

# Cutting Edge Data Science Technique

Owen Zhang

# Agenda

- Intro
- The big picture -- life cycle of data science projects
- Data science competitions and DataRobot in that context
- Secret sauce for DS competition success
- Some examples
- Data science competitions vs the real world
- Beyond technical challenges

# My Background

## Owen

### Current

- Chief Product Officer



### Previous

- VP, Science




MASTER

Highest†  
**1st**

Current†  
**3rd**  
/462,246

168,131.4 points  
Joined 5 years ago  
†Ranking method changed 13 May 2015 (?)





1st/634



1st/414



1st/367



2nd/1687



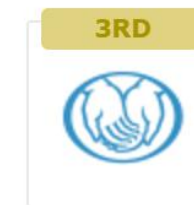
2nd/1604



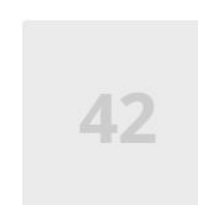
2nd/337



2nd/102

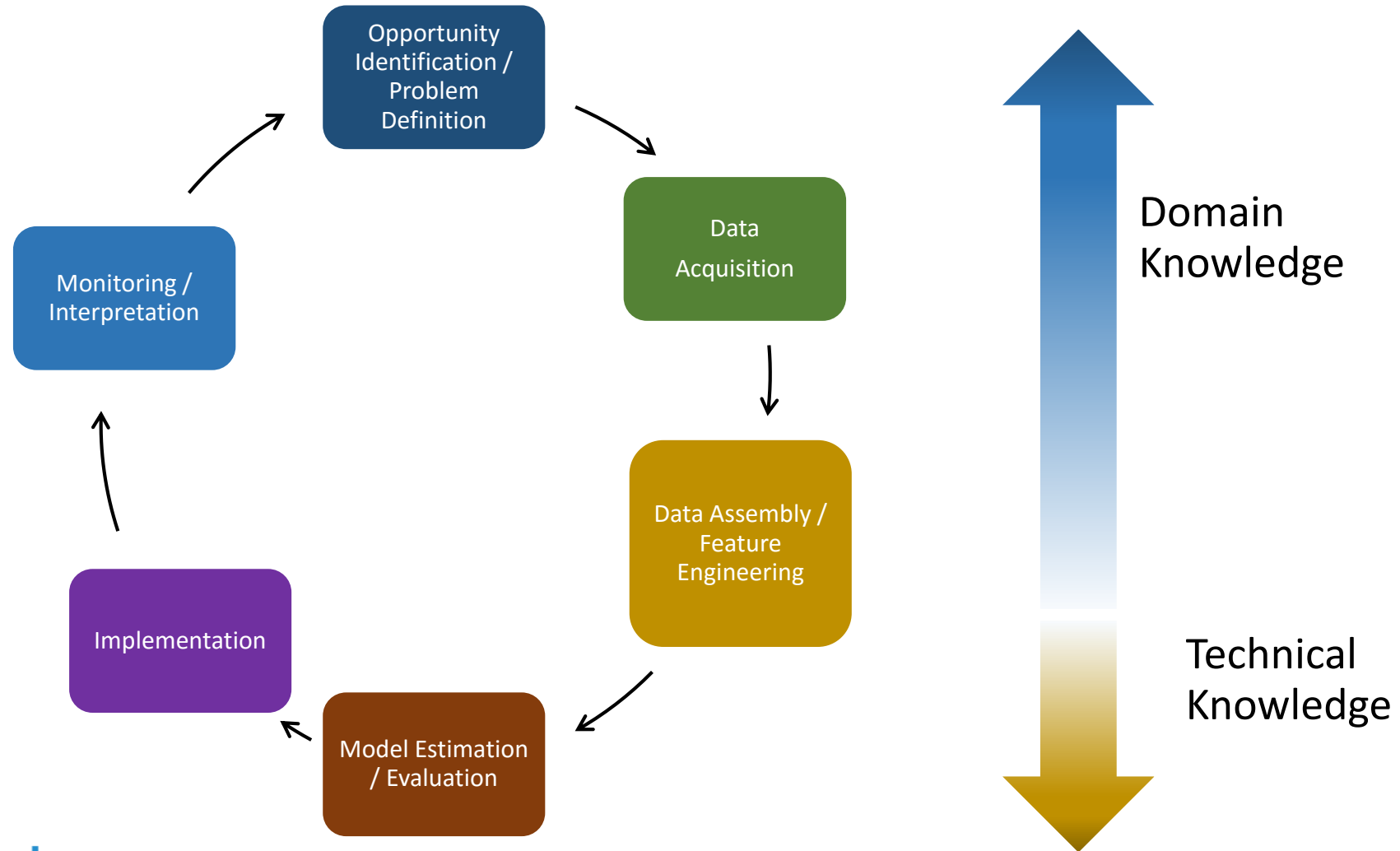


3rd/1568

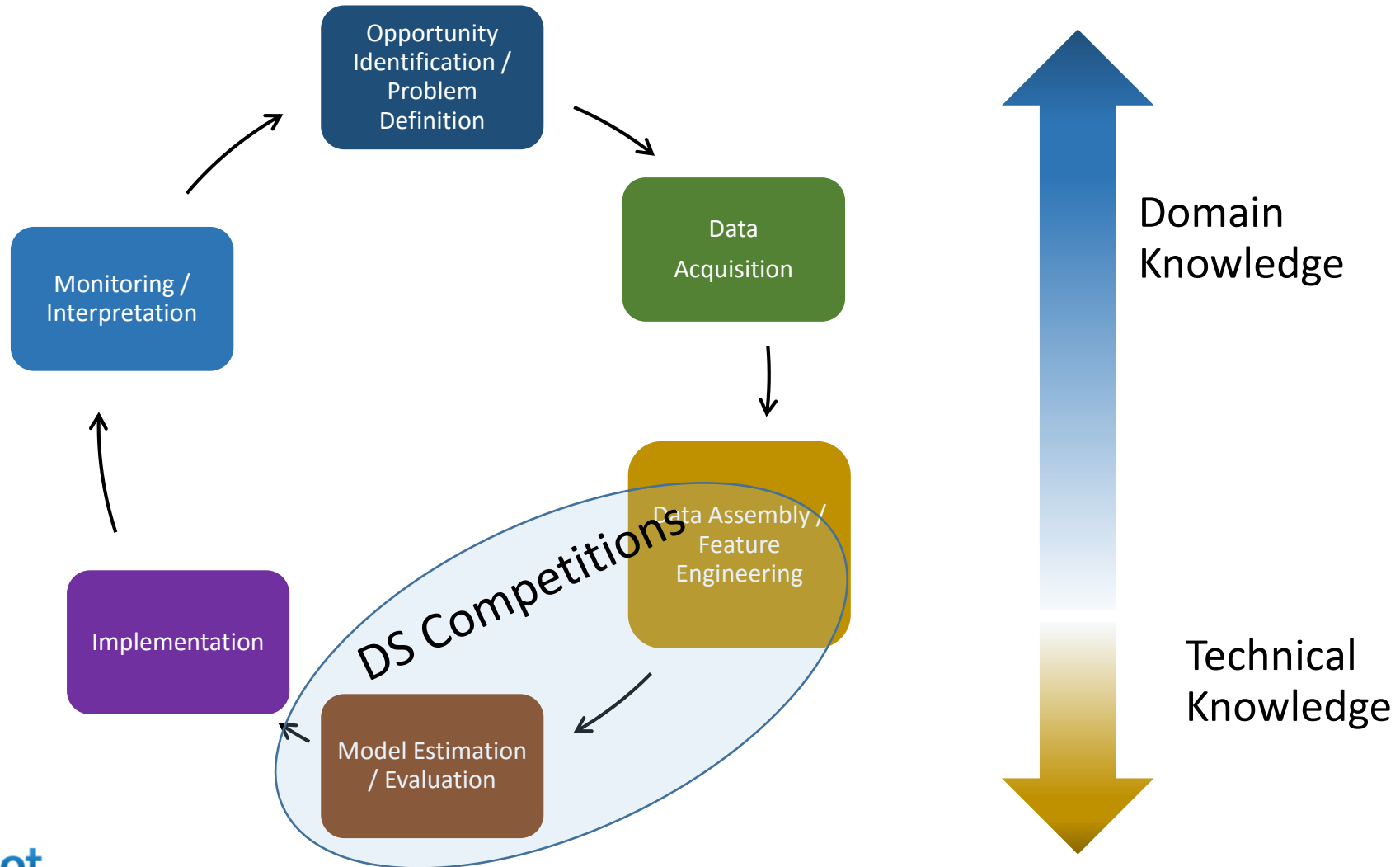


Competitions

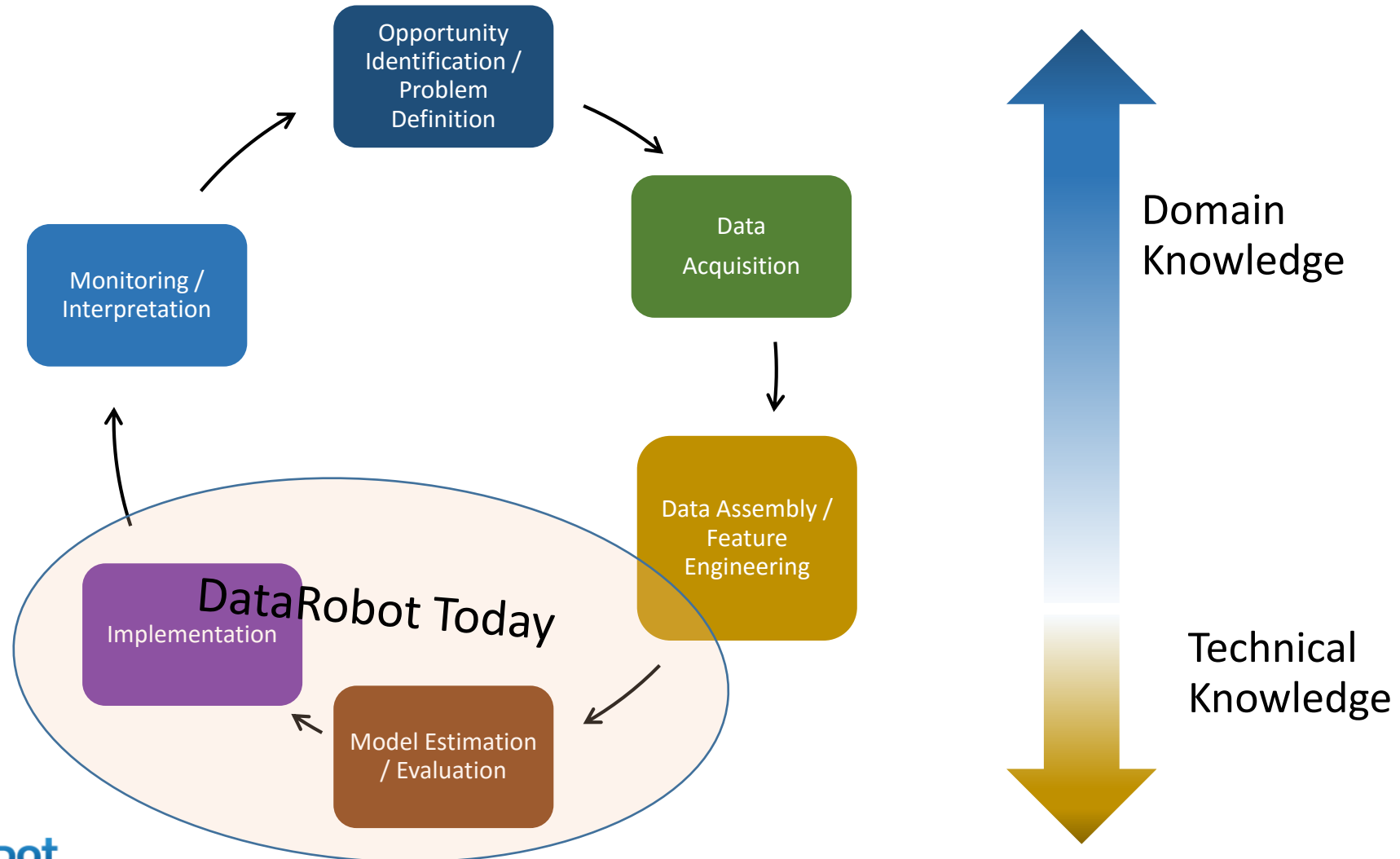
# The Big Picture – Life Cycle of DS



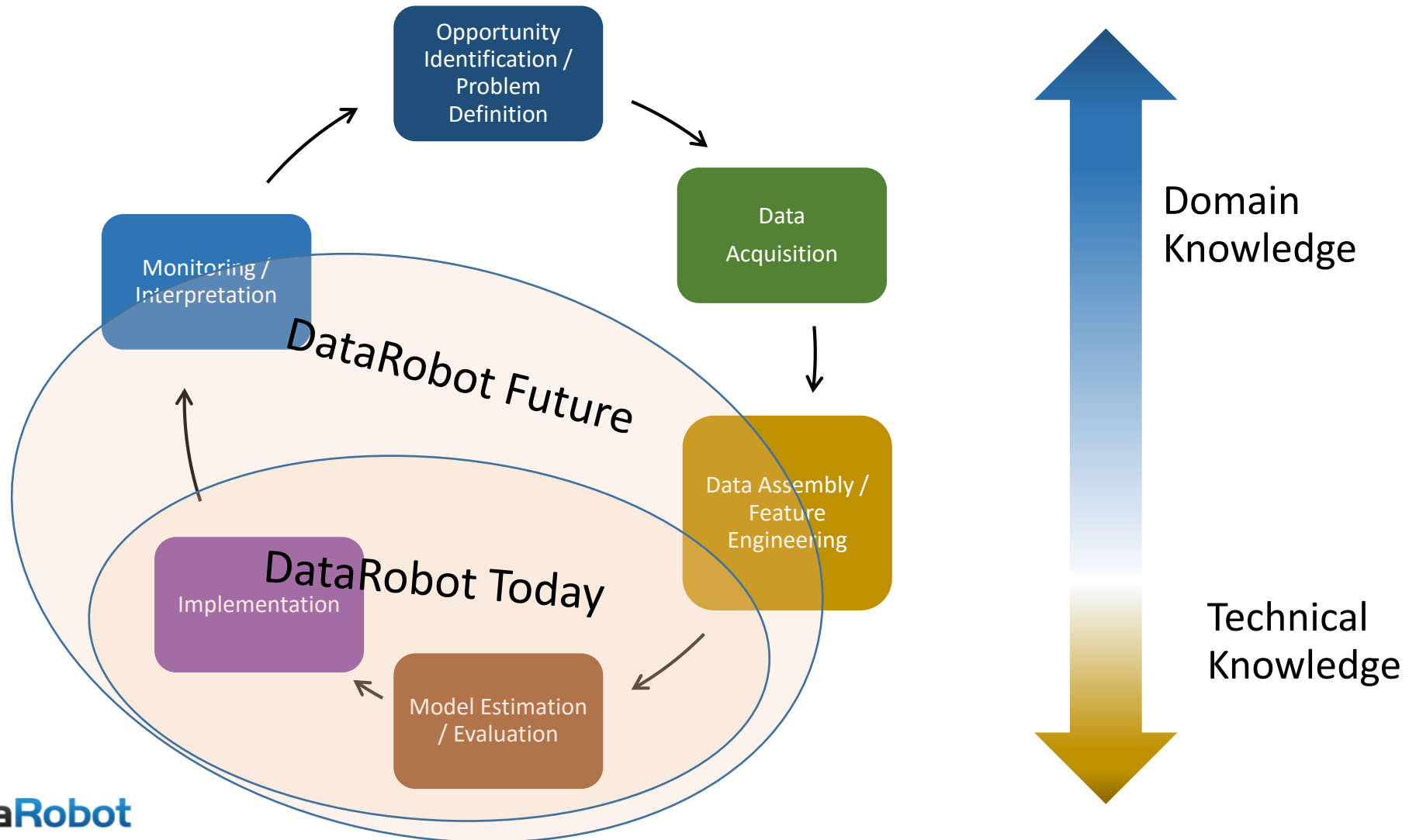
# DS Competition in the Big Picture



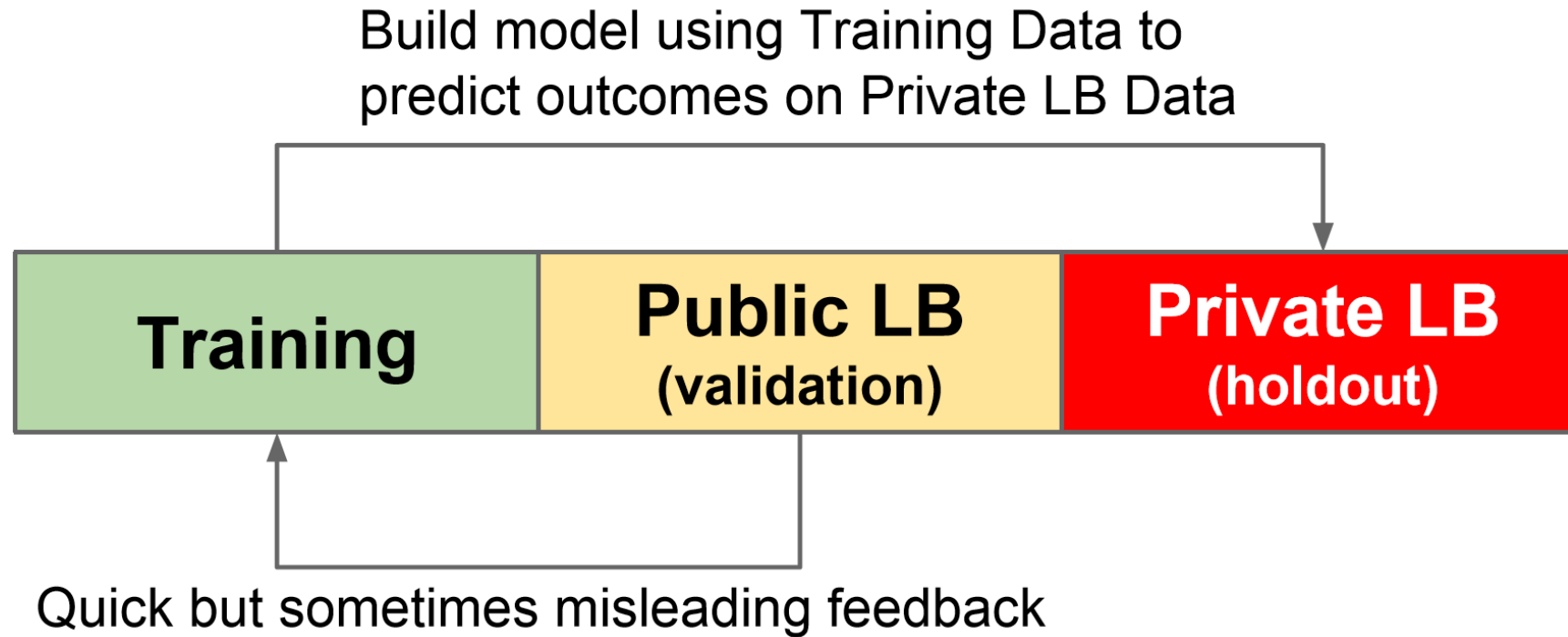
