Wine Quality

ORIE 4741 Final Project Report

ajd242, eer48, lfl42

1 Problem

When valuing a bottle of wine we often consider the producer, its rating, the region which it is from and any reviews we have available to us. Unfortunately, we don't have a formula for determining if this wine is a good buy at its price point, but what if we could? Our goal is to predict the selling price of a bottle of wine given its ratings, reviews, region, etc. This model can help producers understand the value of the wine they are making, and to help consumers make smarter purchases. Ultimately, we hope our model for predicting the price of a bottle of wine will create a more efficient market for buying and selling wine.

2 Dataset

The following table describes the features of importance within our dataset that we used in our regression analyses. We considered adding features such as 'region_2'; however, we would be cutting out too much of the data as there are only 19 sub-regions. We also considered using the 'winery' and 'designation' features but realized these were too granular as there are >11,000 wineries and >22,000 designations.

Features we used	**Description**
Description	Review provided by sommeliers (i.e. professionals, not consumers)
Country	The country in which the wine was produced
Points	As determined from an average of Official Rating Agencies
Variety	The particular grape varietal or blend of a wine
Region	The particular wine region from which the bottle originates
Price (What we are predicting)	The industry standard price as obtained from Wine.com

Importing Messy Data

Our first approach was to import data using Python's pandas.read_csv. The first issue we ran into after importing occurred during an inspection of our raw data in Excel. We noticed that many of the foreign characters such as 'é' were not being displayed accurately in our imported dataset (see Figure 2.1). This improper formatting concerned us because the interpretability of the dataset was difficult for potential readers and us. We found the problem to be that Excel expected characters to be encoded as UTF-8. These incorrect outputs were a result of UTF-8 only including ASCII characters; however, our data had a significant amount of non-ASCII characters, resulting in a UnicodeDecodeError.

We initially tried changing the file type and even started writing code to replace strange character sequences. This process was labor intensive and took way too long. Then we thought it would make sense to simply remove the data containing any foreign characters and in doing so, learned our dataset was biased towards non-foreign wines and reviews. We eventually learned our root problem was the character encoding of a .csv file in which our data was stored. We ended up using .json files instead of .csv to store our data. This type of file does not allow us to examine our raw data in Excel, but when importing the file into the .ipynb notebook we found that the method pandas.read_json read-in foreign languages efficiently and accurately without causing the fields to expand or transform. This was mainly a benefit of how .json files structure and store data in defined fields while also including the correct encodings for reading numbers as Float64 rather than strings.

Examples of some issues:

What it should be	What it opened as
Bodega Carmen Rodríguez	Bodega Carmen RodrÃguez
Château Lagrézette	ChÃteau LagrÃ©zette
Bergström	BergstrÃ¶m
Domaine de la Bégude	Domaine de la BÃ©gude
La Brûlade	La BrÃlade

Figure 2.1

Cleaning the Data

When we examined our data, we realized some entries in price column which are NaN (i.e., the price at index 150922) then we proceeded to remove any values in the points and price columns equal to NaN. Our data trimming resulted in the following transformation: (150930 rows x 10 columns → 137235 rows x 10 columns).

We noticed that we had some "odd" countries that we didn't expect to see. Also, they had an insufficient amount of data points and thus are usually not helpful in predicting prices for more "well-known" countries. We decided to cut out any country with less than 150 wines from that country. This cut caused a reduction of observations from 137235 rows x 10 columns → 136334 rows x 10 columns.

We also decided to remove grape varieties with less than 15 data points. Our reasoning for doing this is to capture the varietals that the average customer will come across when shopping for wine at the average to above average wine store. We capture 196 varietals from our data with some of our top ones displayed in Figure 3.2 below. This elimination resulted in the following transformation of our data 136334 rows x 10 columns → 134711 rows x 10 columns.

Finally, we looked at the regions that wines are from such as Napa Valley, Bordeaux, and Champagne. We included regions of which there are ten or more data points from each region resulting in 610 unique regions. This final exclusion resulted in our data going from 134711 rows x 10 columns → 111009 rows x 10 columns.

Overall, we end up with about 73.5% of our original dataset which still accounts for a healthy proportion of Worldwide wine production.

