Two Sigma Connect: Rental Listing Inquiries

DSI SM3 Capstone Project

Nate Velarde – March 10, 2016

Executive Summary

The objective of this Kaggle Competition is to predict how popular an apartment rental listing is based on the listing content like text description, photos, number of bedrooms, price, etc. The data comes from renthop.com, an apartment listing website that is a portfolio company of Two Sigma Ventures, which is a division of Two Sigma Investments, a New York City-based hedge fund that uses a variety of technological methods, including artificial intelligence, machine learning, and distributed computing, for its trading strategies.

The ~50K apartment listings that comprise the dataset are from New York City and span the three month period between April and July 2016.

After EDA and feature engineering on both the numeric and text features, we constructed separate models for the numeric and text features to determine the effectiveness of each type of feature. We compared these sub-models against each other and also to a combined model ('Big_X') and a stacked model, where we added the predicted probabilities from our text models as features to our base numeric model.

In order to assess the validity of our models, we used Accuracy and Log Loss as our success metrics. Because of the imbalanced nature of the dataset – 70% of the observations were labeled 'low' interest, 23% 'medium' and 7% 'high' – our minimum benchmark was the baseline accuracy ~ 70% - the proportion of the majority class. The Kaggle competition is judged on log loss. From a benchmark perspective, we were targeting a log loss below 0.55 as that would place us in the top 10% of submissions to date.

Our preliminary conclusion is that simple beats complex - our basic numeric features models had superior predictive power than the text, combined and stacked models. Our models did not achieve accuracy levels meaningfully above the baseline accuracy, which was disappointing, but we learned much from implementing our modelling approach and believe it is conceptually sound. At the conclusion of this report, we provide several avenues to pursue to improve upon our results to date.

Problem Statement

We are presented with a supervised learning multi-class classification problem. The listing data is comprised with three types of data:

1. Numeric – number of bedrooms, bathrooms, monthly rent, geographic coordinates

2. Text – apartment features, detailed apartment description

3. Photos – listing photos of apartment

We wish to determine which aspects of a rental listing best predict the interest level (low, medium, high) that a listing receives. The target variable, interest_level, is defined by the number of inquiries a listing has in the duration that the listing was live on the site.

Overview of Data

Our dataset came in the form of a 20 MB json file. A listing of the datafields and a view of the data in a pandas Dataframe is provided below:

- bathrooms: number of bathrooms (numeric)
- bedrooms: number of bathrooms (numeric)
- building_id (numeric – did not use)
- created (numeric – created new features from this variable)
- description (text | further feature engineering)
- display_address (numeric – did not use)
- features: a list of features about this apartment (text | further feature engineering)
- latitude (numeric)
- listing_id (numeric – did not use)
- longitude (numeric)
- manager_id (numeric – created new features from this variable)
- photos: a list of photo links (converted the number of links to a numeric variable)
- price: in USD (numeric)
- street_address (numeric – did not use)
- interest_level: this is the target variable. It has 3 categories: 'high', 'medium', 'low'

	10	**10000**
bathrooms	1.5	1
bedrooms	3	2
building_id	53a5b119ba8f7b61d4e010512e0dfc85	c5c8a357cba207596b04d1afd1e4f130
created	2016-06-24 07:54:24	2016-06-12 12:19:27
description	A Brand New 3 Bedroom 1.5 bath ApartmentEnjoy ...	
display_address	Metropolitan Avenue	Columbus Avenue
features	[]	[Doorman, Elevator, Fitness Center, Cats Allow...
interest_level	medium	low
latitude	41	41
listing_id	7211212	7150865
longitude	-74	-74
manager_id	5ba989232d0489da1b5f2c45f6688adc	7533621a882f71e25173b27e3139d83d
photos	[https://photos.renthop.com/2/7211212_1ed4542e...	[https://photos.renthop.com/2/7150865_be3306c5...
price	3000	5465
street_address	792 Metropolitan Avenue	808 Columbus Avenue

