Capstone Project
Machine Learning Nanodegree
4/12/2017

Definition

Project Overview
Classify interest level in online apartment listings for renthop.com. Renthop provided three months of New York City apartment listing data to kaggle.com for a competition.

Problem Statement
Given the various information apartment managers input into the apartment listing, the problem is to classify online interest level into one of three categories: low interest, medium interest, and high interest. By successfully identifying what features are important in listing apartments, future listings can be improved to ensure that apartment managers and apartment renters are able to best display and acquire information from the listing. A high-level solution is to create a model that accurately predicts the interest level on data in the withheld kaggle test set in the hopes that the model could be used for future apartment listings.

Metrics
Evaluation method is the multi-class logarithmic loss function (log-loss). For each test sample, the model will assign an output that is the probability of high interest, medium interest, and low interest. The estimated probabilities are judged against the actual label. Guesses are penalized based on how far the estimated probability for the label is from the actual label. The method penalizes models for being overly confident while allowing the model to show uncertainty. It is also applicable for cases like this one where the target classes are unbalanced.

$$F = -\frac{1}{N}\sum_{i}^{N}\sum_{j}^{M} y_{ij} \cdot Ln(p_{ij}))$$

Analysis

Data Exploration

The data consists of 49,353 labeled training examples and 74,669 unlabeled test examples. The data features are:

- bathrooms: number of bathrooms
- bedrooms: number of bathrooms
- building_id
- created
- description
- display_address
- features: a list of features about this apartment
- latitude
- listing_id
- longitude
- manager_id
- photos: a list of photo links.
- price: in USD
- street_address
- interest_level: this is the target variable. It has 3 categories: 'high', 'medium', 'low'

Apartment features are available as a list and are inputted by the user so that features that represent the same thing may be listed different (ie pre war vs. Pre war vs. pre-war). All photos related to the apartment listings were downloaded but only the count of images for each apartment listing was used.

Exploratory Visualization

The accompanying Jupyter notebook details an exploration of most of the input features. In general summaries of numeric features (mean, median, IQR), histograms, table counts, and counts vs. interest level are provided where appropriate.

Figure below shows a histogram plot of apartment price-per-bedroom (x-axis) vs. count of apartments (y-axis).

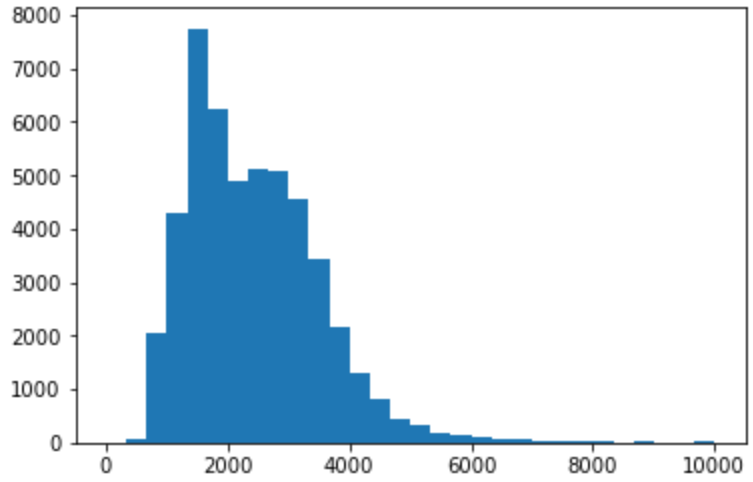


Figure below shows a scatterplot of apartment price-per-bedroom (x-axis) vs. interest level (y-axis). High interest level is scattered around 10,000, medium around 5,000 and low around 0. Note that as price-per-bedroom grows, there are less medium and high interest apartments. Price-per-bedroom is of the most important features for the fitted models.