3 Data Analysis

Exploring the Data: Relationships Among Variables

After visualizing the cleaned 'Points' data, using Python's matplotlib.pyplot.hist() function, we hypothesized the data came from either a Normal Distribution or a t-distribution. We then proceeded to fit both distributions, using scipy.stats.distribution.fit function, to the empirical data. We display the Normal distribution and empirical histogram below in Figure 3.2. As we can see, the distribution seems to fit the data quite well! We also explored different relationships between the features and the price (the output vector). Figure 3.1 displays the average price of a bottle of wine per country. As we can see, this would be an important variable we would like to include in our regressions as we observe significant differences between countries.

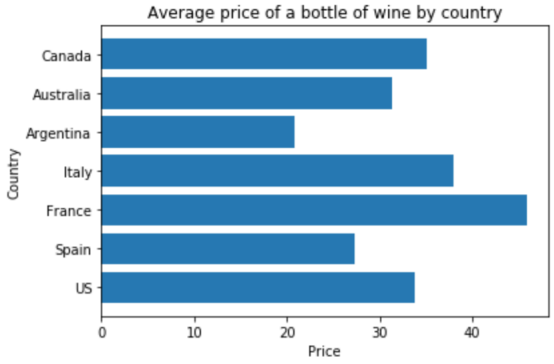

Figure 3.1

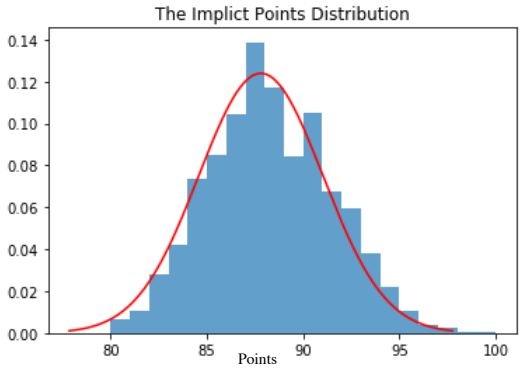

Figure 3.2

4 Feature Engineering

Multi-class features

The 'country' feature of the dataset consists of one of 7 countries for each datapoint, for instance, "France"; the 'variety' feature consists of one of 196 varieties for each datapoint, for instance, "Merlot"; and the 'region_1' feature consists of one of 610 regions for each datapoint, for instance, "Bordeaux". We transformed the 'country', 'variety' and 'region_1' features via one-hot encodings for the purpose of using the country, variety, and region classification information as features in our regression analysis along with an offset of ones.

Example of the feature transformations for 'country':
"US" → [1., 0., 0., 0., 0., 0., 0.]
"France" → [0., 0., 1., 0., 0., 0., 0.]

Natural language features

The 'description' feature of the dataset consists of a or a couple of sentences written by a sommelier about the wine datapoint in question. We felt that this information might provide unique information about the wine that other features were not able to capture. To transform this feature into numerical values to be used in regression analysis, we attempted two sentiment analysis approaches.

Sentiment sums: We obtained a dictionary containing words tagged with their positivity/negativity. For instance, "good" might map to 3 while "atrocious" might map to -5. These values were summed for each word in the description, resulting in a real number value. This creates a new feature vector to be used in regression.

Example of sentiment sum transformation:
"This wine has tremendous flavors and rich undertones." → [5.]
"This wine is so bad it made me vomit." → [-3.]

Sentiment probabilities: We used Python's NLTK sentiment package to calculate sentiment probabilities. The package takes an input sentence and returns a triple containing the probability the sentence is positive, neutral, and negative, respectively. The corresponding probabilities create three new feature columns to be used in regression.

Example of sentiment probability transformation:
"This wine has tremendous flavors and rich undertones." → [0.55, 0.45, 0.0]
"This wine is so bad it made me vomit." → [0.0, 0.63, 0.36]

The most strong correlation between the sentiment transformation features and our data was with negative sentiment probabilities. As can be seen in Figures 4.1 and 4.2, wines with high prices and points do not generate a high probability of having a description that is of negative sentiment. This analysis of the resulting

feature transformations led us to believe that the addition of the sentiment features would improve our model performance. In fact, for the models we developed (described in future sections), we confirmed that the addition of our sentiment features improved predictions: via cross-validation, we examined the performance for each model between these features being added and being left out, and found improved scores when the sentiment language feature vectors were added.

