Machine Learning Engineer Nanodegree

Capstone Project

Ratna Bearavolu

June 20, 2018

I. Definition

Project Overview

For my capstone project, I have chosen to work on a competition posted on kaggle.com - Avito Demand Prediction challenge (https://www.kaggle.com/c/avito-demand-prediction)

Domain Background

Avito Demand Prediction competition wants to predict the likelihood/probability that an advertisement that is posted on their social advertisement website sells the item that is being advertised. Based on various details of an advertisement posted, the company would like to give feedback to the seller on the likelihood that the ad will sell. This will help set expectations of the seller, so they are not frustrated with the website if the item does not sell. It will also give the seller an opportunity to review the posting, to help improve the likelihood of the item being sold (e.g. may be include a detailed description, better picture, etc)

I am interested in solving this problem because of the nature of the inputs that need to be analyzed - tabular data, textual data and image data to predict the likelihood of selling. Typically, in the homeworks/assignments that I have worked on so far, the data was of single type (either tabular data or textual data or image data). When I saw that this problem had a mix of inputs, it piqued my interest in how I would address this issue and how I would set up this problem. Also at a personal level, in the past, there were many of my ads (that I posted on various social advertisement websites) that never

sold. I would have definitely welcomed any feedback on the likelihood of the ad selling from any one of these websites.

Problem Statement

Given an advertisement, i.e, information on item to be sold, such as description, price, image, etc, a '***deal probability***' should be predicted by the system. Deal probability is a continuous variable ranging between 0.0 to 1.0. Higher probability will indicate greater likelihood that the advertisement will sell and lower probability will indicate less likelihood that the advertisement will sell the item.

Datasets and Inputs

The competition data has been provided by Avito (https://www.kaggle.com/c/avito-demand-prediction/data). Of importance for my capstone project were:

- train.csv - Training 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.
- train_jpg.zip - Images from the ads in train.csv.
- test.csv - Test data used by Kaggle to generate the prediction score
    - The columns are identical to train.csv except the deal_probabillity column.

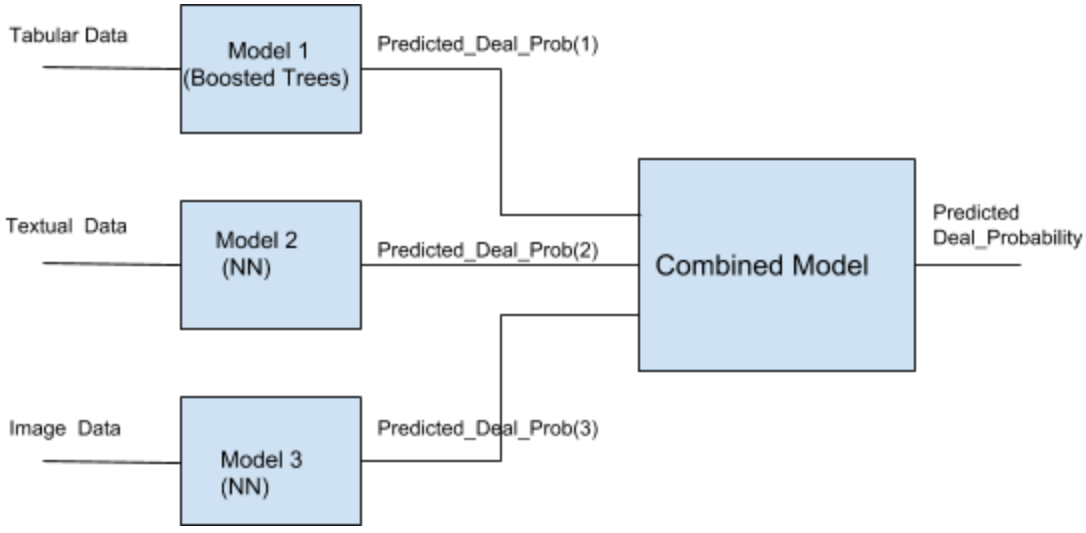
- test_jpg.zip - Images from the ads in test.csv

train.csv has 1.5 Million rows of data. Each row indicates an ad item that was published by the user on Avito website. Not all rows have description and not all rows have an associated image (i.e. the image of the item the user is trying to sell). The size of the file is 908MB. The size of train_jpg.zip is ~40GB.