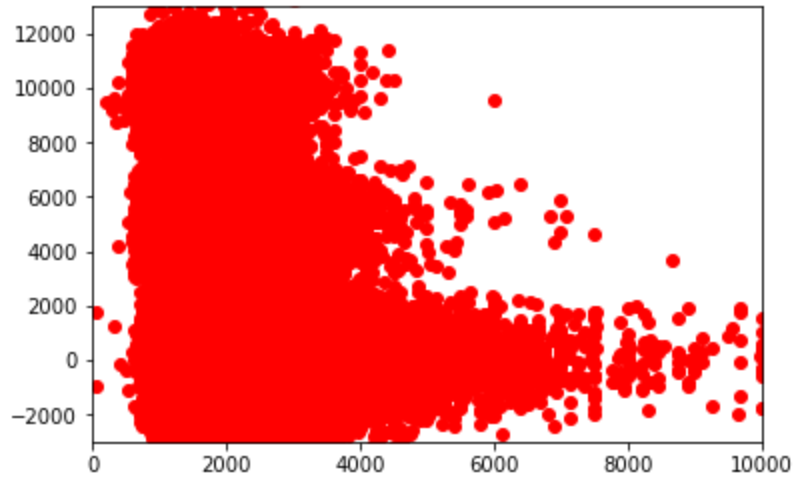
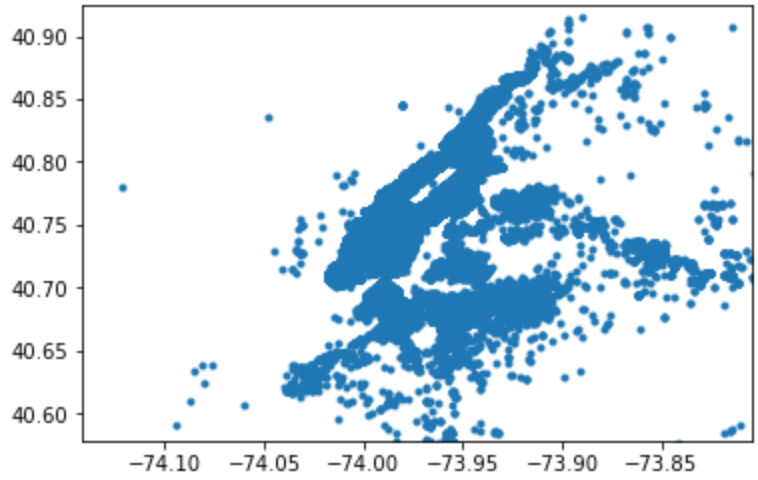


Figure below shows a scatterplot of the apartment listings by latitude and longitude. Representation of the 5 boroughs of New York City can be seen (Manhattan and Brooklyn most represented while Staten Island is the least represented. The accompanying notebook also has a k-means plot and some heat maps.



Algorithms and Techniques

In examining the data, one issue that arose in multiple features was obviously wrong data. For example, listings with 0 bathrooms. There is no 0 option bathroom button to click when making or searching for a listing on the renthop.com website. When looking at the descriptions of 0 bathroom listings, they did not appear to be the type of apartments that do not have any bathrooms. 0 bathroom entries are most likely a mistaken entry. In some data sets I would argue for excluding obviously wrong entries like the 0 bathroom entries. Here I think it makes sense to keep them. By comparing the interest count of 0 bathroom entries to the benchmark rate, 0 bathroom entries skew more heavily towards low interest. While a 0 bathroom entry may be a mistake or nonsense, it has a real effect on interest level and should be left in. Sloppy and unusual entries in many input features show a similar trend. Rather than eliminate bad entries from the training set, it makes more sense to let them remain and be used to help predict interest level.

In some cases though--price, latitude and longitude--leaving in obviously wrong entries can skew the standard deviation and cause feature normalization and scaling to behave differently. For those features I changed the extreme outliers to more limited outliers in the hope of allowing models to use the predictive power of those outliers while not unnecessarily affecting the variance of the data.

Below is a brief overview of the features and how they were changed or ignored. The accompanying Jupyter notebook has further details.

- 'bathrooms': no change
- 'bedrooms': added a feature that modified 0 bedroom entries (studio apartment) to roughly .83

- 'building_id': removed and replaced with counts that id shows up in data and results of the other listings
- 'created': added features for hour, day of week, day of month, and month
- 'description': removed and replaced with length of description feature
- 'display_address': removed as location is provided by latitude and longitude
- 'features': removed and replaced by length of number of features and one hot encoding of the 10 most popular features
- 'interest_level': target variable
- 'latitude/longitude': extreme outliers scaled down.
- 'listing_id': removed
- 'manager_id': removed and replaced with counts that id shows up in data and results of the other listings
- 'photos': removed and replaced by count of number of photos for that listing
- 'price': added another field that also shows the price_per_bed, extreme outliers scaled down
- 'street_address': removed as location is provided by latitude and longitude

Manager_id and building_id have too many unique values (roughly 3,500 and 7,500) to one hot encode. However, when analyzing the test set, it would be useful to know if the manager_id and building_id have been used before and what the past results were. Since managers likely have a similar system for creating entries and buildings have similar desirable (or undesirable) attributes, I created 4 features for each: count of listings and count of high/medium/low interest listings. For each entry I subtracted the result of that apartment interest level in order not to influence the predictions (ie for a manager with only 1 entry, the count of high/medium/low would predict the interest level 100% of the time, thus I have to remove the prediction for that entry, from that entry row).