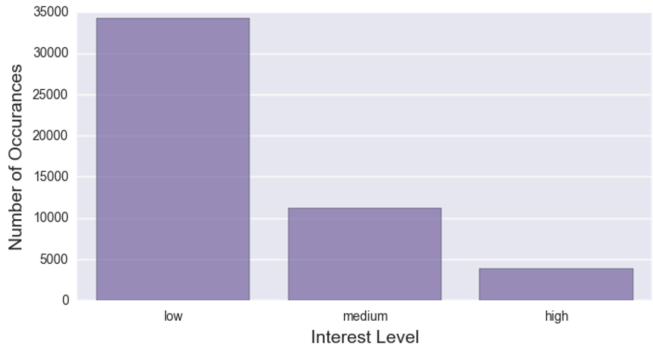
EDA | Visualizations

We performed EDA and create multiple visualizations to try to get a sense of which features impacted interest level. One of the challenges of this dataset is the imbalanced nature of the classes. 70% of the observations were labeled 'low' interest, 23% 'medium' and 7% 'high'. Overall, our EDA and visualizations, while interesting, yielded few clues as to which features provided predictive power in determining interest level.

All of the visualizations can be found in the *EDA_and_Numeric_Feature_Engineering_FINAL* notebook located in the *ipbnb* folder. We show several visualizations excerpted below.

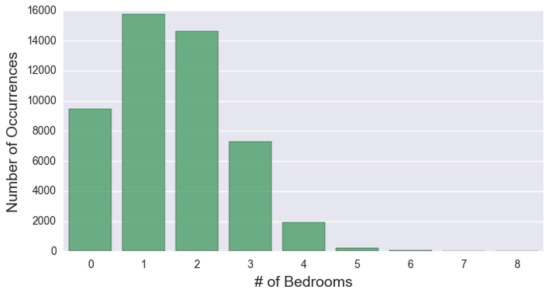
EDA | VISUALIZATIONS

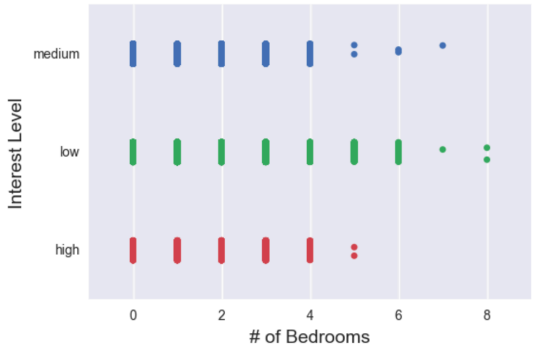
- Imbalanced classes – 'Low' interest level accounted for 70% of the observations, 'Medium' – 23%, 'High' – 7%



EDA | VISUALIZATIONS

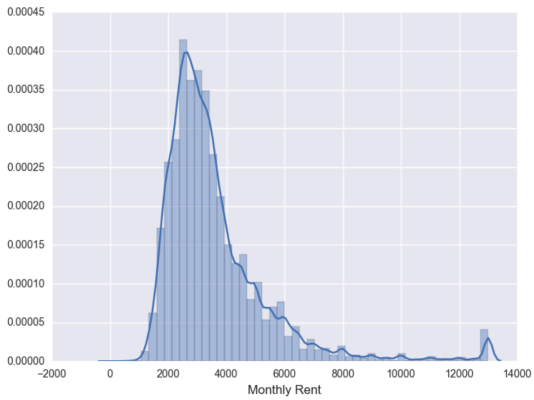
- In general, EDA and visualizations did not yield a lot of insight – interest level is fairly evenly distributed by number of bedrooms – similar results for the other numerical features





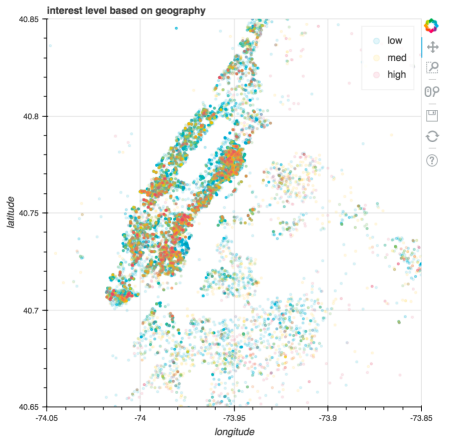
EDA | VISUALIZATIONS

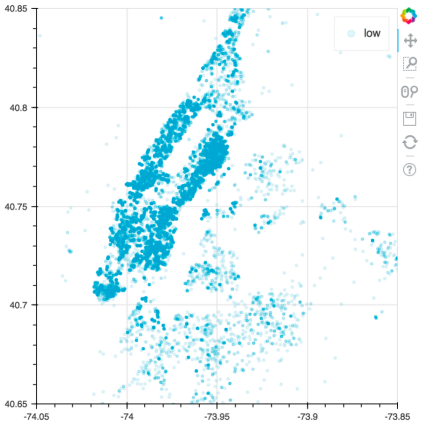
- Monthly rent was an important feature (per our models) – distribution was skewed to the right