As the kaggle competitions is active, to measure the performance of my models on the test data, I have made submissions to Kaggle to generate the performance metric on the models.

Solution Statement

As the data set is labeled and target variable is continuous, I used supervised learning algorithms to design the model (e.g. Boosted Tree Regression and Neural Net Regression techniques). Also, as the input is primarily of three types of data (tabular data, textual data and images), I built three different models for each type of input and then used another regression model to combine the outputs of the three models to get the final prediction.



For Model 1, the input (X_train) was mostly all the columns found in train.csv and the label (y_train) was the corresponding ‘deal_probability’ value. For Model 2, the X_train was the vector representation of the ‘description’ column and y_train was the

corresponding ‘deal_probability’ from train.csv. For Model 3, X_train was the matrix representation of the image and the corresponding ‘deal_probability’ from train.csv. For all three models, the output was the predicted deal_probability value given their respective X input.

The next step in the model building process was to build the combined model that takes the outputs of these three models (individual deal_probability values) and combine them to predict the final value for the target variable.

Metrics

As the competition is being evaluated on the Root Mean Squared Error (RMSE) score, I intend to use this metric for evaluating my models.

RMSE is defined as

$$RMSE = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(target_{i\ actual} - target_{i\ predicted})2}$$

Where n = the number of testing data points, target_actual = actual target variable value (between 0.0 and 1.0), target_predicted = predicted target variable value (between 0.0 and 1.0). The goal would be to build a model that has a value of RMSE close to 0.0

Given the nature of the target variable (continuous), I think RMSE is a good evaluation metric.

II. Analysis

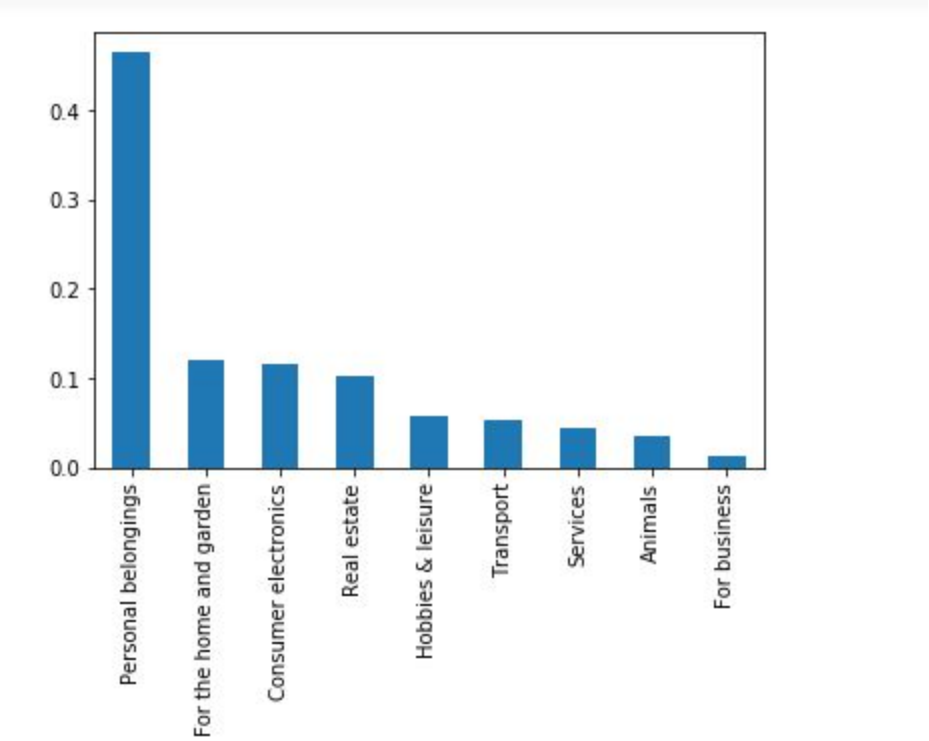
Data Exploration and Visualization:

In this phase, my primary goal was to understand how various attributes (in train.csv) relate to one another and their correlation to the target variable. I sliced and diced the data in several different ways to gain familiarity with the data and see if there were any correlations among the various columns in train.csv. I started with simple bar charts to get the distribution of the data by individual columns and then used scatter plots to plot combination of columns to see if there are any trends among various columns. (E.g, - distribution of deal_probability by parent category, etc). I have included a few graphs of interest, but the python notebook has more details on the ways I grouped the data, statistics on individual groups, etc.

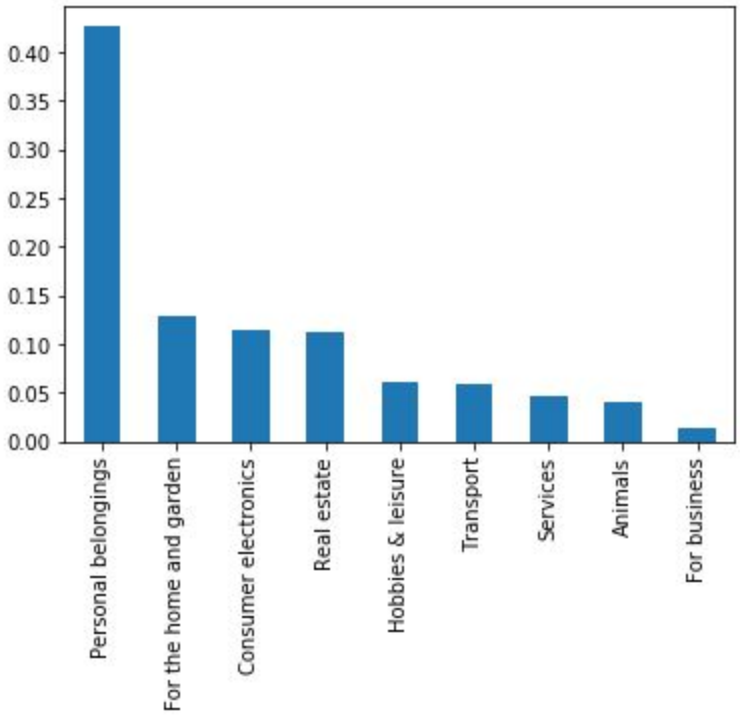
Along with understanding the data, I was also checking if the train data is a balanced set and represents what is found in test data well. For example, below shows the distribution of the ads by ‘parent_category_name’ column in train data and test data to see if the distribution is similar or not. From the graph, it can be seen that the train data distribution is a good representation of what is seen in test data as well.

- parent_category_name - Top level ad category as classified by Avito's ad model.

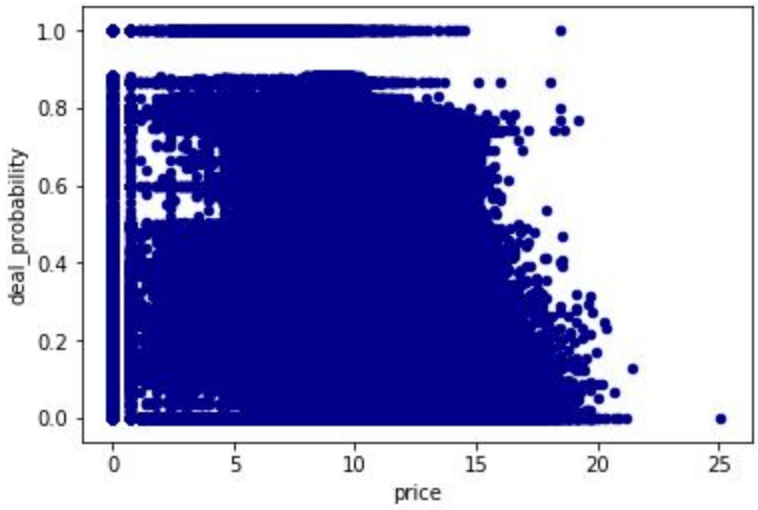
Normalized distribution of ads in train data by parent_category



