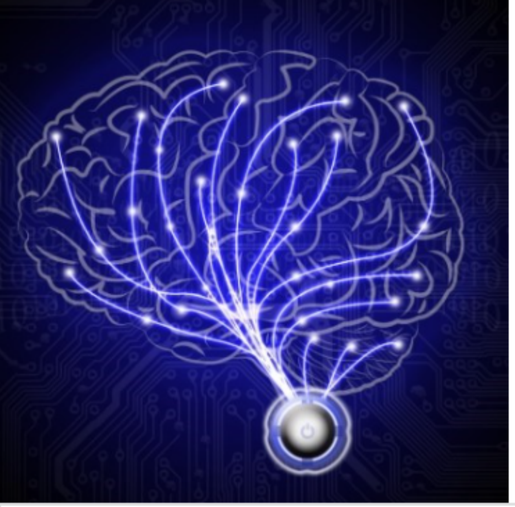
THE TWOSIGMA CONNECT: RENTAL LISTING INQUIRIES

HOW TO WIN A KAGGLE COMPETITION?



ChrisCC

Calgary, AB, Canada

Joined 2 years ago · last seen in the past day

in

Followers 4

Competitions Master

Home | Competitions (14) | Kernels (31) | Discussion (31) | Followers (4) | Contact User | Follow User

Competitions Summary

Competitions Master

Current Rank	Highest Rank
515 of 64,847	309

1	2	2

Competitions: 14
Solo: 14 (100%)
Team: 0

Data Scientist@Shaw

Chris Chen

Top 1% Kaggle Master

KAGGLE?

Your Home for Data Science

jobs board >

Kaggle helps you learn, work, and play

Create an account or Host a competition

Largest and most influencing data science community

Battlefield for talented data scientists from 190 countries

Cradle of pioneering algorithms and approaches

TOOLS?

dmlc
XGBoost

dplyr

pandas

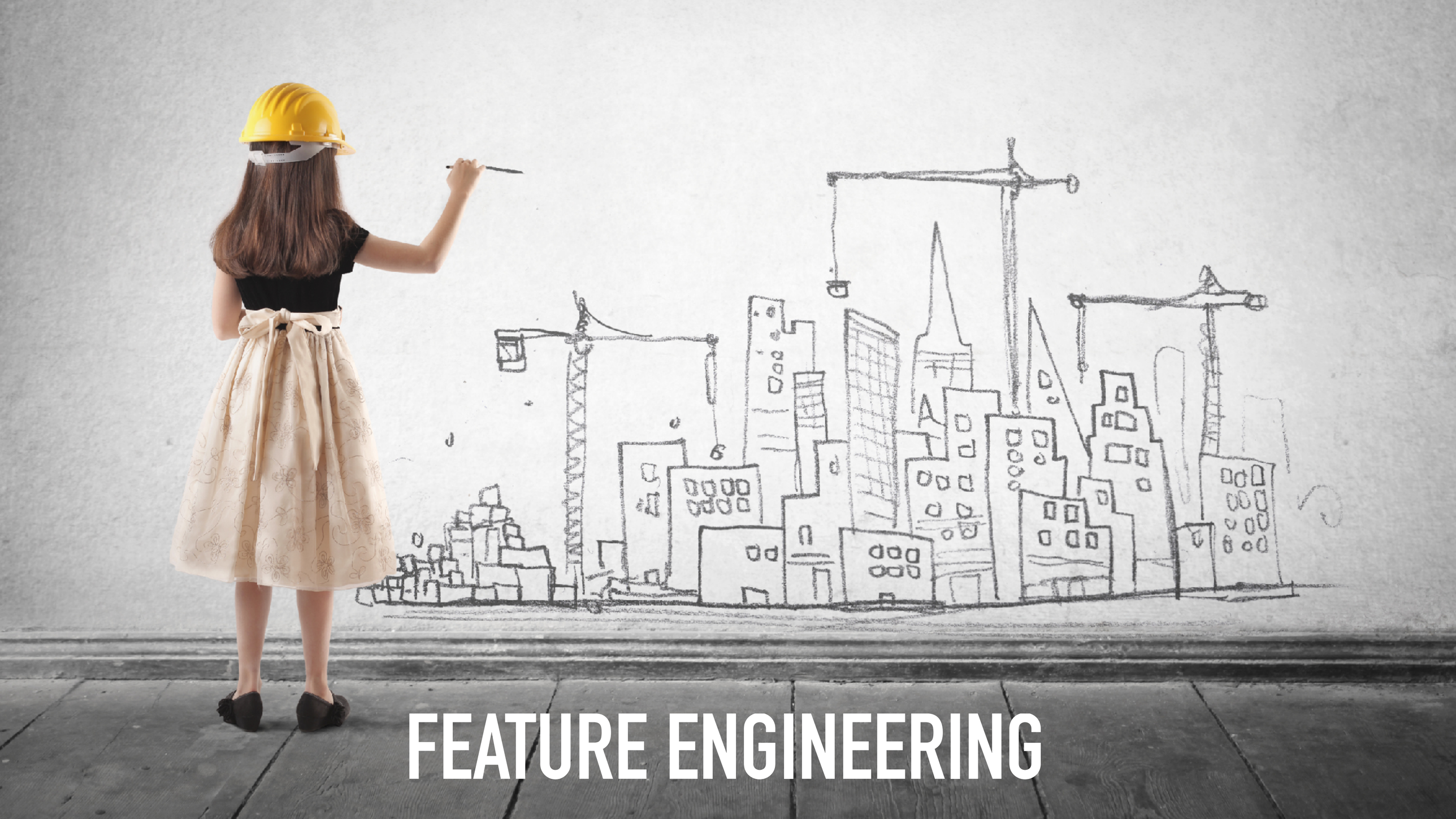
$$y_{it} = \beta' x_{it} + \mu_i + \epsilon_{it}$$

dmlc
mxnet

WHAT WE WILL BE USING

- Anaconda - Python distribution for Data Scientists
  - scikit-learn
  - scipy
  - numpy
  - jupyter notebook
- Machine Learning
  - XGBoost
  - LightGBM
  - Tensorflow + Keras
  - Bayesian Optimization
- Data visualization tools
  - seaborn
  - bokeh

