Intro into Ensembling

Ensembling

The natural world is complex, so it figures that ensembling different models can capture more of this complexity.

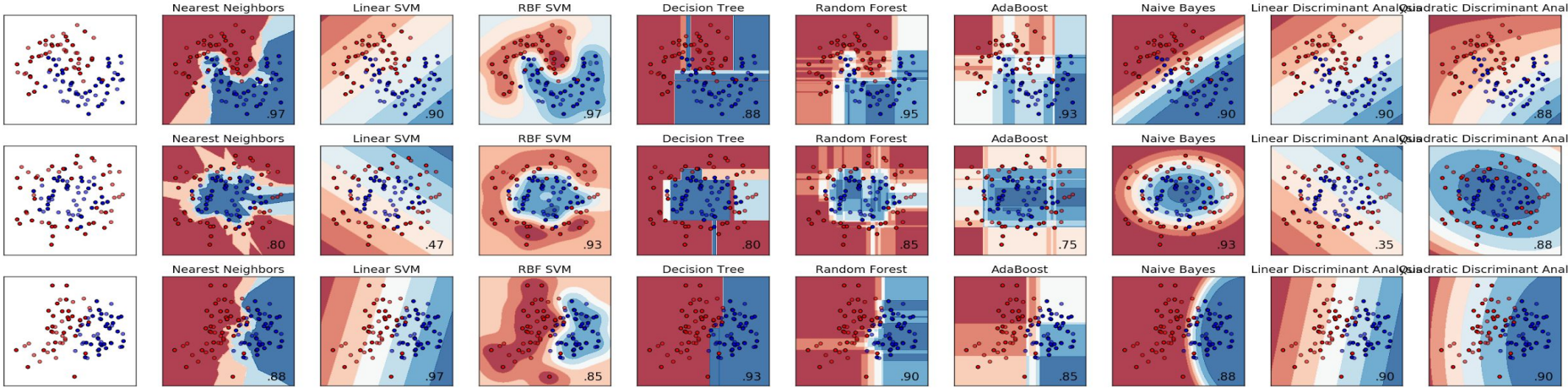
Ben Hamner, Chief Scientist at Kaggle

Supervised learning algorithms are commonly described as performing the task of searching through a hypothesis space to find a suitable hypothesis that will make good predictions with a particular problem. Even if the hypothesis space contains hypotheses that are very well-suited for a particular problem, it may be very difficult to find a good one. Ensembles combine multiple hypotheses to form a (hopefully) better hypothesis.

Wikipedia

Typical supervised learning framework

- Feature engineering
- Sampling
- Hyperparameter optimization
- Applying algos of different origins



Voting

Voting

The idea behind the voting classifier implementation is to combine conceptually different machine learning classifiers and use a majority vote or the average predicted probabilities (soft vote) to predict the class labels.

Such a classifier can be useful for a set of equally well performing model in order to balance out their individual weaknesses.

Reasoning behind majority voting

Ground truth: `1111111111`

We have 3 binary classifiers with a 70% accuracy. We can view these classifiers for now as pseudo-random number generators which output a “1″ 70% of the time and a “0″ 30% of the time.

A voting ensemble of 3 pseudo-random classifiers with 70% accuracy would be correct ~78% of the time.

A voting ensemble of 5 pseudo-random would be correct ~83% of the time.

...

Voting works well with uncorrelated models

Ground truth: `1111111111`

Highly correlated models:

`1111111100 = 80% accuracy`

`1111111100 = 80% accuracy`

`1011111100 = 70% accuracy.`

Their majority voting ensemble:

`1111111100 = 80% accuracy`

Ground truth: `1111111111`

Highly uncorrelated models:

`1111111100 = 80% accuracy`

`0111011101 = 70% accuracy`

`1000101111 = 60% accuracy`

Their majority voting ensemble:

`1111111101 = 90% accuracy`

Weighing

Usually we want to give a better model more weight in a vote.

The reasoning is as follows: The only way for the inferior models to overrule the best model (expert) is for them to collectively (and confidently) agree on an alternative.

We can expect this ensemble to repair a few erroneous choices by the best model, leading to a small improvement only.

Kaggle: Forest Cover Type prediction

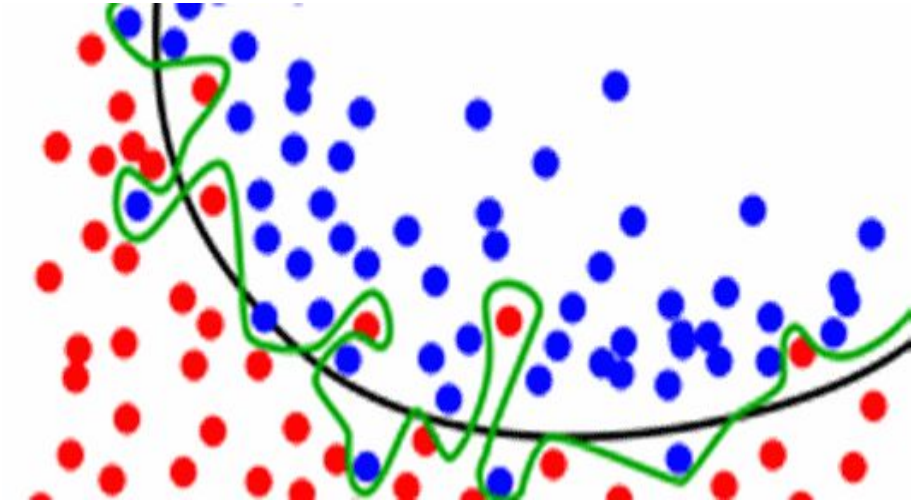
MODEL	PUBLIC ACCURACY SCORE
GradientBoostingMachine	0.65057
RandomForest Gini	0.75107
RandomForest Entropy	0.75222
ExtraTrees Entropy	0.75524
ExtraTrees Gini (Best)	**0.75571**
Voting Ensemble (Democracy)	0.75337
Voting Ensemble (3*Best vs. Rest)	**0.75667**

Averaging and rank averaging

Averaging predictions often reduces overfit. You ideally want a smooth separation between classes, and a single model's predictions can be a little rough around the edges.