# DataRobot in the Big Picture



# DataRobot in the Big Picture -- Future



# Structure of a Typical DS Competition





# The (Not so) Secrete Sauce

- Luck
- Discipline
  - Proper validation framework
- Effort
- Domain knowledge + Feature engineering
- The “right” model structure
- Familiarity with many machine learning tools/packages
- Coding/data manipulation efficiency

# A Little Bit of Philosophy

- There are many ways to over fit
- Beware of “multiple comparison fallacy”
  - There is a cost in “peeking at the answer”
  - A more formal treatment of this cost can be seen in this paper <http://science.sciencemag.org/content/349/6248/636.short>
  - My experience: the first ideas (if they work) are the best

***“Think” more, “try” less***

# Get the Basics Right

- A key step is to understand and replicate the training/test data split
- Have a good toolset and model building/validation pipeline
  - Accumulate packages, utilities, etc. over several competitions
  - Always try to learn from other competitors
- Read up on the subject, if needed
- Be ready to spend a lot of time on a problem
- A good computer is usually a worthwhile investment

# The Right Way of Using Public LB

- Public LB is an “independent” check of solution correctness
  - It happens quite often (to me at least) that my initial submission performs terribly due to misunderstanding of the dataset, or coding errors
- Hope is to see public LB score and local CV score move in tandem
- For certain data split setup, it is crucial to use public LB feedback as “pseudo training data”
  - Train/test split is by time or group, but public/private LB split is random
  - In this situation, public LB data is a much closer match to private LB compared to training data
  - Use training data to build baseline model, then tune/ensemble/stack using public LB scores

# The Toolbox

- Data manipulation – R/Python
  - There is virtually infinite possibilities in feature engineering, many domain dependent
- Modeling tools
  - Gradient Boosting Machine (xgboost recommended)
  - Neural Networks (esp. for multiclass classification problems)
    - CNN a must for image recognition
  - Regularized regression / SGD
  - Factorization machine
  - SVM
  - Random Forest, ExtraTrees, etc.
- Ensemble / Stacking

# Enough Philosophy, Let's See Some Examples

- Brief overview of 4 competitions (where I did well)
- Provide some flavor on “what works”
- Please note:
  - Behind every smart idea there are probably 10 (or 100) silly ideas
  - Behind each success there are 10 or more terrible failures
- These are symptoms of bad performance
  - No good ideas at all
  - Never get validation right – local CV result moves randomly compared to LB
  - Model performance far from leaders and no hope to catch up

# 1. Amazon User Access -- Intro

- One of the most popular competitions on Kaggle to date with 1687 teams
- Use anonymized features to predict if employee access request would be granted or denied
- All categorical features
  - Resource ID / Mgr ID / User ID / Dept ID ...
  - Many features have high cardinality
- A recurring feature engineering challenge – how to convert categorical features into numerical, so that they can be fed into GBM