Normalized distribution of ads in test data by parent category



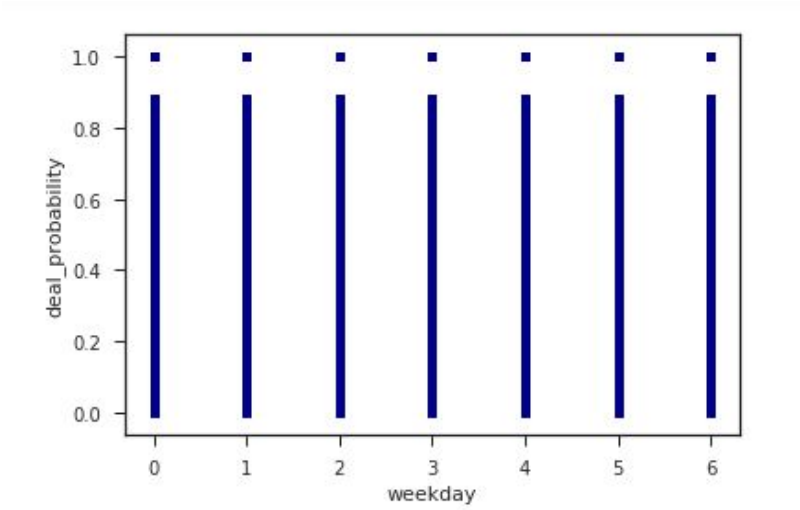
- price - Ad price. Scatter plot of the target variable (deal_probability and price). Wanted to see if there were any patterns, such as low priced items having higher deal_probability, etc. If there were, I was going to further explore or isolate the data to better understand the dynamics of these variables. However, the image below shows a very uniform distribution of data. (Note, price on this graph has been log scaled to address the wide range of data)



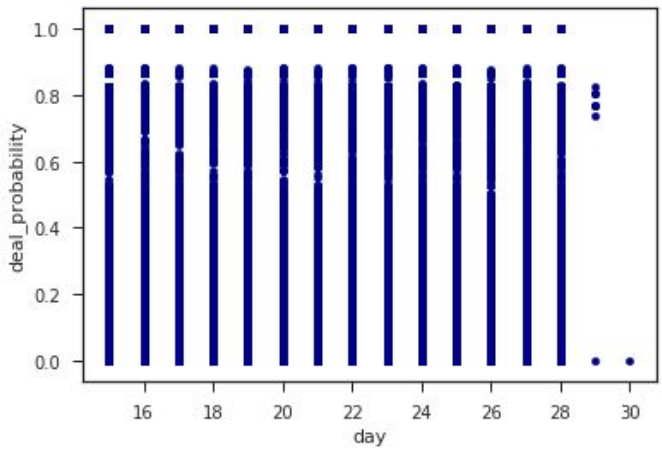
Feature Engineering

- activation_date- Date ad was placed. Broke the data into three subcomponents - day, month and year of the date. Wanted to see if there were any correlations of deal_probability based on when the ad was placed. For example, did the ads that were activated on weekend have a higher deal_probability, or ads placed earlier on in the month have a higher deal_probability. From the visual below, it was clear that no such patterns existed in the data.

Plot showing weekday vs. deal_probability of train data



Plot showing day vs. deal_probability of train data



Take away from this was that the training data come mostly from the last two weeks of the month.

