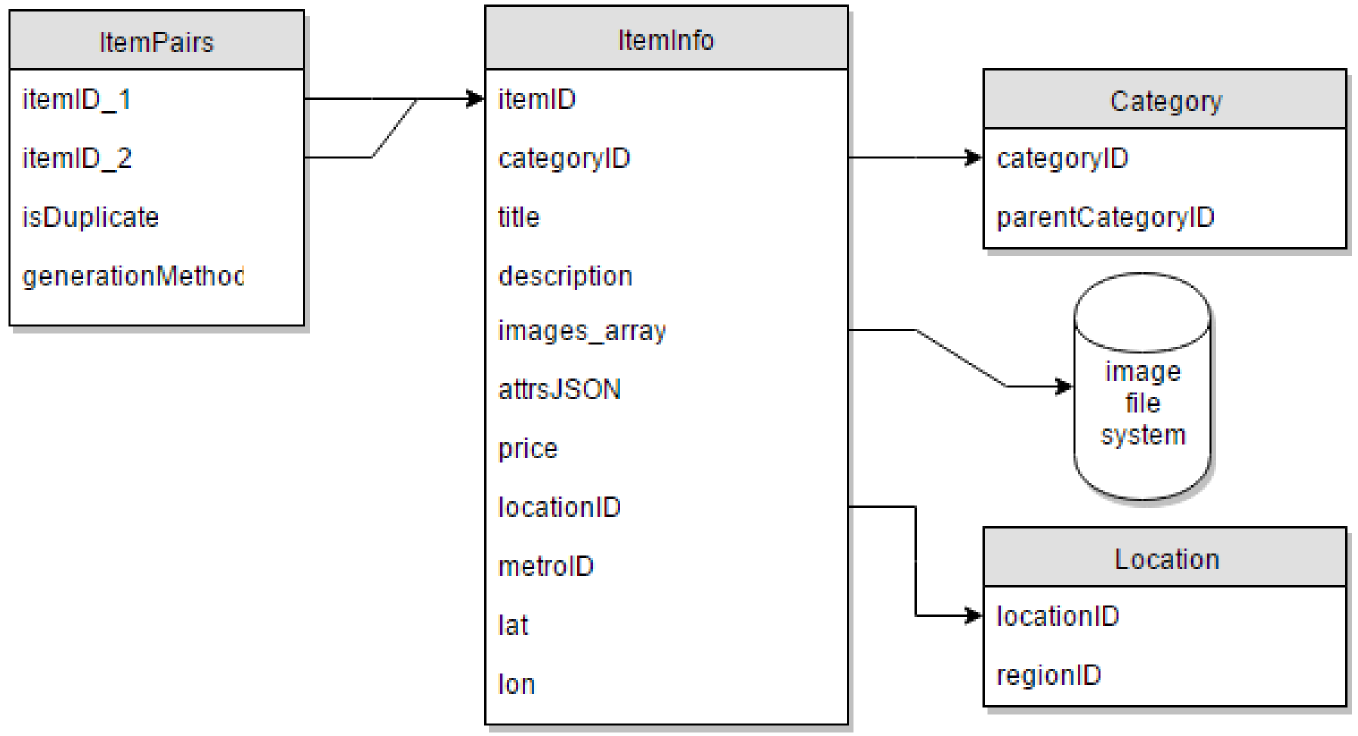
Major challenge:

- Many sellers are posting the Ads multiple times after slightly altering the contents, just to get their product noticed

Goal:

- Detection of duplicate Ads and remove those sellers

Schema


ItemPairs
itemID_1
itemID_2
isDuplicate
generationMethod
ItemInfo
itemID
categoryID
title
description
images_array
attrsJSON
price
locationID
metroID
lat
lon
Category
categoryID
parentCategoryID
image file system
Location
locationID
regionID

Steps

Step 1: Gather information about the Ad pairs:

train = ItemPairs
items = ItemInfo

Step 1a: Gather information about first Ad

item1 = items[selected features]

item1 = item1 left join category

item1 = item1 left join locationID

Rename the columns of Item1 table with a suffix “1”, i.e. itemID_1, categoryID_1 etc

train = train Left-Join item1

Steps cont….

Step 1b: Gather information about second Ad

Item2 = Items[selected features]

Item2 = Item2 left join category

Item2 = Item2 left join location

Rename the columns of Item2 table with a suffix "2", i.e. itemID_2, categoryID_2 etc

train Left-Join Item2

Steps cont.....

Step 2: Add derived attributes:

Step 2a: Same attributes

price_same:	1, if prices are same , otherwise 0
locationID_same:	1 if same, otherwise 0
categoryID_same:	1 if same, otherwise 0
regionID_same:	1 if same, otherwise 0
metroID_same:	1 if same, otherwise 0
lat_same:	1 if same, otherwise 0
lon_same:	1 if same, otherwise 0

Steps continued….

Step 2b: Text Similarity score

$$\left(\frac{L1 \cdot L2}{\sqrt{(L1 \cdot L1)(L2 \cdot L2)}}\right)$$

L1: freq of words in doc 1

L2: freq of words in doc 2

Note: If score is high, then documents are more similar

Description Similarity: Similarity between the descriptions of the two Ads, using the formula

Title Similarity: Similarity between the titles of the two Ads, using the formula

Model

- Used a tree based classification algorithm: xgboost

Why xgboost:

- Very fast
- Scalable
- Highly developed python interface
- Automatic parallel computation on a single machine
- Good results for most data sets
- Customized objective and evaluation
- Tunable parameters

Details of the model

- Objective: binary:logistic
- booster: gbtree
- eval_metric: auc
- max_depth: 5
- subsample: 0.8
- silent: 1
- seed: random_state = 0

Training the model

- num_boost_round = 260
- early_stopping_rounds = 20
- test_size = 0.1

- X_train, X_valid = train_test_split(train, test_size=test_size, random_state=random_state)
- y_train = X_train[target]
- y_valid = X_valid[target]
- dtrain = xgb.DMatrix(X_train[features], y_train)
- dvalid = xgb.DMatrix(X_valid[features], y_valid)

- watchlist = [(dtrain, 'train'), (dvalid, 'eval')]
- gbm = xgb.train(params, dtrain, num_boost_round, evals=watchlist, early_stopping_rounds=early_stopping_rounds, verbose_eval=True)

Validation and Prediction

Validation:

- check = gbm.predict(xgb.DMatrix(X_valid[features]), ntree_limit=gbm.best_ntree_limit)
- score = roc_auc_score(X_valid[target].values, check)

Prediction:

- test_prediction = gbm.predict(xgb.DMatrix(test[features]), ntree_limit=gbm.best_ntree_limit)

Performance

- Training Score: 0.861364
- CV score: 0.860815
- Public LB score: 0.79349
- Private LB score: Unknown

Conclusion: Since CV score is higher than test score, there might be overfitting.

Further improvements

- Involving images: For every pair of Ad, we can compare the images and find out how similar they are. If their similarity score is high, the Ads are more likely to be duplicate.
- Modifications in "same" attributes: In this model, I have used same attributes like lat_same, price_same etc. Instead of that, we can check if they are close because seller can also alter the price or prices might vary with time.
- Removal of features: If I remove some features, that might improve the test score.