# 1. Amazon User Access -- Summary

- Encode categorical features using observation counts
  - This is even available for test data!
- Encode categorical features using average response
- Build different kind of trees + ENET
  - GBM + ERT + ENET + RF + GBM2 + ERT2
  - I didn't know VW (or similar), otherwise might have got better results.
- Code available <https://github.com/owenzhang/Kaggle-AmazonChallenge2013>



# 1. Amazon User Access – Response Encoding

- Every level of a categorical feature is represented by the average value of the response variable in that level in the training data
- However this leads to severe overfitting when the # of rows in each level is low
  - Double dipping the response
- Take the following steps to control overfitting:
  - For training data
    - Use the average response of all the other rows in each level, except for the current row
    - Shrink the row  $\text{avg}(\text{response})$  toward global mean, based on # of rows in that average
    - Apply a small random multiplying factor in  $(.95, 1/.95)$  to avoid concentration of frequent values
  - For test data
    - Still apply shrinkage, but not leave-self-out or randomization

## 2. Allstate User Purchase Option Prediction

- Predict final purchased product (insurance policy) options based on earlier transactions.
- 7 correlated targets, each has 2-4 levels
- This turns out to be very difficult because:
  - The evaluation criteria is all-or-nothing: all 7 predictions need to be correct
  - The baseline “last quoted” is very hard to beat.
    - Last quoted 53.269%
    - #3 (me) : 53.713% (+0.44%)
    - #1 solution 53.743% (+0.47%)
- Key challenges -- capture correlation, and not to lose to baseline

## 2. Allstate Purchase Option -- Summary

- Dependency -- Chained models
  - First build stand-alone model for F
  - Then model for G, given F
  - $F \Rightarrow G \Rightarrow B \Rightarrow A \Rightarrow C \Rightarrow E \Rightarrow D$
  - “Free models” first, “dependent” model later
  - In training time, use actual data
  - In prediction time, use most likely predicted value
- Not to lose to baseline -- 2 stage models
  - Build one more model to predict which one to use: chained prediction, or baseline

# 3. Liberty Mutual Fire Loss Prediction

## DATA OVERVIEW

- ~1 million insurance records
- 300 variables:

**target** : The transformed ratio of loss to total insured value

**id** : A unique identifier of the data set

**dummy** : Nuisance variable used to control the model, but not a predictor

**var1 – var17** : A set of normalized variables representing policy characteristics

**crimeVar1 – crimeVar9** : Normalized Crime Rate variables

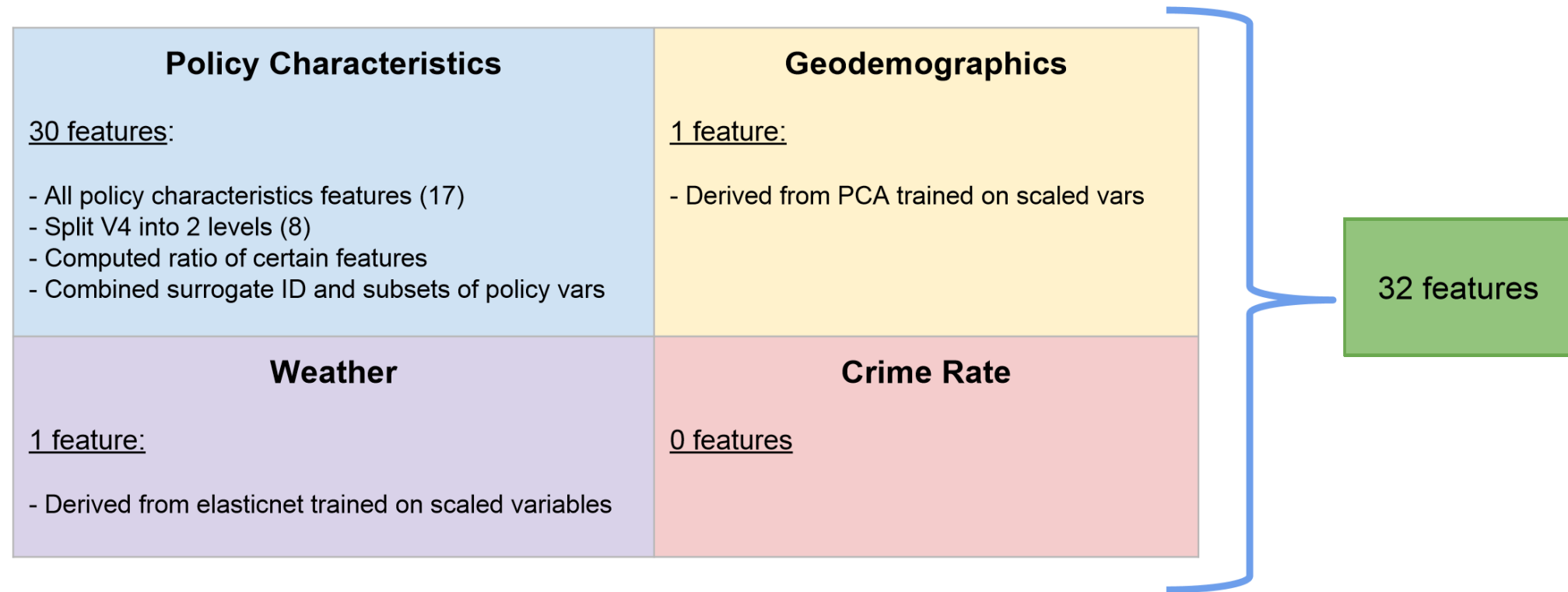
**geodemVar1 – geodemVar37** : Normalized geodemographic variables

