BNP Paribas Claims Classification

Springboard Data Science Intensive

Ritwik Raj

August 24, 2016

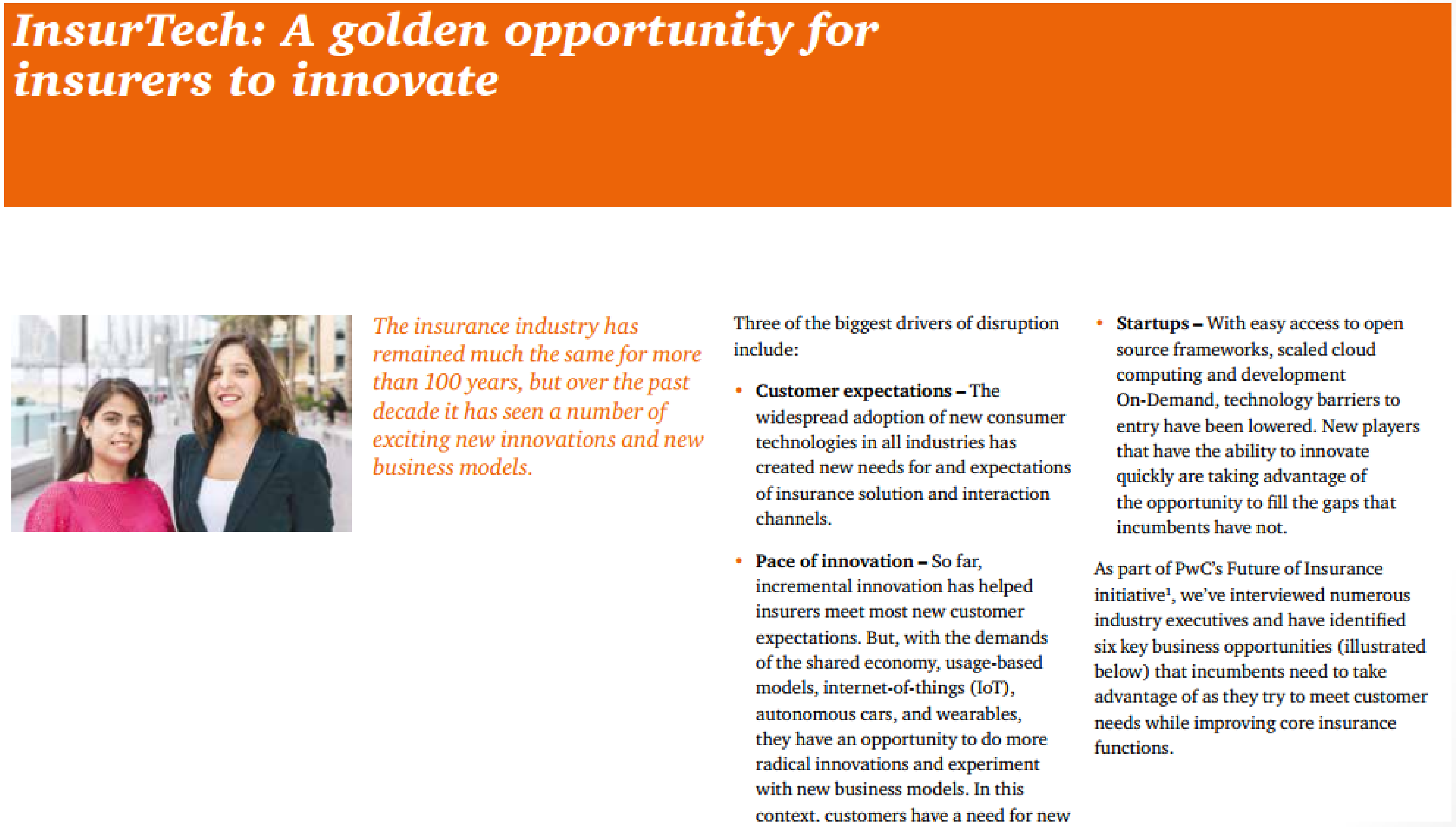
Background

- Insurance companies such as BNP Paribas gather a variety of information about each insurance claim.
- In some cases, they need more information to approve the claim
- The process of gathering information takes time → Customer has to wait, Company has to spend resources
- What if we can make this process quicker by
  - Automatically evaluating claims: Is more information needed?
  - Make decisions based on data, instead of human judgment

The Problem

- “Predict the category of an insurance claim based on features available early in the process, helping BNP Paribas accelerate its claims process and therefore provide a better service to its customers.”
- Kaggle Contest (Jan – Apr 2016)
- Classification problem – 0/1

The Opportunity



InsurTech: A golden opportunity for insurers to innovate

The insurance industry has remained much the same for more than 100 years, but over the past decade it has seen a number of exciting new innovations and new business models.

Three of the biggest drivers of disruption include:

- **Customer expectations –** The widespread adoption of new consumer technologies in all industries has created new needs for and expectations of insurance solution and interaction channels.
- **Pace of innovation –** So far, incremental innovation has helped insurers meet most new customer expectations. But, with the demands of the shared economy, usage-based models, internet-of-things (IoT), autonomous cars, and wearables, they have an opportunity to do more radical innovations and experiment with new business models. In this context. customers have a need for new
- **Startups –** With easy access to open source frameworks, scaled cloud computing and development On-Demand, technology barriers to entry have been lowered. New players that have the ability to innovate quickly are taking advantage of the opportunity to fill the gaps that incumbents have not.

As part of PwC's Future of Insurance initiative[1], we've interviewed numerous industry executives and have identified six key business opportunities (illustrated below) that incumbents need to take advantage of as they try to meet customer needs while improving core insurance functions.

Surpass customer expectations + Enhance pace of innovation!

