Kaggle tips

Dmitry Larko, Sr. Data Scientist @ H2O.ai
dmitry@h2o.ai

December 15th, 2016

About me

Dmitry Larko

Sr. Data Scientist at H2O.ai

San Francisco Bay Area, CA, United States

Joined 4 years ago · last seen in the past day

in http://h2o.ai

Competitions Grandmaster

Home | Competitions (32) | Kernels (0) | Discussion (31) | Datasets (0) | More | Edit Profile

Competitions Grandmaster	
Current Rank 36 of 52,561	Highest Rank 25

Gold	Silver	Bronze
9	8	6

Kernels Contributor		
Unranked		
0	0	0

Discussion Contributor		
Unranked		
0	4	12

Bio

- About 10 years working experience in DB/Data Warehouse field. Until 4 years ago I learned about Kaggle from my dad, who competing on Kaggle as well (and does that better than me)
- My first competition was Amazon.com - Employee Access Challenge, I placed 10th out of 1687, learned a tons of new techniques/algorithms in ML field in a month and I got addicted to Kaggle.
- Until now I participated in 32 completions, was 2nd twice, won once and I am in Kaggle top- 100 Data Scientists.
- Currently I’m working as Senior Data Scientist at H2O.ai

Motivation. Why to participate?

- **To win** – this is the best possible motivation for competition;
- **To learn** – Kaggle is a good place to learn and the best places to learn on Kaggle are forum and kernels from past and current competitions;
- **Looks good in resume** – well... only if you're constantly winning ☺

How to start?

- Learn Python
- MOOC for Machine learning (Coursera, Udacity, Harvard)
- Participate in Kaggle “Getting started” and “Playground” competitions
- Visit Kaggle finished competitions and go through winner’s solution posted at competition’s forum

Kaggle's Toolset (1 of 2)

- Scikit-Learn (scikit-learn.org). Simple and efficient tools for data mining and data analysis. A lot of tutorials, many ML algorithms have scikit-learn implementation.
- XGBoost (github.com/dmlc/xgboost). An optimized general purpose gradient boosting library. The library is parallelized (OpenMP). It implements machine learning algorithm under gradient boosting framework, including generalized linear model and gradient boosted regression tree (GBDT).
  - Theory behind XGBoost:
    - https://homes.cs.washington.edu/~tqchen/pdf/BoostedTree.pdf
  - Tutorial:
    - https://www.kaggle.com/tqchen/otto-group-product-classification-challenge/understanding-xgboost-model-on-otto-data

Kaggle's Toolset (2 of 2)

- H2O (h2o.ai). Fast Scalable Machine Learning API. Has stat-of-the-art models like Random Forest and Gradient Boosting Trees. Allows you to work with really big datasets on Hadoop cluster. It also works on Spark! Check out Sparkling Water: h2o.ai/product/sparkling-water/
- Neural Nets/Deep Learning
  - Keras (github.com/fchollet/keras)
  - MXNet(github.com/dmlc/mxnet)

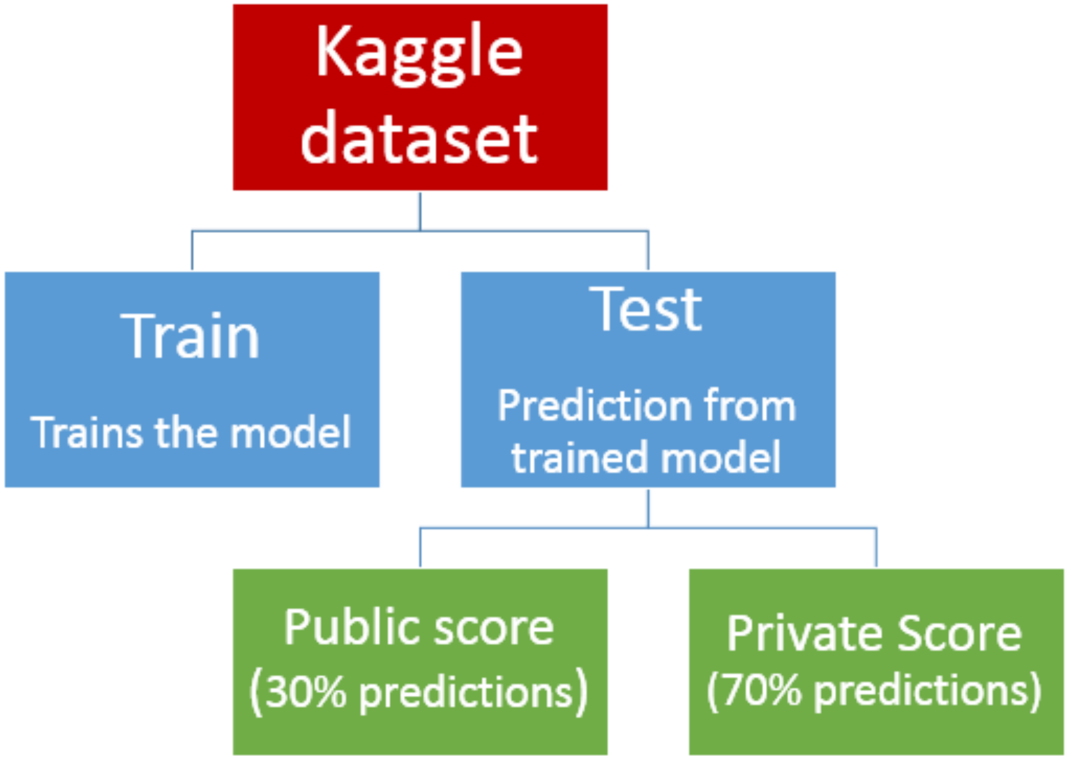
Kaggle's Toolset - Advanced

- Vowpal Wabbit (GitHub) – fast and state-of-the-art online learner. A great tutorial how to use VW for NLP is available here: github.com/hal3/vwnlp
- LibFM (libfm.org) - Factorization machines (FM) are a generic approach that allows to mimic most factorization models by feature engineering. Works great for sparse wide datasets, has a few competitors:
  - FastFM - ibayer.github.io/fastFM/index.html
  - pyFM - github.com/coreylynch/pyFM
- Regularized Greedy Forest (github.com/baidu/fast_rgf) - tree ensemble learning method, can be better than XGBoost, but you need to know how to cook it.
- Light GBM (github.com/Microsoft/LightGBM) – FAST gradient boosting framework based on decision tree algorithms.
- Four leaf clover, gives +100 to Luck

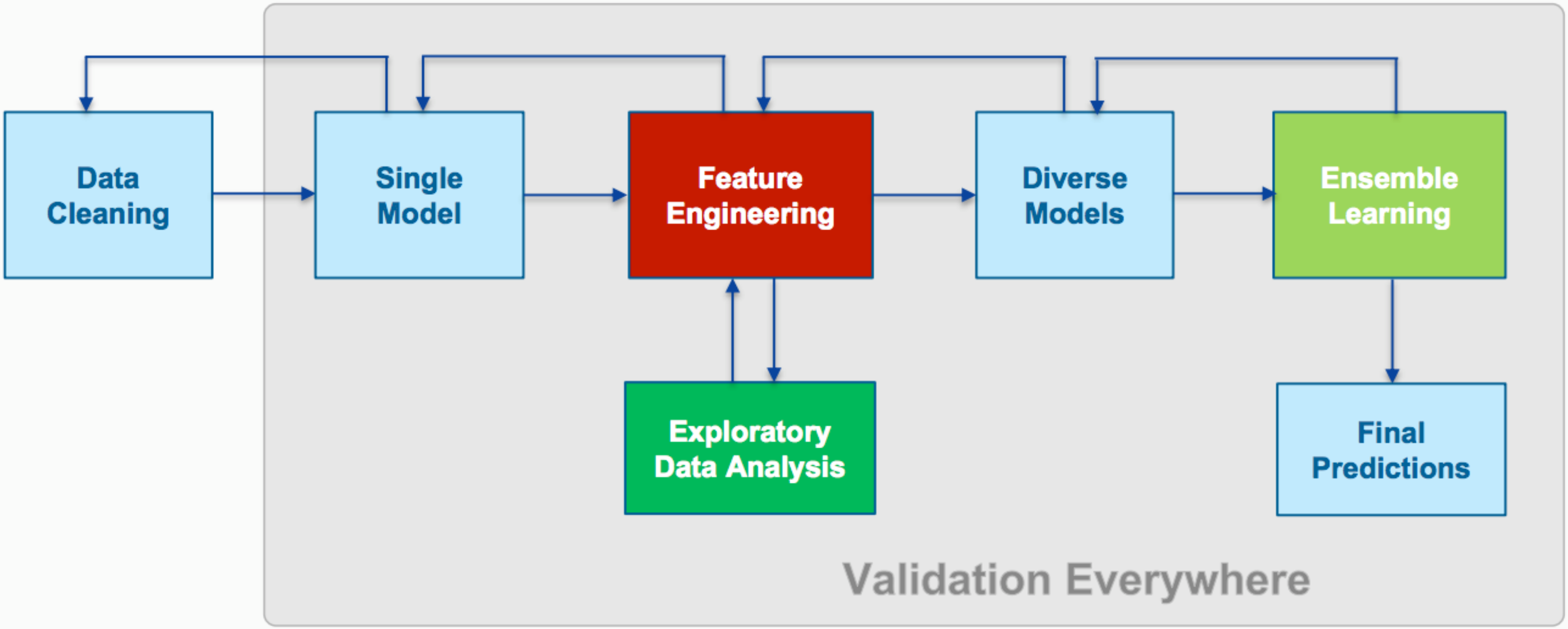
Which past competition to check?

- Well, all of them of course. But if I would need to choose I'd selected these 4:
  - Greek Media Monitoring Multilabel Classification (WISE 2014). Why?
    - Interesting problem, a lot of classes.
  - Allstate Claims Severity. Why?
    - A great dataset to learn and polish you ensembling techniques
  - Caterpillar Tube Pricing. Why?
    - A good example of business problem to solve
    - Need some work with data before you can build good models
    - Has a "data leakage" you can find and exploit
  - National Data Science Bowl. Why?
    - Good Deep Learning competition to start

Kaggle Competition Dataset and Rules



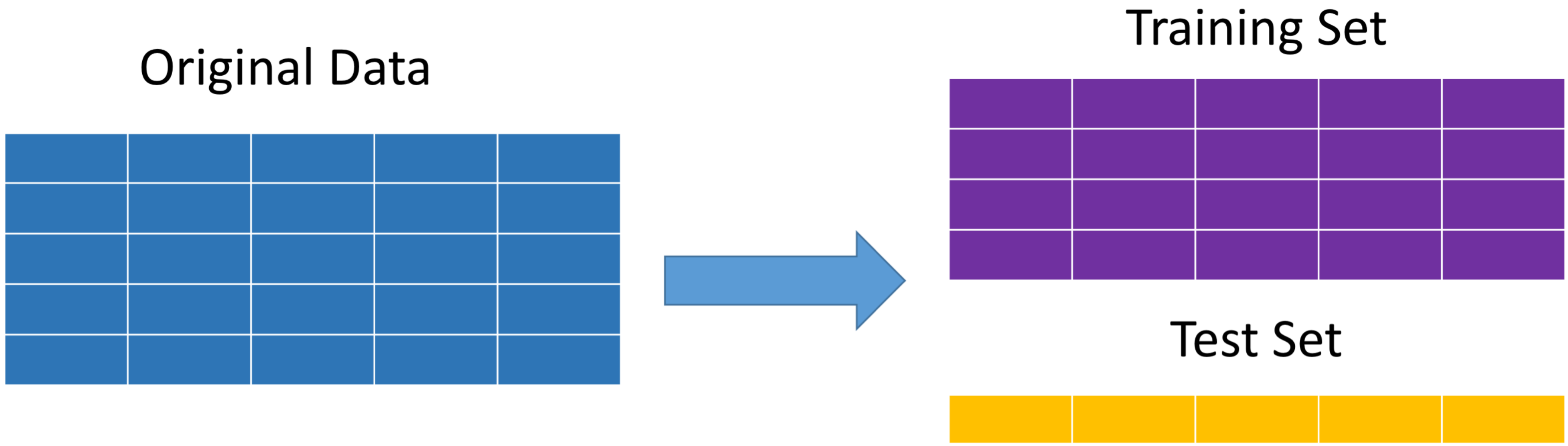
Pipeline


Data
Cleaning
Single
Model
Feature
Engineering
Diverse
Models
Ensemble
Learning
Exploratory
Data Analysis
Final
Predictions
Validation Everywhere