EDA | VISUALIZATIONS

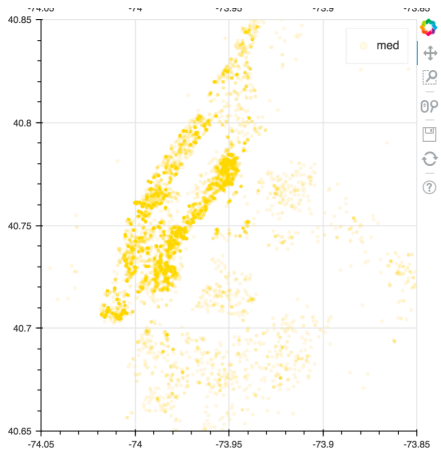
- Location, location, location - meh

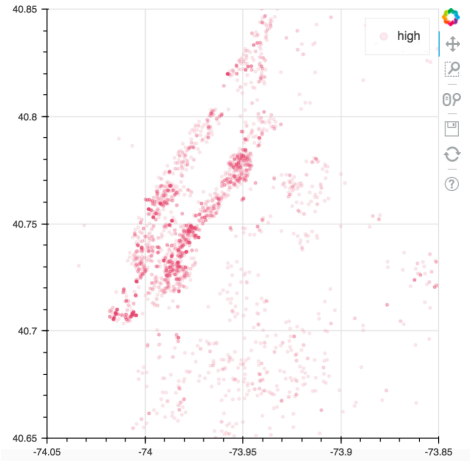




EDA | VISUALIZATIONS

- Location, location, location - meh





Feature Engineering

We conducted some simple feature engineering on the numeric variables. These included:

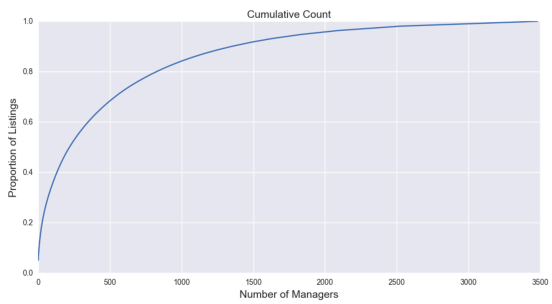
- Number of photos (based on number of photo links)
- Number of features listed
- Length (number of words) of apartment description
- Binned manager_id by number of listings per manager
- Converted listing date created into a pandas datetime object and binned the day of the month the listing was created

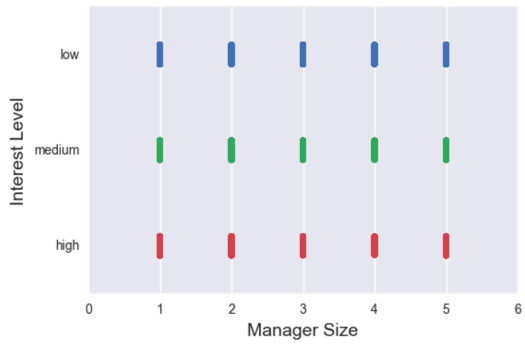
The intuition behind most of our engineered features was straightforward. Since 1/3 of managers accounted for 80% of listings, we decided to bin the manager_id by the number of listings per manager on the hypothesis that larger managers achieved their scale by better managing their buildings and/or had superior quality properties and thus would be able to achieve higher interest level in their listings all else equal. However, our visualization of interest level by manager bin size is fairly evenly distributed indicating that manager size had little predictive power. The ineffectiveness of manager bin size was confirmed via by our modelling (logistic regression with lasso coefficients and decision tree feature importance).

Number of photos yielded the most concrete takeaway from our EDA. One sure way to ensure a low interest level is to not include a picture in a listing.