The Data

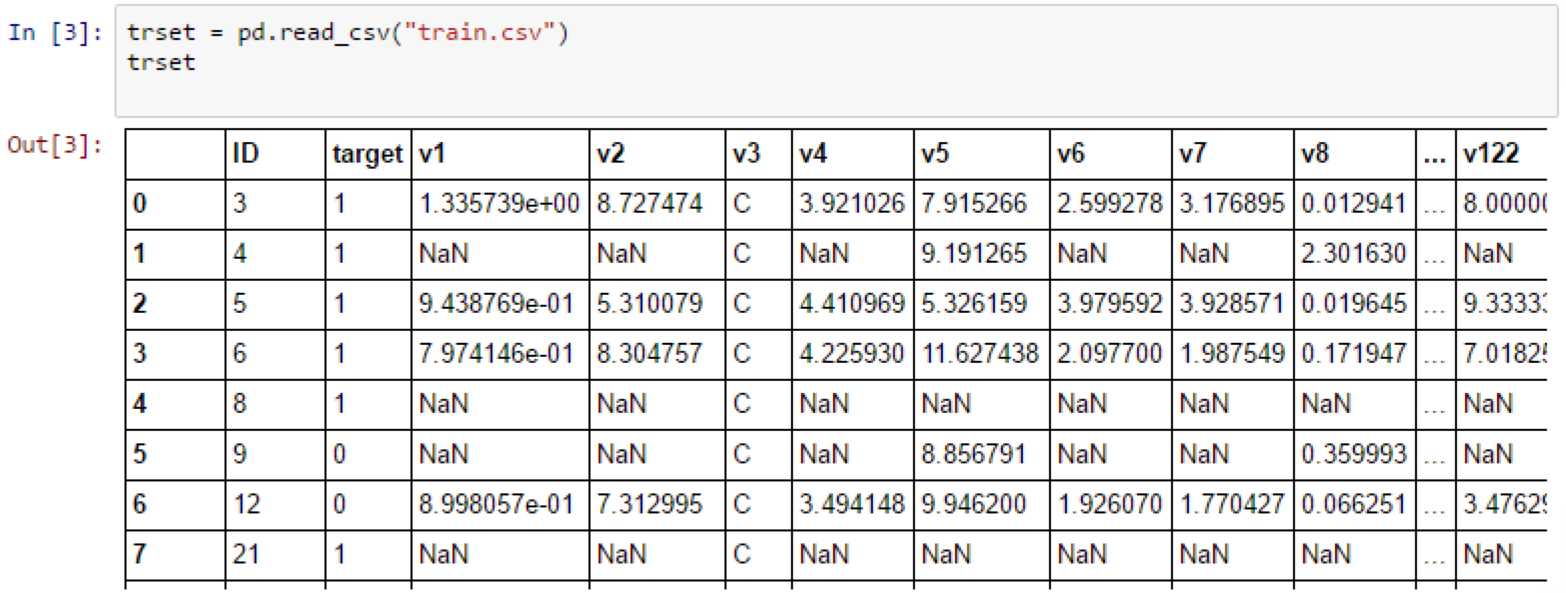
- 114000+ rows of training data with 131 numerical and categorical features
- Features are "anonymized"
- Target is 0/1 – Need more info/Approved

- 114000 rows of test data also provided

- Data is what is available to BNP PC at the time the claim is received

Data Exploration

- When we see this:



```
In [3]: trset = pd.read_csv("train.csv")
        trset
```

Out[3]:

	ID	target	v1	v2	v3	v4	v5	v6	v7	v8	...	v122
0	3	1	1.335739e+00	8.727474	C	3.921026	7.915266	2.599278	3.176895	0.012941	...	8.00000
1	4	1	NaN	NaN	C	NaN	9.191265	NaN	NaN	2.301630	...	NaN
2	5	1	9.438769e-01	5.310079	C	4.410969	5.326159	3.979592	3.928571	0.019645	...	9.3333
3	6	1	7.974146e-01	8.304757	C	4.225930	11.627438	2.097700	1.987549	0.171947	...	7.0182
4	8	1	NaN	NaN	C	NaN	NaN	NaN	NaN	NaN	...	NaN
5	9	0	NaN	NaN	C	NaN	8.856791	NaN	NaN	0.359993	...	NaN
6	12	0	8.998057e-01	7.312995	C	3.494148	9.946200	1.926070	1.770427	0.066251	...	3.4762
7	21	1	NaN	NaN	C	NaN	NaN	NaN	NaN	NaN	...	NaN