Validation

- Main idea you need to model competition train/test split, so you can test all your ideas locally:
  - Train/test is random:
    - Dataset is big or no access to good hardware:
      - Train/test split
    - Dataset is small and/or we have enough computational power:
      - N-fold cross validation
  - Train/test split is not random (good example is time-series):
    - Train/test split

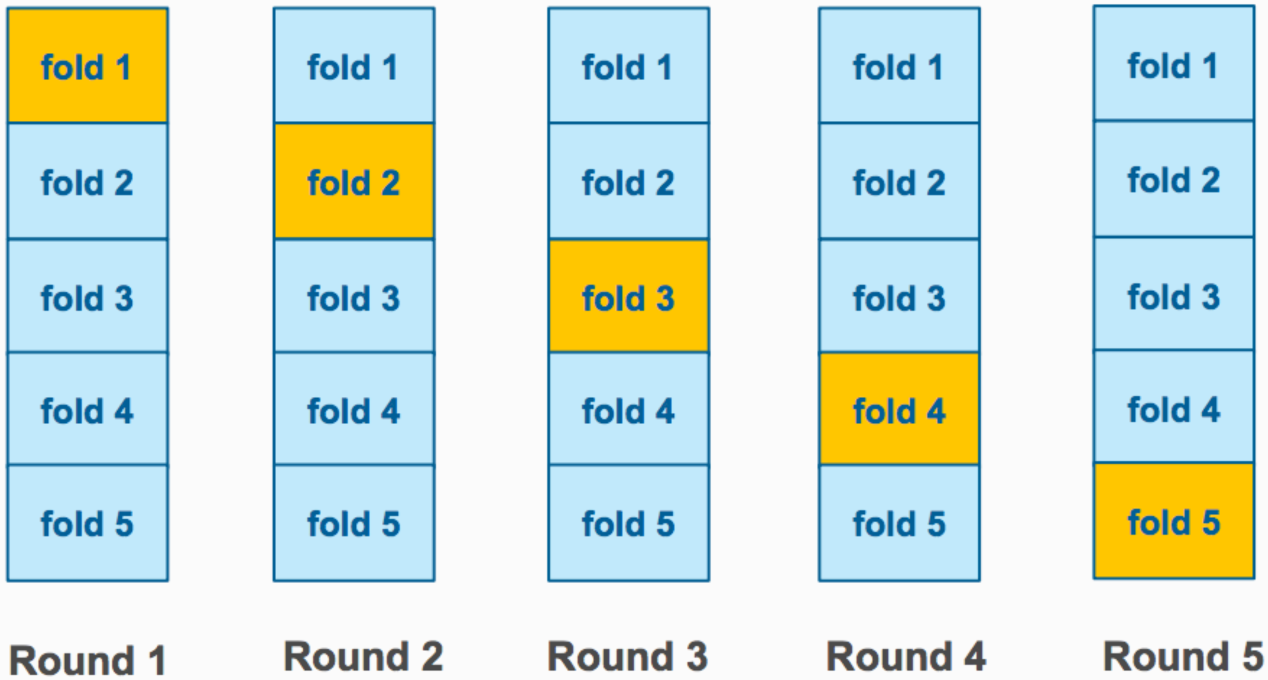
Train/Test split



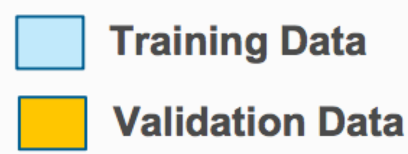
Train/validation split tips

- Rule-of-thumb: 80/20 or 70/30
- How to identify best validation set?
  - LB feedback
  - Adversarial validation. Combine train and test sets into one and train classifier to classify train and test examples correctly. When choose a number of misclassified examples that the model was most certain about. It means that they look like test examples but in reality are training examples
    - fastml.com/adversarial-validation-part-one
    - fastml.com/adversarial-validation-part-two

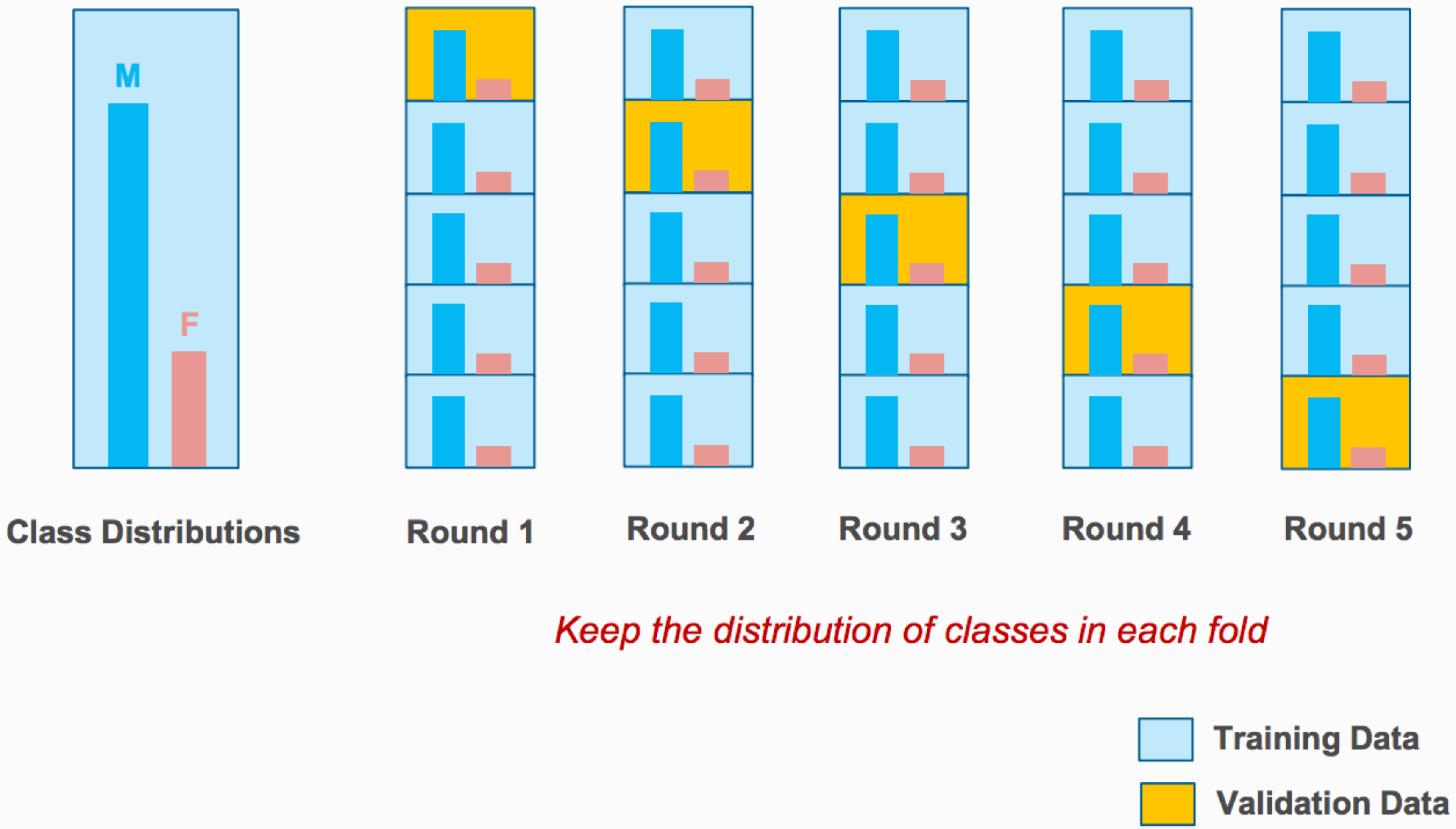
5-fold Cross-Validation



score(CV) = the average of evaluation scores from each fold
You can also repeat the process many times!



Stratified 5-foldCross-Validation

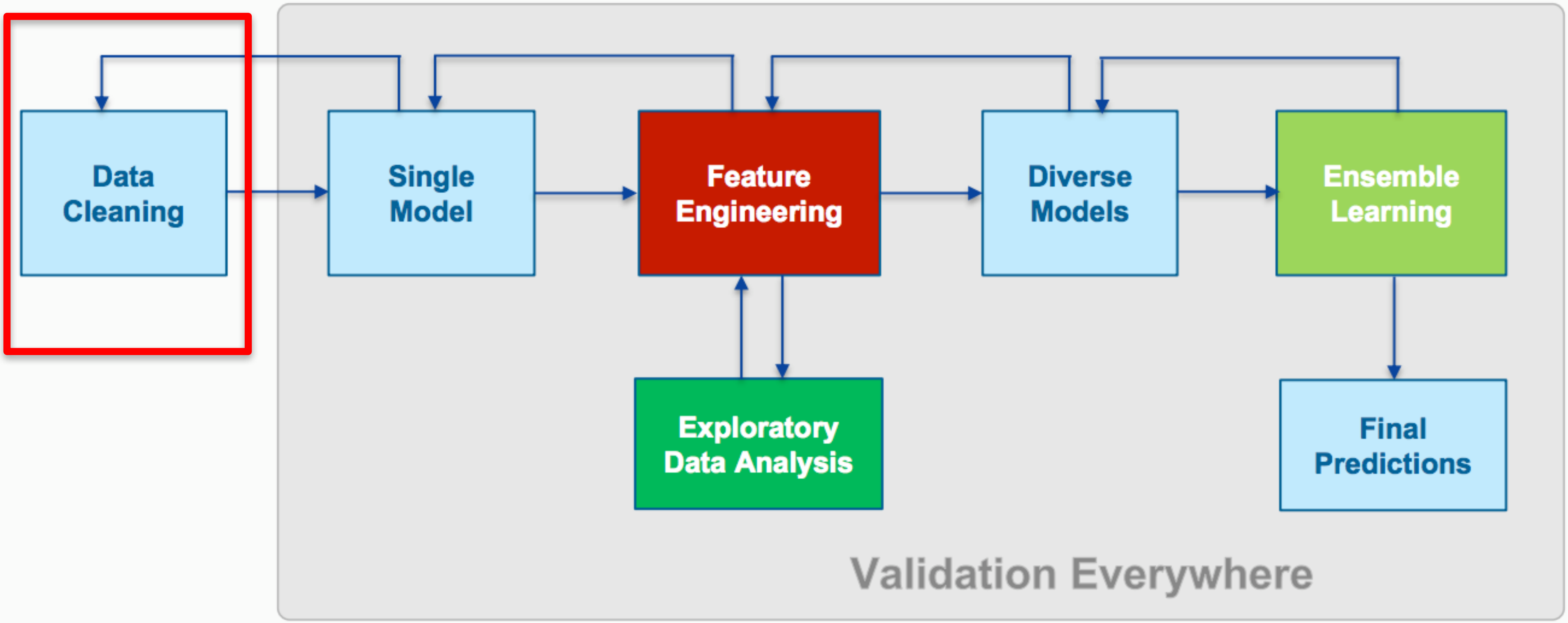


Keep the distribution of classes in each fold

Cross Validation tips

- It is normal to experience big shake up on private LB if not using local CV correctly
- 5 or 10 folds are not always the best choices (you need to consider the cost of computation time for training models)
  - Which K to choose?
    - Depends on your data
    - Mimic the ratio of training and testing in validation process
    - Find a K with lowest gap between local CV and public LB scores
    - Standard deviation of K-fold CV score matters more than mean:
      - To find balanced cross-validation: train the model, measure scores and select CV with smaller standard deviation
    - Stratified K-fold CV is important for imbalanced dataset, especially for classification problems.