ENGINEERED FEATURE VISUALIZATIONS

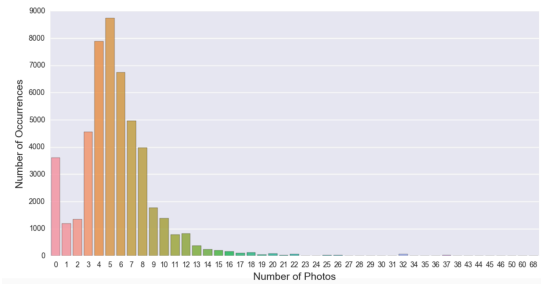
- Not much added insights

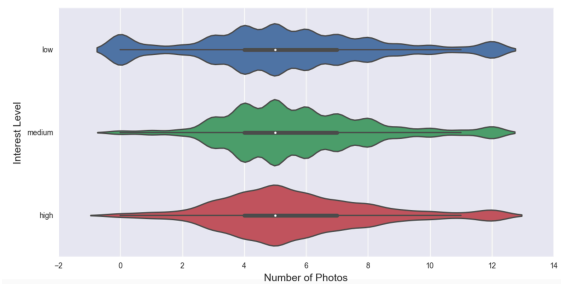




ENGINEERED FEATURE VISUALIZATIONS

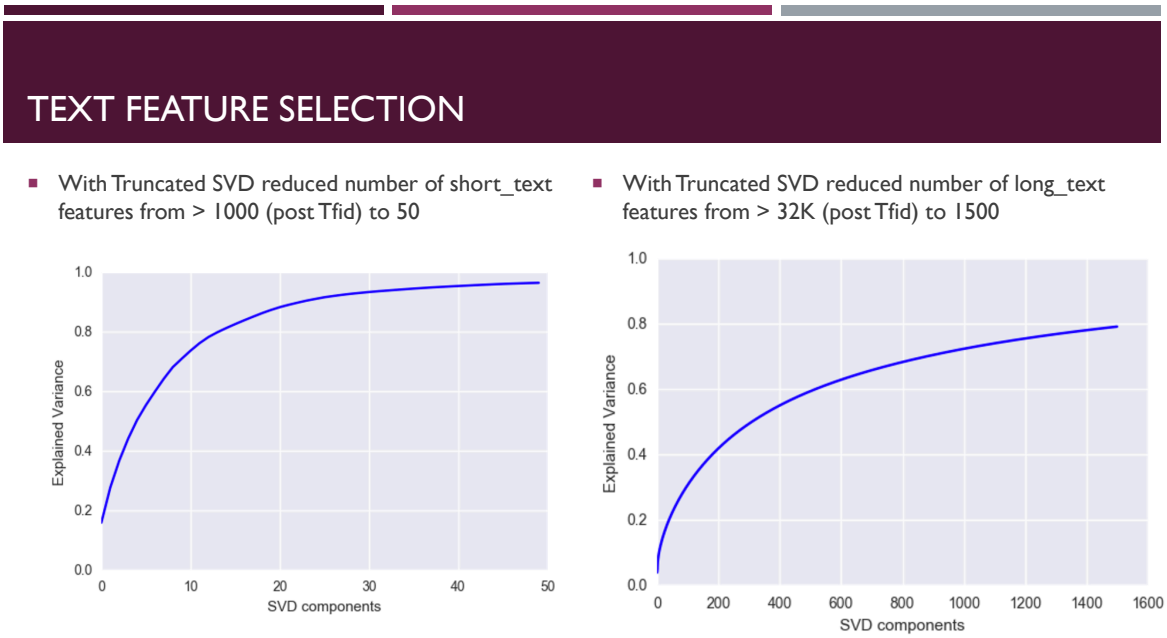
- Number of photos in listing





Our feature engineering on our text features was more involved. We cleaned the text using regex and eliminated stop-words before transforming the text using a Tfid-Vectorizer. On the long text, we tried porter stemming and lemmatization from the NTLK library, but these did not yield materially different results, so in the final version of our models, we ended up not using these techniques on the long-text..