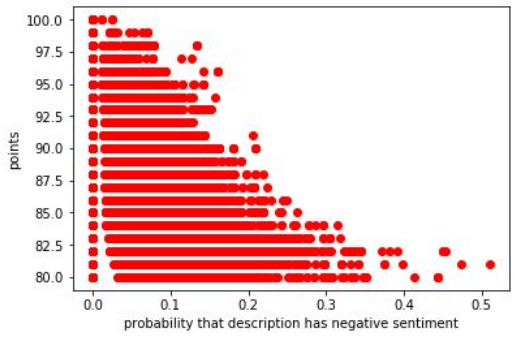

Figure 4.1

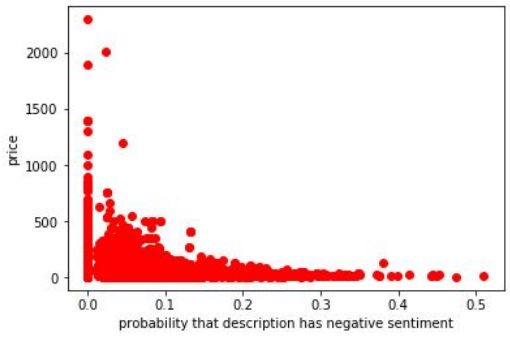

Figure 4.2

5 Validation and Testing

We split our overall dataset into train and test sets, 90% and 10%, respectively. We noticed that the order of the data samples seemed to group wines with similar point rankings close together. For this reason, we randomly select samples to place into the training and test sets (as opposed to merely splitting by index) to ensure that our distributions for train and test sets are not skewed.

To tune the hyperparameters of each of our models, as well as measure the performance across the different models, we use 10-fold cross-validation of the training set. For instance, we train a Decision Tree model using only the features that did not need sentiment analysis feature engineering. We then train the same Decision Tree model with the added features described above using feature engineering. We compare the results of each model using cross-validation, concluding that including the additional engineered feature improves the model performance, and proceed using the better-performing model.

We find it necessary to distinguish between cross-validation and test set so that we may use the former to fine-tune and compare our models while using the latter in the final evaluation of the project on data that no model has seen. The final **Results and Analysis** section contains values computed on this test set.

6 Linear Regression Models

To explore the relationships between a wine's attributes and its price, we decided to run a couple of linear regression models. Specifically, we ran Least Squares to fit the average bottle of wine (taking into account outliers such as the First Growths of Bordeaux) and Huber Regression as a more robust fit in our attempt to predict price. Our reasoning for choosing these two models was to compare the trade-off between minimizing the number of bottles of wine that we predict incorrectly or minimizing the absolute error for any given bottle of wine. We achieved moderate success with Least Squares when comparing metrics such as R^2 but less-than-desirable results for everything else (Figure 9.1). These findings are visually represented in Figures 6.1 and 6.2 below for the Least Squares Linear Regression (as expected, the Huber fits the median and had a much lower tail in its version of Figure 6.2).