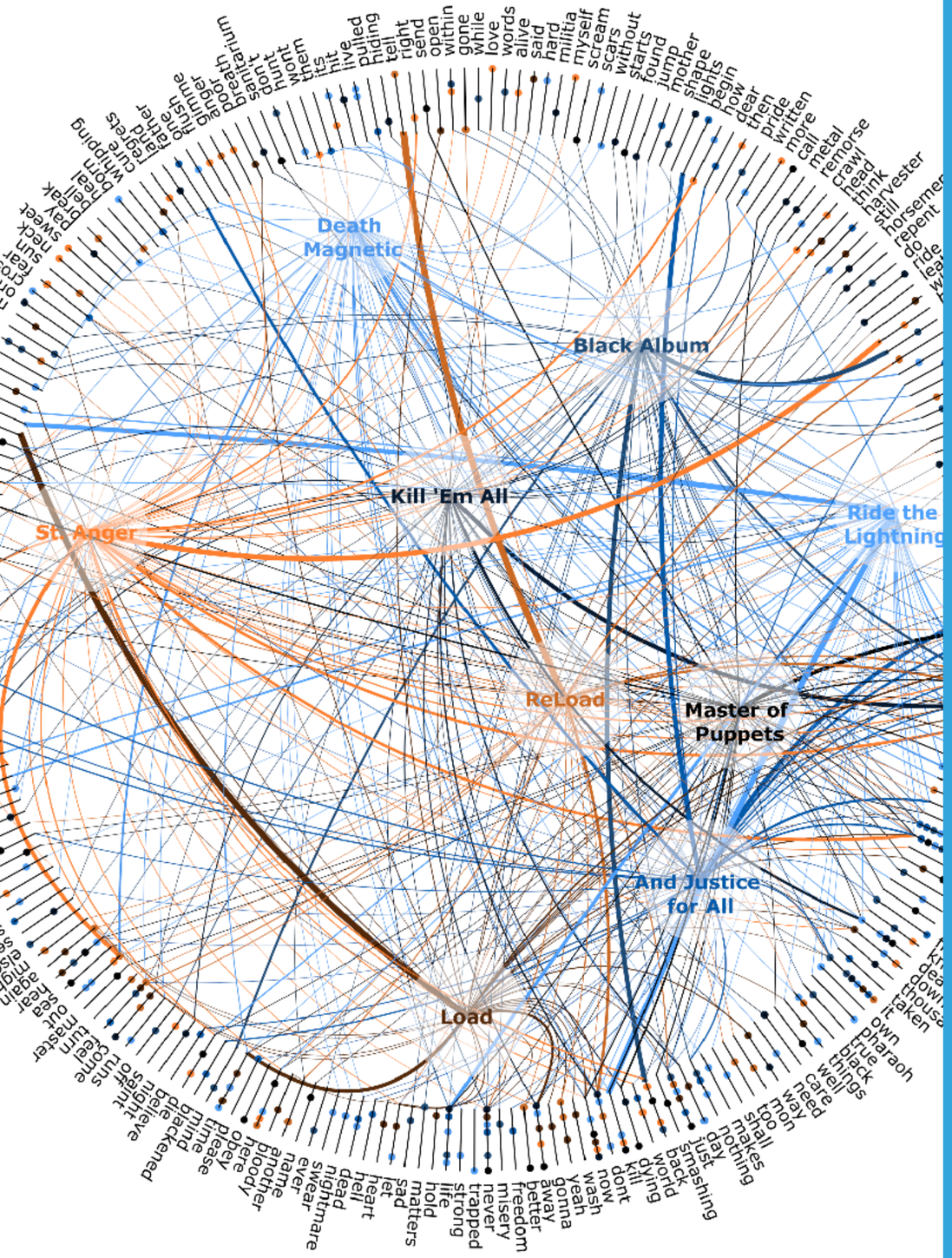
HOW?



FEATURE ENGINEERING

PARAMETER TUNING

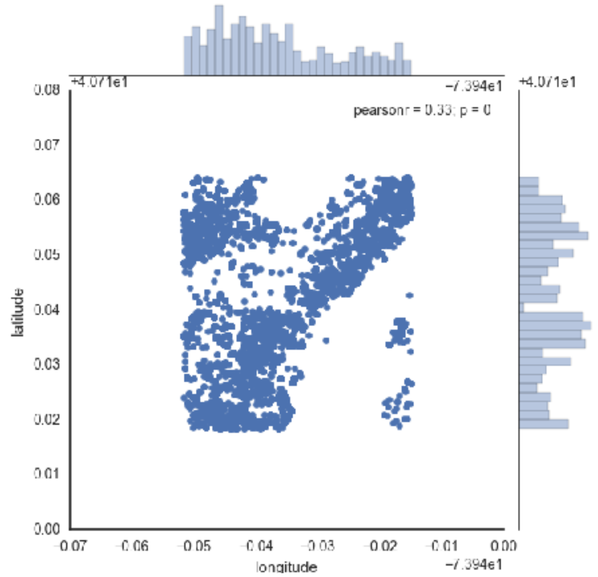

MODEL ENSEMBLE

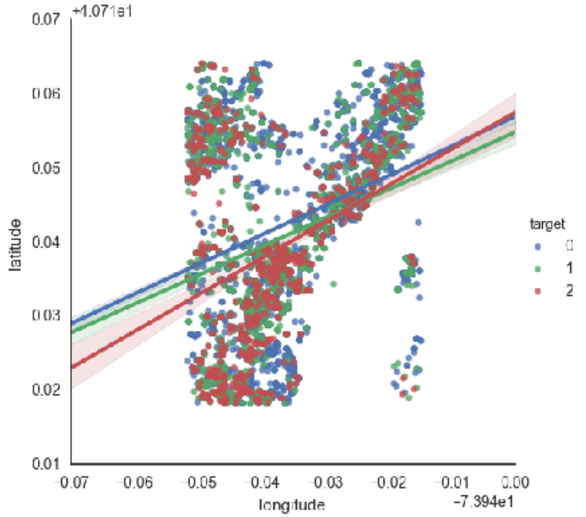


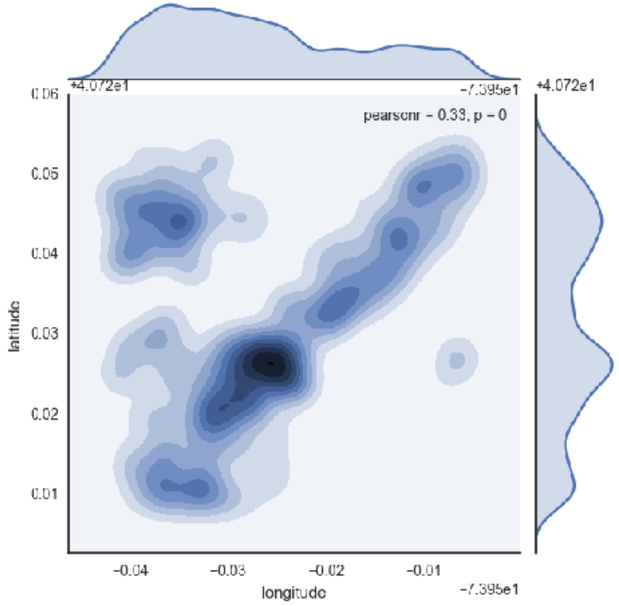
EDA (EXPLORATORY DATA ANALYSIS)