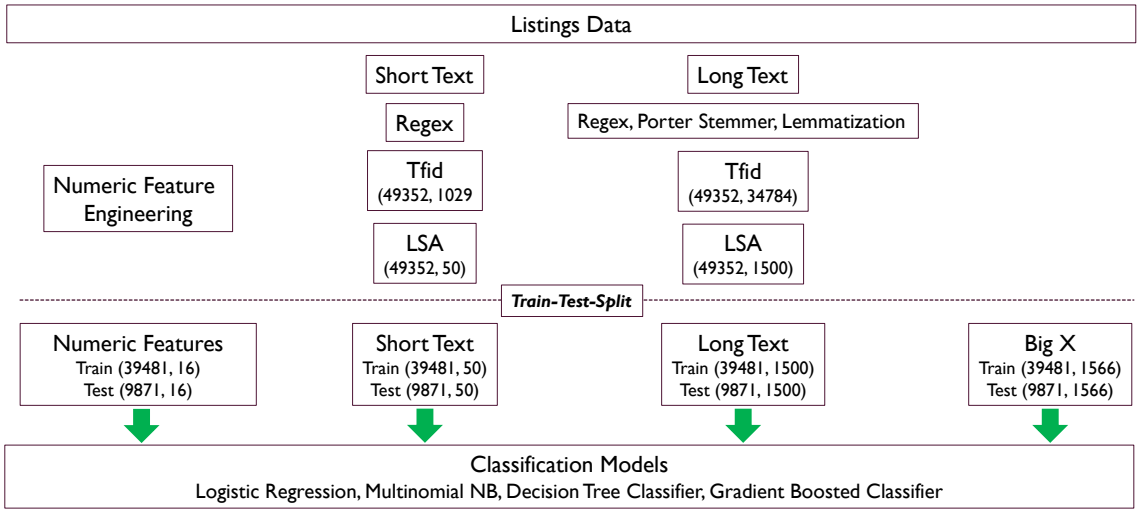
Given the size of our corpus (90K unique words in the uncleaned description feature and 35K post Tfid), we were concerned about dimensionality. Therefore we used LSA (Truncated SVD) for automatic feature selection and reduced the number of text features significantly. However, in the modelling stage we had an issue where our attempt to use Naive Bayes classification models failed when confronted with negative values in the Truncated SVD matrix. As a check, we went back and applied MultinomialNB classifier (which are well suited as a base model for sparse matricies that arise from text data) to our text models without LSA. As the results were largely similar to the models with LSA, we are comfortable that our finding that text features have no predictive power was not due to any implementation errors on our part.



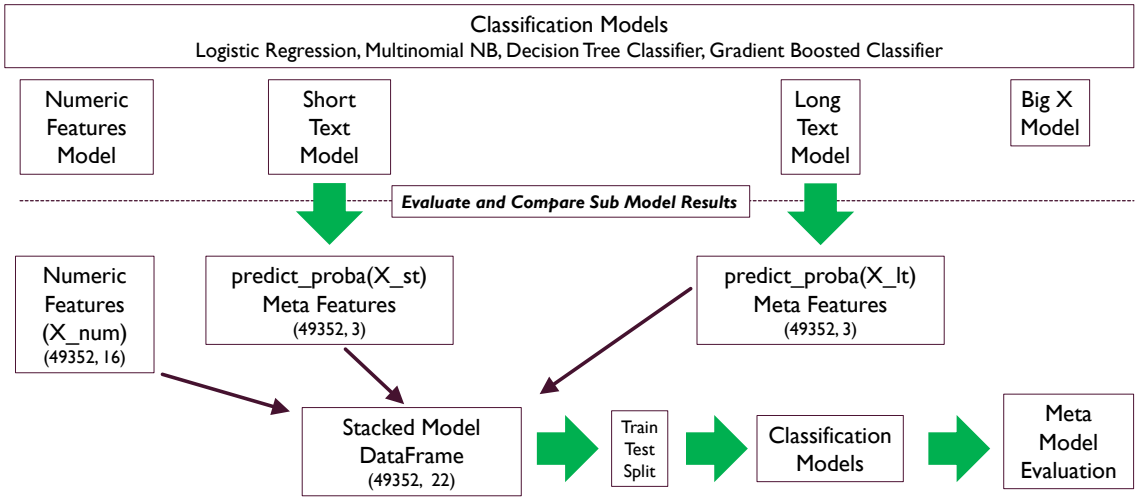
Modelling Approach

Given the different types of data we had to work with, we believed that developing models for each type of data – numeric, short-text (features list) and long-text (description) – would be useful to determine the effectiveness of each type of feature. We compared these sub-models against each other and also to a combined model ('Big_X' model where all features were included in a single model) and to a stacked or meta-model, where we added the predicted probabilities from our text models as features to our base numeric model. A good discussion of stacked modelling can be found here: http://blog.kaggle.com/2016/12/27/a-kagglers-guide-to-model-stacking-in-practice/