When averaging the outputs from multiple different models some problems can pop up. Not all predictors are perfectly calibrated: they may be over- or underconfident when predicting a low or high probability. Or the predictions clutter around a certain range.

The solution is to first turn the predictions into ranks, then averaging these ranks. After normalizing the averaged ranks between 0 and 1 you are sure to get an even distribution in your predictions.



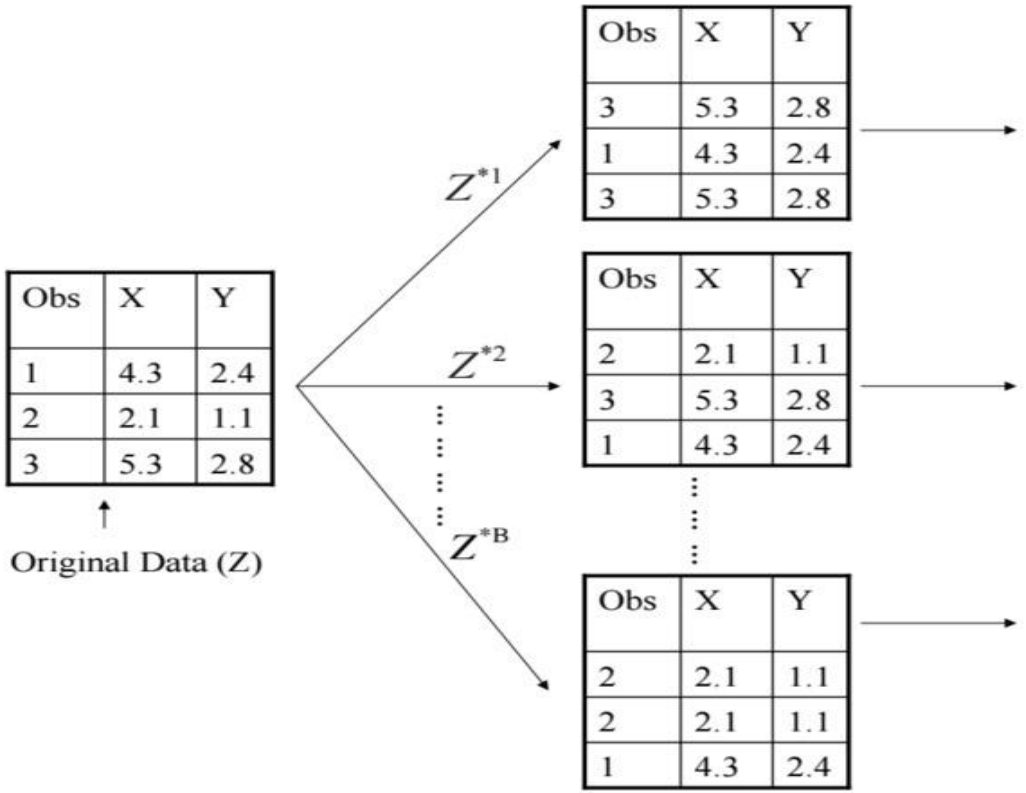
Our goal is not to memorize the training data but to generalize well to new unseen data.

Bagging

Bootstrap

1. Create a random subset of data by sampling

2. Draw N' of the N samples with replacement (sometimes w/o)



Bagging (bootstrap aggregating)

- Repeat K times
  - Create a training set of N’ < N examples
  - Train a classifier on the random training set
- To test, run each trained classifier
  - Each classifier votes on the output, take majority
  - For regression: each regressor predicts, take average

Variations of bagging

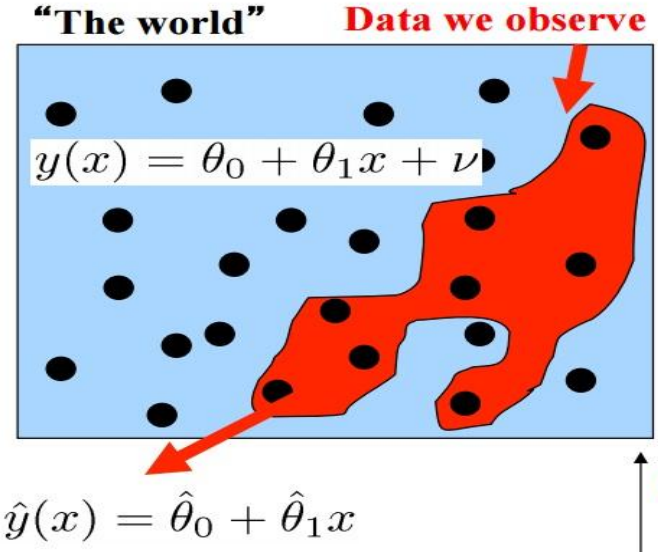
When random subsets of the dataset are drawn as random subsets of the samples, then this algorithm is known as **Pasting**.

When samples are drawn with replacement, then the method is known as **Bagging**.