Data cleaning



Data cleaning

- Data cleaning is the removal of duplicates, useless data, or fixing of missing data
- Reduce dimensional complexity of dataset
  - Make training faster without (significant) hurting the performance
- Apply imputation methods to help (hopefully) utilize incomplete rows
  - Incomplete rows may contain relevant features (don’t just drop them!)
  - In the risk of distorting original data, so be cautious!

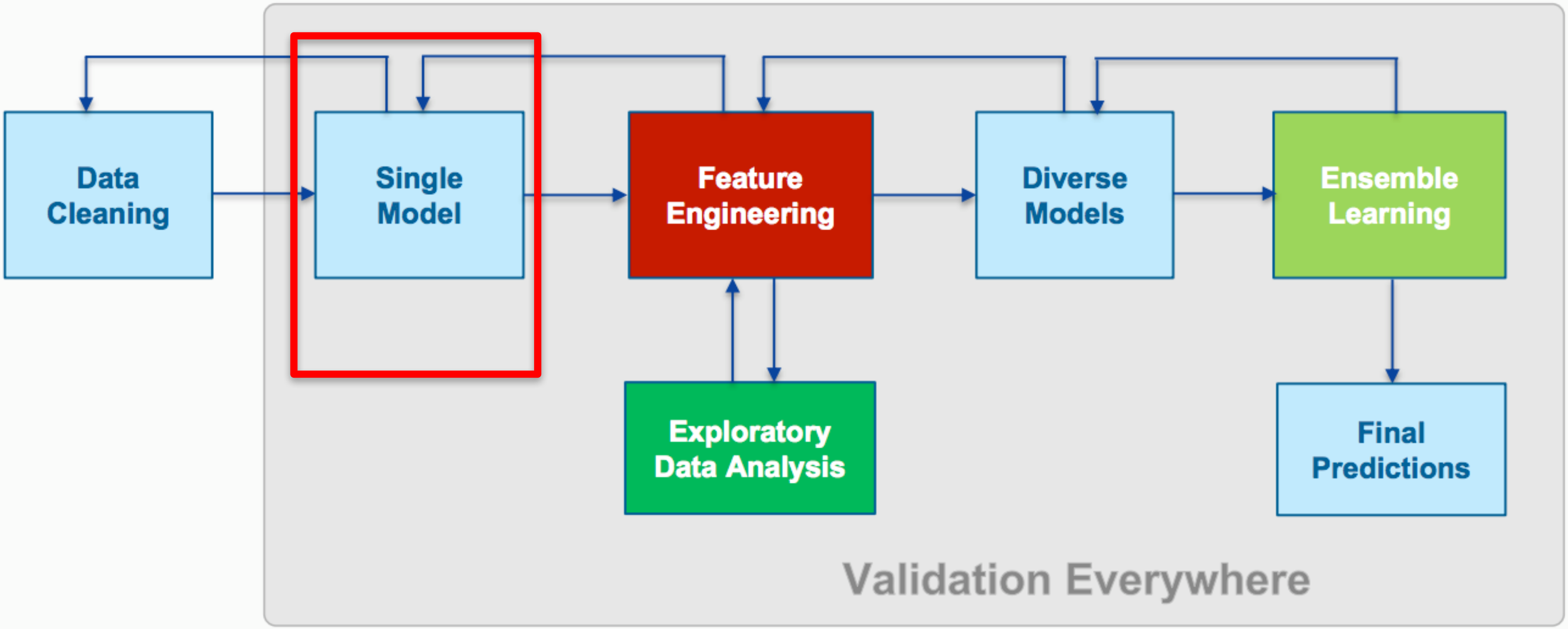
Data cleaning

- Remove duplicate features
- Columns having the same value distribution or variance
  - Only need to keep one of them
- Remove constant features
  - Columns with only one unique value
  - Remove features with near-zero variance
- Remove features with significant amount of missing values
  - Try to use it first in the models, of course

Data cleaning

- Some machine learning tools cannot accept NAs in the input
- Encode missing values to avoid Nas
- Binary features
  - -1 for negatives, 0 for missing values and 1 for positives
- Categorical features
  - Encode as an unique category
    - “Unknown”, “Missing”, ….
- Numeric features
  - Tree-based methods
    - Encode as a big positive or negative number
      - 999, -99999, ….
      - max(x) + 1, min(x)-1
  - Linear, neural nets, etc. methods
    - Encode by splitting into 2 columns:
      - Binary column isNA (0 if not and 1 if yes)
      - In original column replace NAs by mean or median

Building first model



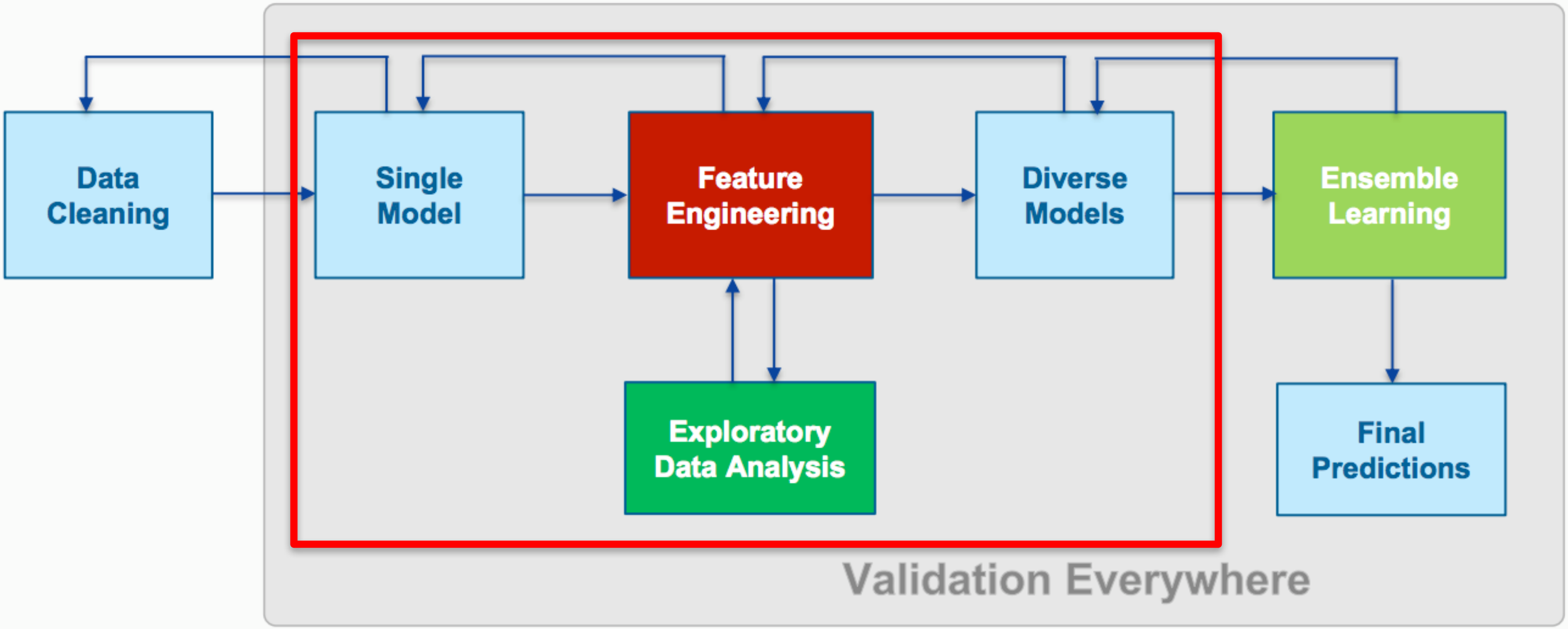
Mostly Used ML models

Model type	Name	Package
Regression	Linear Regression	sklearn.linear_model.LinearRegression
	Ridge Regression	sklearn.linear_model.Ridge
	Lasso Regression	sklearn.linear_model.Lasso
Instance-based	K Nearest Neighbors	sklearn.neighbors.KNeighborsClassifier sklearn.neighbors.KNeighborsRegressor
	Support Vector Machines (SVM)	sklearn.svm.SVC, sklearn.svm.SVR sklearn.svm.LinearSVC, sklearn.svm.LinearSVR
Hyperplane-based	Naive Bayes	sklearn.naive_bayes.GaussianNB sklearn.naive_bayes.MultinomialNB sklearn.naive_bayes.BernoulliNB
	Logistic Regression	sklearn.linear_model.LogisticRegression
Ensemble trees	Random Forests	H2O Distributed Random Forest
	Extremely Randomized Trees	sklearn.ensemble.ExtraTreesClassifier sklearn.ensemble.ExtraTreesRegressor
	Gradient Boosting Machines (GBM)	H2O GBM; Xgboost; LightGBM RGF