I considered adding other columns that had the proportion of low/medium/high interest % listings modified by how certain I am that those percentages are different than the benchmark percentage. The thinking being if a manager produces 50% high interest listings over 200 samples, he's likely doing something statistically significant and that could be used to improve the rating. However I could not think of a good way to produce those percentages. Simply calculating the observed percentages produces issues with small sample size and I could not think of a Laplacian smoothing like procedure that would work in this case. I considered a one-proportion z-test[1] but most manager samples lack sufficient number of samples. I feel like Bayes Theorem could be applied here or a probability density but could not come up with a good way to apply my thoughts to manager_id and building_id.

[1] https://en.wikipedia.org/wiki/Statistical_hypothesis_testing

Benchmark

The benchmark model is simply predicting the probabilities that each interest level is available in the training data set. The benchmark model predicts 0.695, 0.223, and 0.078 for the probabilities of low interest, medium interest, and high interest. The benchmark model received a log-loss score of 0.79075 on the kaggle test set.

Methodology

After some consideration, I chose the following models to train the data on:

- Random Forest: From what I've learned, trees are typically a good classifier. After looking at the data, I felt that trees would be good at making inferences from many of the marginal and unusual samples. Can also use 'feature_importances' to see the relative effect of each column
- Logistic Regression: Simple to implement, outputs probabilities.
- Naive Bayes: Not a good solution for this type of problem but simple enough to train and often effective.
- Adaboost: For many of the reasons I chose Random Forest, I thought a boosting algorithm with some of the similar properties could prove useful.
- SGD Classifier
- XGBoost: Powerful boosting that is used on a large amount of Kaggle entries.
- Voting Classifier: As I altered most of the inputs and modified the feature space, I am hopeful that by combining some of my models I can leverage their individual strengths to produce better predictions
- Feedforward Neural Network

In the accompanying notebook, there is a section with some more thoughts on the models, and what parameters were tuned for the fitting of each model.

I decided to fit each model (except for the neural network) on the training set and score it with a small test set prior to doing feature selection. Since I added and modified many features in the data exploration phase, I wanted to get a preliminary look at the usefulness of the created features before doing feature selection.

Data Preprocessing

I dropped the features mentioned above and standardized the price, latitude and longitude features. I split the training data into a training set, a validation set, and a test set. For some of the models, probability calibration[2] helps the model generalize better and the validation set is used to

[2] http://scikit-learn.org/stable/modules/calibration.html

fit the calibrated classifier. Some cross validation sets were also created for grid search and randomized search of optimal hyper-parameters.

Implementation and Refinement (Part 1)
For a more detailed walkthrough please see the accompanying Jupyter notebook. I fit the 7 models and scored them on the created test set. The random forest and gradient boost models scored well on the created test set but did not generalize well to the kaggle test set. The best model created was the voting classifier which combined the 6 models.

With the models as a guide, three types of feature selection (select from model, variance threshold, and select k best) were performed. Based on that analysis, it was clear that the 'created_' features and the 'apt_feature_' features not adding much to the model. The 'created_' features were either dropped or modified to have larger ranges. All the but 'apt_feature_no_fee' were dropped.

Using tf-idf and a support vector classifier, I created different input features to account for the text in the description and the apartment features. I combined the description and the list of features into one text blob for each listing. I fit a tf-idf to each text blob and then used a support vector classifier to predict high, medium, and low interest probabilities based on the tf-idf representation of each text blob.

Implementation and Refinement (Part 2)
With the now modified feature list I fit a feedforward neural network and refit the 7 original models. I decided on a feedforward neural network using three hidden layers of rectified linear units and a softmax cross entropy cost function. From what I have learned from the deeplearningbook.org, quora.com and the Udacity course is that rectified linear units are typically the best hidden unit to start with. Softmax makes sense since I'm looking for probability outputs and cross entropy is an implementation of the log-loss cost function. I also normalized the inputs for easier gradient descent training.

Results

Model Evaluation and Validation
For the 7 original models, I refit and retuned the hyperparameters. I scored each model on the kaggle test set. The results:

- Feedforward Neural Network: 0.60874
- Logistic Regression: 0.63189

- Stochastic Gradient Descent: 0.65002
- K Nearest Neighbors: 0.65466
- Voting Classifier: 0.68491
- Naive Bayes: 0.73656
- Random Forest: 0.78278
- Benchmark: 0.79075
- Adaboost: 2.01938

In all but the adaboost and voting classifier, the refined model after feature selection performed better. The voting classifier for the second batch of models performed worse than the voting classifier for the first batch of models, despite the second version of all but one of the models in the second batch performing better than the first batch.