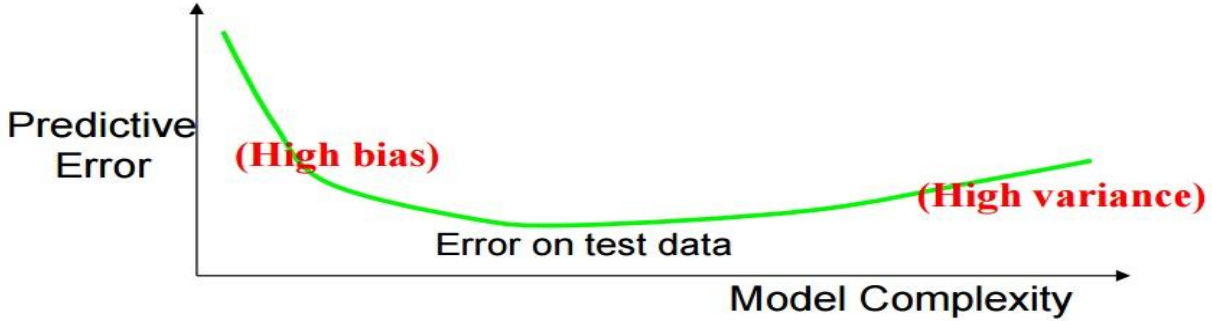
When random subsets of the dataset are drawn as random subsets of the features, then the method is known as **Random Subspaces**.

Finally, when base estimators are built on subsets of both samples and features, then the method is known as **Random Patches**.

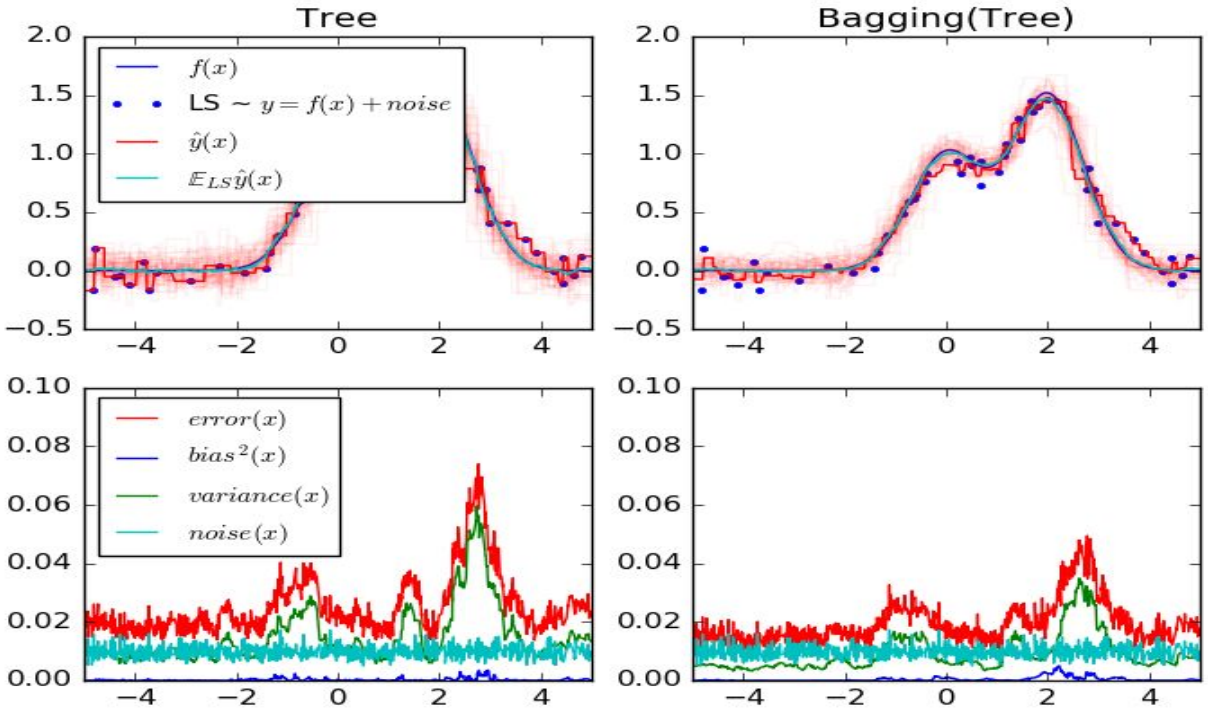
Bias-variance decomposition



- We only see a little bit of data
- Can decompose error into two parts
  - Bias – error due to model choice
    - Can our model represent the true best predictor?
    - Gets better with more complexity
  - Variance – randomness due to data size
    - Better w/ more data, worse w/ complexity