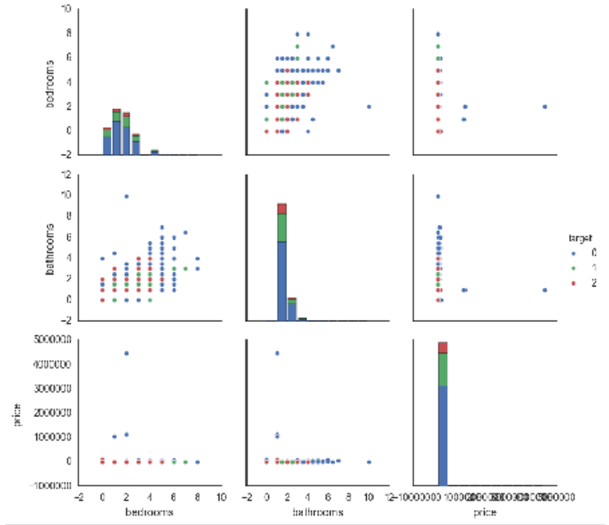
- Maximize insights
- Uncover hidden patterns
- Extract important features
- Detect outliers and anomalies
- Validate underlying assumptions

CONTINUOUS FEATURES

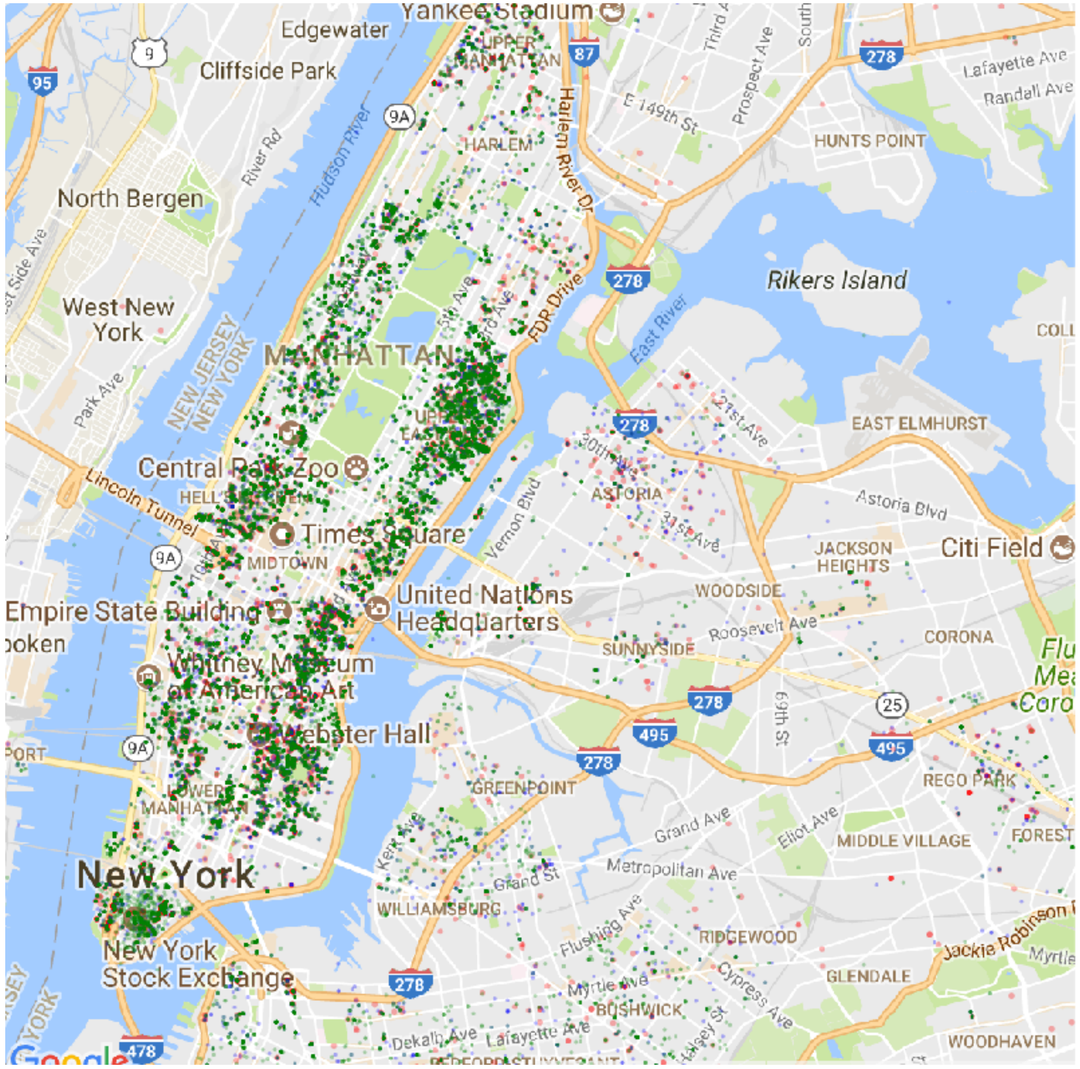








GEOLOCATION EDA



GMapPlot:

- Plot data over a Google Map
- Google API key required

```
from bokeh.io import output_file, show
from bokeh.models import (
  GMapPlot, GMapOptions, ColumnDataSource,
    Circle, DataRangeId
)
map_options = GMapOptions(lat=40.7145,  lng=-73.9425,
                          map_type="roadmap", zoom=11)
plot = GMapPlot(
    x_range=DataRangeId(), y_range=DataRangeId(),
    map_options=map_options
)
plot.title.text = "Newyork"
plot.api_key = "YOURAPIKEY"
source = ColumnDataSource(
    data=dict(
        lat=full_data["latitude"].tolist(),
        lon=full_data["longitude"].tolist()
    )
)
circle = Circle(x="lon", y="lat", size=5, fill_color="red")
plot.add_glyph(source, circle)
show(plot)
```