Justification
All models except for the adaboost outperformed the benchmark model. Five of the models were significantly better than the benchmark model. The top score on kaggle is currently 0.50031 and the neural network score is currently in place 1350 out of 2000. In an open ended competition such as this, it is difficult to say whether the problem is solved but the neural network does significantly outperform the benchmark model.

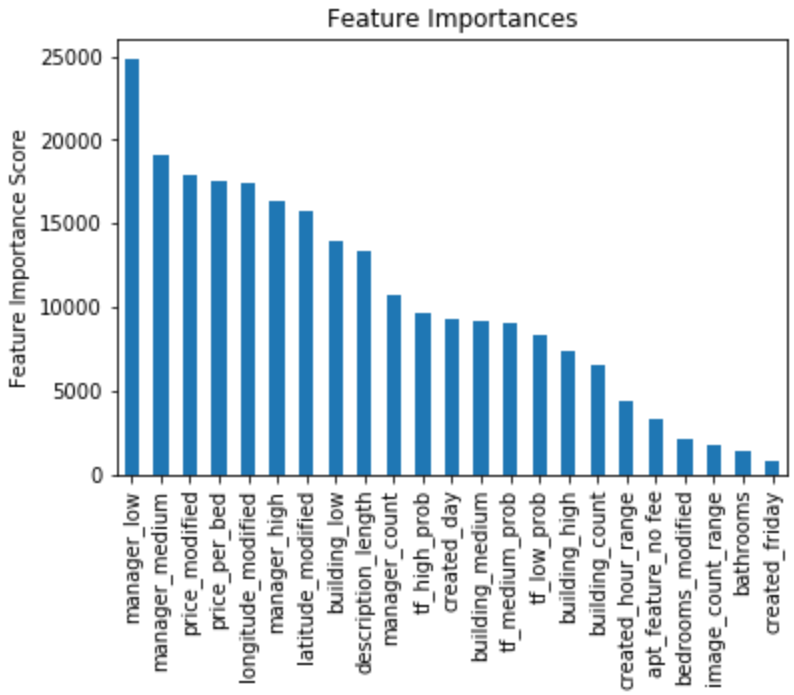
I would speculate that the neural network was the most successful because the model was best able to weigh the importance of price-per-bed and location (latitude and longitude) and interactions of the ‘manager_’ and ‘building_’ features. In the abstract, renting an apartment is paying money for an amount of space in a location. The number of bedrooms is a rough proxy for that amount of space. I would speculate that mapping that abstraction in NYC is difficult due to the complexities of the city itself and that the neural network was the most flexible in handling those difficulties.

For the the ‘manager_’ and ‘building_’ features, there is some interaction between the overall count of instances and the prior results for each manager and building. I was unable to come up with a good function mapping for this but can speculate that a well-tuned neural network would able to make inferences that I could not.

Conclusion

Free-Form Visualization

Figure below shows the relative feature importance of the xgboost model. While the most successful prediction model was the neural network, it is difficult to visualize interactions between the hidden layers.



Reflection

It's hard to know how to end a project such as this one as there is no correct or final answer. On my withheld test set, I can overfit several models (xgboost, neural net, and voting classifier) to a log loss much lower than what is on the kaggle leaderboard. However when I use that overfitted model on the kaggle test set I predictably score much lower. I can continue to fit and tune the most promising models and get better scores on kaggle but even that does not guarantee that the model is actually better. I may just be overfitting to the revealed portion of the kaggle test set and not actually generalizing well to the hidden portion of the kaggle test set. The best model I came up with was the feedforward neural net, followed by the second logistic regression model, and then the voting classifier for the inferior batch of first models.

Improvement

While I learned many things by working on this dataset, the main discovery I had was how much there is to know and how many different directions solving a problem such as this one can take. As I worked on the project, I wrote a list of ideas to try and directions to explore. While I implemented and explored some of them, going through the entire list would take months at least and likely lead to many different techniques to try. It's also difficult to tell how successful each attempt is as there are a multitude of factors that can influence the measures of success. My

experience here emphasizes to me the importance of pipelines, being able to make incremental changes, analyze the results, and then hopefully improve the model.

The biggest improvement would be to design an effective pipeline that is “stubbed” at certain points of further interest. Features could be transformed or dropped and then tested on a variety of models in order to gauge improvement.