



FALL 2016

CSCI B - 659

APPLIED MACHINE LEARNING

FINAL PROJECT

- 1. Predicting frequent prescribers of Opiates using Random Forest**
- 2. Predicting severity of loss in insurance claims using Gradient Boosting**

Team Members:

- 1. Harish Annavajjala(hannavaj@iu.edu)**
- 2. Ramprasad Bommaganty(rbommaga@indiana.edu)**

DATASET-I: US Opiate Prescriptions/Overdoses

Data Description:

This dataset contains summaries of prescription records for 250 common opioid and non-opioid drugs written by 25,000 unique licensed medical professionals in 2014 in the United States for citizens covered under Class D Medicare. The increase in overdose fatalities is a well-known problem, and the search for possible solutions is an ongoing effort [1]. In 2010, an estimated two million people reported abusing prescription pain medication for the first time within the previous 12 month period. This number amounts to 5,500 people a day abusing prescription pain meds for the first time [2]. Our primary interest in this dataset is predicting frequent prescribers of opiate drug.

This is a classification task having a binary target variable and having around 250 features.

Data Statistics:

```
In [38]: data.head(5)
```

```
Out[38]:
```

	NPI	Gender	State	Credentials	Specialty	ABILIFY	ACETAMINOPHEN.CODEINE	ACYCLOVIR	ADVAIR.DISKUS	AGGRENO
0	1710982582	M	TX	DDS	Dentist	0	0	0	0	0
1	1245278100	F	AL	MD	General Surgery	0	0	0	0	0
2	1427182161	F	NY	M.D.	General Practice	0	0	0	0	0
3	1669567541	M	AZ	MD	Internal Medicine	0	43	0	0	0
4	1679650949	M	NV	M.D.	Hematology/Oncology	0	0	0	0	0

5 rows x 256 columns

```
In [34]: data.index
```

```
Out[34]: RangeIndex(start=0, stop=25000, step=1)
```

```
In [35]: data.values
```

```
Out[35]: array([[1710982582L, 'M', 'TX', ..., 0L, 0L, 1L],
 [1245278100L, 'F', 'AL', ..., 0L, 35L, 1L],
 [1427182161L, 'F', 'NY', ..., 0L, 25L, 0L],
 ...,
 [1346270956L, 'M', 'AZ', ..., 0L, 0L, 1L],
 [1023116514L, 'F', 'IN', ..., 0L, 0L, 1L],
 [1518913672L, 'M', 'NV', ..., 0L, 0L, 0L]], dtype=object)
```

```
In [36]: data.describe()
```