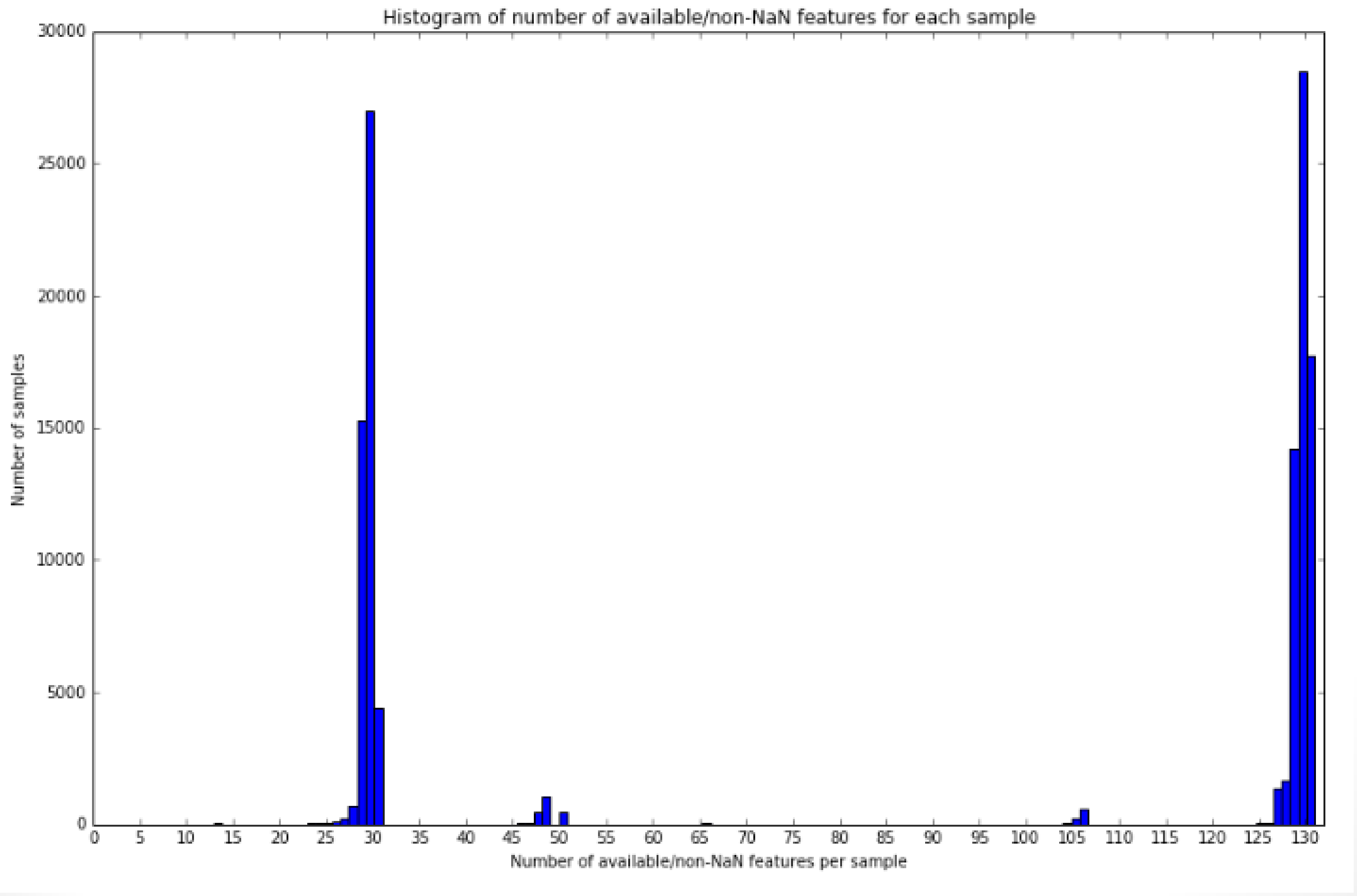
- We are thinking: Missing Values!!

Data Exploration

- Some metrics about missing values in the training data:
  - 34% sparse – the percentage of values “missing”
  - Average of ~85 features available per row (out of 131)
  - Only 15% of rows are data complete