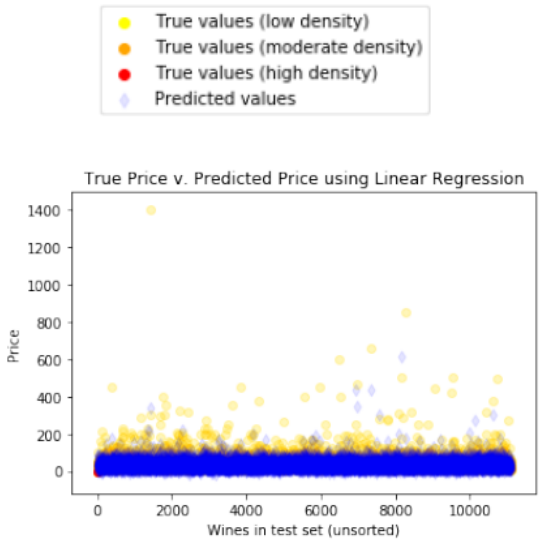

Figure 6.1

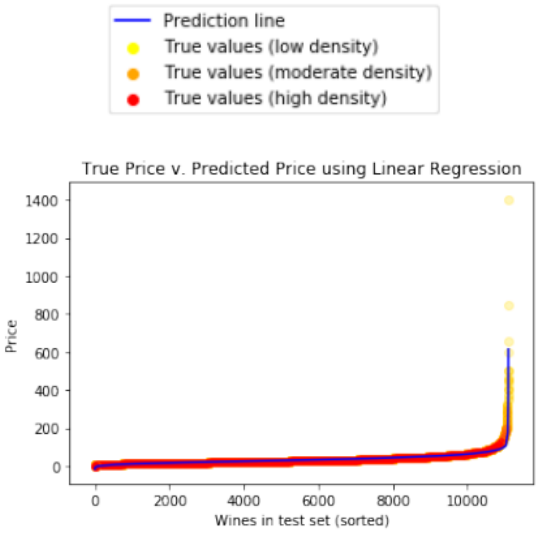

Figure 6.2

7 Decision Trees and Other Models

Interpreting the results from our linear models, we found that the wine prices tend to have complicated non-linear relationships. To combat this obstacle, we decided to use decision trees to replicate how a sommelier would think about wine. I.e., isolate the variety, is it a Cabernet Sauvignon, okay now what country is it from, France? Great, if I know it is from Bordeaux and is rated 91 points then the price will likely be 28 dollars.

We implemented several different tree algorithms, the first being a Decision Tree Regression. At each step, we separate the data by splitting the feature that minimizes the mean squared error. We use the decision tree to attempt to improve the accuracy of the model. We also implemented a Random Forest Regression to reduce the variance of our predictions. This was done by averaging ten individual decision trees on the training data to improve the precision of the predictions. We also used a couple of Gradient Boosting models that, similar to random forests, sum together individual weak learners using gradient descent to minimize loss.

When doing so, we found that these tend to produce much better predictions than linear models when measured by Mean Absolute Error, Median Absolute Error, Explained Variance, and R-squared scores. Figure 9.1 shows a table displaying each algorithm and their performance metrics. From this summary, we can safely conclude that wine prices tend to have a complicated relationship between their attributes which is better described by trees rather than linear regressions.

We also considered fitting a Support Vector Machine to the data to minimize an epsilon-intensive loss objective that focuses on errors outside the epsilon-intensive generalization bounds. However, before running Support Vector Regression, we learned the run-time is $O(n_{features} * n^2_{observations})$ which considering we have over 100,000 data points would cost us at least 10e13 flops. Due to this enormous run-time, we decided it would be better to omit this model from our selection.

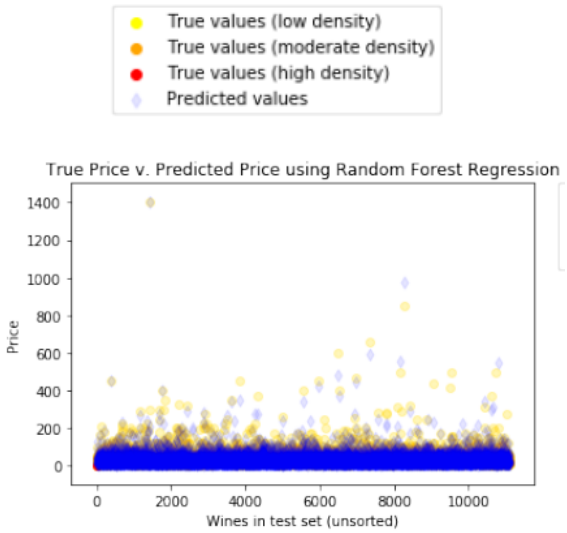

Figure 7.1

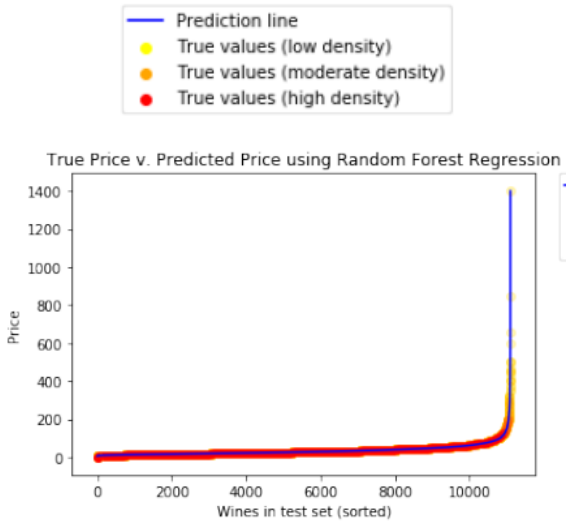

Figure 7.2

8 Natural Language Processing Extension

We wanted to further explore the "description" feature of our data. In order to do so, we use natural language processing to examine the "description" column of each sample, which contains a or a couple of sentences from a sommelier describing the qualities of the wine. We use word embeddings, which map each description to a high dimensional vector space. This enables us to use a neural network to directly learn and predict the price of a wine given its description.