I have also used pairwise plot to visualize the feature data in train.csv. Did not include it in the report due to image resolution limitation (it was looking like an eye-chart). Please refer to the output in the ipython notebook. Briefly, used the encoded categorical values when creating the pairwise grid (please see below for the details on encoding of categorical values). The grid confirmed some of the observations I had seen earlier on - train data was mostly covering the last two weeks of the month, absence of any 'apparent' correlations among the columns, etc. When I see data like this (that doesn't seem to have too many visible patterns), I appreciate the ML algorithms/techniques that learn function(s) over this data to uncover the relationship(s) that can predict the dependent variable.

Algorithms and Techniques

When considering algorithms for Model 1, I was leaning towards using an algorithm that uses decision trees, primarily due to the ease of understanding how the various features are used to predict the target variable and how well the trees handle missing and categorical values. After some initial experimentation with a few variations of tree based models (DecisionTreeRegressor, eXtreme Gradient Boosted trees),I decided to use XGB (https://xgboost.readthedocs.io/en/latest/). I was mostly motivated to use this due to the low RMSE score observed on my initial training data, as well as seeing that numerous Kaggle winning competitions used XGB model. XGB model uses iterative tree ensemble techniques. Each iteration (a new tree) tries to minimize the error observed in the previous iteration.

For Model 2, I used a two layer Neural Net to process the description data. My primary motivation came from the literature I read about for sentiment analysis using NN . In the sentiment analysis case, the textual data is converted to multi dimensional input space (via tf-idf or word2vec) and a NN is trained to identify if the sentiment is positive or negative. In my case, I used TF_IDF and SVD to convert the description column data into multidimensional data, which is used as input to my two layer NN to predict the 'deal_probability'

For Model 3, I used a CNN to train and predict the target variable. I found that this was very similar to the Dog Breeder Classifier project, with the exception that in this competition, the target variable is continuous and not discrete. I changed the last layer of CNN to have a sigmoid function that would give the output value between 0 and 1.

Benchmark

As this an active Kaggle competition, there hasn't been a published model yet. So, as a benchmark model, I used a simple linear regression model based only on tabular data in train.csv (i.e, excluded description and image data). I systematically compared the RMSE score of benchmark model with that of the various combined models. My hypothesis is that the combined model, as they are using additional forms of data/information, should perform better than the benchmark solution.

III. Methodology

Data Preprocessing

- Below are the steps I followed to prepare the data in train.csv
- For Model 1:
    - Checked for missing values - calculated the percent missing. Below is a snapshot of the output from the function.

```
Missing Train data ********************************
param_1 has 4.09571750883317 of data missing
param_2 has 43.536753437486695 of data missing
param_3 has 57.3733690562343 of data missing
price has 5.677839385296497 of data missing
image_top_1 has 7.48877229577285 of data missing
Missing Test data *********************************
param_1 has 4.505957461873424 of data missing
param_2 has 45.87166970210724 of data missing
param_3 has 60.24943060904181 of data missing
price has 6.015482713723207 of data missing
image_top_1 has 8.380372828152105 of data missing
```

    - Dropped 'param_3' column as it had more than 57% and 60% missing data in train and test data. (As a future improvement, I may go back to this and fill in a 'placeholder' value for the missing data and run thru the various models to see if it improves the RMSE score)

- Classified the number of columns as categorical, ordinal, continuous, binary
  - Apart from price, all the columns were categorical columns. Below is the count of cardinal values by columns

```
region :28
city :1733
parent_category_name :9
category_name :47
param_1 :372
param_2 :272
image_top_1 :3063
user_type :3
weekday :7
```

- **Handling high cardinal categorical values**: As seen from the output above, most of the columns have high cardinality (e.g., image_top has about 3000 cardinality). One-hot encoding is not the appropriate technique, as this will increase the dimensionality of the data rapidly. During my exploration process, I label-encoded the various categories. This does address the dimensionality increase issue, but inherently imposes an ordinal structure and depending on the model I choose, may lead to incorrect results. I searched for other techniques of handling high cardinal categorical columns and found the paper *'A Preprocessing Scheme for High-Cardinality Categorical Attributes in Classification and Prediction Problems' by Daniele Micci-Barreca* (https://kaggle2.blob.core.windows.net/forum-message-attachments/225952/7441/high%20cardinality%20categoricals.pdf). The technique converts categorical column values into continuous range of values using target statistics. I applied this technique to convert all the categorical columns found in train.csv and test.csv

- For Model 2:
  - Model 2 only used the 'description' column of train.csv as input data. 7.73% of the data was missing description information. Used a placeholder value for the missing data.