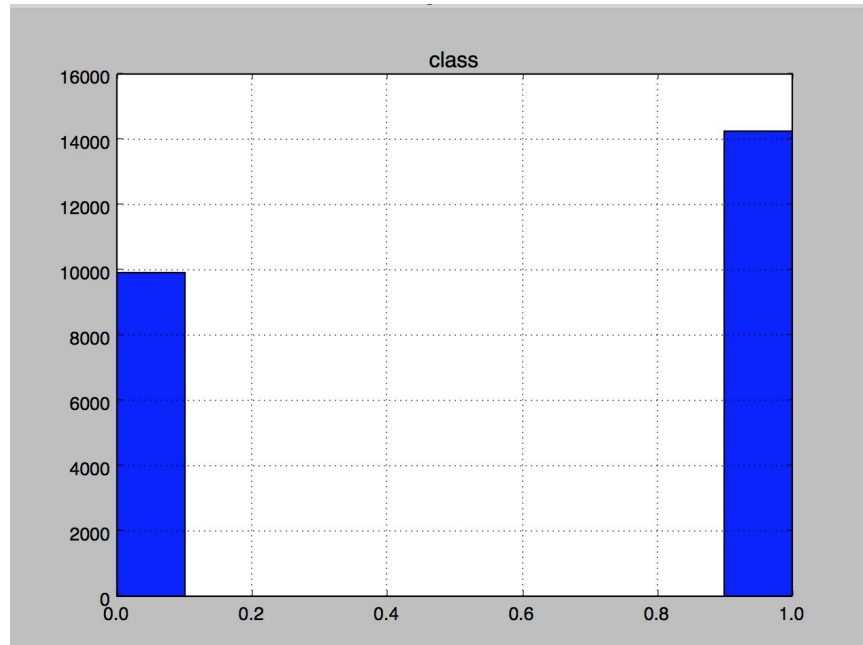
```
Out[36]:
```

	NPI	ABILIFY	ACETAMINOPHEN.CODEINE	ACYCLOVIR	ADVAIR.DISKUS	AGGRENOX	ALENDRONATE.SODIUM	ALLOPI
count	2.500000e+04	25000.000000	25000.000000	25000.00000	25000.000000	25000.000000	25000.000000	25000.0
mean	1.498162e+09	3.157160	2.370400	1.05368	7.041000	0.708440	8.962840	9.30572
std	2.877233e+08	20.755819	11.631601	6.66110	25.898648	5.224049	36.520987	29.3754
min	1.003002e+09	0.000000	0.000000	0.00000	0.000000	0.000000	0.000000	0.00000
25%	1.245473e+09	0.000000	0.000000	0.00000	0.000000	0.000000	0.000000	0.00000
50%	1.497842e+09	0.000000	0.000000	0.00000	0.000000	0.000000	0.000000	0.00000
75%	1.740406e+09	0.000000	0.000000	0.00000	0.000000	0.000000	0.000000	0.00000
max	1.992999e+09	770.000000	644.000000	356.00000	1105.000000	275.000000	2431.000000	790.000

8 rows x 252 columns

```
In [37]: data.shape  
Out[37]: (25000, 256)
```

The class distribution for the whole data set is shown in a histogram below:



As we can see the data is not skewed towards any one class variable, so accuracy would be a good measure here instead of something like ROC curves.

Feature Engineering and Feature Selection:

There are huge number of columns, so we have used few techniques to reduce the dimensionality of the data. We have eliminated the features which have very little variance and then applied a chi-square test on the remaining data to select the best set of features out of them.

We have used scikit learn packages for doing this dimensionality reduction. [3]

Algorithms Implemented:

1) Naive Bayes:

We have implemented Naive Bayes algorithm for this data with our own code. Here all the features are continuous type - it is the count of number times a drug has been prescribed by a doctor. The results are shown below:

The confusion matrix for Naive Bayes is as follows:

		Predicted	
Actual	True Negative: 237		False Positive: 796
	False Negative: 197	True Positive: 1270	

accuracy= 60.28 %
time taken= 5.45757198334

As we see the results are not so great for Naive Bayes.

2) *Random Forest:*

We have implemented decision tree with our own code and used n-cross validation.

The results are as shown below:

Incorrectly classified= 255 Correctly classified= 2164

Accuracy for a depth of 5 is 89.4584539066 %

The confusion matrix is as follows:

		Predicted	
Actual	True Negative: 998		False Positive: 0
	False Negative: 210	True Positive: 1166	

TIME TAKEN= 66.3640549183

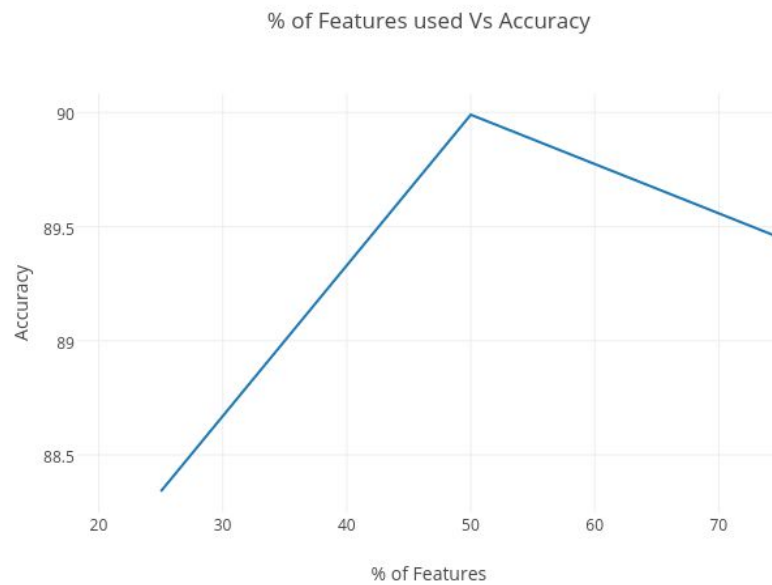
Analysis:

As we see the results for random forest algorithm are much better than the Naive Bayes. This could be because probably the features are highly correlated and a Naive Bayes assumption that the features are independent might not have worked well here.