MODELLING WORKFLOW I



MODELLING WORKFLOW II



For each sub-model, we ran 3 models – Logisitic Regression with Lasso, Decision Trees Classifier and Gradient Boosted Classifier. We tuned each of these models via GridSearch CV – please see individual model notebooks in ipynb folder. For our stacked model, we used the predicted probabilities of the best scoring text sub-models. We did attempt a simple majority voting ensemble model, but this was computationally expensive on the text and combined models.

From a coding perspective, we ran our models by using little 'p' pipelines. This was accomplished via helper functions.

```
def evaluate_model(model):

    model.fit(X_train, y_train)

    train_score = model.score(X_train, y_train)
    test_score = model.score(X_test, y_test)

    y_hat = model.predict(X_train)
    y_pred = model.predict(X_test)

    y_hat_proba = model.predict_proba(X_train)
    y_pred_proba = model.predict_proba(X_test)

    return {'model': model,
            'X': X,
            'X_test': X_test,
            'X_train': X_train,
            'y': y,
            'y_test': y_test,
            'y_train': y_train,
            'y_hat': y_hat,
            'y_pred': y_pred,
            'y_hat_proba': y_hat_proba,
            'y_pred_proba': y_pred_proba,
            'train_score': train_score,
            'test_score': test_score
           }
```

```
def model_summary(model, data_dictionary):

    class_rpt = classification_report(data_dictionary['y_test'], data_dictionary['y_pred'])
    log_loss_score_train = log_loss(data_dictionary['y_train'], data_dictionary['y_hat_proba'])
    log_loss_score_test = log_loss(data_dictionary['y_test'], data_dictionary['y_pred_proba'])

    print ("Model Summary Report:\n")
    print (model)
    print ("\n")
    print ("Classification Report:\n")
    print (class_rpt)
    print ("Accuracy on training set: {:4f}".format(model.score(data_dictionary['X_train'],
                                                                 data_dictionary['y_train'])))

    print ("Accuracy on test set:     {:4f}".format(model.score(data_dictionary['X_test'],
                                                                data_dictionary['y_test']))))
    print ("\n")
    print ("Log Loss on training set: {:4f}".format(log_loss_score_train))
    print ("Log Loss on test set:     {:4f}".format(log_loss_score_test))
```

Example output generated from model helper functions:

Tuning Model 1 via GridSearch

```
params = {'C': [0.0001, 0.001, 0.01, 0.1, 1.0, 10.0],
          'penalty': ['l2']
         }

gs_clf1 = GridSearchCV(clf1,
                       params,
                       n_jobs=-1,
                       cv=10,
                       verbose=1
                      )
```

```
X_num_gs_clf1_dictionary = evaluate_model(gs_clf1)
```

```
Fitting 10 folds for each of 6 candidates, totalling 60 fits

[Parallel(n_jobs=-1)]: Done  42 tasks      | elapsed:    8.0s
[Parallel(n_jobs=-1)]: Done  60 out of  60 | elapsed:   12.6s finished
```

```
model_summary(gs_clf1, X_num_gs_clf1_dictionary)
```

```
Model Summary Report:

GridSearchCV(cv=10, error_score='raise',
       estimator=LogisticRegression(C=0.001, class_weight=None, dual=False, fit_intercept=True,
          intercept_scaling=1, max_iter=100, multi_class='ovr', n_jobs=-1,
          penalty='l2', random_state=82, solver='liblinear', tol=0.0001,
          verbose=0, warm_start=False),
       fit_params={}, iid=True, n_jobs=-1,
       param_grid={'penalty': ['l2'], 'C': [0.0001, 0.001, 0.01, 0.1, 1.0, 10.0]},
       pre_dispatch='2*n_jobs', refit=True, return_train_score=True,
       scoring=None, verbose=1)


Classification Report:

             precision    recall  f1-score   support

          0       0.00      0.00      0.00       765
          1       0.69      1.00      0.82      6813
          2       0.00      0.00      0.00      2293

avg / total       0.48      0.69      0.56      9871

Accuracy on training set: 0.695702
Accuracy on test set:     0.690001
```