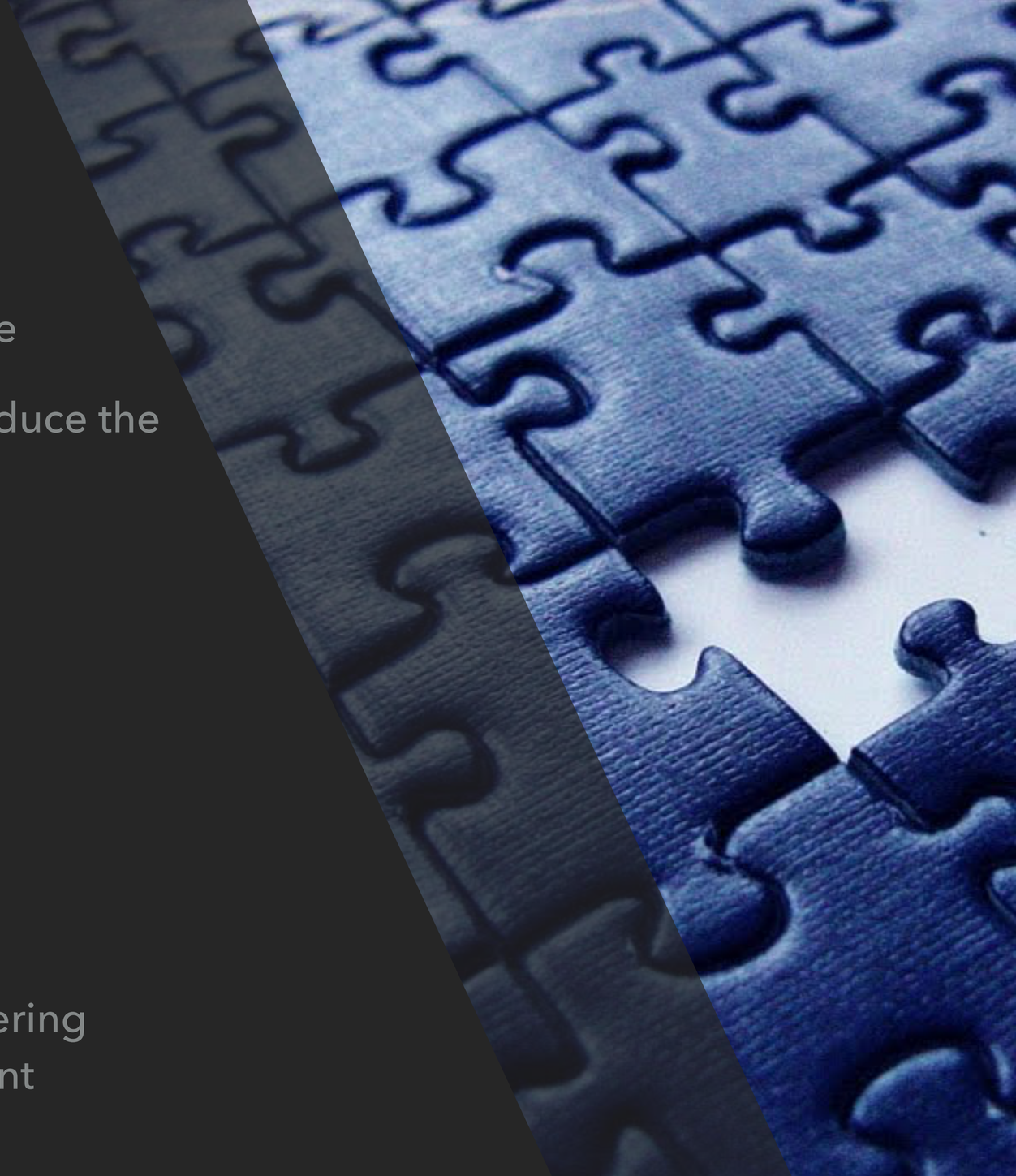
FEATURE ENGINEERING

COMING UP WITH FEATURES IS DIFFICULT, TIME-CONSUMING, REQUIRES EXPERT KNOWLEDGE. "APPLIED MACHINE LEARNING" IS BASICALLY FEATURE ENGINEERING.

Andrew Ng

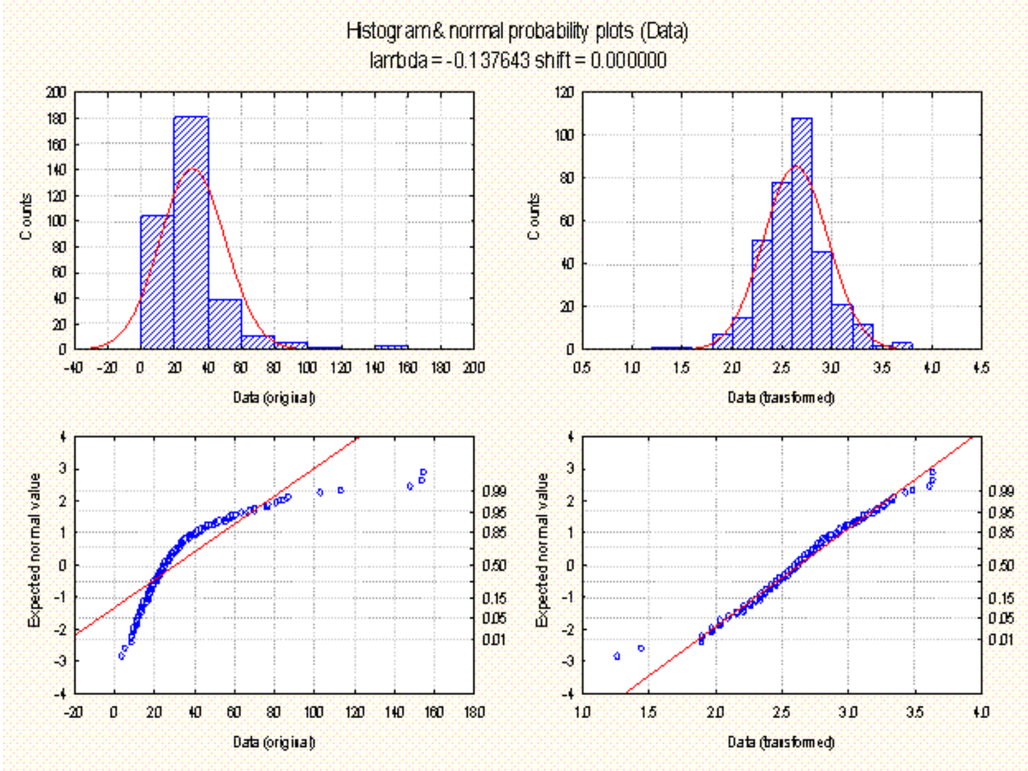
MISSING VALUES IMPUTATION

- **Numeric Features**
  - Impute with mean for normal distributed feature
  - Impute with median for skewed distribution (reduce the impacts of outliers)
- **Categorical Features**
  - Majority imputation
- **Supervised learning imputation**
  - Regression/ Classification
- **XGBoost/LightGBM**:
  - Missing values are gracefully treated by considering missing value as candidate for tree splitting point