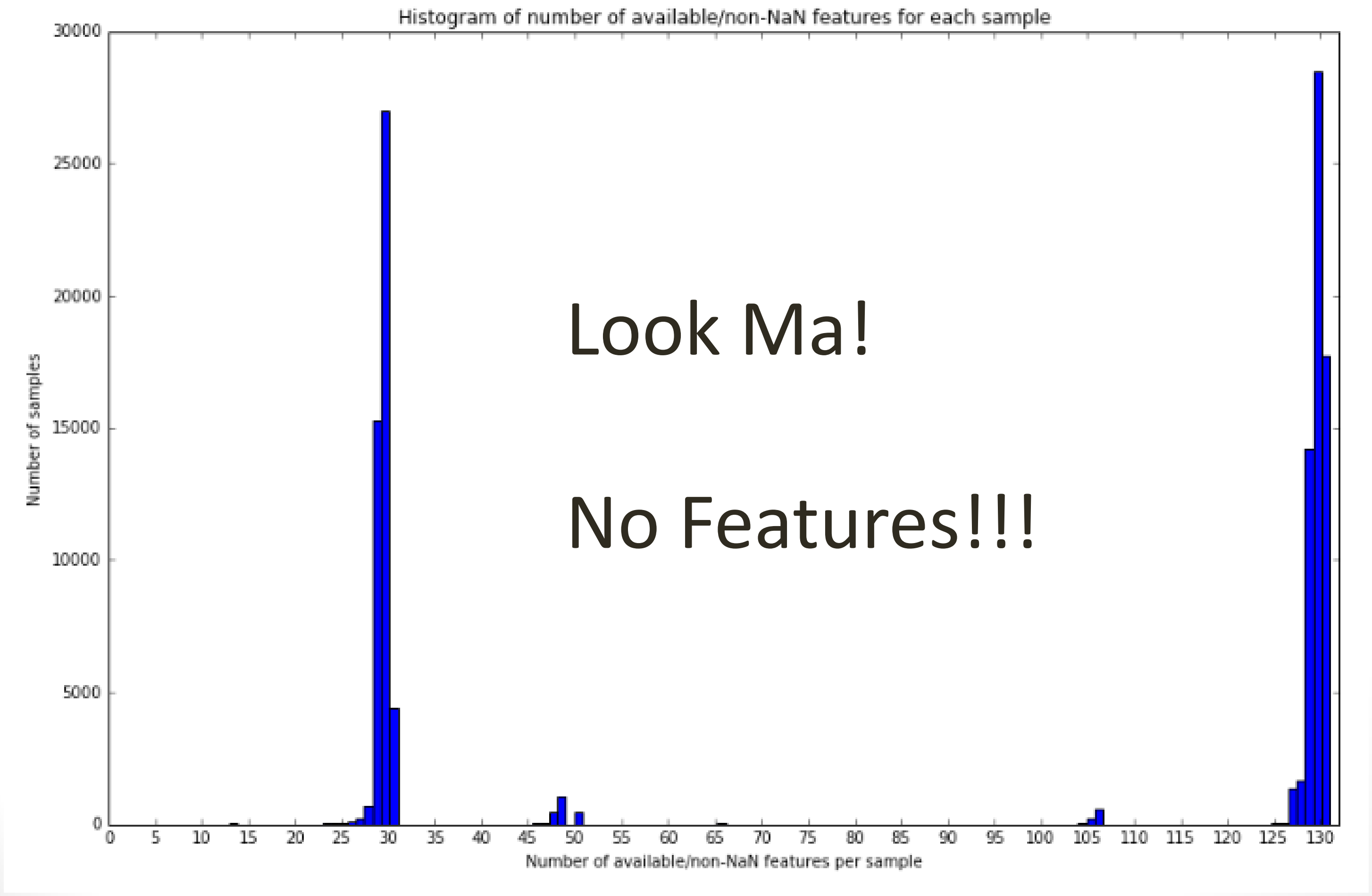
Data Exploration

- Let's look at the picture on a per-row basis
- Plotting the number of non-NaN features per row



Data Exploration

- Let’s look at the picture on a per-row basis
- Plotting the number of non-NaN features per row

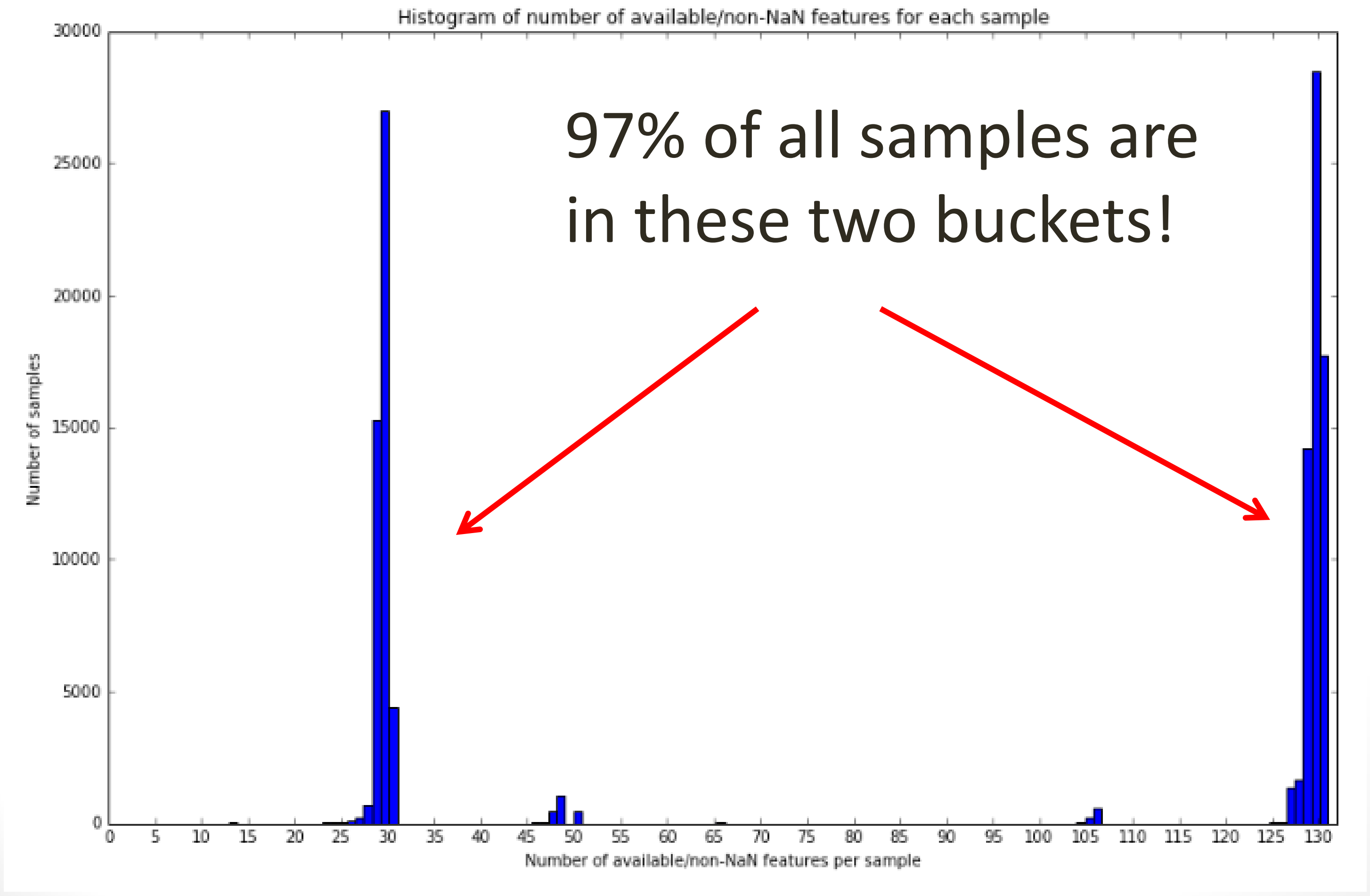


Look Ma!

No Features!!!