We have implemented our random forest algorithm for different sets of features. And each set is created by selecting n best features that we got after performing a chi-square test on the data.

Each set contains a varying fraction of the original feature set.

Below graph shows our accuracy for three cases: i) When we take only the top 25% of the whole feature set , ii) top 50% of the whole feature set , iii) top 75% of the whole feature set.



From the above graph we could see that we get best result when we take the top 50% of the whole feature set. In the other two cases we are getting little bit less accuracy probably because in the first case we are eliminating a lot of features and in the last case we are including too many features.

DATASET2: ALLSTATE INSURANCE CLAIM SEVERITY DATA

Dataset Source and Motivation:

The second largest personal insurer in the United States[4], AllState Corporation deals with over 16 million households and counting[5]. It was a no-brainer that such a company would want to leverage the potential of machine learning and increase its effectiveness in terms of customer service.

Through a Kaggle competition, AllState challenged machine learning enthusiasts to predict the severity of an insurance claim. Our motivation in choosing this dataset was to work with a real world dataset and make predictions that are credible with minimal error.

Dataset Description:

The dataset consists of 188318 rows and 132 columns. Each row corresponds to one insurance claim and each column is an unknown feature which has no description in the given documentation. There are 116 categorical variables and 14 continuous variables. The target class variable is a feature known as 'loss' which is continuous valued, thereby prompting us to use regression techniques to solve the challenge. Train.csv and Test.csv were the two files given for training the model and subsequent testing.

```
In [3]: data.head(5)
```

```
Out[3]:
```

	id	cat1	cat2	cat3	cat4	cat5	cat6	cat7	cat8	cat9	...	cont6	cont7	cont8	cont9	cont10	cont11	cont12	cont13	cont14
0	1	A	B	A	B	A	A	A	A	B	...	0.718367	0.335060	0.30260	0.67135	0.83510	0.569745	0.594646	0.822493	0.71484
1	2	A	B	A	A	A	A	A	A	B	...	0.438917	0.436585	0.60087	0.35127	0.43919	0.338312	0.366307	0.611431	0.30449
2	5	A	B	A	A	B	A	A	A	B	...	0.289648	0.315545	0.27320	0.26076	0.32446	0.381398	0.373424	0.195709	0.77442
3	10	B	B	A	B	A	A	A	A	B	...	0.440945	0.391128	0.31796	0.32128	0.44467	0.327915	0.321570	0.605077	0.60264
4	11	A	B	A	B	A	A	A	A	B	...	0.178193	0.247408	0.24564	0.22089	0.21230	0.204687	0.202213	0.246011	0.43260

5 rows × 132 columns

```
In [4]: data.index
```

```
Out[4]: RangeIndex(start=0, stop=188318, step=1)
```

```
In [5]: data.values
```

```
Out[5]: array([[1L, 'A', 'B', ..., 0.822493, 0.714843, 2213.18],
               [2L, 'A', 'B', ..., 0.611431, 0.304496, 1283.6],
               [5L, 'A', 'B', ..., 0.195709, 0.774425, 3005.09],
               ...,
               [587630L, 'A', 'B', ..., 0.339244, 0.503888, 5762.64],
               [587632L, 'A', 'B', ..., 0.654753, 0.721707, 1562.87],
               [587633L, 'B', 'A', ..., 0.810511, 0.72146, 4751.72]], dtype=object)
```

```
In [6]: data.describe()
```

```
Out[6]:
```

	id	cont1	cont2	cont3	cont4	cont5	cont6	cont7	cont8
count	188318.000000	188318.000000	188318.000000	188318.000000	188318.000000	188318.000000	188318.000000	188318.000000	188318.000000
mean	294135.982561	0.493861	0.507188	0.498918	0.491812	0.487428	0.490945	0.484970	0.486437
std	169336.084867	0.187640	0.207202	0.202105	0.211292	0.209027	0.205273	0.178450	0.199370
min	1.000000	0.000016	0.001149	0.002634	0.176921	0.281143	0.012683	0.069503	0.236880
25%	147748.250000	0.346090	0.358319	0.336963	0.327354	0.281143	0.336105	0.350175	0.312800
50%	294539.500000	0.475784	0.555782	0.527991	0.452887	0.422268	0.440945	0.438285	0.441060
75%	440680.500000	0.623912	0.681761	0.634224	0.652072	0.643315	0.655021	0.591045	0.623580
max	587633.000000	0.984975	0.862654	0.944251	0.954297	0.983674	0.997162	1.000000	0.980200

```
In [7]: data.shape
```

```
Out[7]: (188318, 132)
```