**weatherVar1 – weatherVar236** : Normalized weather station variables

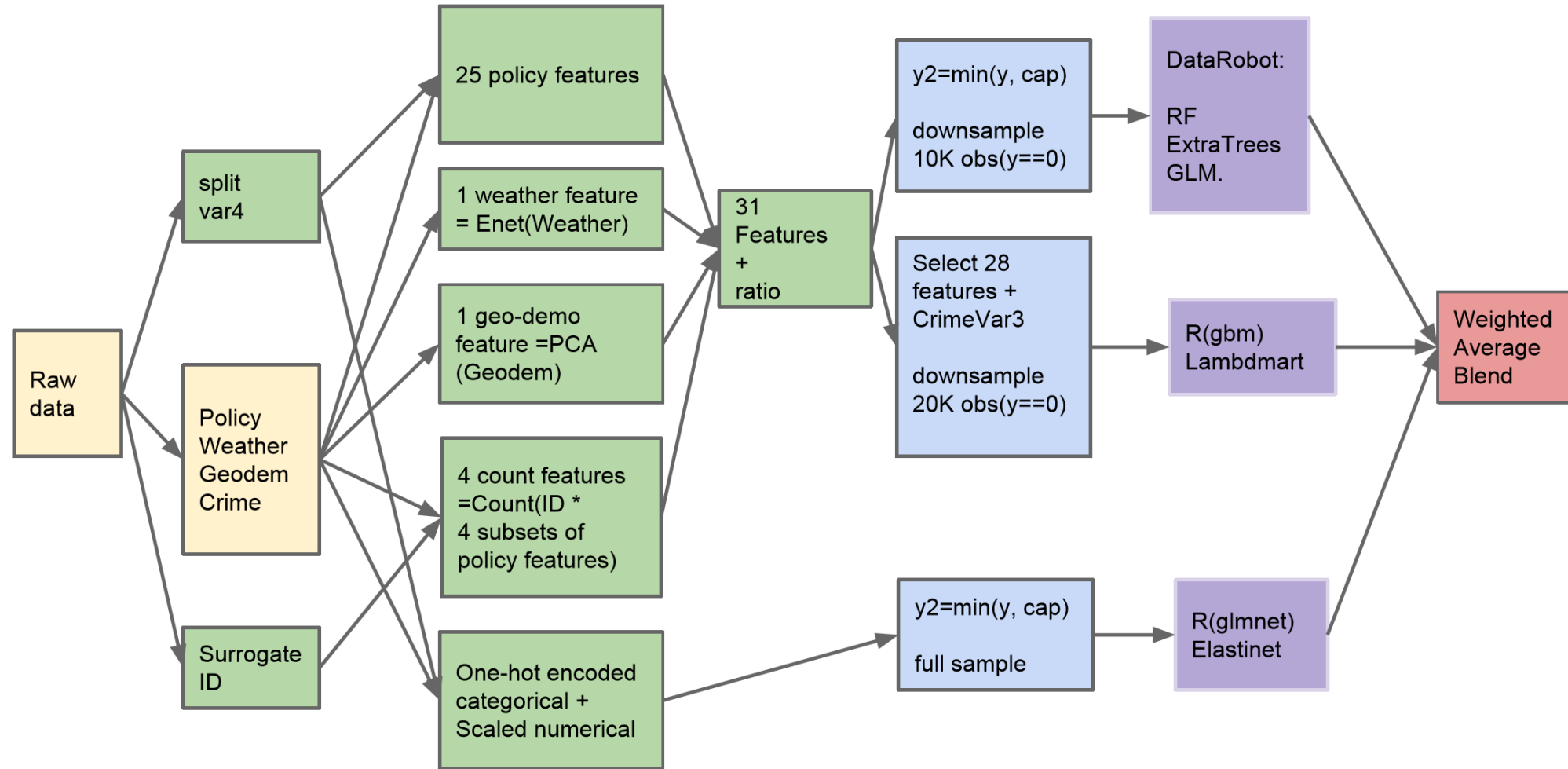
Numeric Variable Name	Variable Type		
target	Continuous		
id	Discrete		
var10	Continuous		
var11	Continuous		
var12	Continuous		
var13	Continuous		
var14	Continuous		
var15	Continuous		
var16	Continuous		
var17	Continuous		
crimeVar1 – crimeVar9	Continuous		
geoDemVar1 – geoDemVar37	Continuous		
weatherVar1 – weath	Categorical Variable Name	Variable Type	Possible Values
var1		Ordinal	1, 2, 3, 4, 5, Z*
var2		Nominal	A, B, C, Z*
var3		Ordinal	1, 2, 3, 4, 5, 6, Z*
var4*		Nominal	A1, B1, C1, D1, D2, D3, D4, E1, E2, E3, E4, E5, E6, F1, G1, G2, H1, H2, H3, I1, J1, J2, J3, J4, J5, J6, K1, L1, M1, N1, O1, O2, P1, R1, R2, R3, R4, R5, R6, R7, R8, S1, Z*
var5		Nominal	A, B, C, D, E, F, Z*
var6		Nominal	A, B, C, Z*
var7		Ordinal	1, 2, 3, 4, 5, 6, 7, 8, Z*
var8		Ordinal	1, 2, 3, 4, 5, 6, Z*
var9		Nominal	A, B, Z*
dummy		Nominal	A, B

### 3. Liberty Mutual Fire Loss -- Feature Engineering

- Broke feature set into 4 components
- Created surrogate ID based on identical crime, geodemographics and weather variables