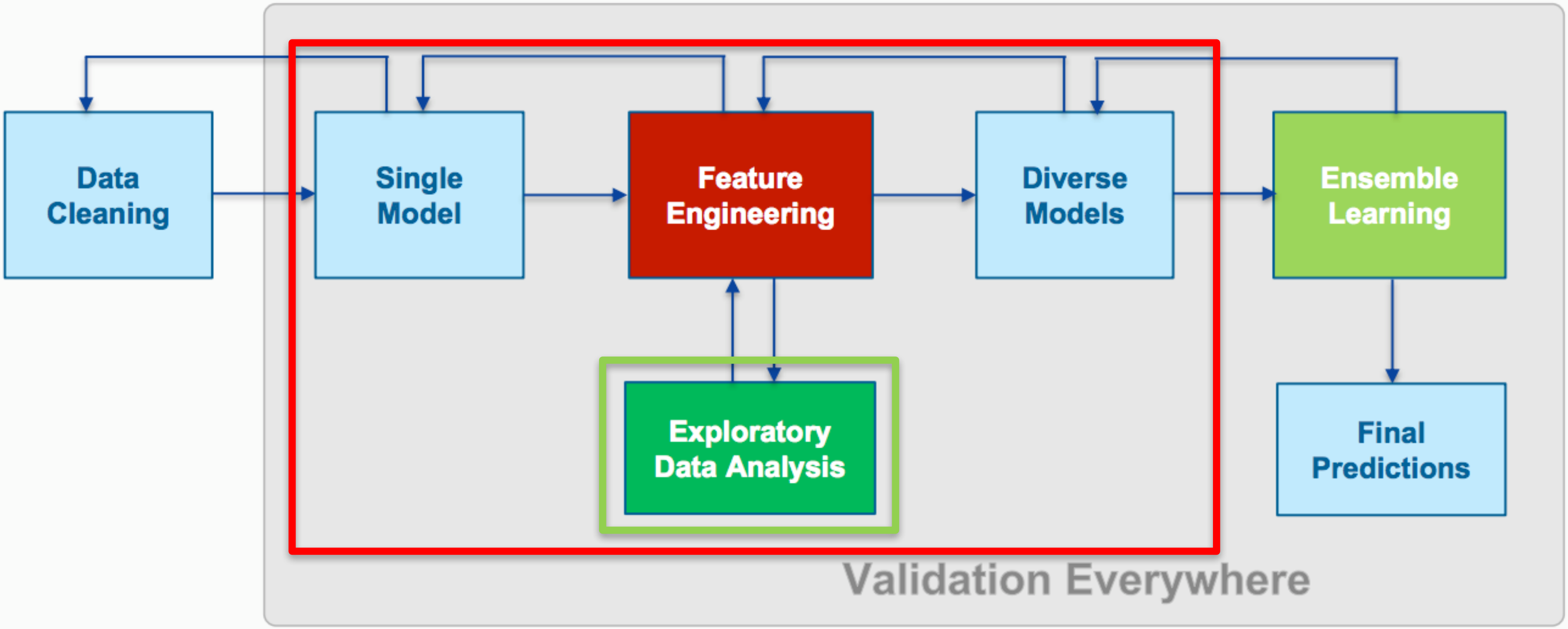
Mostly Used ML models cont'd

Model type	Name	Package
Neural Network	MLP CNN RNN	Keras MXNet
Recommendation	Matrix Factorization	sklearn.decomposition.NMF
	Factorization machines	pyFM
Clustering	K-Means	sklearn.cluster.MiniBatchKMeans H2O k-means (auto k!!!)
	(H)DBSCAN	sklearn.cluster.DBSCAN hdbscan
Dimensionality reduction	tSNE	tsne
	PCA	sklearn.decomposition.PCA
	Autoencoder	H2O autoencoder

Main cycle


Data Cleaning
Single Model
Feature Engineering
Diverse Models
Ensemble Learning
Exploratory Data Analysis
Final Predictions
Validation Everywhere

Exploratory data analysis



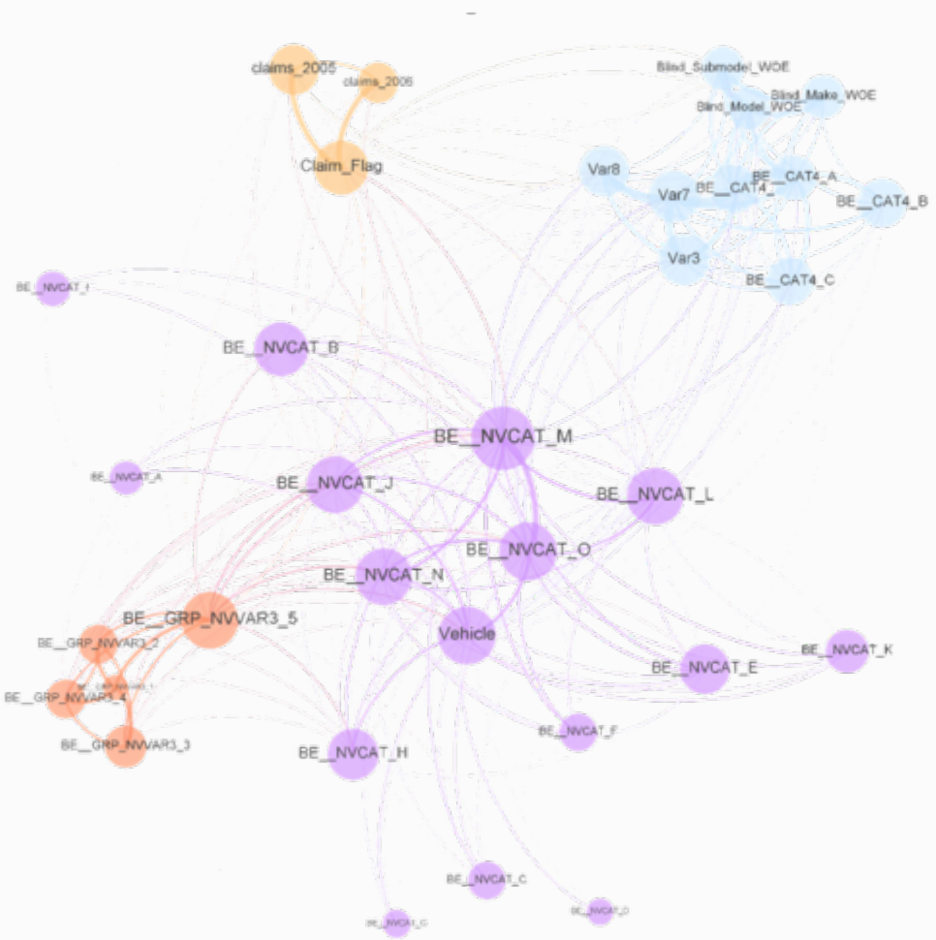
Correlation Graph

The nodes of this graph are the variables in a data set. The weights between the nodes are defined by the absolute value of their pairwise Pearson correlation.

To create:

- calculate Pearson correlation between columns/variables
- build undirected graph where each node is a column/variable
- connection weights between nodes are defined by Pearson correlation absolute values; weights below a certain threshold are not displayed
- node size is determined by number of connections (node degree)
- node color is determined by a graph communities calculation
- node position is defined by a graph force field algorithm

Free graph software: https://gephi.org/

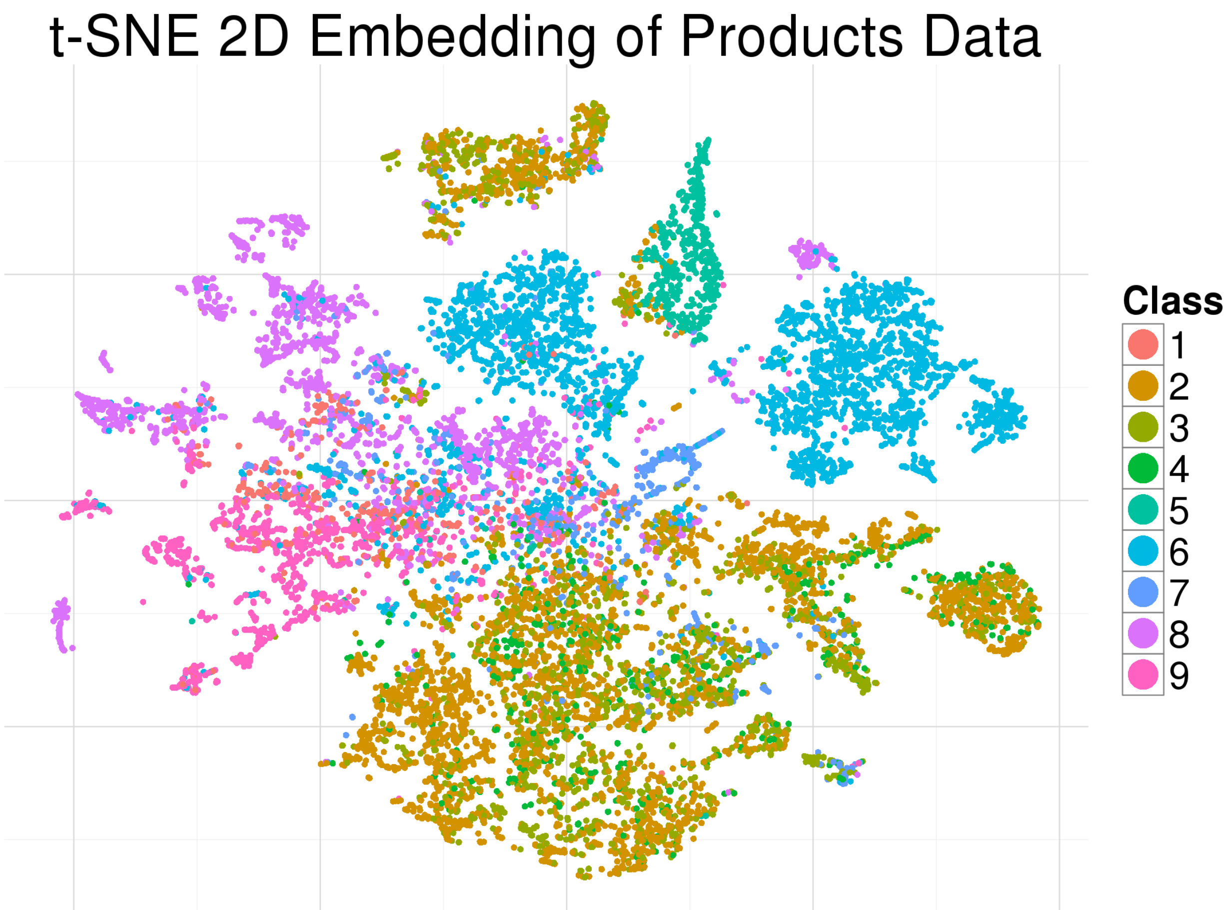


2-D projections

786 dimensions to 2 dimensions
with PCA

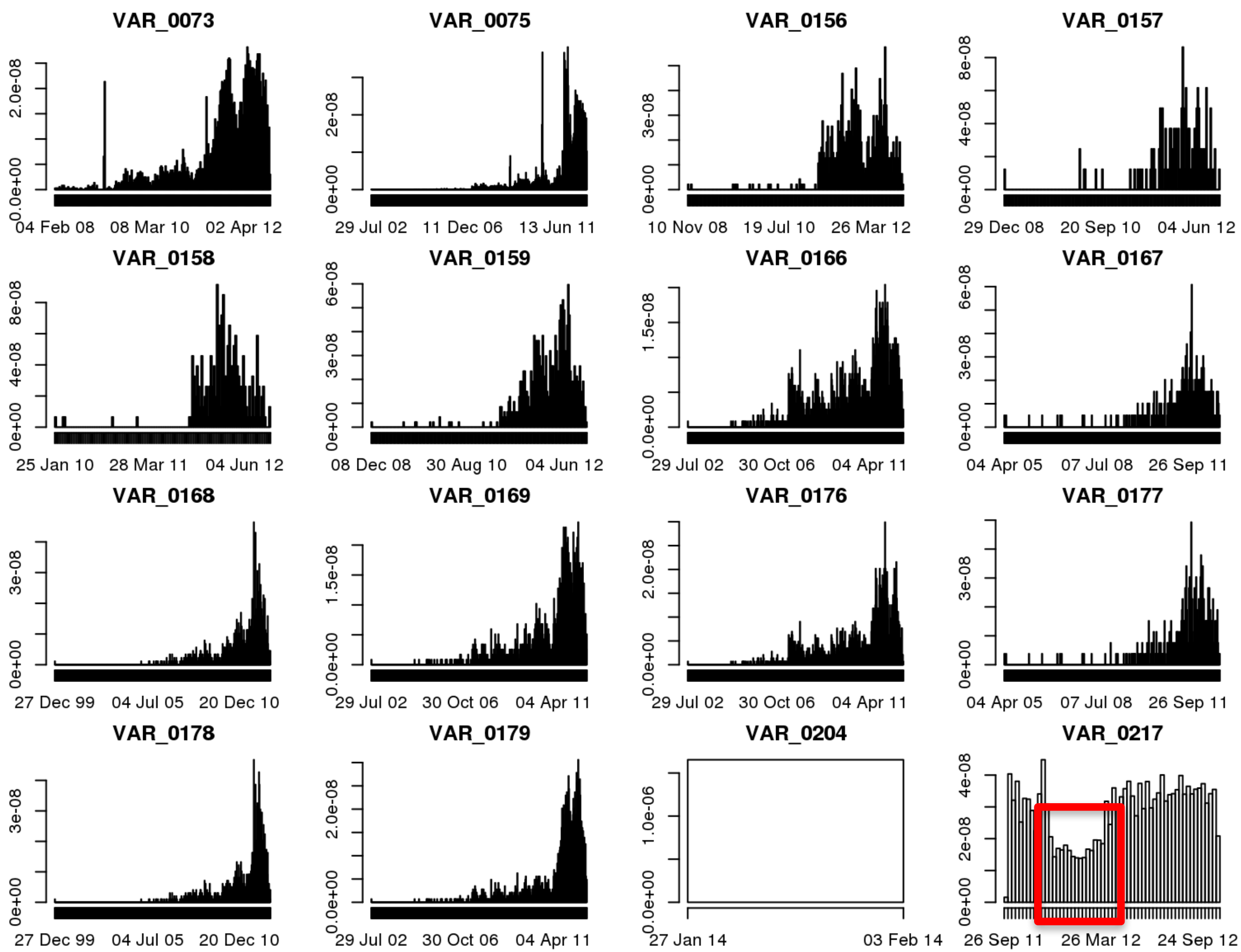
786 dimensions to 2 dimensions
with autoencoder