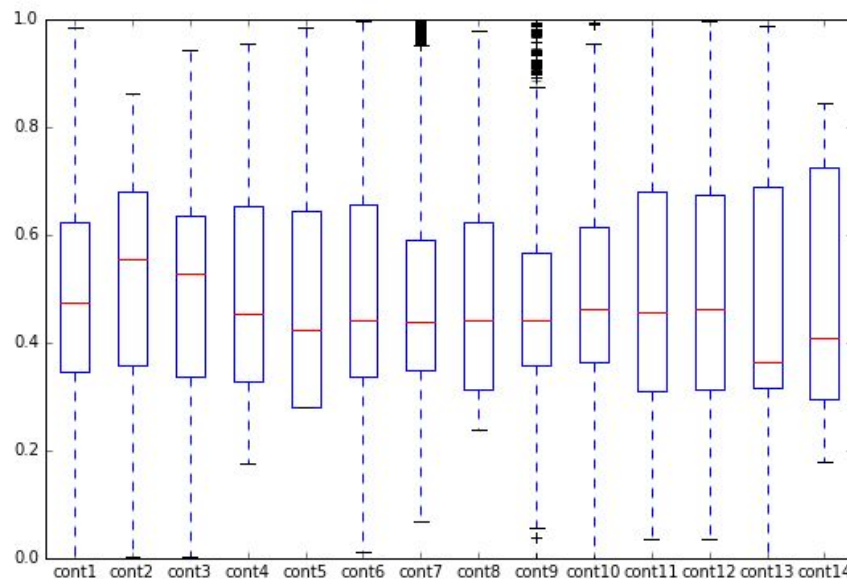


Figure showing the boxplots of all the continuous features in the dataset.

Steps taken to solve the problem:

1. Since this was a regression problem, all the categorical attributes had to be converted into numerical attributes. This was done using One Hot Encoding, thereby converting categorical attributes like 'cat80' with values such as 'A', 'B', 'C' etc. to binary valued attributes like cat80_A, cat80_B, cat80_c with values 0 and 1.
2. This conversion lead to an increase in the number of attributes to 1007 in total. Now, feature selection was performed by calculating the unique count of each categorical attribute before the One Hot Encoding was performed and choosing only those attributes as features, which had a unique count of 8 or less. The reasoning behind this was that an attribute with too many unique

values did not contribute to an increase in model effectiveness, which was vindicated by our results on tuning and test sets, presented in the next section.

3. With the feature set down to 286 columns, we ran four different regression algorithms with default parameters and evaluated the performances using two metrics: Root Mean Squared Error and Mean Absolute Error.
4. Out of all the four algorithms, Gradient Boosting regression performed the best, followed by Random Forest Regression. Linear Regression did not fare very well, while SVM was computationally expensive and did not seem to fit very well.
5. We picked Gradient Boosting regression as our approach to solve the problem, based on the above mentioned results and used a tuning set consisting of 20% of training data to tune the parameters and finally give out the prediction results.

Note :

1. Additional feature engineering techniques like log transformation(RMSE Score: 2295.70865501) and square root transformation for the continuous attributes were tried with little success (RMSE Score without log : 2125.61532548). The distributions hardly changed, prompting us to persist with the existing values of continuous attributes.
2. We also tried converting the continuous valued attributes to discrete binned attributes by using Pandas.cut() method. However, the resulting RMSE scores were not any better (RMSE Score: 2203.47883054).

Results:

The following are the results for the baseline RMSE and MAE scores for four algorithms using default parameters:

Algorithm	Baseline RMSE Score	Baseline MAE Score
Random Forest Regressor	2304.78	1467.01
Support Vector Regressor	2236.675437	1384.3456783
Linear Regression	2114.2168364	1319.10685668
Gradient Boosting Regressor	1954.94810341	1209.31268665

Table: Baseline RMSE and MAE scores for all tested algorithms with default parameters.