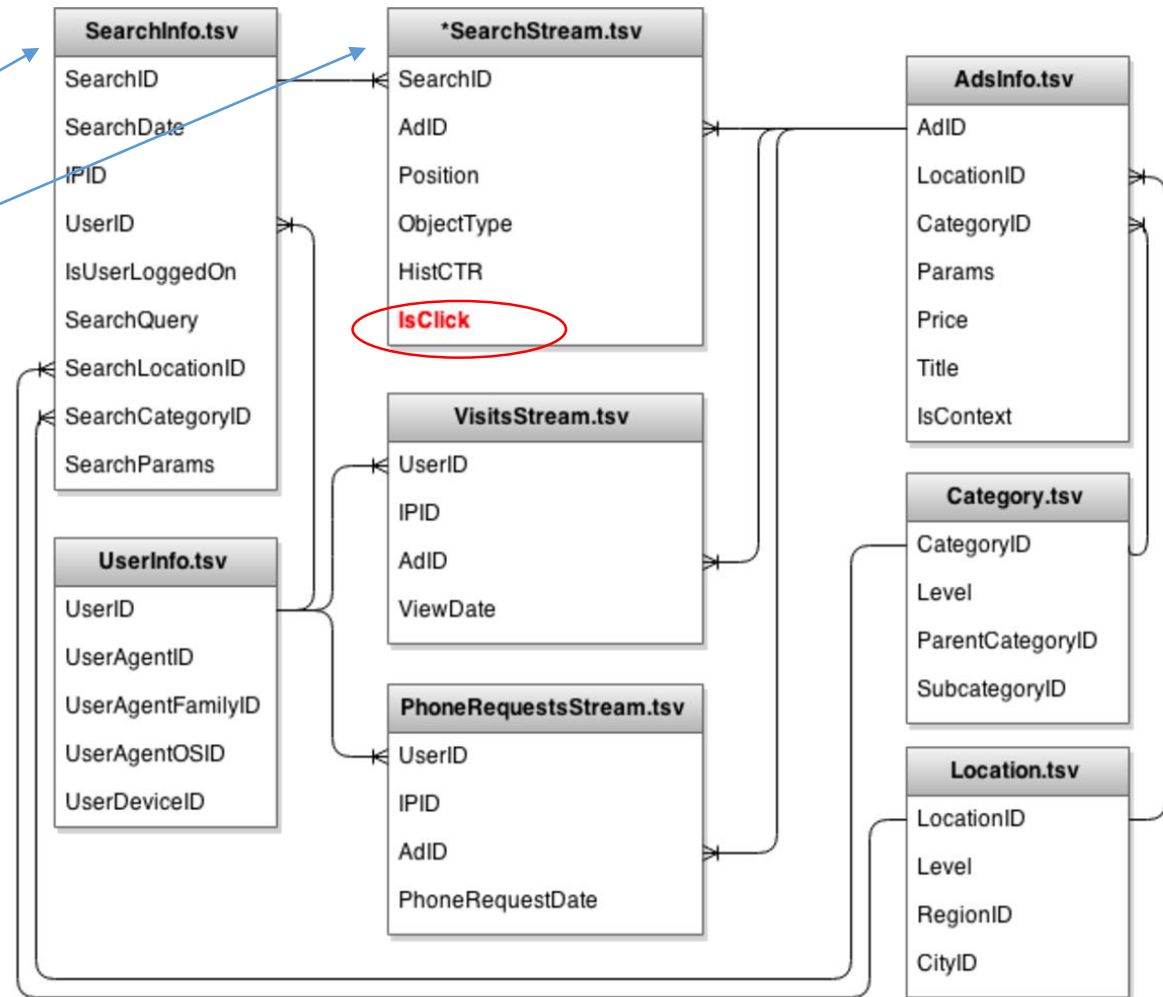


### 3. Liberty Mutual Fire Loss – Model Structure



## 4. Avito Context Ads Click Through Rate Prediction

- Big Data?
  - 112159462 rows
  - 392346948 rows
- It is all relative
- All data processing done in R on a single 16core/256GB machine



## 4. Avito Summary

- This comp has fairly large data and nontrivial structure. I spent about  $\frac{2}{3}$  of time doing feature engineering and  $\frac{1}{3}$  training models.
- It is interesting that pure xgboost models performed so well that I never had the need to build other models to add to the ensemble.
- The structure of the data provide interesting feature engineering opportunities.
- The data are large enough that there is virtually no risk of overfitting public LB.
- Code available <https://github.com/owenzhang/kaggle-avito>



## 4. Avito Features

- raw features, such as Position, HistCTR,
- simple time based features, such as time of day and day of the week
- sequence based features, such as # of ads seen up to a given impression
- average prior response, such as # of ads clicked up to a given impression
- text based features, using ngram/tfidf/svd, such as SearchQuery
- text similarity, for example, similarity between SearchQuery and Title
- simple text stats, such as # of characters in Title
- price based features, such as average price for given category views of a given user
- entropy based features how diverse/concentrated a user's history is
- categorical features that are encoded with click rate, adjusted for credibility

# Bane of DS Competitions – Data Leakage

- It is extremely difficult to set up a “smart data scientist proof” DS competition
- Many kind of leaks can ruin a dataset:
  - Sequential ID/rows that are highly correlated with target
  - Name, timestamp, or other attributes of data files
  - Data elements that are dependent on the target variable
- In actual practice, we must be vigilant against data leaks
  - Watch out for models that are “too good to be true”
  - Some leaks can be quite subtle
  - A good rule – never use data elements that don’t exist at prediction time