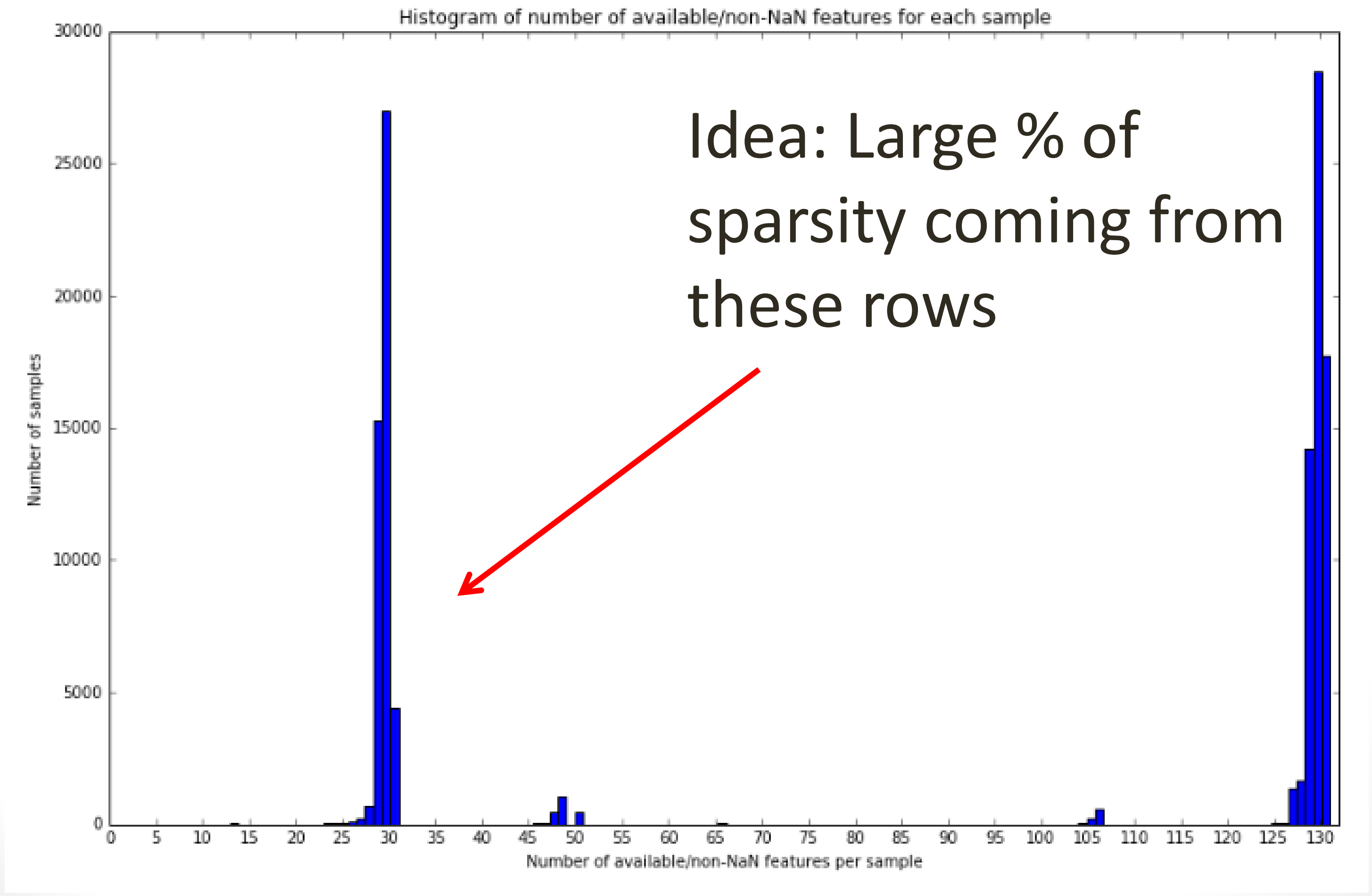
Data Exploration

- Let’s look at the picture on a per-row basis
- Plotting the number of non-NaN features per row



