Complete Script

December 10, 2017

```
In [1]: from html.parser import HTMLParser
        import pandas as pd
        import numpy as np

        import statsmodels.formula.api as smf

        from sklearn.linear_model import LogisticRegressionCV as logCV
        from sklearn.svm import SVC
        from sklearn.model_selection import train_test_split,StratifiedShuffleSplit
        from sklearn.metrics import classification_report
        from sklearn.preprocessing import MinMaxScaler
```

1 Initial Data Visualization:

Visualizations can be found within most of our python files. Visualizing the data helped us understand how different models and interaction terms preformed. However, our initial visualization can be found here.

2 Initial Exploration of Various Models:

- Trees (Standard Tree, Random Forest Classifier, Bagged Trees)
- Linear Model (to get a sense of relevant predictors)

3 Exploration of Features/Data Cleaning:

- Testing for Significant Interaction Terms
- Using NLP to classify housing listings by the type of home
- Data Cleaning (describe more)
- Data Cleaning 2 (describe more)

4 Model Exploration

- Model 1 : Fitting a Random Forest Classifier on the 'typed' housing listings (see NLP) and exploring fitting a series of 2-class Random Forest Classifiers to improve prediction:
- Model 2: SVMs

  - SVM Initial Exploration
  - Hyperparameter Selection (of gamma and c)
- Model 3: Nural Nets with subpar success
- Model 4: KNN and logistic regression on cleaned, typed, and pre-processed data

5 Final Model:

We broke our final model into two complete functions. One cleans and pre-processes the data (cleanPreprocessData()), while the other actually runs our learning model. To clean the data, we took the most successful aspects of each individual model to create terms that begin to separate interest levels. For example, in New York, as expected, a balcony and access to a common outdoor space are highly valued; therefore, we created an interaction term "outdoor_score" that attempts to capture how "outdoor-sy" the building is. We began to generate these terms for a wide range of rental units and combinations. In addition, we used basic NLP to classify the units. This is used to create market expectations for feature combinations, price points, and more importantly price/feature. Using this, we are able to tell if a specific unit is listed above or below predicted market value for that specific type of home. For example, what renters look for in a studio is very different than the market for penthouses. Some of these interaction terms are "price_per_feature", "price_per_num_lux", and "price_feature_ratio" among others. However, the most important interaction term is the market immediately surrounding that specific rental. We determined this through computing an expected interest level for every building and unit manager. The initial exploration of these terms can be found here in cell 6. This specific separation allowed us to use a multi-class logrithmic classifier to determine the interest level of units.