DATA PREPROCESSING – NUMERIC FEATURES

- **Box-Cox transformation**
  - Reduce data skewness by shifting the distribution
  - $$y = \begin{cases} {(x^{\lambda}) - 1}/{\lambda} & when\ \lambda \neq 0 \\ \log(x) & when\ \lambda = 0 \end{cases}$$
  - scipy.stats.boxcox
  - Requires the input data to be positive
  - Not needed by tree models



DATA PREPROCESSING – NUMERIC FEATURES

- **Standardization**
  - To standardize features by removing the mean and scaling to unit variance
  - sklearn.preprocessing.StandardScaler
  - Required by SVM/ K-means. Helps linear models such as Linear Regression, Logistic Regression, LASSO/Ridge and NN to converge faster.
  - Not needed by tree models

$$x_{new} = \frac{x - \mu}{\sigma}$$

```
from sklearn.preprocessing import StandardScaler

data = [[0, 0], [0, 0], [1, 1], [1, 1]]
scaler = StandardScaler()
print(scaler.fit(data))
```

DATA PREPROCESSING – NUMERIC FEATURES

- **Normalization**
  - Scale individual samples to have unit norm (e.g. 0-1)
  - sklearn.preprocessing.Normalizer
  - Helps models such as Linear Regression, Logistic Regression, LASSO/Ridge and NN to converge faster.
  - Not needed by tree models

$$x_{new} = \frac{x - x_{min}}{x_{max} - x_{min}}$$

DATA PREPROCESSING – CATEGORICAL FEATURES

- **One Hot Encoding**
- Encode categorical **integer** features using a one-hot aka one-of-K scheme
- OHE is the most adaptive encoding approach. Works well for both linear models and tree models
- sklearn.preprocessing.OneHotEncoder
- Categorical text features need to be transformed to integers prior to OHE.
- Sparse matrix is recommended for outputs

Sample	Category	Numerical
1	Human	1
2	Human	1
3	Penguin	2
4	Octopus	3
5	Alien	4
6	Octopus	3
7	Alien	4

OHE

Sample	Human	Penguin	Octopus	Alien
1	1	0	0	0
2	1	0	0	0
3	0	1	0	0
4	0	0	1	0
5	0	0	0	1
6	0	0	1	0
7	0	0	0	1

```
from sklearn.preprocessing import OneHotEncoder
enc = OneHotEncoder()
enc.fit([[0, 0, 3], [1, 1, 0], [0, 2, 1], [1, 0, 2]])
```

DATA PREPROCESSING – CATEGORICAL FEATURES

- **Label Encoding**
- Encode categorical features with integers between 0 and # of levels -1
- Works for tree models such as Random Forest, GBDT, GBM, XGBoost and LightGBM
- Not work for linear models
- sklearn.preprocessing.OneHotEncoder
- Pros:
  - RAM efficient
  - Works for categorical features with large number of levels
  - Easier to implement

Sample	Category	Numerical
1	Human	1
2	Human	1
3	Penguin	2
4	Octopus	3
5	Alien	4
6	Octopus	3
7	Alien	4

Label-encoded

DATA PREPROCESSING – TEXT FEATURES

- **TF-IDF encoding**
  - Transform a count matrix to a normalized tf or tf-idf representation
  - **Term Frequency**
    - Raw count of a term in a document, i.e. the number of times that term t occurs in document dInverse document frequency
  - **Inverse document frequency**
    - How much information the word provides, that is, whether the term is common or rare across all documents
  - [sklearn.feature_extraction.text.TfidfVectorizer]
  - Sparse matrix is the recommended representation format for outputs

	text
0	I have a friend whose name is Kevin
1	Kevin's best friend is Joe
2	Joe is not a friend of mine
3	Kevin and Joe both know another Kevin

	best	friend	joe	kevin	know
0	0.000000	0.707107	0.000000	0.707107	0.000000
1	0.670819	0.428175	0.428175	0.428175	0.000000
2	0.000000	0.707107	0.707107	0.000000	0.000000
3	0.000000	0.000000	0.366260	0.732521	0.573818

```
from sklearn.feature_extraction.text import TfidfVectorizer

encoder = TfidfVectorizer(stop_words='english', max_features=200)
text_features =encoder.fit_transform(documents)
pd.DataFrame(text_features.todense(), columns=sorted(encoder.vocabulary_))
```

FEATURE INTERACTIONS

- **Numerical-to-numerical**
  - Basic operations (+ - * /)
    - Works for tree models
  - Polynomial
    - May work for both linear and tree models
- **Categorical-to-categorical**
  - N-way combinations
    - A, B - > AB
    - **Try**: features that complements each other, e.g. state vs product
    - **Not** try: features that are hierarchical, e.g state vs county
- **Numerical-to-categorical**
  - Aggregations (sum, median, mean, std) of numerical features group by categorical features

THINK OUT OF THE BOX. . .

HOW TO HACK HOPSCORE?

HopScore: a proprietary algorithm that sorts and ranks all apartment listings on our site based on a set of different factors. These factors fall into three categories:

- **Listing quality**
  - Photos
  - Descriptions
- **Manager Performance**
  - Reply rate
  - Activeness
- **Listing freshness**
  - A listing's freshness will decrease over time
  - A listing may occur many times in the dataset and the interest level may be different

Relevance	Listing Quality	Manager Performance	Listing Freshness
bathrooms	Fair	Fair	Low
bedrooms	Fair	Fair	Low
building_id	Fair	Fair	Low
created	Low	Fair	High
description	High	High	Low
display_address	High	Fair	Low
features	High	High	Low
latitude	High	Fair	Fair
listing_id	Low	Fair	High
longitude	High	Fair	Fair
manager_id	Fair	High	Low
photos	High	Fair	Low
price	High	Fair	Fair
street_address	High	Fair	Low

Feature-Hopscore relevance matrix

A few key **findings**:

- Manager seems the most important factor as it may also impact listing quality
- lat/lon/desc can be used as the unique id for a listing which will also allow us to identify if a listing had been posted many times

HOW TO HACK HOPSCORE?

- **Listing quality**
  - % of upper case words
  - # of '*', '!', '$', '<' etc
  - HTML tags
  - Phone number in desc
  - email address in desc
  - is street address the same as display address
- **Manager Performance**
  - count/mean/median/std/min/max of bathrooms/bedrooms/long/lat/price/# photo/# feature/ len of desc group by manager id (**key** to success)
- **Listing freshness**
  - How many times a property had been listed?
  - How long since a property had been listed?