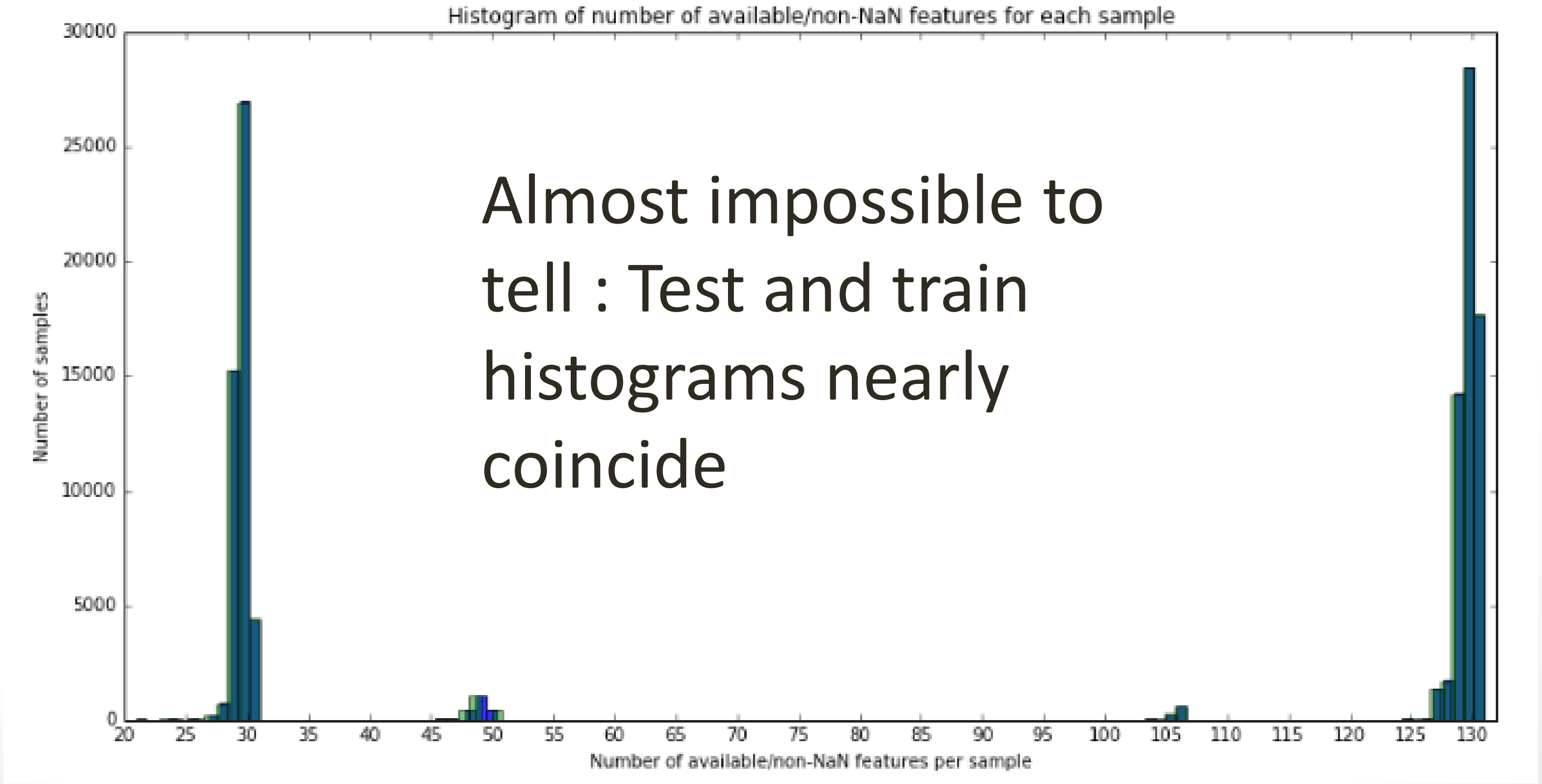
Data Exploration

- Let's look at the picture on a per-row basis
- Plotting the number of non-NaN features per row



Data Exploration

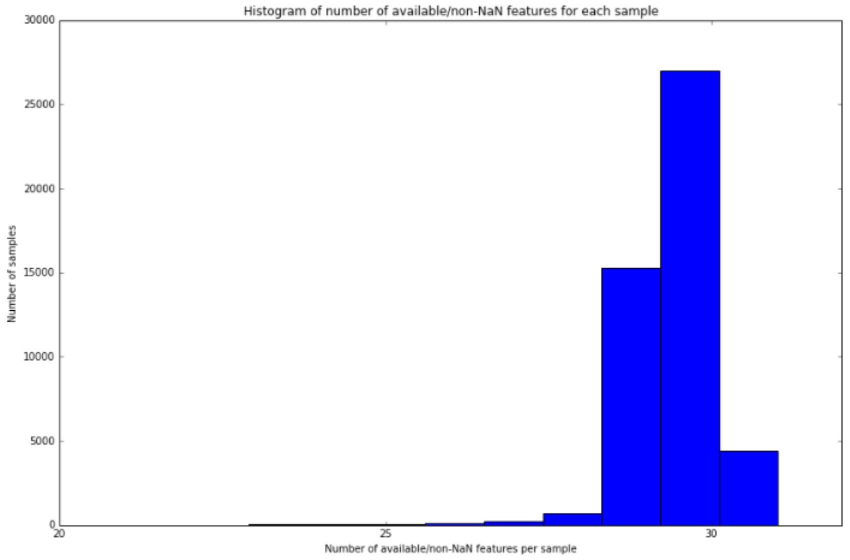
- Solution to reducing sparsity: Split the Data into two parts
- But, wait! What about the test data? Does it have this split too?



Almost impossible to tell : Test and train histograms nearly coincide

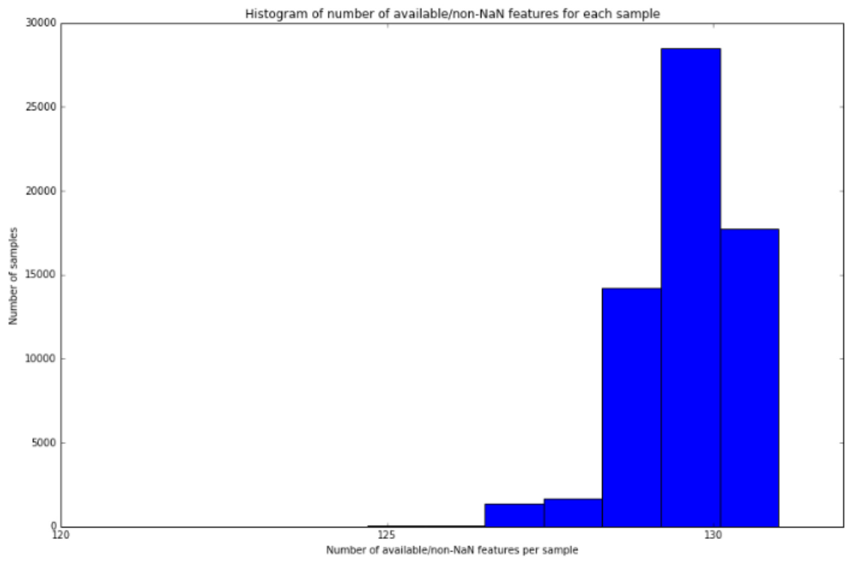
Data Exploration

- So now we have two data sets:



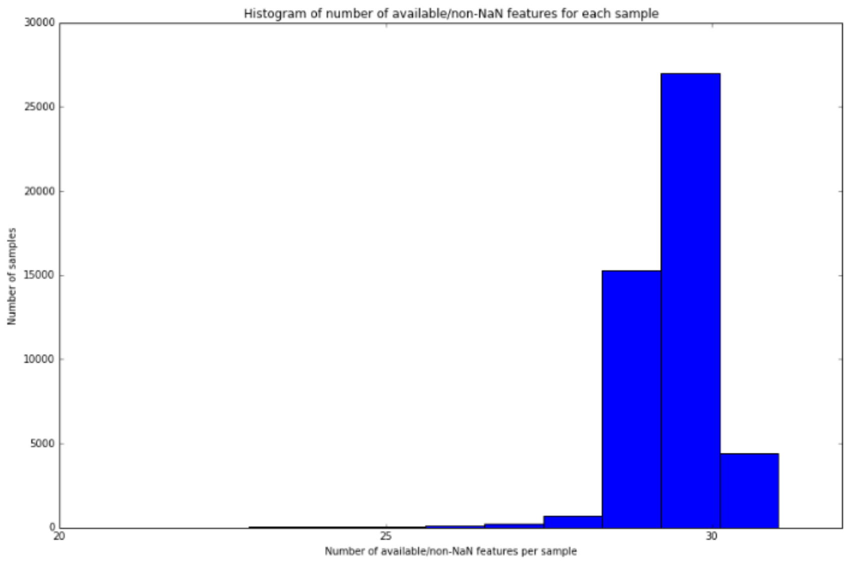
~63k rows, 131 features

~47k rows, 31 features



Data Exploration

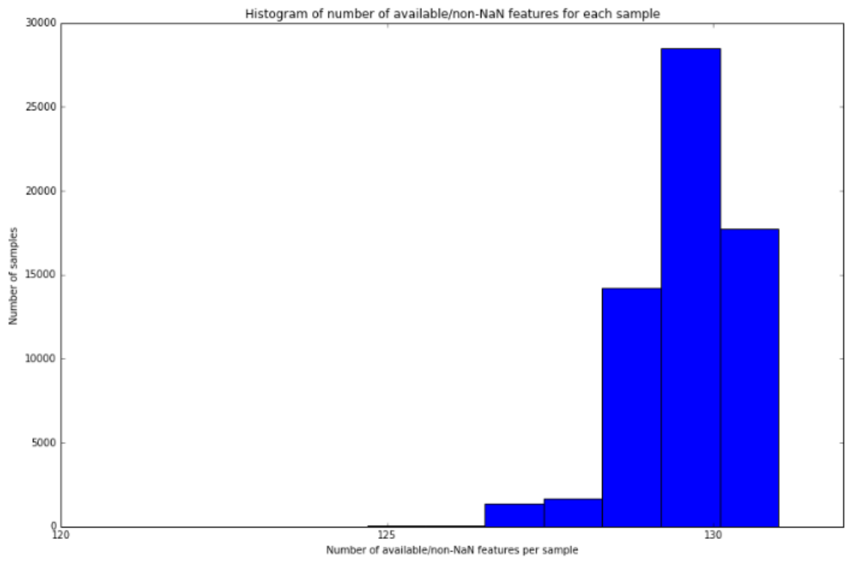
- So now we have two data sets:



~63k rows, 131 features

We will use two classifiers to make predictions on either set

~47k rows, 31 features



More Data Exploration

- All training data: 19 categorical features and 112 numerical features
- Smaller subset: 19 Categorical, 12 Numerical
- Larger subset: 19 Categorical, 112 Numerical

- New data sets are 3.9% and 0.8% sparse

- Unique Values for Categoricals
  - # of Uniques → # of new features (via one-hot encoding)
  - Large # of uniques is a problem
  - V22 has 18k uniques – we will drop it from the feature set

To impute or not?

- Most classification algorithms do not work with missing values
- Common solution: impute (estimate) values based on
  - Distribution of available values
  - Mean/median
- Pros:
  - "Complete" data set
  - We can use several algorithms to classify
- Cons:
  - Based on patterns in missing values seen so far, missing values are absent for a reason
  - Imputation would add spurious relationships among data
- Decision: We will not impute

XGBoost

- XGBoost is an implementation of boosted trees
- Advantages:
  - Handles missing values well
  - In-built ensembling
  - In-built regularization
  - Fast and can run on multiple cores

- Using XGBoost → Imputation not required

Parameter Tuning

- XGBoost has a number of parameters for tuning
  - Tree depth
  - Criteria for partitioning leaf nodes
  - Regularization parameters – L1 and L2
  - Learning rate (or gradient descent rate)
  - Sampling parameters for creation of new trees

- 5-fold cross validation and predictions on holdout data from training to prevent overfitting
- GridSearchCV and RandomizedSearchCV to explore parameter space

XGBoost Outputs

- Returns predicted probabilities for the 1 class
- Feature importance – weights of features in the trees used for prediction

- Can be used to do a successive feature reduction : remove lowest ranked features – re-run the classifier

Results

- **30-feature data set (Best results on holdout training data)**
- Log loss: 0.44355929663
- Accuracy: 0.800154104791
- Area under ROC curve 0.771802793489

- **130-feature data set (Best results on holdout training data)**
- Log loss: 0.475450764217
- Accuracy: 0.776955380577
- Area under ROC curve 0.747324417547