# DS Competitions vs the Real World

- Implementation is a curial component in reality
  - Time to market and run-time performance are important
  - A fast to implement 80% solution may outperform a hard-to-implement 95% solution, especially in a fast moving environment
- Complex models carry higher cost in both maintenance and interpretation
- Generally speaking, complex models are less robust against “unknown unknown”

# Practical Lessons Learned from Kaggle

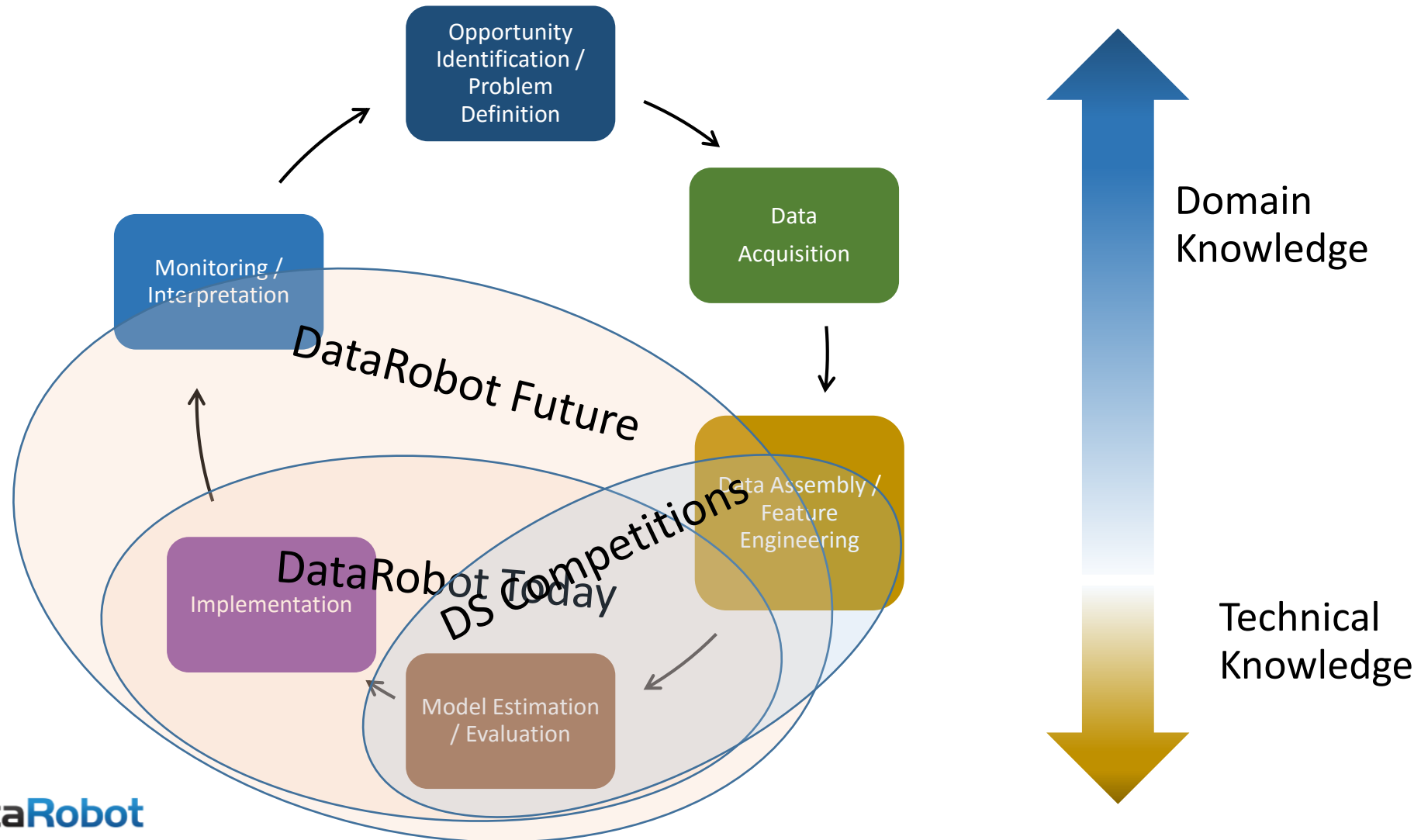
- A group of highly competitive experts can quickly create models that outperform existing baseline (very often created by competent internal team), even with little domain knowledge and anonymized data
- Winning models, however, are usually NOT good candidates for direct production use, because
  - They are far too complex, often hard or impossible to implement and interpret
  - Very often a significant gain of prediction accuracy comes from data leakage, not real signal – there is no LB to game in the real world

# Then What Good can Come from DS Comps?

Plenty:

- Identify potential data issues / leakage
- Provide benchmark of “upper end” of predictability
- Provide ideas for feature engineering for the production model
- Screen all possible modeling approaches and find the most promising ones
- Recruit talent
  - Good test of practical technical capability
  - Also shows that the sponsor is cool

# Beyond Technical Challenges



# Crucial Tasks that still Require Us

- Identify opportunities and define problems
- Distinguish between correlation and causality
  - We don't do a good job either, but can better utilize external knowledge
- Evaluate which signals/patterns are more likely to persist over time or changing environment
- Find useful data sources

# Useful Resources

- <http://www.kaggle.com/competitions>
- <http://www.kaggle.com/forums>
- <http://statweb.stanford.edu/~tibs/ElemStatLearn/>
- <http://scikit-learn.org/>
- <http://cran.r-project.org/>
- [https://github.com/JohnLangford/vowpal\\_wabbit/wiki](https://github.com/JohnLangford/vowpal_wabbit/wiki)
- <http://keras.io>
- <https://github.com/dmlc/xgboost>
- ...