    - Used TF-IDF and Truncated SVD from sklearn to get the top 125 svd components of description column data.
- For Model 3:
    - Model 3 used the images data. I used the Keras image processing package to read them and convert them into 3d array representations.

Implementation and Refinement

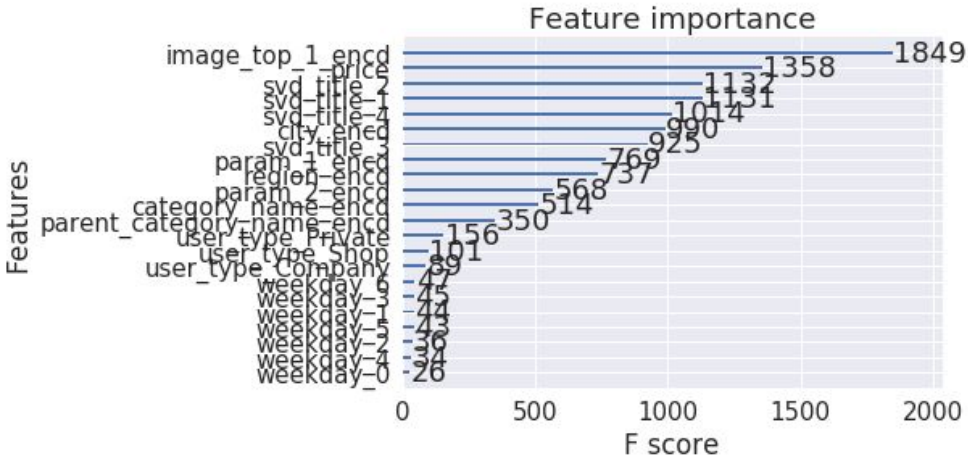
Once the data was pre-processed and ready, the code for train, validation and testing were relatively straight forward. Most of the time was spent tuning the various model parameters.

Model 1 - tuning XGB model:

XGB model comes with many parameters that can be tuned to deal with bias-variance tradeoff. Off importance for controlling overfitting are ‘max_depth’, ‘min_child_weight’,’gamma’, ‘subsample’ and ‘colsample_bytree’. (http://xgboost.readthedocs.io/en/latest/parameter.html)

With an initial set of booster parameter values (*'eta': 0.3, 'subsample': 1.0, 'colsample_bytree': 0.8, 'objective': 'binary:logistic', 'max_depth':5, 'min_child_weight':3*), performed cross validation on the training data to determine the best number of estimators needed. The number of estimators (trees) for this set of params was 472. Also, below is the screenshot of Feature Importance based on the booster parameter criteria. Feature Importance is computed based on the number of times it was used to make splits in all generated trees. I found this feature extremely useful as well as interesting to see which feature(s) played an important role in predicting the target variable. In future, for other applications, I can use this feature as part of feature selection exercise.

```
Shape of selected columns train data (1503424, 22)
CPU times: user 1h 5min 58s, sys: 3.09 s, total: 1h 6min 1s
Wall time: 1h 6min 1s
```