One of the unexpected struggles we faced was in a basic task – the train-test-split. As we wanted apples-to-apples comparisons between the sub-models, we wanted to have the same train and test instances for all of the sub-models. To accomplish this we had to combine the feature sets, perform the test-train split and then break the feature sets again into numeric, short-text and long-text components again. From Scott Tarlow's presentation, we discovered that this could have been accomplished by using Stratified Shuffle Split in sklearn – http://scikit-learn.org/stable/modules/generated/sklearn.model_selection.StratifiedShuffleSplit.html

Results

Our overall finding is that simpler is better than complex. Our simple numeric features models outperformed our combined and stacked models. NLP of the text features did not seem to have any predictive value. Relative to baseline accuracy of 70%, our accuracy scores across all of our models were disappointing. Our best model was the X_num sub model using a Boosted Gradient Classifier which had strong training accuracy and log loss scores. Unfortunately, these scores could be sustained using test data. Our results are summarized in the slides below:

X_NUM SUB-MODEL RESULTS

- Gradient Boosted Classifier model yielded the best results

	Accuracy		Log Loss	
Model	**Train**	**Test**	**Train**	**Test**
Logistic Regression	0.6851	0.6922	0.7639	0.7688
Decision Tree Classifier	0.7055	0.6976	0.6917	0.7336
Gradient Boosted Classifier	**0.7767**	**0.7193**	**0.5264**	**0.6647**

X_SHORT_TEXT SUB-MODEL RESULTS

- Suspect results or no real information provided by text data

	Accuracy		Log Loss	
Model	**Train**	**Test**	**Train**	**Test**
Logistic Regression	0.6952	0.6923	0.9079	0.9095
Decision Tree Classifier	0.6953	0.6923	0.7857	0.7921
Gradient Boosted Classifier	0.6953	0.6923	0.8096	0.8133
Multinomial Naïve Bayes Classifier	0.6982	0.6862	1.0162	1.0162

X_LONG_TEXT SUB-MODEL RESULTS

- Suspect results or no real information provided by text data

Model	Accuracy		Log Loss	
	Train	Test	Train	Test
Logistic Regression	0.6953	0.6923	0.9096	0.9113
Decision Tree Classifier	0.6953	0.6923	0.7874	0.7924
Gradient Boosted Classifier	0.6953	0.6923	0.8092	0.8134
Multinomial Naïve Bayes Classifier	0.6958	0.6902	0.7979	0.8033

BIG 'X' MODEL SUMMARY RESULTS

- Simple beats complex

Model	Accuracy		Log Loss	
	Train	Test	Train	Test
Logistic Regression	0.6951	0.6922	0.7639	0.7688
Decision Tree Classifier	0.6998	0.6924	0.7298	0.7436
Gradient Boosted Classifier	0.6953	0.6923	0.7790	0.7846

STACKED MODEL RESULTS SUMMARY

- Used 'y_pred_proba' from text models as additional features to base 'X_num' model
- Simplest 'X_num' model performs the best
- Text features add no value

Model	Accuracy		Log Loss	
	Train	Test	Train	Test
Logistic Regression	0.6957	0.6900	0.7634	0.7705
Decision Tree Classifier	0.7029	0.6954	0.6879	0.7034
Gradient Boosted Classifier	0.7798	0.7099	0.5175	0.6707

Outside of the bottom-line scores, from an automated feature selection perspective, it was interesting to see which features each model found most/least important and in what order. Price and location scored high in all three models.