Word Embedding

The "description" feature of each datapoint is transformed into a sentence embedding matrix as follows. The string description is split by word, and each word is mapped to a 300-dimensional word embedding vector. We use Google's word2vec dataset for these word embedding vectors, which consists of a dictionary of 3 billion word to word vector mappings. In order to use these word embeddings as input to the neural network, we must ensure the input dimensions for each datapoint's description has uniform size; we set a maximum sequence length for each sentence. Thus, our input to the neural network is a 20 by 300 float matrix for each string description for the datapoint, which is a sentence embedding matrix representation of the natural language in the description.

The Neural Network

The neural network feeds the sentence embedding matrix input into a Long-Short Term Memory layer, then through a dense layer with a linear activation function (because our predictions are real numbers). We use mean absolute error within the neural network as its training loss function.

Results

Our neural network obtains a final training set loss of 13.0487 and testing set loss of 14.1788. The model does not even perform well on the training set, indicating that this model underfits and is not a good model for predicting the price of the wine. These results are not surprising to us, as it makes sense that using only the description of a wine cannot fully capture the important information about a wine enough to predict the wine's price. Direct analysis of individual descriptions gives us further insight; the language used in descriptions is relatively similar whether or not the wine is expensive, as it mainly describes flavors as opposed to using strongly positive or negative words. Further, the set of descriptions covers a small vocabulary subset of natural language, limiting the ability of a neural network to accurately distinguish between datapoints. However, it is not the case that the "description" feature and natural language analysis is not useless for prediction. As

was seen in previous models, transforming the "description" feature to sentiment score vectors and including these added features improves the overall performance. We conclude that information can be extracted from the description and does help prediction, but is not a good predictor alone for the price of wine.

9 Results and Analysis

The following tables (Figure 9.1 ad Figure 9.2) contain final results of each tuned model, computed on the test set. The test set, partitioned at the start, has not been seen or trained on by any model before computing these final scores.

Regressions	**Mean Abs Error**	**Median Abs Error**	**Explained Variance**	R^2
Least Squares (LS)	13.10	8.63	0.43	0.43
Huber	13.23	7.31	0.23	0.21
Decision Tree	6.85	0.0	0.61	0.61
Random Forest	7.72	3.10	0.75	0.75
Gradient Boosting (Huber)	12.17	7.09	0.41	0.40
Gradient Boosting (LS)	12.82	8.41	0.59	0.59
NLP Neural Network	14.18	7.53	0.20	0.18

Figure 9.1

Regressions	**Mean**	**Variance**	**Skew**	**Kurtosis**
Real Data (t-distribution)	**25.09**	**inf**	-	-
Least Squares (LS) (t-distribution)	31.35	487.88	0.00	10.67
Huber (t-distribution)	28.96	164.88	0.00	1.35
Decision Tree (t-distribution)	24.90	inf	-	-
Random Forest (t-distribution)	26.14	3485.87	-	-
Gradient Boosting (Huber) (t-distribution)	27.36	349.79	-	-
Gradient Boosting (LS) (t-distribution)	28.28	2416.45	-	-
NLP Neural Network (t-distribution)	28.33	166.04	0.00	0.78
Real Data (exponential-distribution)	**34.42**	**925.08**	**2.00**	**6.00**
Least Squares (LS) (exponential-distribution)	34.39	2534.27	2.00	6.00
Huber (exponential-distribution)	29.64	1087.95	2.00	6.00
Decision Tree (exponential-distribution)	34.39	923.39	2.00	6.00
Random Forest (exponential-distribution)	34.35	838.00	2.00	6.00
Gradient Boosting (Huber) (exponential-distribution)	31.57	476.55	2.00	6.00
Gradient Boosting (LS) (exponential-distribution)	34.77	429.37	2.00	6.00
NLP Neural Network (exponential-distribution)	29.23	559.52	2.00	6.00

Figure 9.2: '-' implies a value of 'nan'

Discussion of Regression Output

In examining the output of the regressions, we find that the Least Squares and Huber regressions tended to perform worse when compared to their tree rivals. This out-performance is likely due to the complicated relationship between the price of wine and it's attributes. We also observed the Huber Loss function tended to produce worse predictions compared to its Least Squares Loss cousins. The reason why we used mean and median absolute value regression metrics was for the ease of user interpretation.