The following line plots show the varying RMSE and MAE values for different n_estimators parameter(which deals with number of trees) in the Gradient Boosting regression model.

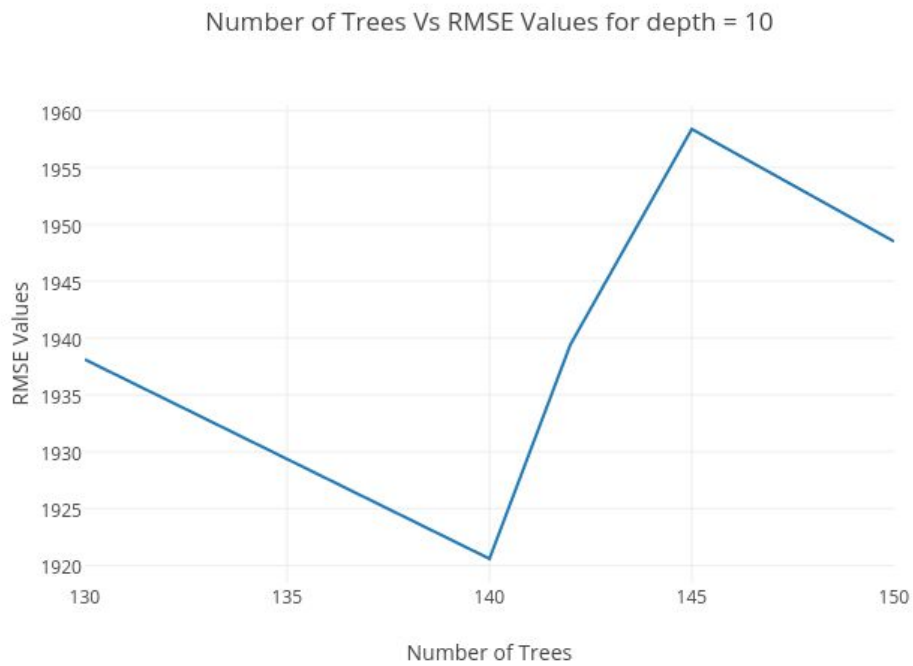


Figure showing the plot of number of Trees Vs the RMSE Values for Gradient Boosting Regressor.

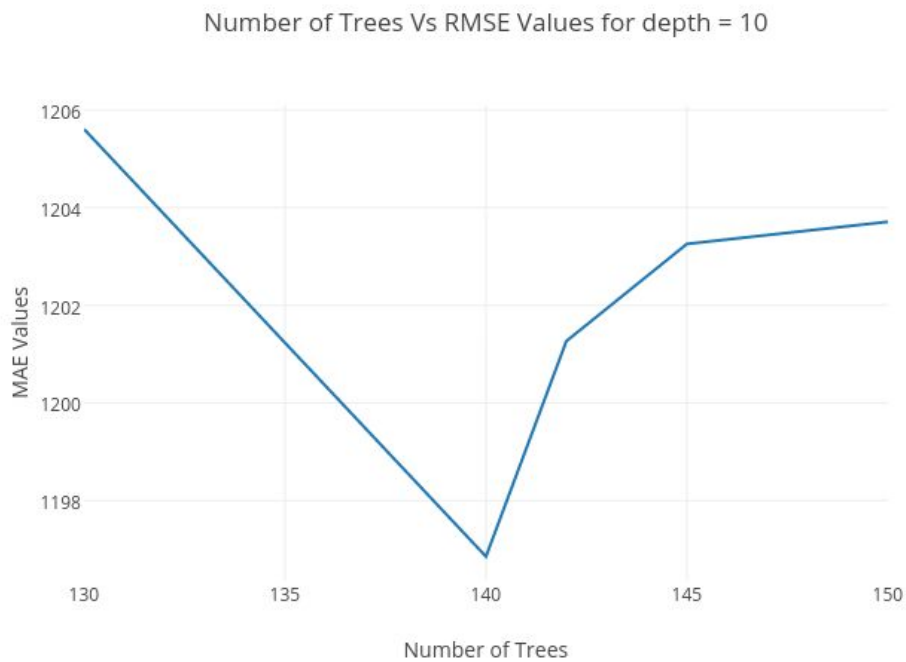
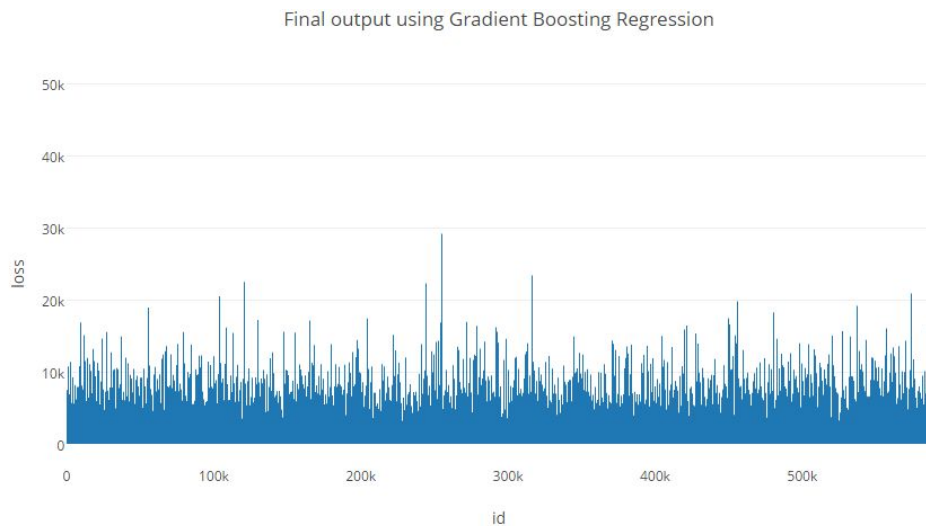