2-D projections



https://www.kaggle.com/benhamner/otto-group-product-classification-challenge/t-sne-visualization

Distributions

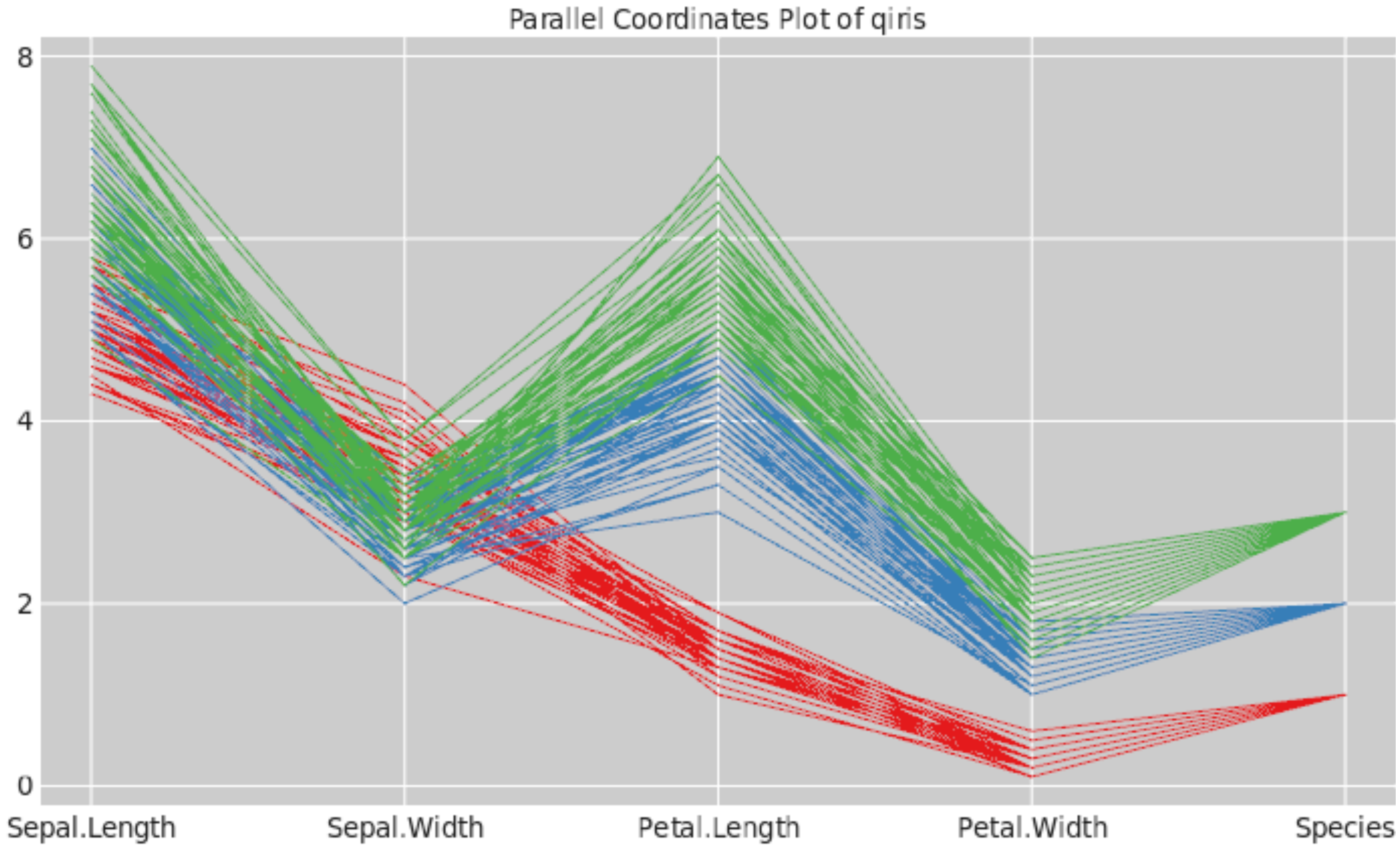


https://www.kaggle.com/darraghdog/springleaf-marketing-response/explore-springleaf/notebook

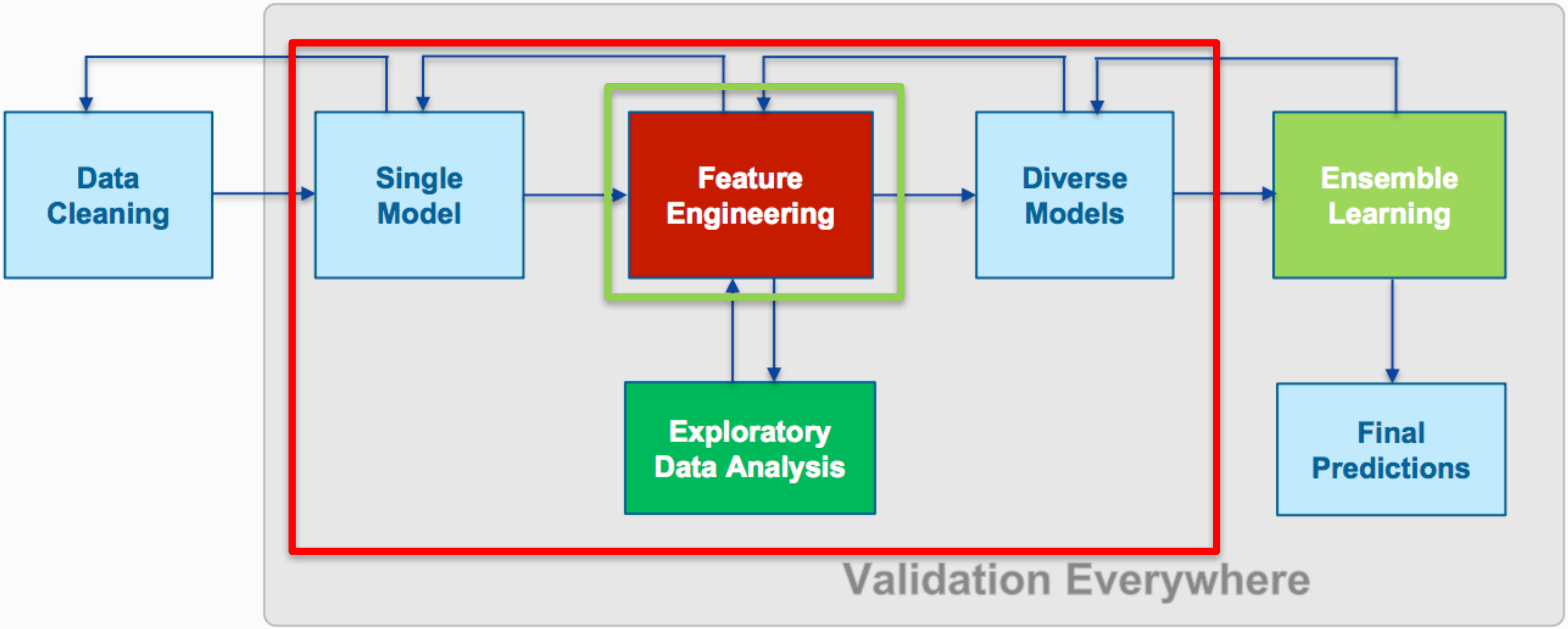
Pair plots

Fare
Age
ParentsAndChildren
SiblingsAndSpouses
Pclass
Survived
Died
Survived

Parallel coordinates plot



Feature Engineering



Feature Engineering

- Extract more new gold features, remove irrelevant or noisy features
  - Simpler models with better results
- The most important factor for the success of machine learning
- Key Elements
  - Data Transformation
  - Feature Encoding
  - Feature Extraction
  - Feature Selection

Data Transformation

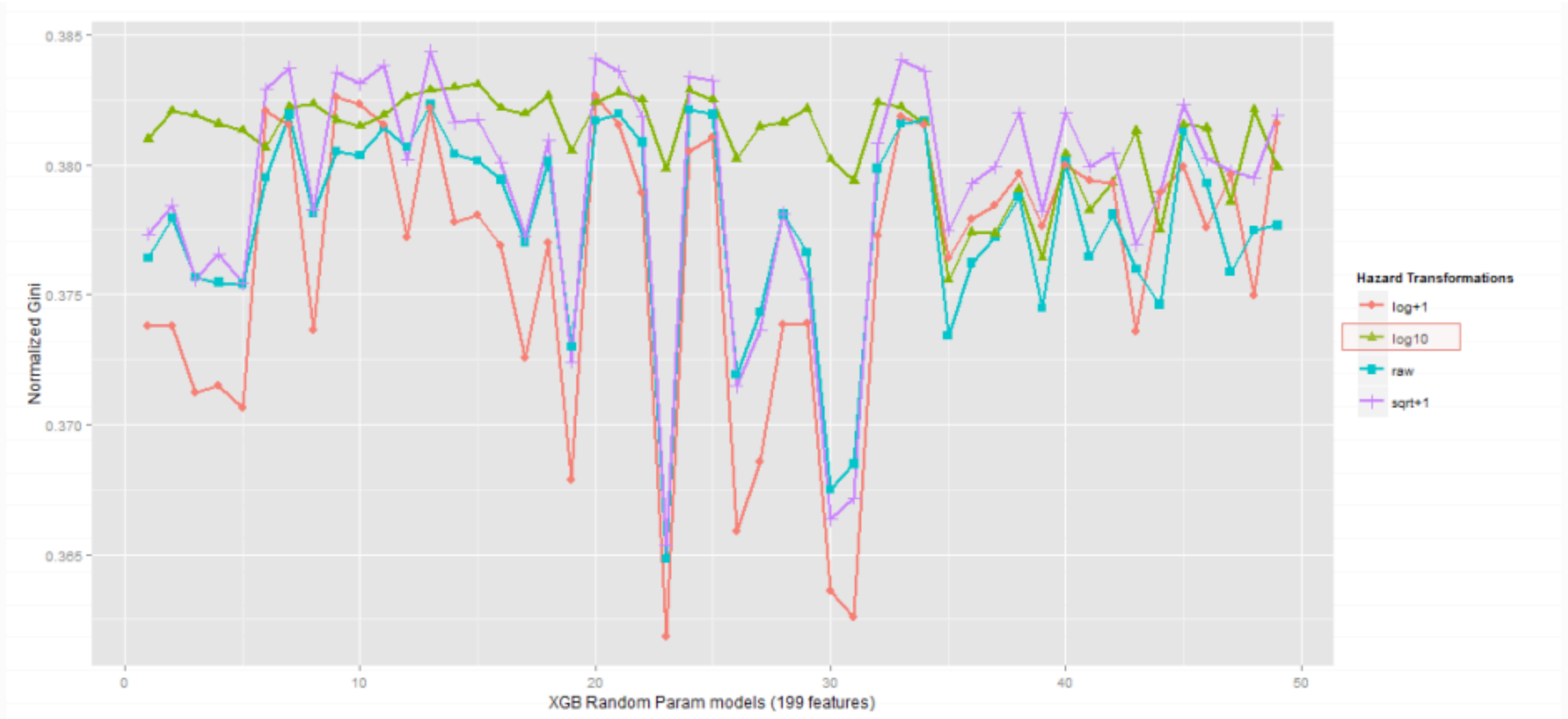
- Feature Scaling
  - Rescaling
    - Turn numeric features into the same scale (e.g., [-1,+1], [0,1], …)
    - Python scikit-learn: MinMaxScaler
  - Standardization
    - Removing the mean ($\mu = 0$) and scaling to unit variance ($\sigma = 1$)
    - Python scikit-learn: StandardScaler
  - To avoid features in greater numeric ranges dominating those in smaller numeric ranges
  - Critical for regularized linear models, KNN, SVM, K-Means, etc.
  - Can make a big difference to the performance of some models
  - Also speed up the training of gradient decent
  - But not necessary to do it for tree-based models