WIDE WILD WEST

- **More manger features**
  - # of active days
  - # of postings
  - Mean postings response time span
  - Aggregate listing quality/ listing freshness features by manager
- **More listing freshness features**
  - Price changes since last listing/ first listing for a property
  - Interest level of last listing/ first listing for a property
  - Is this posting the last one for a property
  - is this posting the first one for a property
- **Location, location, location!**
  - Perform k-means to cluster listings into geo-segments
    - Aggregate attributes by geo-segments
    - Distance between a listing and the centroid of a cluster
  - Create clusters after filtering certain keywords such as “supermarket”, “subway”, “bus”, “park” etc
    - Aggregate attributes by geo-segments
    - Distance between a listing and the centroid of a cluster
  - Not allowed by the competition but you may want to give it a try from knowledge learning perspective:
    - Retrieve postal code/ neighbourhood by lat/lon, perform similar aggregations mentioned before
    - Combine public demographic stats by joining on postal code

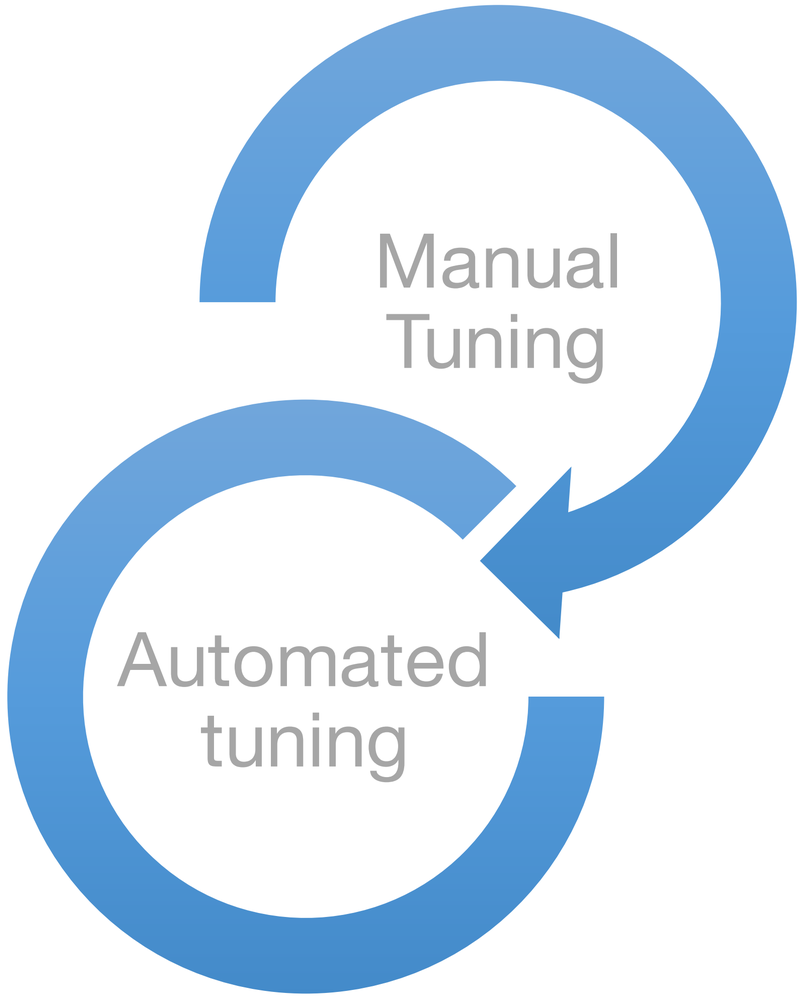
MODEL TUNING

TUNING A MACHINE LEARNING ALGORITHM IS OFTEN A “BLACK ART” THAT REQUIRES EXPERT EXPERIENCE, UNWRITTEN RULES OF THUMB, OR SOMETIMES BRUTE-FORCE SEARCH

Jasper Snoek

Author of “Practical Bayesian Optimization of Machine Learning Algorithms “

MANUAL TUNING VS AUTOMATED TUNING



Manual tuning

- Greedy search
- Tune one parameter at a time
- Use cross validation if possible
- Otherwise use holdout dataset

Automated tuning

- Tune multiple parameters at the same time
- Grid search, or
- Bayesian Optimization
- Findings discovered by manual tuning can be leveraged to determine parameter space for automated tuning

XGBOOST

THE NAME XGBOOST, THOUGH, ACTUALLY REFERS TO THE ENGINEERING GOAL TO PUSH THE LIMIT OF COMPUTATIONS RESOURCES FOR BOOSTED TREE ALGORITHMS. WHICH IS THE REASON WHY MANY PEOPLE USE XGBOOST.

Tianqi Chen

XGBOOST OVERVIEW

- State of art accuracy thanks to its enhanced
  - object function - **second order derivation** (GBDT uses only first order)
  - regularization - tree **complexity as regularization**
- Faster processing
  - Parallel processing
    - **Feature based**, as compared to RF which is tree based
    - Fit **Residual** between trees, vs Gradient (traditional GBDT, e.g. GBM)
    - Optimized memory usage - **pre-sorted blocks**
  - Smartly deal with **sparse features** and **missing values**
- Supports both **Tree** booster (gbtree) and **Linear** Booster (gblinear)