Figure showing the Number of Trees Vs the MAE Values for Gradient Boosting Regressor.

The following figure shows a bar chart of the final predicted output using the Gradient Boosting Regression with the following parameters:

learning_rate = 0.1
n_estimators = 140
max_depth = 10
max_features = 'sqrt'



Algorithm Performance:

The Gradient Boosting algorithm performed better than the other three algorithms that we tested on using the transformed dataset. On the validation set that was created using 20% of the training set, the best performance of the Gradient Boosting algorithm was:

RMSE Score: 1920.57634692

MAE Score: 1196.85370969

Since, Gradient Boosting Regressor of the sklearn package builds an iterative and additive model by fitting a tree on the negative gradient of the loss function[6], it is robust to overfitting and performs well. On the Kaggle Public Leaderboard, the MAE score when run on the unknown test dataset was 1281.85819. While the model seems to have overfit slightly on the unknown test dataset, this was a better performance among our other submissions.

2534

new

Ramprasad + Harish

1281.85819


2

Fri, 09 Dec 2016 14:16:54

Your Best Entry ↑

You improved on your best score by 45.86973.

You just moved up 132 positions on the leaderboard.

 Tweet this!

Future Enhancements and Final Thoughts:

It is a well known fact that many Kagglers are fond of XGBOOST due to its superior performance and stellar results. For future enhancements of this project, we would also like to use it to train our model. Also, the number of features can further be filtered using perhaps the correlations between different features and recursively eliminating them.

Our project involved two datasets, one dealing with classification and the other with regression, which we believe gave us a chance to enhance our understanding of commonly used feature engineering and machine learning techniques. We would like to sincerely thank the course team of CSCI B 659 and Prof. Sriraam Natarajan for their guidance and support all throughout the semester.

REFERENCES:

1. <https://www.kaggle.com/apryor6/us-opiate-prescriptions>
2. <http://www.addictions.com/opiate/10-opiate-addiction-statistics/>
3. http://scikit-learn.org/stable/modules/feature_selection.html
4. <https://en.wikipedia.org/wiki/Allstate>
5. <https://www.kaggle.com/c/allstate-claims-severity>
6. <http://scikit-learn.org/stable/modules/generated/sklearn.ensemble.GradientBoostingRegressor.html>
7. http://scikit-learn.org/stable/supervised_learning.html#supervised-learning
8. <https://plot.ly/>
9. <https://www.analyticsvidhya.com/blog/2016/02/complete-guide-parameter-tuning-gradient-boosting-gbm-python/>
10. <https://www.kaggle.com/achalshah/allstate-claims-severity/allstate-feature-analysis-python>
11. <https://www.kaggle.com/nminus1/allstate-claims-severity/allstate-eda-python>
12. <http://machinelearningmastery.com/start-here/>