Once I had this, my next step was to to use grid search to further fine tune some booster parameters (especially max_depth and min_child_weight to avoid over fitting). Due to limitations of the aws instance, the grid search did not successfully run to completion. I terminated the run after 9 hours. Unfortunately, there were no intermediate results that I could have used. For pedagogical purposes, I ran the grid search on a small training set (~5000 records). Below is the snapshot of the optimized scores of the grid search. From the output of the optimized scores, the combination of 3 and 5 for max_depth and min_child_weight respectively, had the lowest score. I decided to use these values for my Model 1

```
In [245]: if flg_mdl_1 and flg_mdl_1_xgb_gridsearch:
              optimized_GBM.grid_scores_

Out[245]: [mean: -0.09520, std: 0.01919, params: {'min_child_weight': 3, 'max_depth': 3},
           mean: -0.09056, std: 0.02037, params: {'min_child_weight': 5, 'max_depth': 3},
           mean: -0.09852, std: 0.01888, params: {'min_child_weight': 3, 'max_depth': 5},
           mean: -0.09806, std: 0.02859, params: {'min_child_weight': 5, 'max_depth': 5},
           mean: -0.09412, std: 0.01474, params: {'min_child_weight': 3, 'max_depth': 6},
           mean: -0.10422, std: 0.02328, params: {'min_child_weight': 5, 'max_depth': 6}]
```

Model 2:

- Number of epochs to train the NN: As this was a fairly simple NN (2 layers with 64 nodes each), started small (around 15 epochs) and went upto 45 epochs. For multiple runs, noticed that the ‘val_loss’ metric stopped improving anywhere after 20 epochs. So, decided to use 30 epochs.

- Number of SVD components to use from the tf-idf matrix: Based on sklearn recommendation (http://scikit-learn.org/stable/modules/generated/sklearn.decomposition.TruncatedSVD.html) ,used 100 SVD components derived from the TF-IDF matrix. Also, experimented with the using 125 components. Didn't see any difference in the RMSE score. Decided to use 125 as my final value.

Model 3 - processing image data:

Processing the entire training image data (1.3 million images) turned out to be a very resource intensive task.

Converting and storing the image arrays:

With the large volume of images in the training dataset, I was not able to process and store the image arrays in the RAM. My initial approach was to use a different instance of aws with bigger RAM. I changed the aws instance from p2.xlarge instance with 60 GiB of RAM to p2.8xlarge instance with 488 GiB of RAM , but I still ran into memory constraints. Then,I started exploring options available in python to handle large volume datasets and learned about h5py. Used h5py to convert the images to image arrays and stored the hp5y file on hard disk. This was a very time consuming task. It took about 6-8 hours to process the training dataset (~1.3 M images) and testing (~500 K images) dataset and used about 600GB of hard disk.

Next, I wrote helper methods (e.g Generator method that reads from the h5py file into RAM during training) to get chunks of data that the RAM can handle.

Hurdles with the volume of training data

With training data processed and stored as hdf5 file and functions to read the file in chunks, I started to train the CNN. Unfortunately, a single epoch was taking more than a few hours (I am speculating that it was the disk io). After letting it run overnight and not seeing very promising results (i.e., the net was running its second epoch), I decided to terminate the process. Because at this rate, training would have taken days and my aws expenses were going to be really high.

Next, I decided to use a smaller training data set. I was hoping that about 500K images will work faster. However, this did not speed up the process that much either. The runtime was still in the order of an hour or so for a single epoch.

As my final attempt, I used a very small training data set (about 5000 images). I chose this number because, the Dog Classifier project used about 6000 images and the current aws instance (p2.xlarge) ran quickly on this set. I know the accuracy may not be good (as the training data set sample is about 0.3%) and the combined model may not be very predictive. But, I wanted to do this as a learning exercise. As expected, in the next section, you will notice that this model (trained on very limited amount of data) performed very poorly.

(As a side note, this exercise made me appreciate the idea of transfer learning more, where a user can take and apply the learnings of pre-trained network.I will be exploring this option as a future improvement to the project)

IV. Results

Below table summarizes the various models used and the RMSE score generated on the test data. As this is an active Kaggle competition, had to submit the prediction scores on the test data to Kaggle.com to get the RMSE score for all the models I used.

The order of rows, reflects the order in which I built models, evaluated and combined them piecemeal. As soon as I had Model 2, I built a combined model that used only outputs from Model 1 and Model 2. I labeled this ‘Combined Model 1 and 2’.