If the mean absolute value is eight dollars, then we can expect the output of our model to be roughly within an eight dollar band from our predicted value. A median absolute value of zero means at least fifty percent of our predictions are the correct price, which is excellent! We find that the best regression, regarding median and mean absolute values, is the Decision Tree Regression. The Decision Tree Regression is substantially more accurate than other models we tried, although Random Forest has a very similar Mean Absolute Error value. In regards to a more precise model that predicts a closer wine price on average using metrics such as R^2 and

Explained Variance, Random Forest Regression is the clear winner; outputting amazing R^2 and Explained Variance values of 75% on >10,000 test data points.

Fitted Distributions of the Test Set Output vs. the Regression Prediction Output

The Figures 9.3, 9.4, 9.5, and 9.6 compare the fitted distribution of the test set data with the fitted distribution of the prediction set against the normalized empirical histogram of the test set data. The best algorithms, by this measure, are the Decision Tree Regression and the Random Forest Regression when compared across all algorithms. We calculated this by computing the area between the fitted distribution to the empirical data and the fitted distribution to the prediction data. This matches the results obtained from comparing the Mean, Variance, Skew, and Kurtosis metrics, of the fitted distributions, for each algorithm in Figure 9.2.

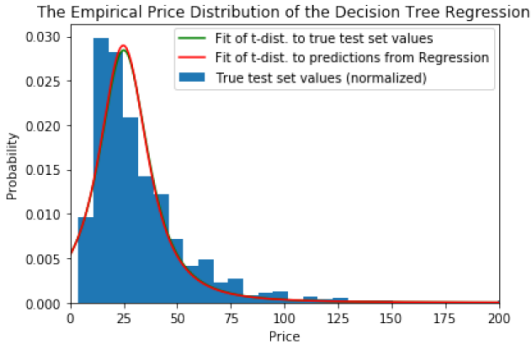

Figure 9.3

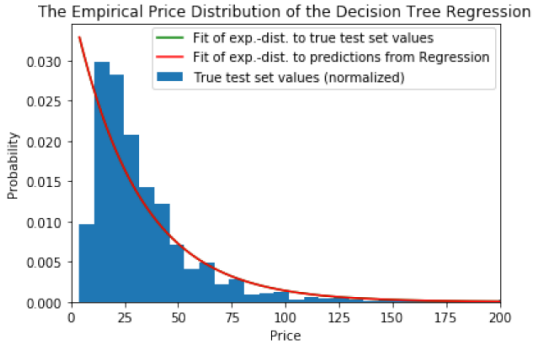

Figure 9.4

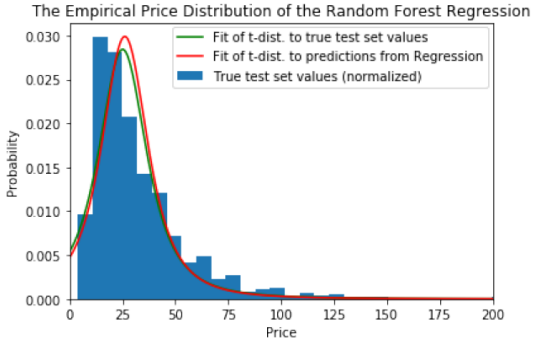

Figure 9.5

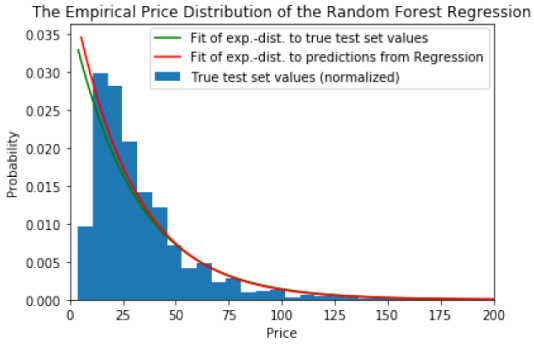

Figure 9.6

10 Sources

Python Packages Used	Julia Packages Used
pandas	lowrankmodels
numpy	DataFrames
pickle	PyPlot
matplotlib	
sklearn	
scipy	
nltk	
keras/tensorflow	
Foogle word2vec	