Ultimately, our model is two parts: a logrithmic classifier for the best data and a SVM for any data that cannot be classified well with the log model. We determine data quality by whether we know a units expected interest level, "prob_buildManager". If we have this information, we are able to classify the interst_level of a unit with 92% accuracy. Wihtout it, our accuracy falls to only 71%, not much better than guessing. Therefore, given this dataset, our expected accuracy is approximatly 81%. In future models, there may be a way to predict "prob_buildManager" by running a KNN on the lat long data. If we can do this with high accuracy we can improve our overall model. Essentially, we found that the mantra of real-estate agents is valid. When buying or renting a home, **'LOCATION, LOCATION, LOCATION.'**

```
In [2]: # These are our helper functions and classes.
        # They each assist with a different part
        # of cleaning and preprocessing the data.

        class MLStripper(HTMLParser):
            def __init__(self):
                self.reset()
                self.strict = False
                self.convert_charrefs= True
                self.fed = []
            def handle_data(self, d):
                self.fed.append(d)
            def get_data(self):
                return ''.join(self.fed)
        def strip_tags(html):
            s = MLStripper()
            s.feed(html)
            return s.get_data()

        def descrClean(x):
            des = strip_tags(x)
            return des.lower()

        # Function to Classify Unit Types
        def unitType(x, types):
            homeType = {
                }
            for lst in types:
                homeType[lst[0]] = False

            for lst in types:
                for w in lst:
                    if w in x:
                        return lst[0]
            return 'other'

        #Dealing with lofts or studios that have no rooms
        def pricePerRoom(row):
            if row['rooms']==0:
                return row['price']/.5
            else:
                return row['price']/row['rooms']

In [3]: def cleanPreprocessData(train, test):
            print('Cleaning...')
            df = train
            test['test'] = True

            #Merge the two files to clean and comput interaction terms.
            df = df.append(test)
            df.reset_index(drop=True, inplace=True)
            df['test'].fillna(False, inplace=True)

            #Clean the column names for regressions and ML Models
            df.columns = [c.replace(' ', '_') for c in df.columns]
            df.columns = [c.replace('-', '_') for c in df.columns]
            df.columns = [c.replace('/', '_') for c in df.columns]

            #Confirm types for columns with numbers
            df['bedrooms'] = df['bedrooms'].apply(float)
            df['bedrooms'].fillna(0, inplace=True)
```

```
df['bathrooms'].fillna(0, inplace=True)

#Drop meaningless columns in data
df.drop(['index', 'level_0'], axis=1, inplace=True)

#Map Interest levels to values for OLS Regression
df['interestVal'] = df['interest_level'].map({'high': 1,
                                              'medium': 0.5,
                                              'low':0})

#Clean the HTML from descriptions to allow for NLP
df['description'] = df['description'].apply(descrClean)

# Aggregate to create one laundry in building
# column that isn't case sensitive
df['laundry_in_building'] = df.apply(lambda row:
                                     row['Laundry_in_Building'] or
                                     row['Laundry_In_Building'],
                                     axis=1)

# Drop old laundry in building columns
df = df.drop(['Laundry_in_Building', 'Laundry_In_Building'], axis=1)

cleanedDf = df

print('Cleaning Complete. ' +
      'Processing descriptions to determine type...')

# To determine the type of rental unit, we conduct a basic NLP
# Define basic unit types
apt = ['apartment', 'apt']
condo = ['condominium', 'condo']
walkUp = ['walk_up', 'walk-up', 'walkup', 'walk up']
studio = ['studio']
ph = ['ph', 'penhouse']
townhome = ['townhome', 'duplex', 'townhouse']
loft = ['loft']

types = [apt, condo, walkUp, studio, ph, townhome, loft]

#Determine rental type
df['type'] = df['description'].apply(lambda x : unitType(x, types))

#Determine if a type has been found
df['foundType'] = ~df['type'].str.contains('other')

#Create binary dummy columns for each type
df = pd.concat([df, pd.get_dummies(df['type'])], axis=1)
```

```
#Combine and drop the two loft column
df['loft'].fillna(False, inplace=True)
df['loft'] = df[['loft', 'Loft']].apply(lambda row :
                                       row['loft'] or
                                       row['Loft'],
                                       axis=1)
df.drop('Loft', axis=1, inplace=True)

cleanedTyped = df

print('Typing Complete. Generating Interaction Terms...')

# Generate interaction terms to find differentiators
# Luxury Score Term - higher the score means the more
# luxury items included
df['lux_score'] = (df['Exclusive'] + df['Doorman'] +
                   df['Outdoor_Space'] + df['New_Construction'] +
                   df['Roof_Deck'] + df['Fitness_Center'] +
                   df['Swimming_Pool'] + df['Elevator'] +
                   df['Laundry_in_Unit'] +
                   df['Hardwood_Floors']) / 10

# Group data by buildings and agents to determine expected interest
# -----MAGIC FEATURE-----
agentGroup = df.groupby(['manager_id']).mean()