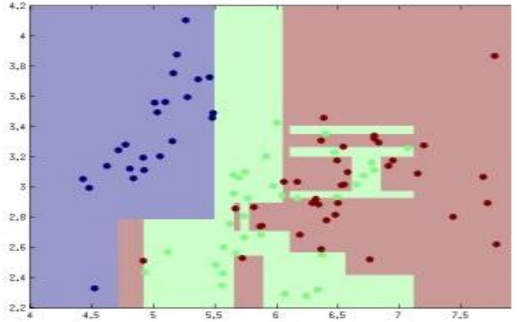


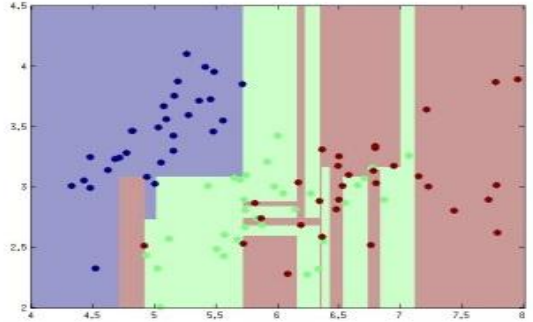
Single estimator versus bagging

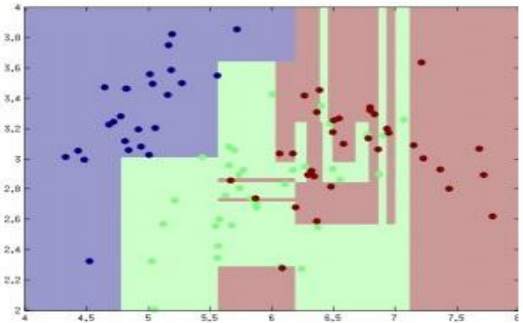


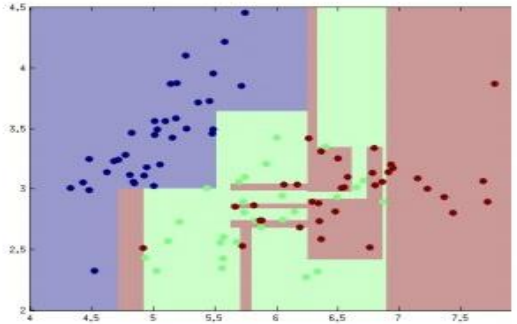
Individual decision trees

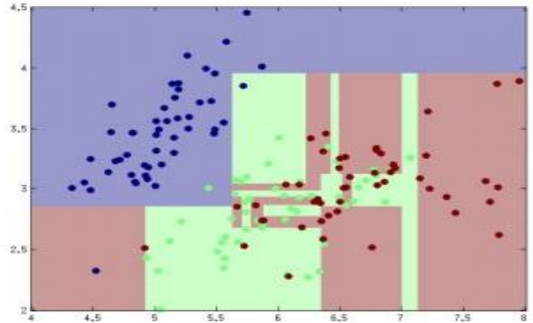
Simulates “equally likely” data sets we could have observed instead, & their classifiers







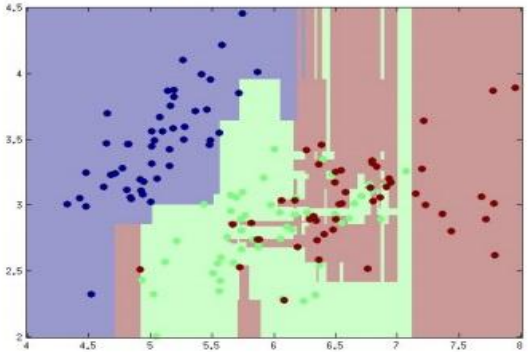




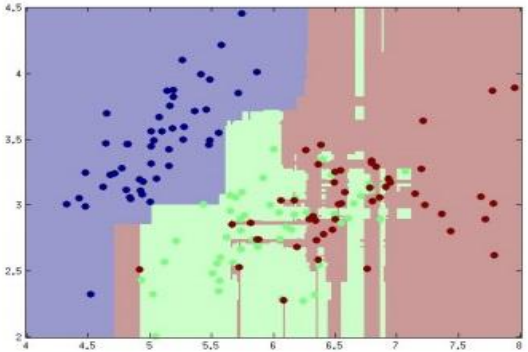
Bagged decision trees

- Reduces memorization effect
  - Not every predictor sees each data point
  - Lowers “complexity” of the overall average
  - Usually, better generalization performance