Data Transformation

- Predictor/Response Variable Transformation
  - Use it when variable shows a skewed distribution make the residuals more close to “normal distribution” (bell curve)
  - Can improve model fit
    - log(x), log(x+1), sqrt(x), sqrt(x+1), etc.

Variable Transformation

In Liberty Mutual Group: Property Inspection Prediction



Different transformations might lead to different results

Feature Encoding

- Turn categorical features into numeric features to provide more fine-grained information
  - Help explicitly capture non-linear relationships and interactions between the values of features
  - Some machine learning tools only accept numbers as their input
    - xgboost, gbm, glmnet, libsvm, liblinear, etc.

Feature Encoding

- Labeled Encoding
  - Interpret the categories as ordered integers (mostly wrong)
  - Python scikit-learn: LabelEncoder
  - Ok for tree-based methods
- One Hot Encoding
  - Transform categories into individual binary (0 or 1) features
  - Python scikit-learn: DictVectorizer, OneHotEncoder
  - Ok for K-means, Linear, NNs, etc.

Feature Encoding

- Target mean encoding:
  - Instead of dummy encoding categorical variables and increasing the number of features we can encode each level as the mean of the response.
  - To avoid overfitting it is better to calculate weighted average of the overall mean of the training set and the mean of the level:

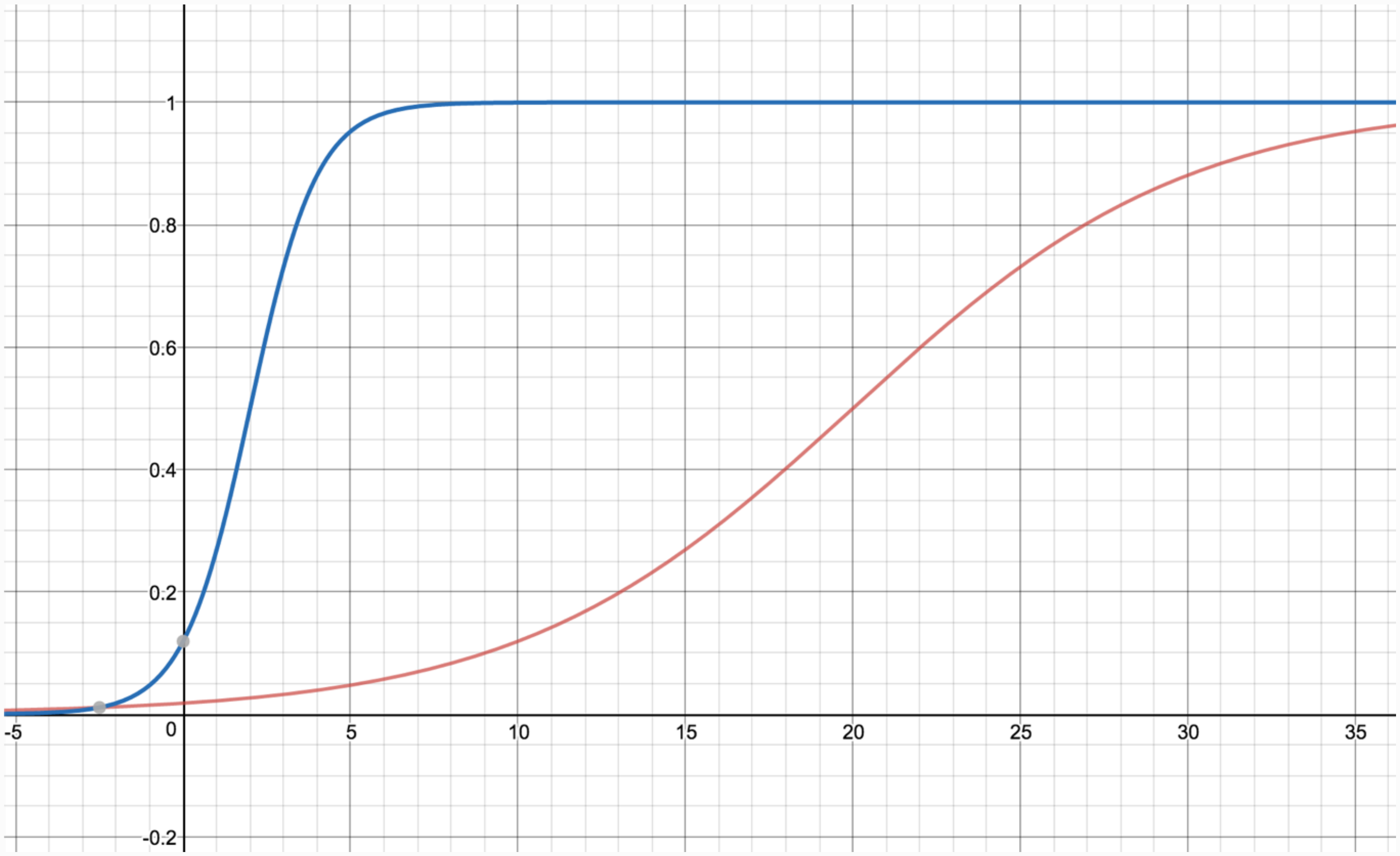
$$\lambda(n) * mean(level) + \big(1 - \lambda(n)\big) * mean(dataset)$$

  - The weights are based on the frequency of the levels i.e. if a category only appears a few times in the dataset then its encoded value will be close to the overall mean instead of the mean of that level.

Feature Encoding – target mean encoding cont'd

$$\frac{1}{1 + exp(\frac{-(x-k)}{f})}$$

x = frequency
k = inflection point
f = steepness



Feature Extraction: HTML Data

- HTML files are often used as the data source for classification problems
  - For instance, to identify whether a given html is an AD or not
- Possible features inside
  - html attributes (id, class, href, src, ...)
  - tag names
  - inner content
  - javascript function calls, variables, ...
  - script comment text
- Parsing Tools
  - BeautyfulSoup (Python), etc

Feature Extraction: Textual Data

- Bag-of-Words: extract tokens from text and use their occurrences (or TF/IDF weights) as features
- Require some NLP techniques to aggregate token counts more precisely
  - Split token into sub-tokens by delimiters or case changes
  - N-grams at word (often 2-5 grams) or character level
  - Stemming for English words
  - Remove stop words (not always necessary)
  - Convert all words to lower case
- Bag-of-Words Tools
  - Python: CountVectorizer, TfidfTransformer in scikit-learn package

Feature Extraction: Textual Data

- Deep Learning for textual data
  - Turn each token into a vector of predefined size
  - Help compute “semantic distance” between tokens/words
    - For example, the semantic distance between user query and product titles in search results (how relevant?)
  - Greatly reduce the number of text features used for training
    - Use average vector of all words in given text
    - Vector size: 100~300 is often enough
  - Tools
    - Word2vec, Doc2vec, GloVe

Feature Extraction

- There usually have some meaningful features inside existing features, you need to extract them manually
- Again you can use counts as features
- Some examples
    - Location
        - Address, city, state and zip code .... (categorical or numeric)
    - Time
        - Year, month, day, hour, minute, time ranges, ... (numeric)
        - Weekdays or weekend (binary)
        - Morning, noon, afternoon, evening, ... (categorical)
    - Numbers
        - Turn age numbers into ranges (ordinal or categorical)

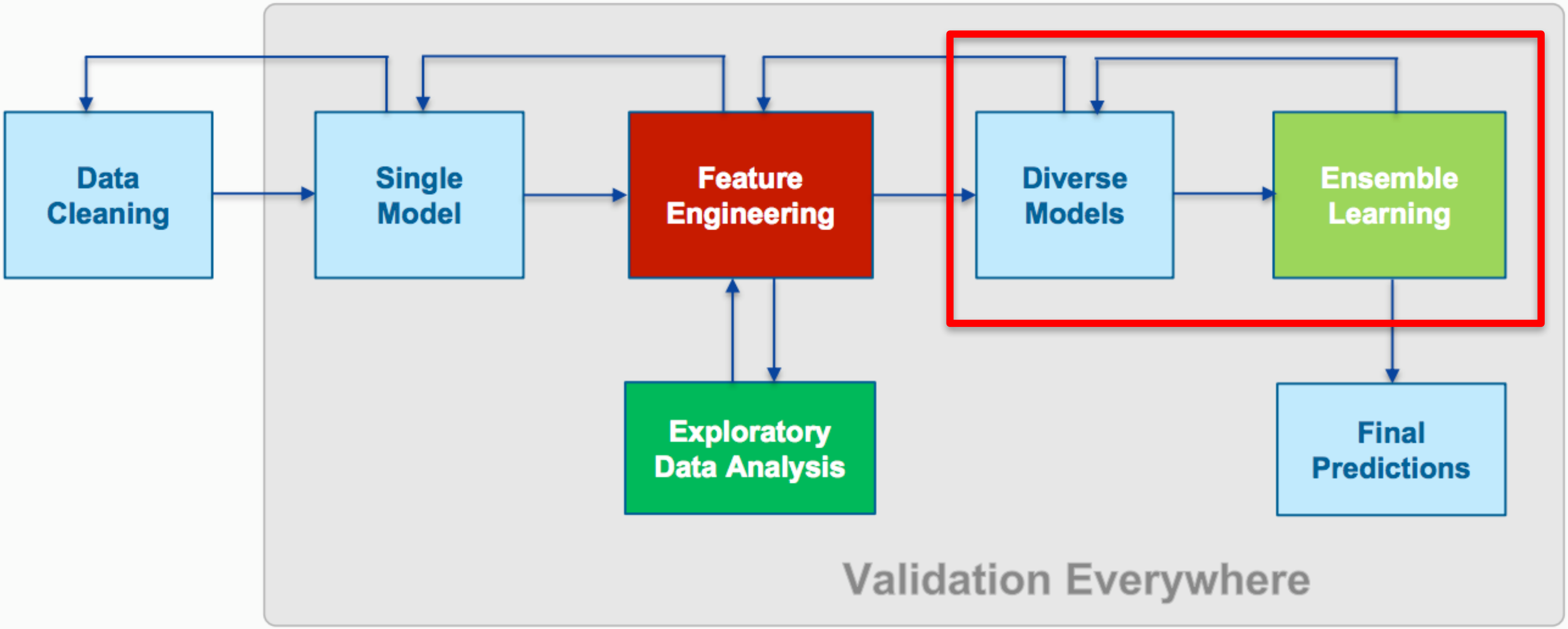
Feature Selection

- Reduce the number of features by removing redundant, irrelevant or noisy features
- Feature Explosion after Feature Extraction!
  - More than 100,000 unique token features from textual data is common
  - Hard to put all of them in memory!
  - PCA or truncated SVD can help to select top-N informative features
    - With the risk of ignoring non-linear relationships between features and dropping important features
    - Use them only if there is no other choice