buildingGroup = df.groupby(['building_id', 'manager_id']).mean()

buildingAvg = buildingGroup[['interestVal']]
buildingAvg.columns = ['prob_interest_building']
buildingAvg.reset_index(inplace=True)

managerAvg = agentGroup[['interestVal']]
managerAvg.columns = ['prob_interest_manager']
managerAvg.reset_index(inplace=True)

#Merge back to original DF
df = df.merge(managerAvg, on='manager_id', how='left')
df = df.merge(buildingAvg, on=['building_id', 'manager_id'],
              how='left')

#Compute expected interest by building and manager
df['prob_buildManager'] = (df['prob_interest_building']+
                           df['prob_interest_manager'])/2

#Count rooms and determine price per room
df['rooms'] = df['bedrooms']+df['bathrooms']
```

```
df['price_per_room'] = df[['price', 'rooms']].apply(pricePerRoom,
                                                    axis=1)

# Number of Luxury Features Term
df['num_luxury'] = (df['Exclusive'] + df['Doorman'] +
                    df['Outdoor_Space'] + df['New_Construction'] +
                    df['Roof_Deck'] + df['Fitness_Center'] +
                    df['Swimming_Pool'] + df['Elevator'] +
                    df['Laundry_in_Unit'] + df['Hardwood_Floors'])

# Number of Features per Listing
df['num_features'] = df['features'].apply(len)

# ADA compatible interaction term
# 1 if both elevator and wheelchair access, 0 if one or
# neither are included
df['ada'] = df['Elevator'] * df['Wheelchair_Access']

# Create transformed term that creates a score for outdoor spaces
# Higher the score, the more of these features are included
df['outdoor_score'] = (df['Outdoor_Space'] + df['Balcony'] +
                       df['Common_Outdoor_Space'] +
                       df['Garden_Patio'] +
                       df['Roof_Deck'] + df['Terrace']) / 6

# Create interaction term for fitness oriented
# 1 if both swimming pool and fitness center are included,
# 0 if one or neither included
df['fitness_oriented'] = df['Fitness_Center'] * df['Swimming_Pool']

# Create interaction term for doorman/exclusive
# 1 if both are included, 0 if one or neither are included
df['door_excl'] = df['Doorman'] * df['Exclusive']

# Create interaction term for cats and dogs allowed
# 1 if both are allowed, 0 if one or neither are allowed
df['pets_allowed'] = df['Cats_Allowed'] * df['Dogs_Allowed']

#Compute price per feature and price per luxury feature.
#If no features exist, the value is empty
df['price_per_feature'] = df['price']/df['num_features']
df['price_per_feature'].replace(np.inf, np.nan, inplace=True)

df['price_per_num_lux'] = df['price']/df['num_luxury']
df['price_per_num_lux'].replace(np.inf, np.nan, inplace=True)

#Determine expected prices by type of unit
g1 = df.groupby(['type']).mean()
```

```
g1.reset_index(inplace=True)

#Columns we wish to average
avgs = g1[['type','lux_score', 'num_features',
           'num_luxury','outdoor_score', 'price_per_num_lux',
           'price_per_feature']]

pd.options.mode.chained_assignment = None  # default='warn'

#Rename columns and merge back to original DF
avgs.columns = ['avg_'+x for x in avgs]
avgs.rename(columns={'avg_type':'type'}, inplace=True)
df = pd.merge(df, avgs, on='type')

#If no price was found, set the price for the column as average
# to avoid skewing the data
df['price_per_num_lux'].fillna(df['avg_price_per_num_lux'],
                               inplace=True)
df['outdoor_score'].fillna(df['avg_outdoor_score'], inplace=True)
df['lux_score'].fillna(df['avg_lux_score'], inplace=True)
df['price_per_feature'].fillna(df['avg_price_per_feature'],
                               inplace=True)

df['price_lux_ratio'] = (df['price_per_num_lux']/
                         df['avg_price_per_num_lux'])
df['outdoor_ratio'] = df['outdoor_score']/df['avg_outdoor_score']
df['lux_ratio'] = df['lux_score']/df['avg_lux_score']
df['price_feature_ratio'] = (df['price_per_feature']/
                             df['avg_price_per_feature'])


#Compute the number of photos included in the listing
df['numPhotos'] = df['photos'].apply(len)

#Listing id is an arbitrary int label assined to each listing.
# not useful for classification
df.drop(['listing_id'], axis=1, inplace=True)

#Output new training and testing datasets
train = pd.DataFrame(df[df['test']==False].dropna())
train.reset_index(drop=True, inplace=True)
train.drop('test', inplace=True, axis=1)

test = pd.DataFrame(df[df['test']])
test.reset_index(drop=True, inplace=True)
test.drop('test', inplace=True, axis=1)

train.to_json('./cleaned/train.json')
```

```
            test.to_json('./cleaned/test.json')
            print('Cleaning and Preprocessing complete.')
            return cleanedDf, cleanedTyped, train, test

In [4]: def runModel():

            train = pd.read_json('./cleaned/train.json')
            test = pd.read_json('./cleaned/test.json')

            #Determine the columns with which to run an OLS, exclude the
            # indicator column
            data = train.drop('interestVal',
                              axis=1).select_dtypes(exclude=['object'])

            #join columns to build to equation
            equation = ('+').join(data.columns)

            #run the OLS to determine significant columns
            model = smf.ols('interestVal~'+equation, data=train).fit()

            #make a DF of significant features
            sig_features = pd.DataFrame(model.pvalues, index=data.columns,
                                        columns={'P_Value'})

            sigCols = sig_features[sig_features['P_Value']<.1].index.values