When combining the two models, I considered XGB, NN and a linear regression models to see which model performed the best. From the results, it can be seen that they all compared very similarly, with XGB performing slightly better compared to the other two models.

During evaluation of how best to combine Model 1 and 2, I explored an alternate approach, where instead of combining outputs of the two models, combine the data of Model 1 and Model 2 and use a single model to predict the target variable. This is the ‘Alternate combined model 1 and 2 data’. The RMSE score came close to the ‘Combined Model 1 and 2’, but did not perform as well as the combined model.

Also, from the table below, it can be seen that the individual models by themselves (Model 1 or Model 2) did not have a good RMSE score. But when the results from Model 1 and 2 were combined, the RMSE scores started to decrease. This reflects my initial expectation that model should perform better with every new addition of data/information (textual and images).

Model	Description	Learner	Num. Training and Validation records	Num. Testing records	RMSE score
Benchmark	Used tabular data from train.csv. Did not consider title and description column	Decision Tree Regressor	1503434	508438	0.2410
Model 1	Used tabular data from train.csv. Included SVD components from the 'title' column	XGB Regressor	1503434	508438	0.2302
Model 2	Used ~125 SVD components of the 'description' column	Neural Net	1503434	508438	0.2450
Combined Model 1 and 2	Used individual outputs from Model 1 and 2 as inputs to predict the final target value	XGB Regressor	1503434	508438	0.2293
Combined Model 1 and 2	Used individual outputs from Model 1 and 2 as inputs to predict the final target value	Neural Net	1503434	508438	0.2294
Combined Model 1 and 2	Used individual outputs from Model 1 and 2 as inputs to predict the final target value	Linear Regressor	1503434	508438	0.2295
Alternate	This model	XGB	1503434	508438	0.2297

combined Model 1 and 2 data	combined the tabular data input with the top 20 svd components of the description data	Regressor			
Model 3	Used a very small sample of Images from train.csv	CNN	4000	508438	0.3032
Combined Model 1, 2 and 3	Used outputs from model 1,2,3 as inputs to predict the final target value	XGB Regressor	4000	508438	0.2533

V. Conclusion

When I first read about the project, the most appealing part of the project was that it had various types of data to work with (tabular, textual, image). Apart from seeing an opportunity to work with and explore different techniques that worked for particular data types (e.g. CNN for image data, NN for analyzing textual data), I wanted to validate that data when viewed as individual sources, may not be very predictive, but combined together can tell a rich story. For example, from the table above, it can be seen that individual RMSE scores of Model 1, 2 and 3 were not that stellar (exception being Model 1) compared to the Benchmark model. However, when they were used together, the overall predictiveness improved. This can be clearly seen from the RMSE score of the ‘Combined Model 1 and 2’.

During the model development phase, I was very encouraged with how the RMSE score was improving whenever a different facet of data was being added (i.e., benchmark only considered tabular data, whereas Combined model considered both tabular and description data) to predict the target variable. I was very positive that by adding image data, I would certainly improve the RMSE score. Unfortunately, due the challenges outlined above, I was not able to validate my hypothesis.

This leads me into future improvements for this project. As a future improvement, I am going to do literature survey on availability of pre-trained CNNs that when given an image can generate a score/grade that rates the image in-terms of its ‘quality’. At this point, I am not clear on how best to define ‘quality’. But I am hoping, this pre-trained CNN would have considered things like brightness, centered image, etc to rank the image. I would use this score as the input to the combined model and train the combined model.

Also, when reading about XGB model, I came across another similar Tree ensemble model called Light GBM from microsoft that trains faster, has low memory usage and better accuracy than most of the boosting algorithms (https://www.analyticsvidhya.com/blog/2017/06/which-algorithm-takes-the-crown-light-gbm-vs-xgboost/). I want to compare Light GBM performance to XGB performance on Model 1 data in terms of RMSE score as well as see if I could perform effective grid search on Light GBM to better tune the parameters of the model.