XGBOOST HYPER PARAMETERS – TREE COMPLEXITY

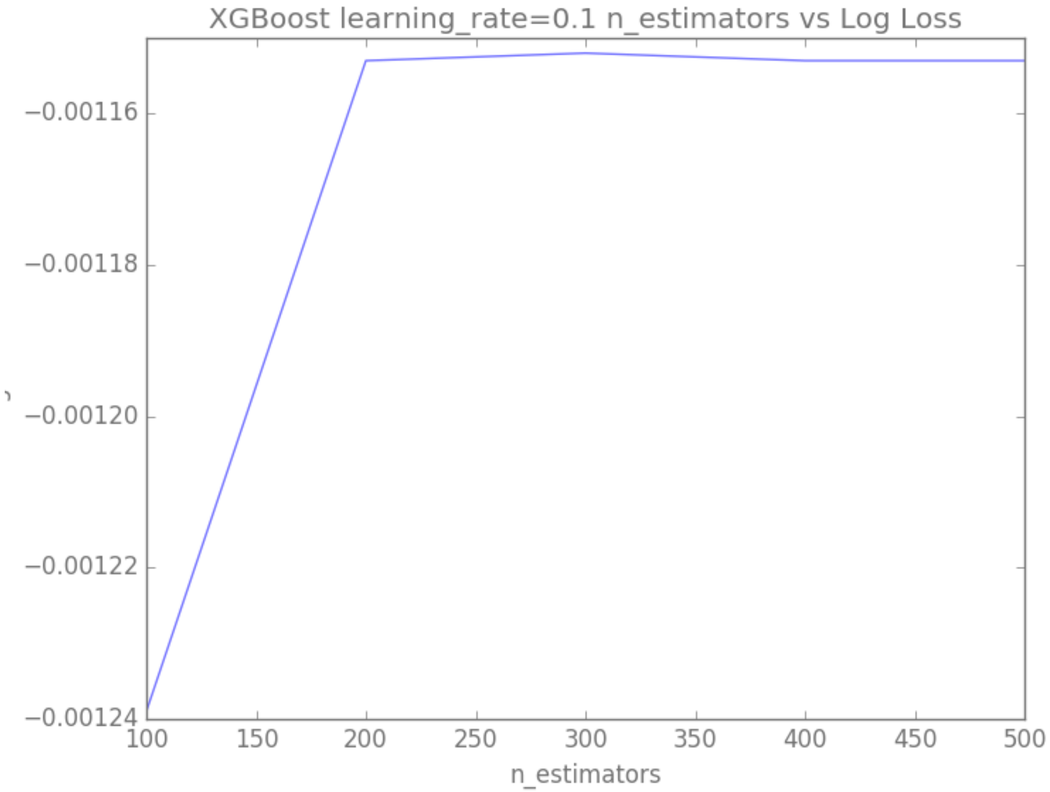
- **max depth** of the tress (max_depth)
  - Controls over-fitting as higher depth will allow model to learn relations very specific to a particular sample however can lead to over-fitting
  - Typical value range: 3 - 10
- **min child weight** of a leaf (min_child_weight)
  - Defines the minimum sum of weights (hessian) of all observations required in a child which is used to control over-fitting. Higher values prevent a model from learning relations which might be highly specific to the particular sample selected for a tree.
  - Typical value: 1-20
- **gamma**
  - minimum loss reduction (number of leaves + L2 norm of leaf scores) required to make a further partition on a leaf node of the tree. the larger, the more conservative the algorithm will be.
  - **better** than min_child_weight in terms of control overfitting
  - Typical values: 0-2

XGBOOST HYPER PARAMETERS – STOCHASTIC PROCESS

- **subsample**
  - Denotes the fraction of observations to be randomly samples for each tree.
  - Lower values make the algorithm more conservative and prevents overfitting but too small values might lead to under-fitting.
  - Typical values: 0.7-1
- **column sample by tree** (colsample_bytree)
  - Denotes the fraction of columns to be randomly samples for each tree.
  - Typical values: 0.3-0.9

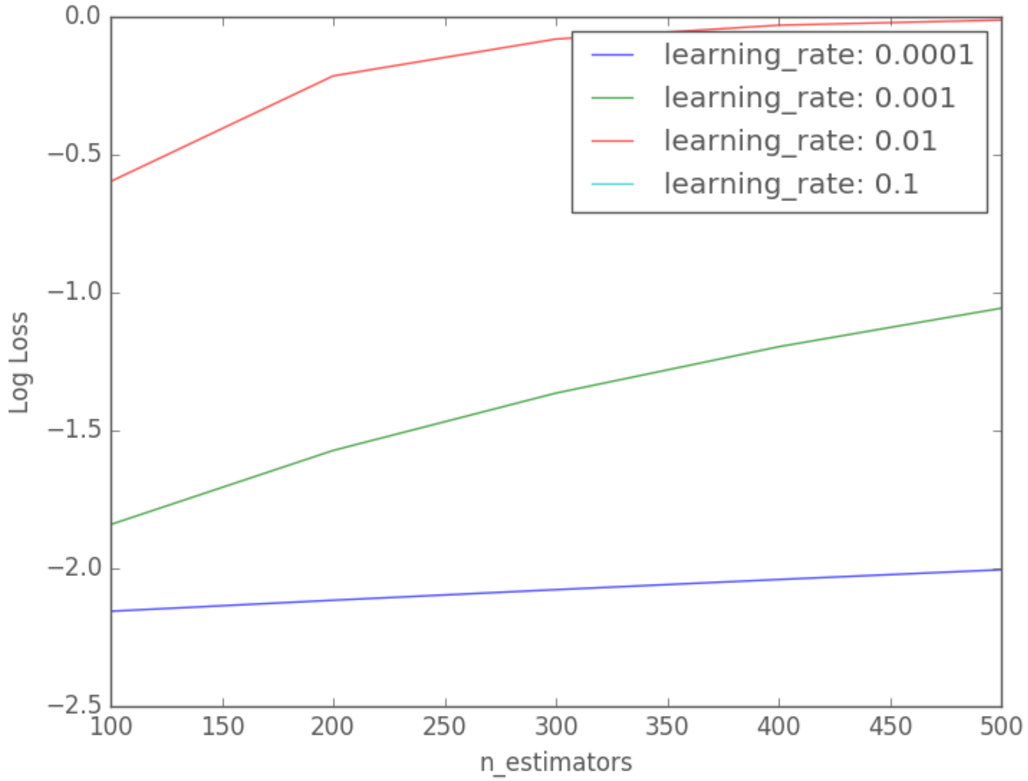
XGBOOST HYPER PARAMETERS – BOOSTING

- **number of trees** (n_estimators, num_of_round)
  - Typically the larger the better, however there's a turn-around point where the model will go overfitting afterwards.
  - The optimal number depends on other parameters, particularly on eta
  - Use early stopping to tune numbers of trees



XGBOOST HYPER PARAMETERS – BOOSTING

- **learning rate** (learning_rate, eta)
  - **Smaller** learning rate yields **better** results but may needs more boosting rounds(trees) to converge.
  - Typical range: 0.01-0.1
  - Use **bigger** eta (0.1) to **tune** other parameters to save time
  - Use **smaller** eta to **train** the actual model once other parameter have been tuned.