Feature Selection

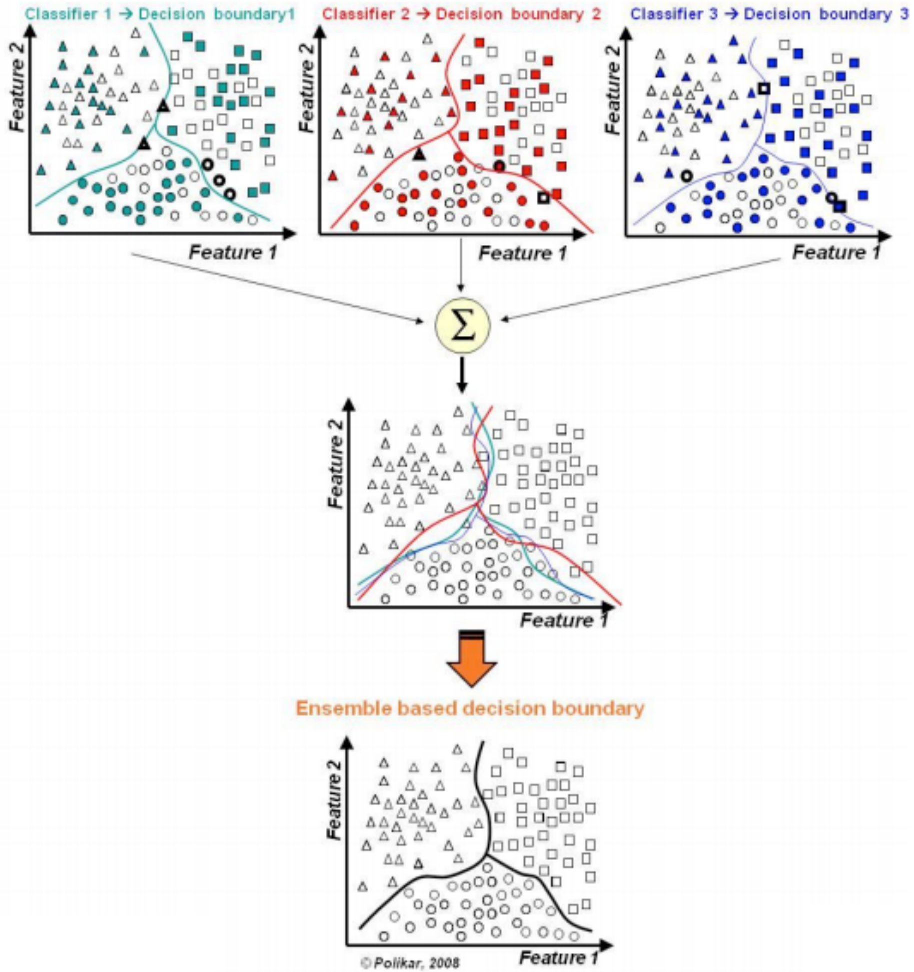
Type	Name	Python
Feature Importance Ranking	Gini Impurity	sklearn.ensemble.RandomForestClassifier sklearn.ensemble.RandomForestRegressor sklearn.ensemble.GradientBoostingClassifier sklearn.ensemble.GradientBoostingRegressor
	Chi-square	sklearn.feature_selection.chi2
	Correlation	scipy.stats.pearsonr scipy.stats.spearmanr
	Information Gain	sklearn.ensemble.RandomForestClassifier sklearn.ensemble.RandomForestRegressor sklearn.ensemble.GradientBoostingClassifier sklearn.ensemble.GradientBoostingRegressor xgboost
	L1-based Non-zero Coefficients	sklearn.linear_model.Lasso sklearn.linear_model.LogisticRegression
Feature Subset Selection	Recursive Feature Elimination (RFE)	sklearn.feature_selection.RFE
	Boruta Feature Selection	Boruta

Feature Engineering



Ensemble Learning - Illustration

- Assume that diverse models see different aspects of the problem space and are wrong in different ways
- Simplest way: the average (possibly weighted) of model predictions
  - Less variance
  - Less generalization error
  - Less chance of overfitting
  - Better chance to win!



Ensemble Learning - Blending

- Simplest way: Weighted average:

$$f(x) = \alpha * f_1(x) + \beta * f_2(x) + \gamma * f_3(x)$$

- Not so simple way:

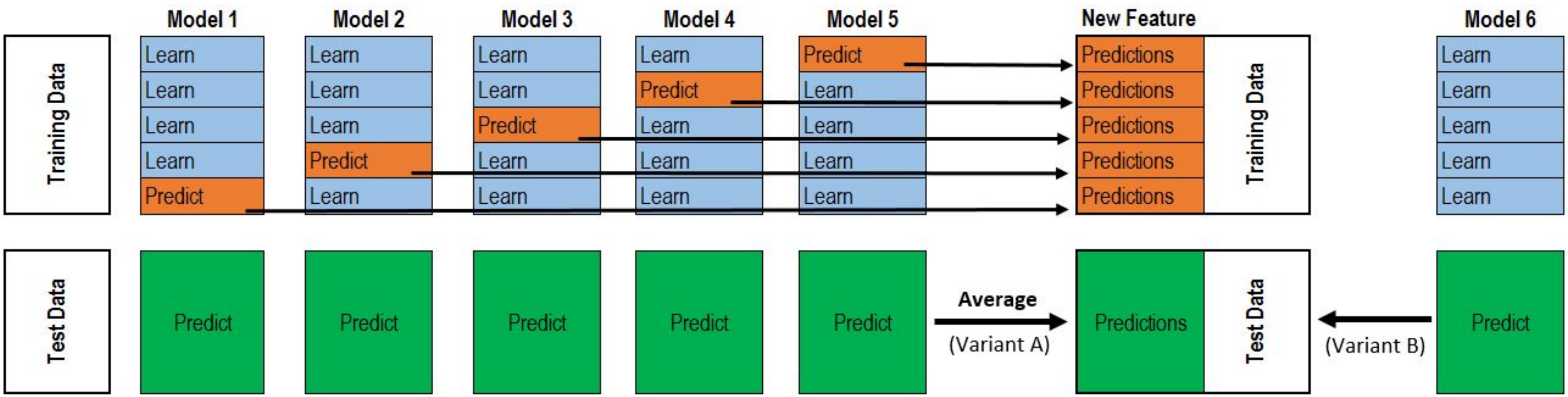
$$f(x) = \alpha_1 * f_1(x)^{\alpha_2} + \beta_1 * f_2(x)^{\beta_2} + \gamma_1 * f_3(x)^{\gamma_2}$$

- And everything in between
- How to find coefficients?
  - By using any optimization package:
    - hyperopt
    - scipy.optimize.fmin

Stacking

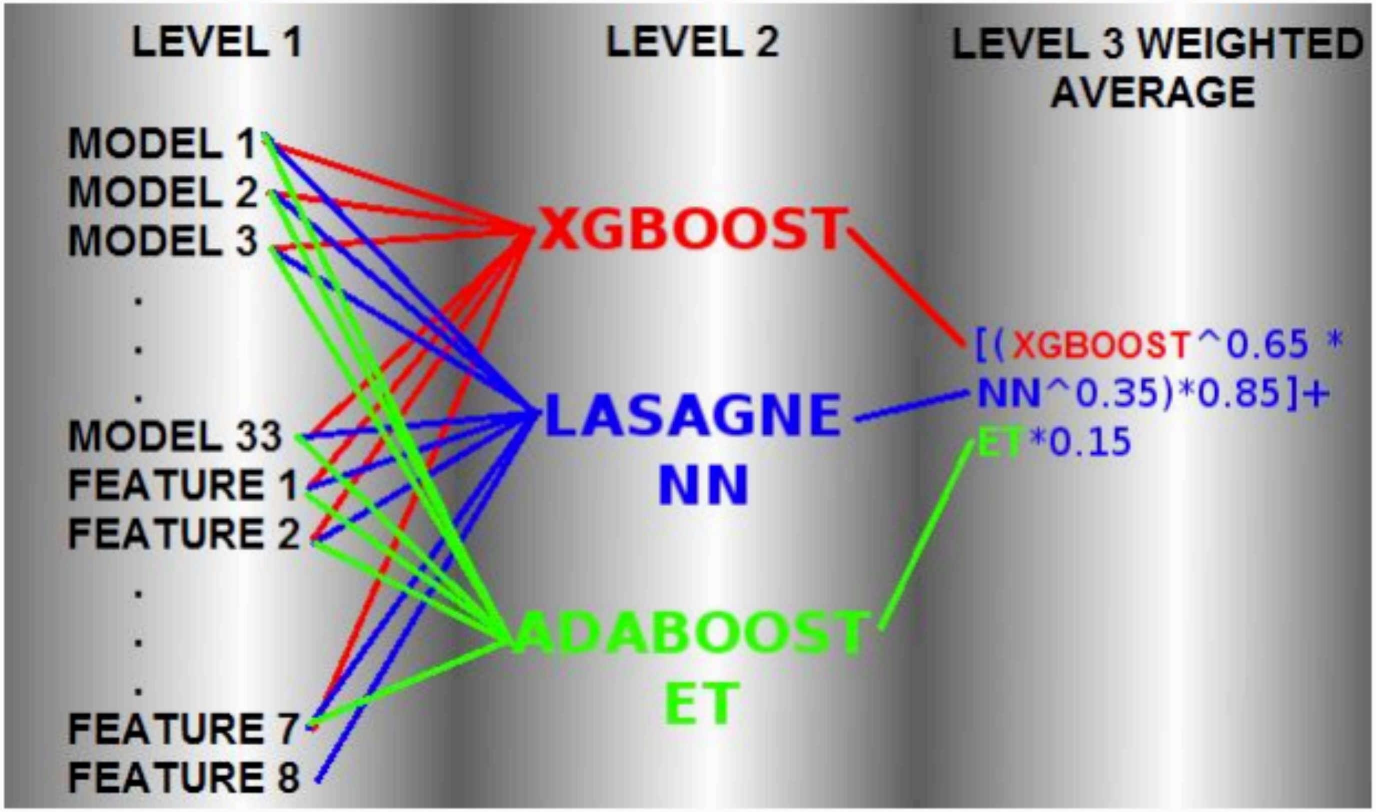
- Using the predictions of current level models as features for training next level models
- What we called “out-of-fold predictions” or “meta-features”
  - Usually we only build 2 levels of models, 4 levels at most
  - Can combine meta-features with original feature sets for training
  - Potential risk of overfitting if cross validation process is not correct
- Also called “Stacking”