Results

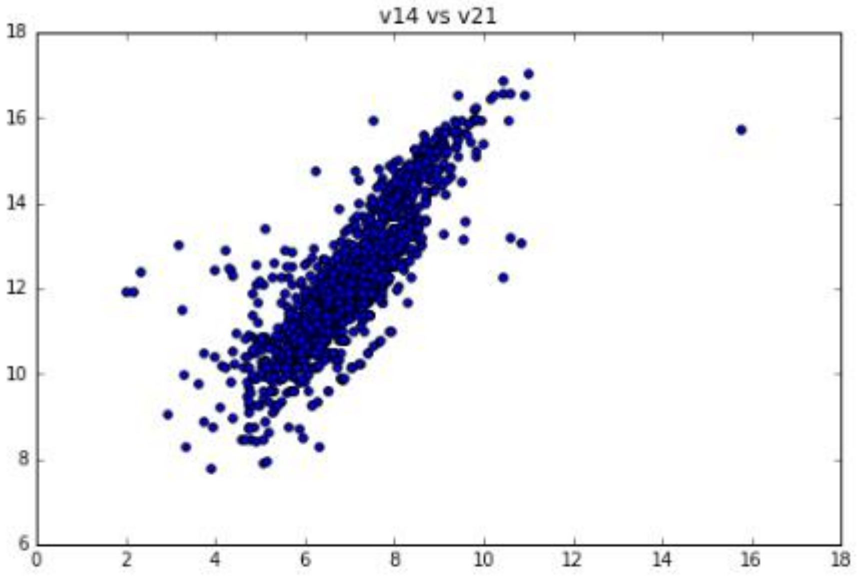
- **30-feature data set (Best results on holdout training data)**
- Log loss: 0.44355929663
- Accuracy: 0.800154104791
- Area under ROC curve 0.771802793489

- **130-feature data set (Best results on holdout training data)**
- Log loss: 0.475450764217
- Accuracy: 0.776955380577
- Area under ROC curve 0.747324417547

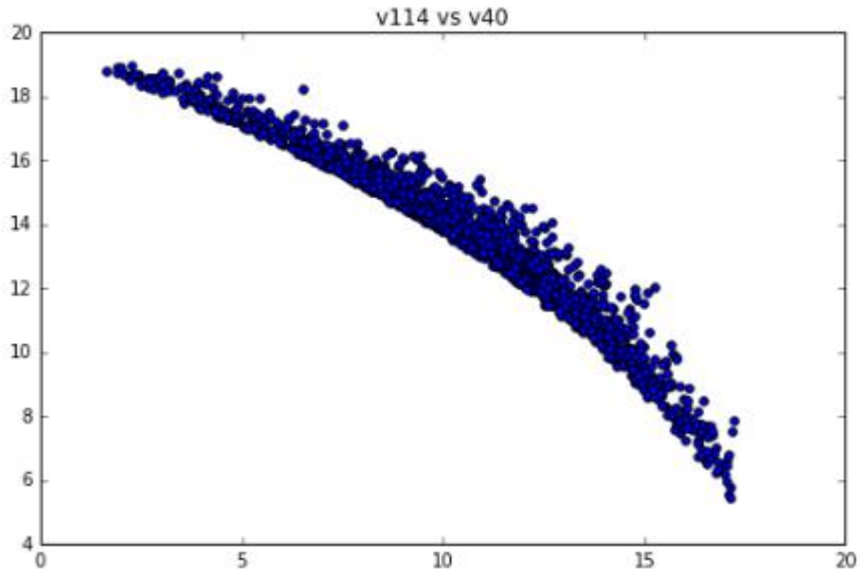
- Results are competent, but not great
- 75-80% accuracy with virtually no information about the data is not insignificant
- However, high misclassification means it is not reliable as a decision maker on it’s own

Some Other Approaches Attempted

- Removing correlated features

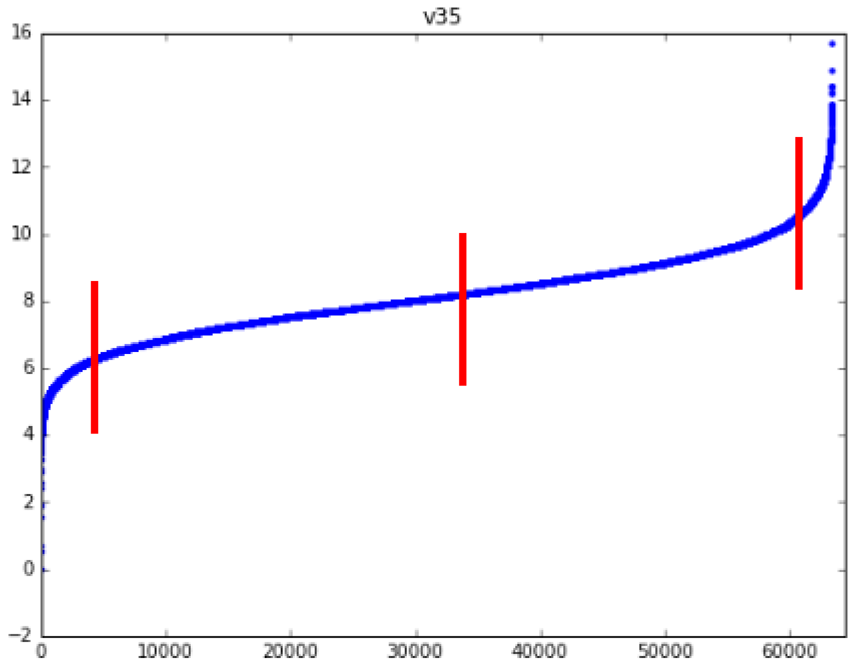


Led to minor improvements but not always replicable



Some Other Approaches Attempted

- Removal of features with large # of missing values
- Creating pair wise combinations of categorical variables
- Reducing numerical values to percentile-based buckets (5%, 50%, 95%)



Conclusions and Recommendations

- Stronger feature engineering is required
  - Major challenge is anonymization of data
  - In this case, feature engineering → De-anonymization
- XGBoost is recommended for classification tasks with missing values
- Recommendation to self: Not the best project for DSI Capstone
  - Challenges mainly around deciphering the data
  - In a real-world case, the labels would have tangible meaning
  - Difficult problems are good for learning, not necessarily important