            print('The data has {} significant columns'.format(len(sigCols)))
            print('The significant columns are: ')
            print(sig_features[sig_features['P_Value']<.1])

            sigCols = np.append(sigCols, 'interest_level')

            #Create a simplified df with only the significant columns
            validLogTest = test[~pd.isnull(test['prob_interest_building'])]

            simpleTrain = train[sigCols]
            simpleTest = validLogTest[sigCols]

            X_train, X_test, y_train, y_test = train_test_split(simpleTrain.drop('i
                                                                simpleTrain['intere
                                                                test_size=0.2,
                                                                random_state=42)

            print('Running Logistic Regression on best data...')
            print('Training Logistic Regression...')
            logReg = logCV(cv=10)
            logReg.fit(X_train, y_train)
            print(classification_report(logReg.predict(X_test), y_test))
```

```
logReg.fit(simpleTrain.drop('interest_level', axis=1),
           simpleTrain['interest_level'])

print(str(len(simpleTest)) + ' samples. Predicting Interest Levels...')
logPreds = logReg.predict(simpleTest.drop('interest_level', axis=1))

validLogTest['interest_level'] = logPreds

print('Running SVM on lower quality data.')

# Because some of the data is still unknown,
# we have to use an SVM to classify about 48% of the test data

# Drop the columns that are objects, but keep the
# interest_level classification
svm_tr = train.drop(['interestVal','building_id','created',
                     'description','display_address','features',
                     'manager_id','photos','type','street_address',
                     'prob_interest_manager', 'prob_interest_building',
                     'prob_buildManager'],axis=1)

# Break into training and test sets to train SVM model
X_svm_train, X_svm_test, y_svm_train, y_svm_test = train_test_split(svm
                                                                    svm
                                                                    ran

scaling = MinMaxScaler(feature_range=(-1,1)).fit(X_svm_train)

# Train SVM with C and gamma values calculated in hyperparameter_select
svm_model = SVC(cache_size=7000,
                decision_function_shape='multinominal',
                C=1.7782794100389228,
                gamma=9.9999999999999995e-07)
print('Training SVM on lower quality data...')
svm_model.fit(X_svm_train, y_svm_train)
print(classification_report(svm_model.predict(X_svm_test), y_svm_test))

# Drop features of type object from test set
svmTest = test[pd.isnull(test['prob_interest_building'])]
svmTestSimple = svmTest.select_dtypes(exclude=['object'])
svmTestSimple.drop(['interestVal', 'prob_buildManager', 'interest_level
                    'prob_interest_building', 'prob_interest_manager'],
                   axis=1, inplace=True)
svmTestSimple.reset_index(drop=True, inplace=True)

print(str(len(svmTestSimple)) + ' samples. Fitting SVM of training...')
svm_model.fit(svm_tr.drop('interest_level', axis=1),
              svm_tr['interest_level'])
```

```
            # Predict unclassified data with SVM model
            print('Predicting Interest Levels for Test data...')
            svm_preds = svm_model.predict(svmTestSimple)
            svmTest['interest_level'] = svm_preds

            # Add classification results from SVM to classification results
            # from log regression
            test = pd.concat([validLogTest, svmTest])

            print('Predictions complete.')
            return test['interest_level'], test
```

```
In [5]: train = pd.read_json('./raw_data/train_data.json')
        test = pd.read_json('./raw_data/test_data.json')

        cleanedDF, cleanedTyped, train, test = cleanPreprocessData(train, test)
```

```
Cleaning...
Cleaning Complete. Processing descriptions to determine type...
Typing Complete. Generating Interaction Terms...
Cleaning and Preprocessing complete.
```

```
In [57]: preds, test = runModel()
```

```
The data has 7 significant columns
The significant columns are:
                         P_Value
Common_Outdoor_Space    0.029706
No_Fee                  0.000172
bathrooms               0.027459
prob_buildManager       0.000000
prob_interest_building  0.000000
prob_interest_manager   0.000000
rooms                   0.001446
Running Logistic Regression on best data...
Training Logistic Regression...
             precision    recall  f1-score   support

       high       0.83      0.81      0.82       646
        low       0.98      0.92      0.95      6226
     medium       0.70      0.88      0.78      1512

avg / total       0.92      0.90      0.91      8384

3776samples. Predicting Interest Levels...
Running SVM on lower quality data.
Training SVM on lower quality data...
```

```
             precision    recall  f1-score   support

       high       0.10      0.65      0.17        93
        low       0.98      0.72      0.83      8010
     medium       0.07      0.48      0.12       281

avg / total       0.94      0.71      0.80      8384

3658 samples. Fitting SVM of training...
Predicting Interest Levels for Test data...
Predictions complete.
```

Our expected accuracy is

$$\frac{.71(3658) + .90(3776)}{3658 + 3776} = 80.65\%.$$

```
In [61]: preds.to_csv('test_predictions.csv')
```

```
In [63]: test.to_csv('test_with_preds.csv')
```