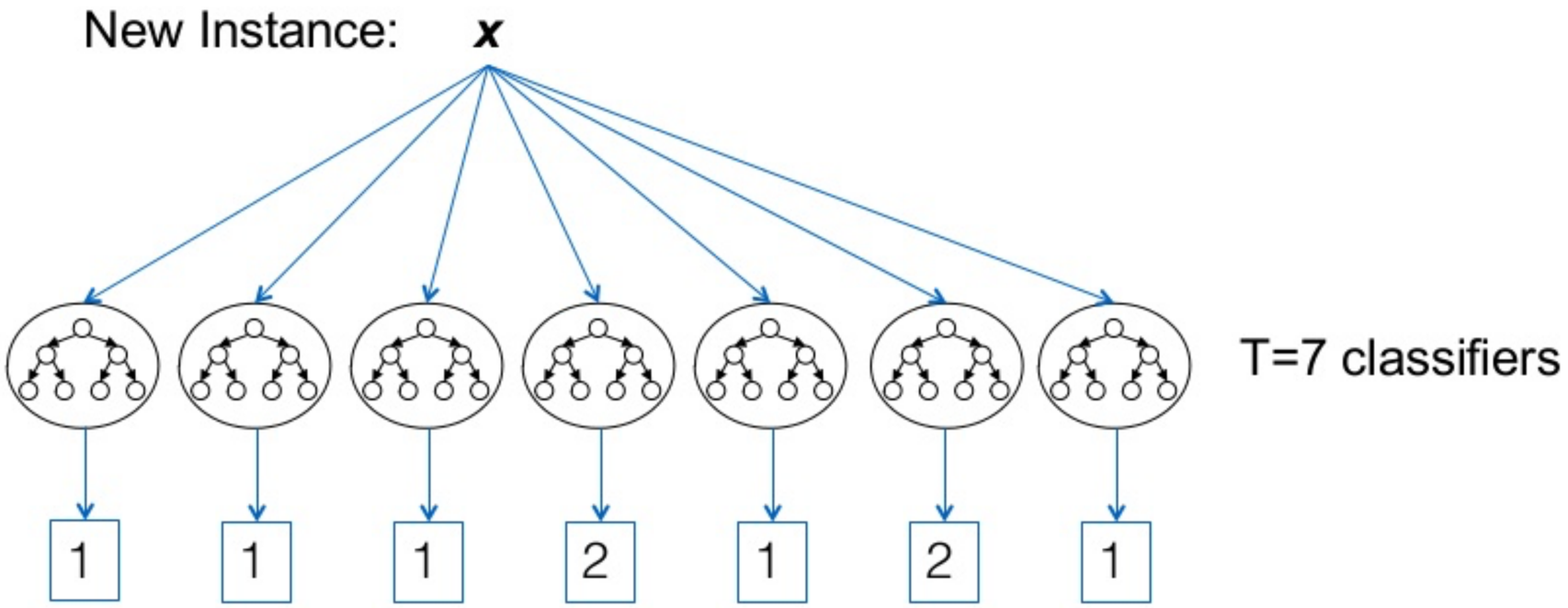
MODEL ENSEMBLE

ESSENTIALLY ALL MODELS ARE WRONG, BUT SOME ARE USEFUL.

George E.P. Box

WHAT IS AN ENSEMBLE?

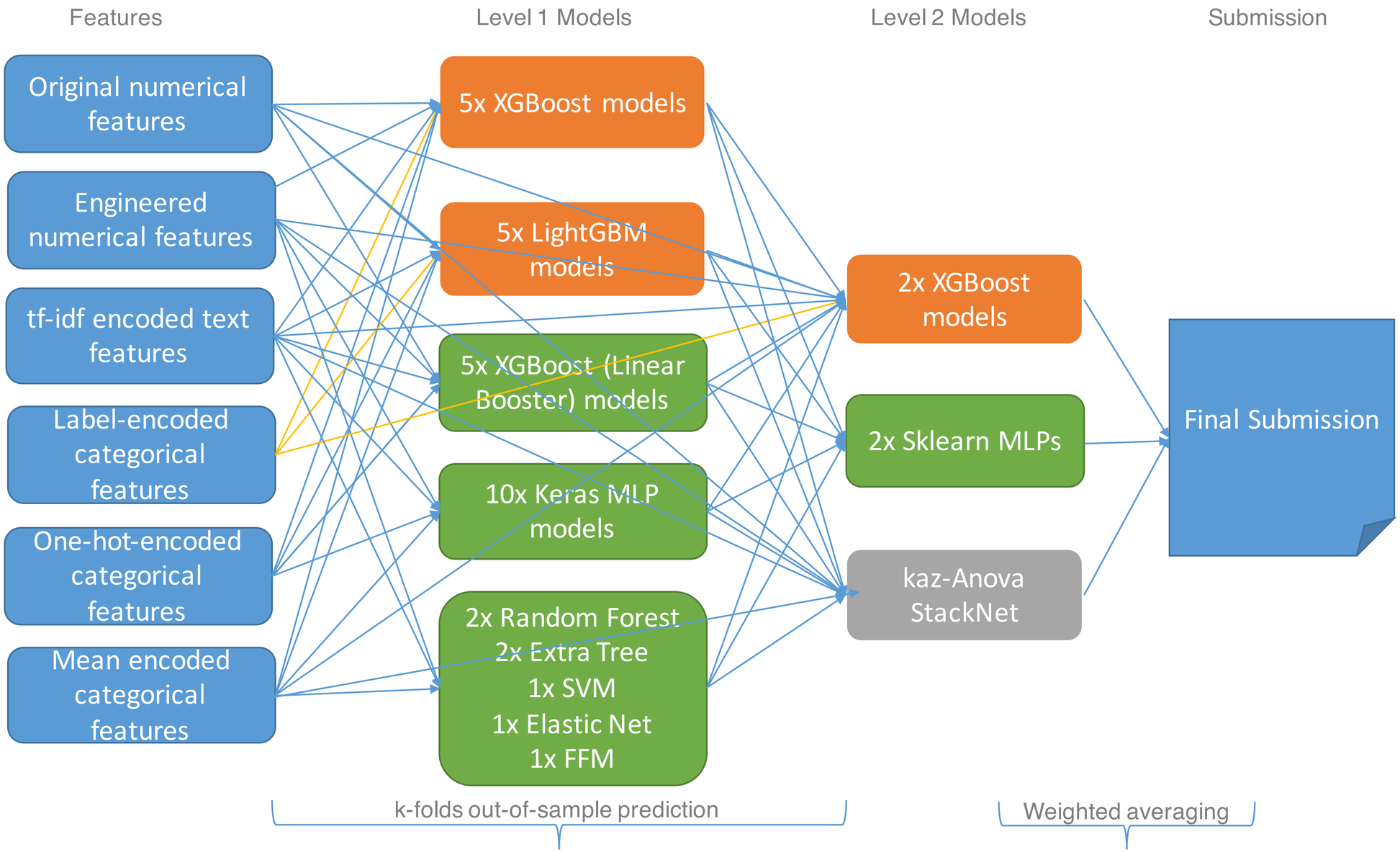
An ensemble is a combination of learners that output a final predictions.



In general, together they perform better than any of the single leaners

STACKING – THE ULTIMATE SOLUTION FOR ENSEMBLE

MY FINAL SOLUTION




QUESTIONS?