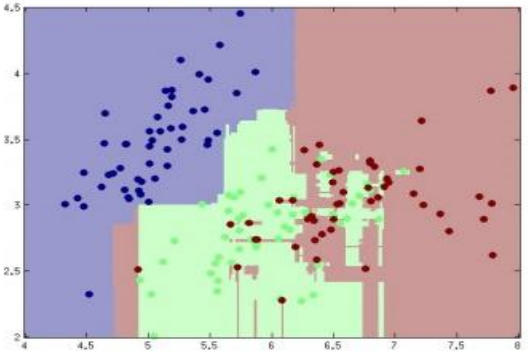
Avg of 5 trees



Avg of 25 trees



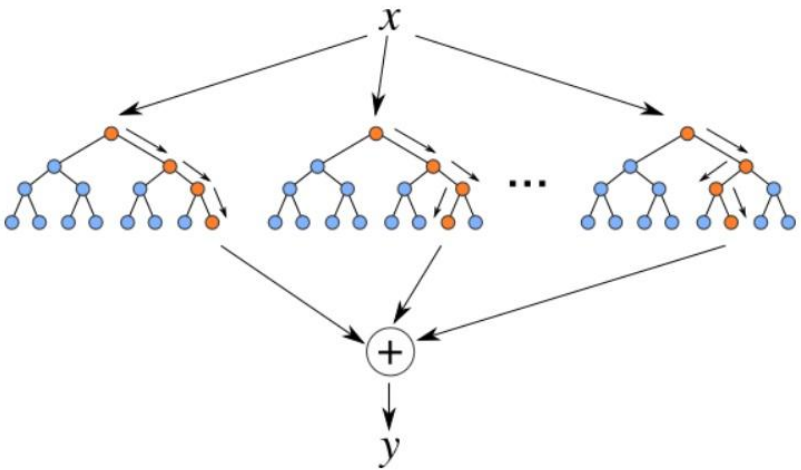
Avg of 100 trees



Random forest

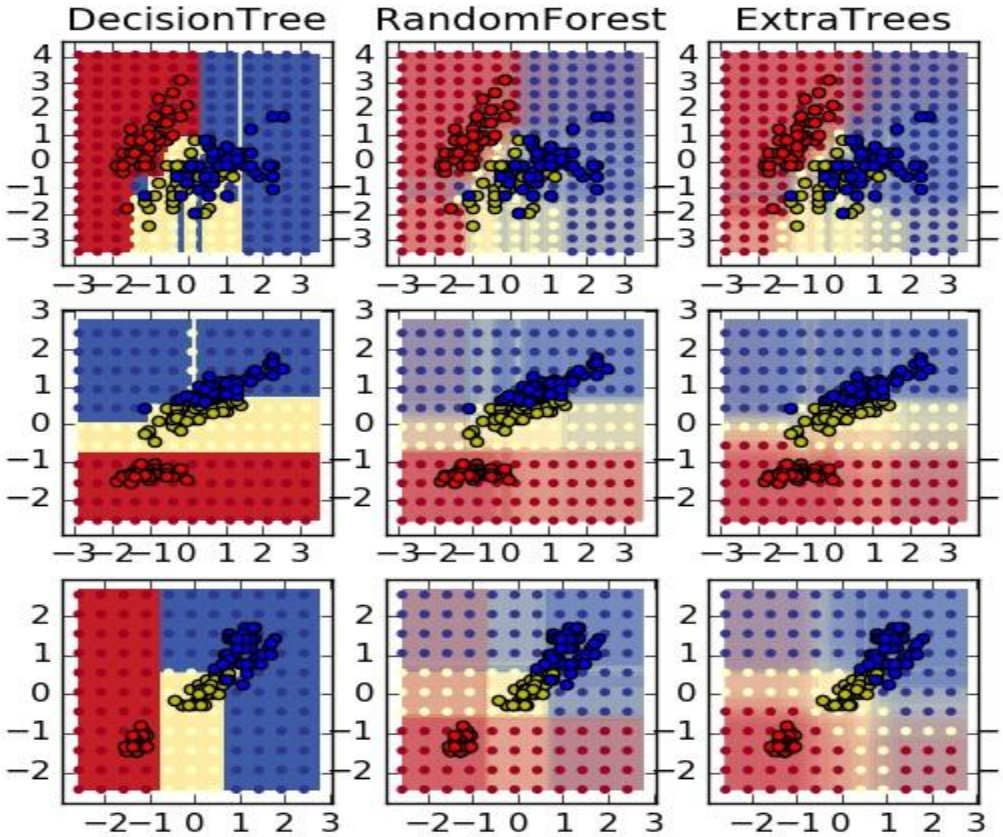
- Bagging applied decision trees
- Problem
  - With lots of data, we usually learn the same classifier
  - Averaging over these doesn’t help
- Introduce extra variation in learner
  - At each step of training, only allow a subset of features
  - Enforces “diversity” (“best” feature not available)
  - Average over these learners (majority vote)

Extremely randomized trees



In extremely randomized trees randomness goes one step further in the way splits are computed. As in random forests, a random subset of candidate features is used, but instead of looking for the most discriminative thresholds, **thresholds are drawn at random for each candidate feature and the best of these randomly-generated thresholds is picked as the splitting rule**. This usually allows to reduce the variance of the model a bit more, at the expense of a slightly greater increase in bias.

Classifiers on feature subsets of the Iris dataset



Summary of bagging techniques

- Bagging methods are used as a way to reduce the variance of a base estimator, by introducing randomization into its construction procedure
- Constitute a simple way to improve with respect to a single model, without making it necessary to adapt the underlying base algorithm
- Work best with strong and complex models (fully developed decision trees)
- Don’t work well with linear base models
- Introduce additional computational costs that can be parallelized

Averaging

$$F(x) = \sum_{m=1}^{M} \gamma_m h_m(x)$$

In general, averaging works well when

- Randomness is an inherent part of the learning process
- Sampling techniques are used
- Models from different families are present

Pros:

- Easy to implement
- Efficient in practice
- Almost doesn’t overfit

Cons:

- Can be too restrictive

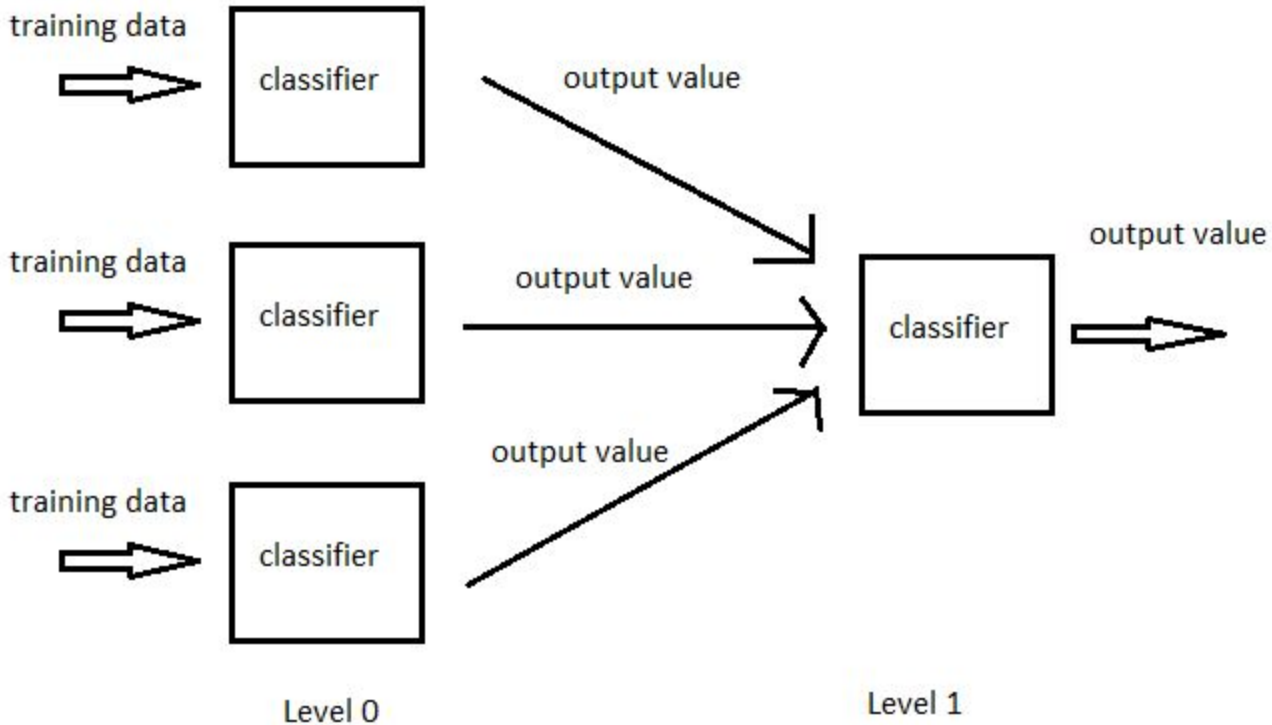
Stacking/Blending

Stacked Generalization

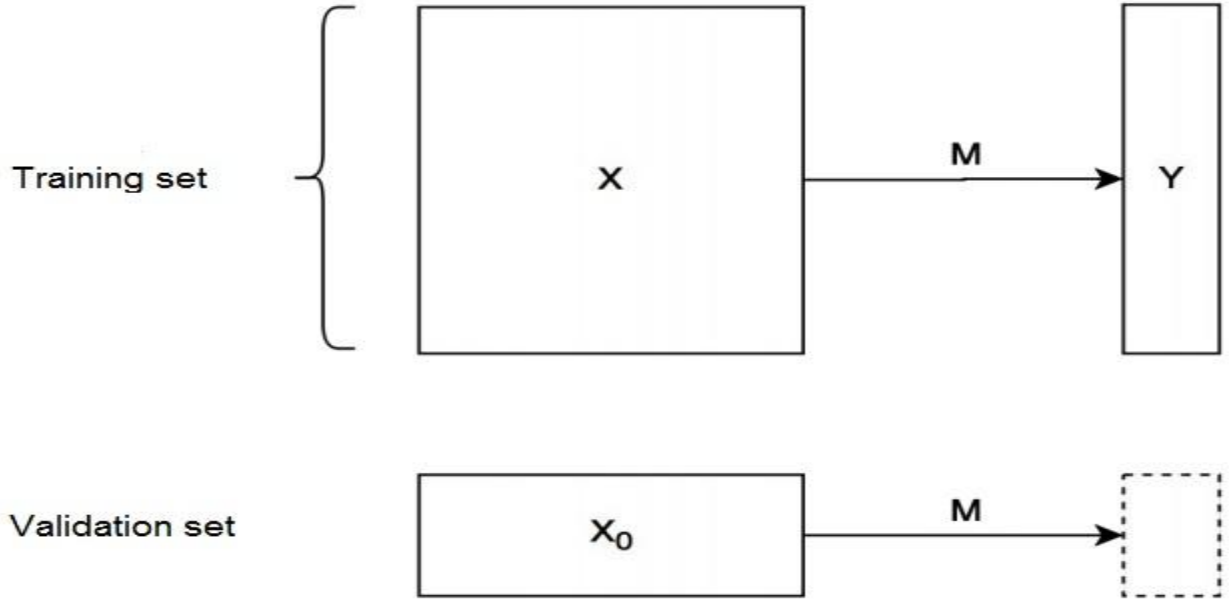
[...] stacked generalization is a means of non-linearly combining generalizers to make a new generalizer, to try to optimally integrate what each of the original generalizers has to say about the learning set. The more each generalizer has to say (which isn't duplicated in what the other generalizer's have to say), the better the resultant stacked generalization.