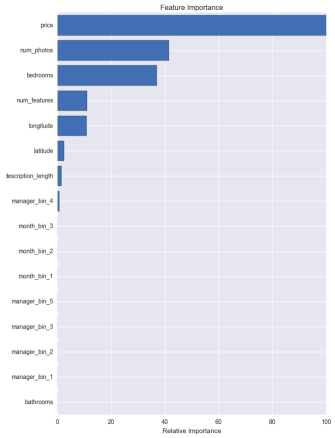
X_NUM SUB-MODEL FEATURE IMPORTANCE

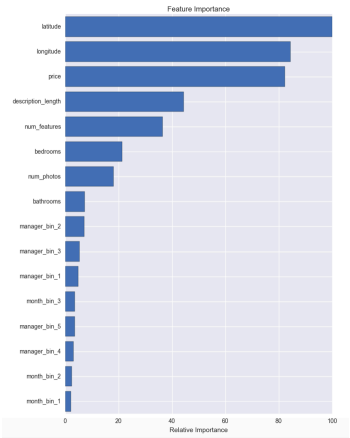
- Logistic Regression with Lasso Feature Coefficients

	Feature	coef_high	coef_low	coef_medium	high_abs_coef	low_abs_coef	med_abs_coef
2	price	-0.123141	0.238975	-0.152796	0.123141	0.238975	0.152796
0	bathrooms	-0.034013	0.050512	-0.041195	0.034013	0.050512	0.041195
7	longitude	0.025883	-0.008005	0.011990	0.025883	0.008005	0.011990
3	num_photos	0.025404	-0.064929	0.050286	0.025404	0.064929	0.050286
12	manager_bin_5	-0.019418	0.043558	-0.028478	0.019418	0.043558	0.028478
4	num_features	-0.015543	-0.051190	0.062809	0.015543	0.051190	0.062809
6	latitude	-0.013946	0.005717	-0.008155	0.013946	0.005717	0.008155
9	manager_bin_2	0.013721	-0.048916	0.037766	0.013721	0.048916	0.037766
15	month_bin_3	-0.013227	0.016471	-0.005910	0.013227	0.016471	0.005910
1	bedrooms	0.013056	-0.087178	0.067962	0.013056	0.087178	0.067962
13	month_bin_1	0.012039	-0.018708	0.008659	0.012039	0.018708	0.008659
5	description_length	0.010005	-0.072147	0.069865	0.010005	0.072147	0.069865
8	manager_bin_1	0.004093	0.029940	-0.033786	0.004093	0.029940	0.033786
10	manager_bin_3	0.000494	-0.013699	0.013629	0.000494	0.013699	0.013629
14	month_bin_2	0.000426	0.002642	-0.003236	0.000426	0.002642	0.003236
11	manager_bin_4	0.000347	-0.010477	0.010382	0.000347	0.010477	0.010382

X_NUM SUB-MODEL FEATURE IMPORTANCE

- Decision Trees. Classifier versus Boosted Gradient Classifier





Suggestions for Further Research | Final Reflections

The high feature importance of 'unprocessed' latitude and longitude data in all our models is a clear signal that this area could yield significant improvements with additional work. We could reverse geocode from geographic coordinates to extract neighborhoods and listing distance from the center of those respective neighborhoods. Chris Petersen suggested we put the data in Postgres and use PostGIS in order to implement this idea.

Other than extracting the number of photos, the photo data was unused. The competition provides a 78.5 GB file of listing images organized by listing id. This would be an ideal data set to run through a machine learning program.

Even though the text data yielded little information in this iteration, there are several potential areas of improvement. In combination with more surgical regex, we could tune the n_gram parameter in TfidfVectorizer to capture potentially useful features like square footage information, other useful 1-3 word combinations, extracting broker names (as certain brokers may have skill at generating higher interest in their listings), etc. Due to time constraints and the fear that tuning the n_gram parameter to capture 1-3 word combination would be very computationally expensive, I did not attempt these suggestions.

In conclusion, despite disappointment in the results generated to date, I learned much from the overall project, especially in terms of feature engineering, feature extraction, NLP and the of building stacked/meta-models.

Guide to Notebooks and Files

Notebook File Name (ipynb folder)	**Description**
EDA_and_Numeric_Feature_Engineering	EDA, Visualizations and Numeric Feature Engineering
Rental Text Feature Engineering	Short and Long Text Cleaning , Tfid, LSA
Building_X	Combining dataframes, train-test-split, break-up into sub-models, pickle dataframes for use in sub-models
X_num_Sub_Models	Numeric Feature Sub-Models
X_st_Sub_Models	Short Text Sub-Models
X_lt_Sub_Models	Long Text Sub-Models
Big_X_Models	Combined Features Models
Stacked_Models	Stacked/Meta Models
Pickled_Files (folder)	Pickled Data and Models
Docs (folder)	Contains Capstone Report and Presentation