Stacking



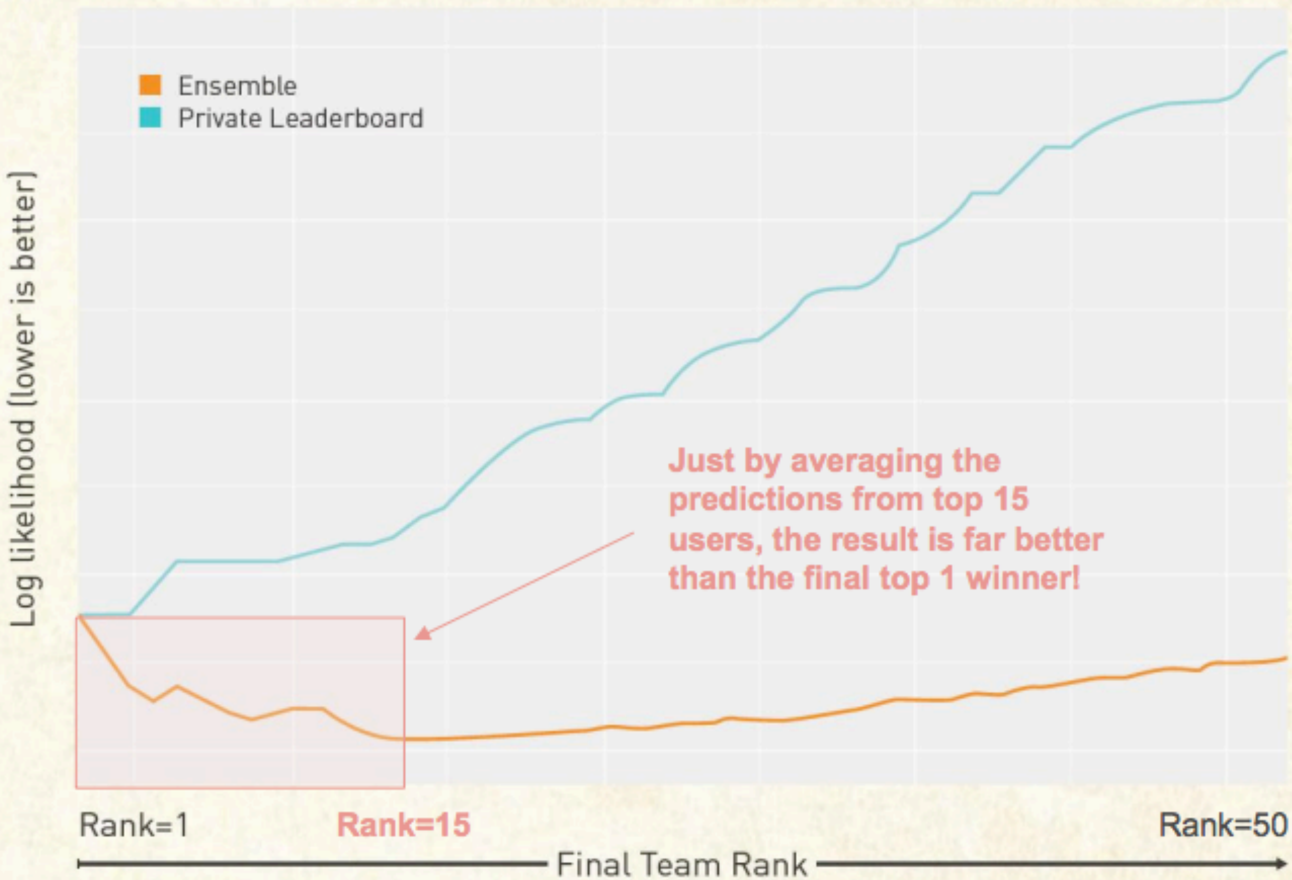
Model 1
Model 2
Model 3
Model 4
Model 5
New Feature
Model 6
Training Data
Learn
Predict
Predictions
Test Data
Predict
Average
(Variant A)
Predictions
(Variant B)

Stacked Generalization
Used by Gilberto Titericz Junior (Kaggle #1)
in Otto Product Classification Challenge



Reference: https://www.kaggle.com/c/otto-group-product-classification-challenge/forums/t/14335/1st-place-winner-solution-gilberto-titericz-stanislav-semenov

The Power of Ensemble Learning



Value of Ensembling for the Kaggle Photo Quality Prediction Competition
(results courtesy of Ben Hamner, Kaggle CTO)

Reference: Booz Allen Hamilton. (2015). "*Field Guide to Data Science*," page 93.

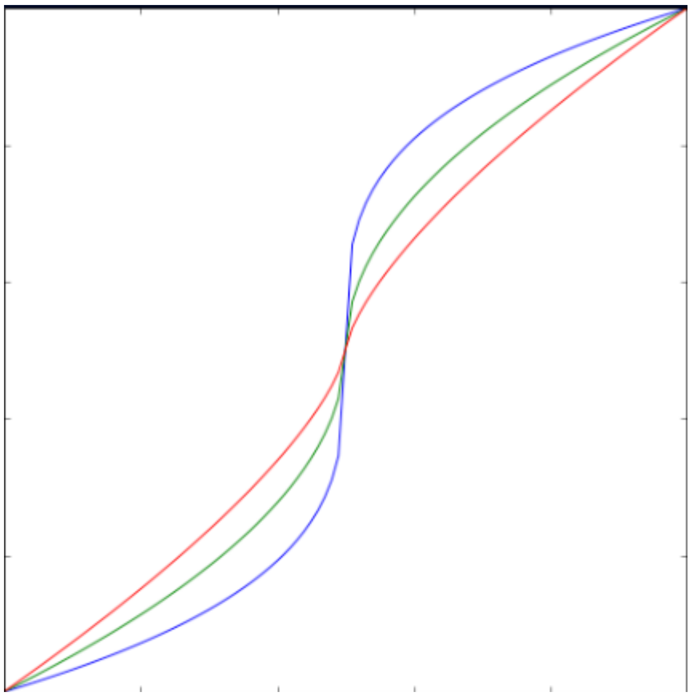
Ensemble Learning - Tips

- Always start from simple blending first
- Even simple averaging of multiple models of the same type can help reduce variance
- Generally work well with less-correlated good models
  - Models with similar performance but their correlation lies in between 0.85~0.95 (that's ideal)
  - Gradient Boosting Machines usually blend well with Neural Nets and Linear models
- Must Read: Kaggle Ensembling Guide by MLWave

Teaming strategy

- Usually we agree to team up at pretty early phase but keep working independently to avoid any bias in our solutions
- After teaming up:
  - Code sharing can help you to learn new things but a waste of time from competition perspective;
  - Share data – especially some engineered features;
  - Combine your models' outcomes using k-fold stacking and build a second level meta-learner (stacking)
  - Continue iteratively add new features and building new models
- Teaming up right before competition end:
  - "Black-box" ensembling – linear and not so linear blending using LB as validation set.
    - Linear: $f(x) = \alpha * f_1(x) + \beta * f_2(x) + \gamma * f_3(x)$
    - Non linear: $f(x) = \alpha_1 * f_1(x)^{\alpha_2} + \beta_1 * f_2(x)^{\beta_2} + \gamma_1 * f_3(x)^{\gamma_2}$

Metrics tips



- Depending on the competition metric you can do some tricks as well.
  - Logloss:
    - $-\frac{1}{N}\sum_{i=1}^{N}\sum_{j=1}^{M} y_{ij}\log(p_{ij})$
      – where N is the number of examples, M is the number of classes, and y_{ij} is a binary variable indicating whether class j was correct for example i.
  - Calibrate result by using formula:
    - $y_{new} = 0.5 * \left(\left(2 * abs(y - 0.5)\right)^{beta}\right) * sign(y - 0.5) + 0.5$
    - *beta* coefficient can be found by using LB feedback or CV
  - Using geometric mean instead average

Metrics tips

- MAE:
  - $\frac{1}{N}\sum_{i=1}^{N}|y_i - y_i'|$
    - where N is the number of examples
- Most of the ML algorithms optimize RMSE instead of MAE, so small adjustment can be useful:
  - $y_{new} = \alpha * y + \beta$
- Also, using median instead of averaging (mean) might be beneficial

Metrics tips

- RMSLE:

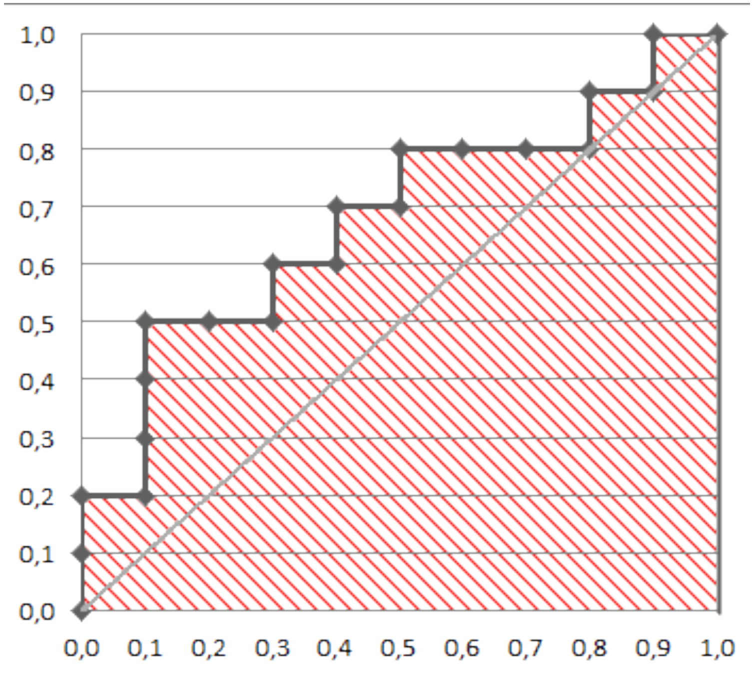
$$\epsilon = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(\log(p_i + 1) - \log(a_i + 1))^2}$$

  - where *n* is the number of examples

- Most of the ML algorithms optimize RMSE, so log-transforming of target variable will make it to optimize RMSLE instead:
  - $y_{new} = \log(y + 1)$
- Don’t forget to transform it back:
  - $y_{old} = exp(y_{new}) - 1$

Metrics tips

- AUC: Area Under ROC Curve, total area is 100%:
  - so AUC = 1 is for a perfect classifier for which all positive come after all negatives
  - AUC = 0.5 - randomly ordered
  - AUC = 0 - all negative come before all positive
  - so AUC $\in [0,1] \in [0,1]$
  - typically we don't have classifiers with AUC < 0.5



- AUC is sensitive to order (ranking), not to specific values:
  - $y_{new} = \frac{rank(y)}{N}$; N – number of rows
  - by this transformation we can blend in ensemble any classifiers, no matter how different they distributions were

Summary

- Know your tools
- Know your data
- Know your metric
- Right validation schema is a key
- Never stick to favorite models only, try all of them
  - Always include linear models into ensemble
  - … and Neural Nets
  - … and kNN
- Good features beat ensemble
  - My team got 10th place on BNP Paribas Competition with ensemble of 120 models
  - Branden Murray's single model would be on 8th (his team placed 4th)

Thank you!

Q & A