Wolpert (1992) Stacked Generalization

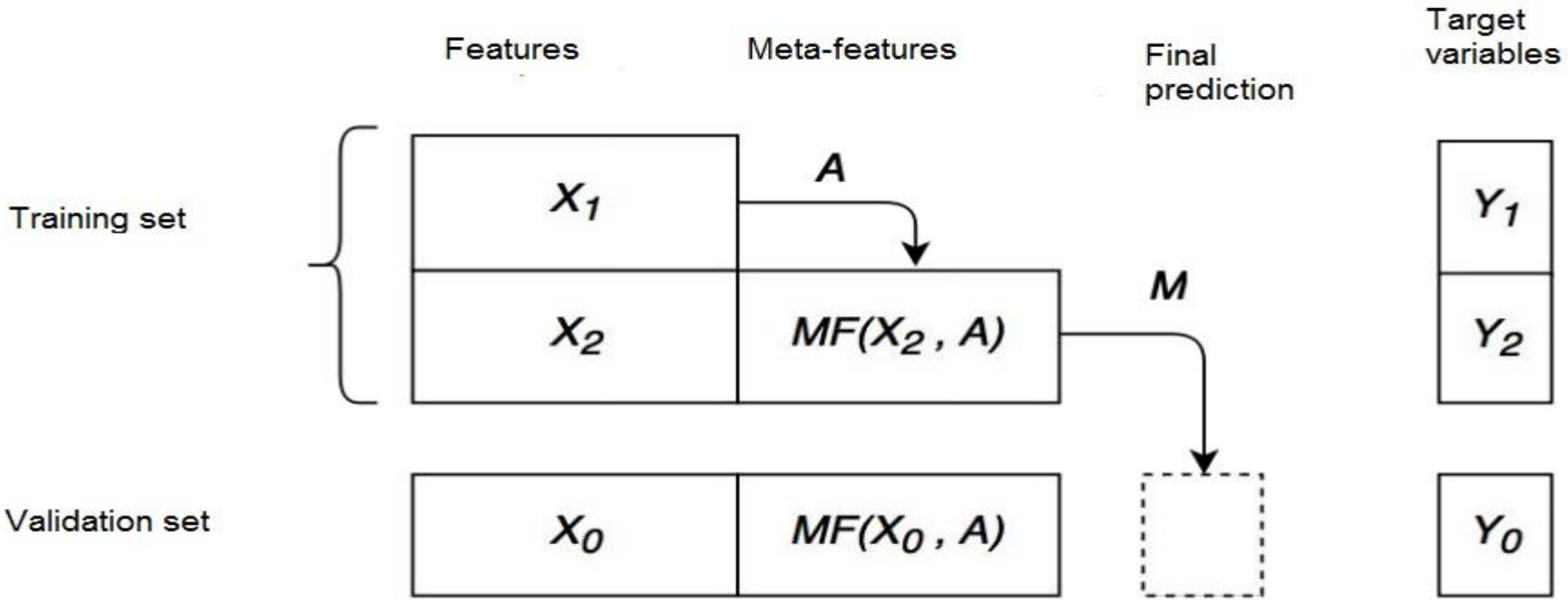
Concept Diagram of Stacking



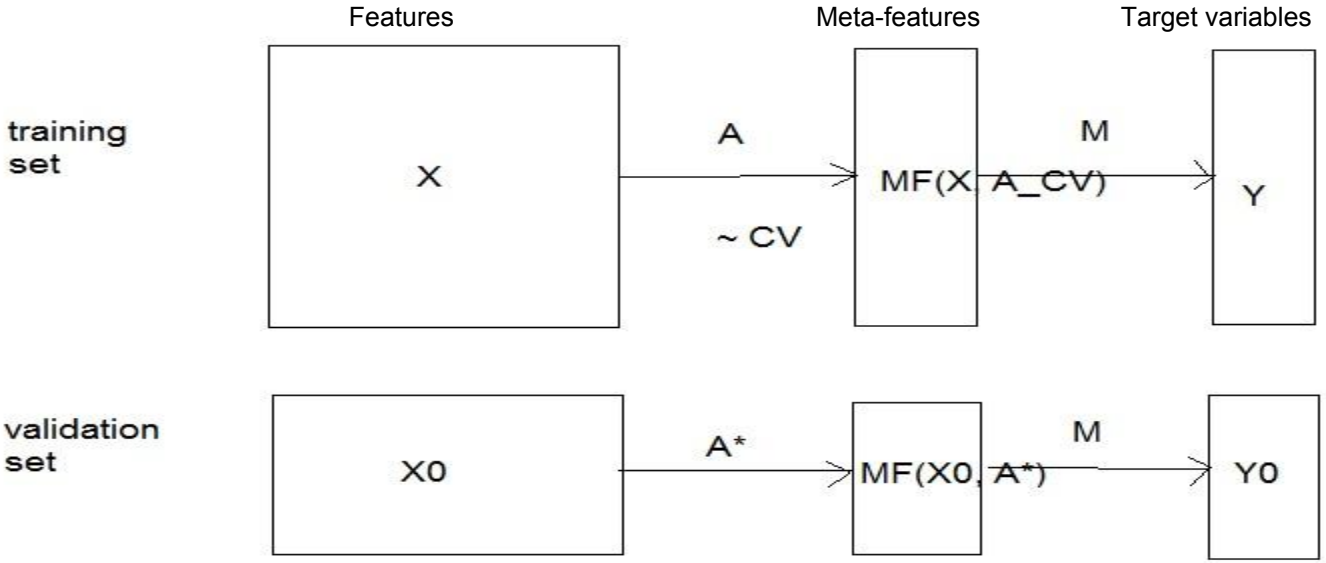
Supervised learning - typical design



Blending - typical design



Stacking - typical design



Blending versus stacking

Blending has a few benefits:

- It is simpler than stacking.
- It wards against an information leak: The generalizers and stackers use different data.
- You do not need to share a seed for stratified folds. Anyone can throw models in the ‘blender’ and the blender decides if it wants to keep that model or not.

The cons are:

- You use less data overall
- The final model may overfit to the holdout set.
- Your CV is more solid with stacking (calculated over more folds) than using a single small holdout set.

As for **performance**, both techniques are able to give **similar results**.

Stacking becomes more preferable when the data is limited.

Words of caution

It’s not a good idea to use in-sample predictions for stacking/blending!
The problem is overfitting.
The more “over-fit” base prediction, the more weight it will get in the generalizer.

- Use out of sample predictions
  - Take half of the training data to build model 1
  - Apply model 1 on the rest of the training data, use the output as input to model 2
- Use cross-validation partitioning when data limited
  - Partition training data into K partitions
  - For each of the K partition, compute “prediction 1” by building a model with OTHER partitions

Stacking in practice - Netflix Prize

Leaderboard

Showing Test Score. Click here to show quiz score

Display top 20 **leaders.**

Rank	Team Name	Best Test Score	% Improvement	Best Submit Time
Grand Prize - RMSE = 0.8567 - Winning Team: BellKor's Pragmatic Chaos				
1	BellKor's Pragmatic Chaos	0.8567	10.06	2009-07-26 18:18:28
2	The Ensemble	0.8567	10.06	2009-07-26 18:38:22
3	Grand Prize Team	0.8582	9.90	2009-07-10 21:24:40
4	Opera Solutions and Vandelay United	0.8588	9.84	2009-07-10 01:12:31
5	Vandelay Industries !	0.8591	9.81	2009-07-10 00:32:20
6	PragmaticTheory	0.8594	9.77	2009-06-24 12:06:56
7	BellKor in BigChaos	0.8601	9.70	2009-05-13 08:14:09
8	Dace	0.8612	9.59	2009-07-24 17:18:43

Netflix Prize

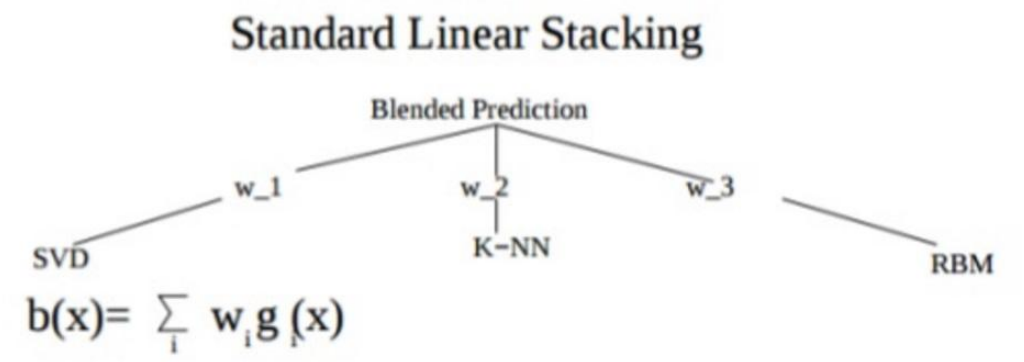
This is a truly impressive compilation and culmination of years of work, blending hundreds of predictive models to finally cross the finish line. We evaluated some of the new methods offline but the additional accuracy gains that we measured did not seem to justify the engineering effort needed to bring them into a production environment.

Netflix Engineers

A number of papers and novel methods resulted from this challenge:

- Feature-Weighted Linear Stacking
- Combining Predictions for Accurate Recommender Systems
- The BigChaos Solution to the Netflix Prize

Feature weighted linear stacking



Feature weighted linear stacking

The hope is that the stacking model learns which base model is the best predictor for samples with a certain feature value. Linear algorithms are used to keep the resulting model fast and simple to inspect.

Pros:

- Easy to implement
- Easy to interpret
- Easy to generalized

Cons:

- Can be too restrictive
- **Easy to overfit**

Nonlinear stacking

Pros:

- Highest accuracy gains
- Find useful interaction between the original features and meta-model features

Cons:

- A lot of effort required
- Very easy to overfit

How to choose models for Stacking?

It is usually desirable that the level 0 generalizers are of all “types”, and not just simple variations of one another (e.g., we want surface-fitters, Turing-machine builders, statistical extrapolators, etc., etc.). In this way all possible ways of examining the learning set and trying to extrapolate from it are being exploited. This is part of what is meant by saying that the level 0 generalizers should “span the space”.

Wolpert (1992) Stacked Generalization

Good ensembles are made from models that are:

- **Diverse**. Different algorithms could be trained on different features and different samples. When correlation is low between two models, ensembling these models usually gives better results, opposed to adding already closely correlated models together.
- **Independent**. Algorithms will overfit (“memorize”) when they are trained on some data, and their predictions are used to train on that same data. (stratified) K-fold training should be employed when stacking.
- **Decentralized**. Algorithms can be trained to focus on a small aspect of a classification or regression problem.

Otto Group Product Classification Challenge

otto group

Completed • $10,000 • 3,514 teams

Otto Group Product Classification Challenge

Tue 17 Mar 2015 – Mon 18 May 2015 (9 months ago)

Dashboard

Home
Data
Make a submission

Information
Description
Evaluation
Rules
Prizes
Timeline

Forum

Scripts
New Script
New Notebook

Leaderboard
Public
Private

My Submissions

Private Leaderboard

1. Gilberto Titericz & Stanislav Semenov
2. ¯_(ツ)_/¯
3. i dont know
4. Dmitry & Davut
5. Mikhail Trofimov
6. Optimistically Convergent
7. x2x4x8
8. all your Gesellschaft mit beschränkter Haftung are belong to us
9. team
10. sjouili

836 Scripts

Understanding XGBoost Model on Otto Data
96 Votes / 10 months ago / R

Competition Details » Get the Data » Make a submission

Classify products into the correct category

Get started on this competition through Kaggle Scripts

The Otto Group is one of the world's biggest e-commerce companies, with subsidiaries in more than 20 countries, including Crate & Barrel (USA), Otto.de (Germany) and 3 Suisses (France). We are selling millions of products worldwide every day, with several thousand products being added to our product line.

A consistent analysis of the performance of our products is crucial. However, due to our diverse global infrastructure, many identical products get classified differently. Therefore, the quality of our product analysis depends heavily on the ability to accurately cluster similar products. The better the classification, the more insights we can generate about our product range.

Evaluation

Submissions are evaluated using the multi-class logarithmic loss. Each product has been labeled with one true category. For each product, you must submit a set of predicted probabilities (one for every category). The formula is then,

$$logloss = -\frac{1}{N}\sum_{i=1}^{N}\sum_{j=1}^{M} y_{ij}\log(p_{ij}),$$

where N is the number of products in the test set, M is the number of class labels, log is the natural logarithm, y_{ij} is 1 if observation i is in class j and 0 otherwise, and p_{ij} is the predicted probability that observation i belongs to class j.

otto group

Completed • $10,000 • 3,514 teams

Otto Group Product Classification Challenge

Tue 17 Mar 2015 – Mon 18 May 2015 (9 months ago)

Dashboard

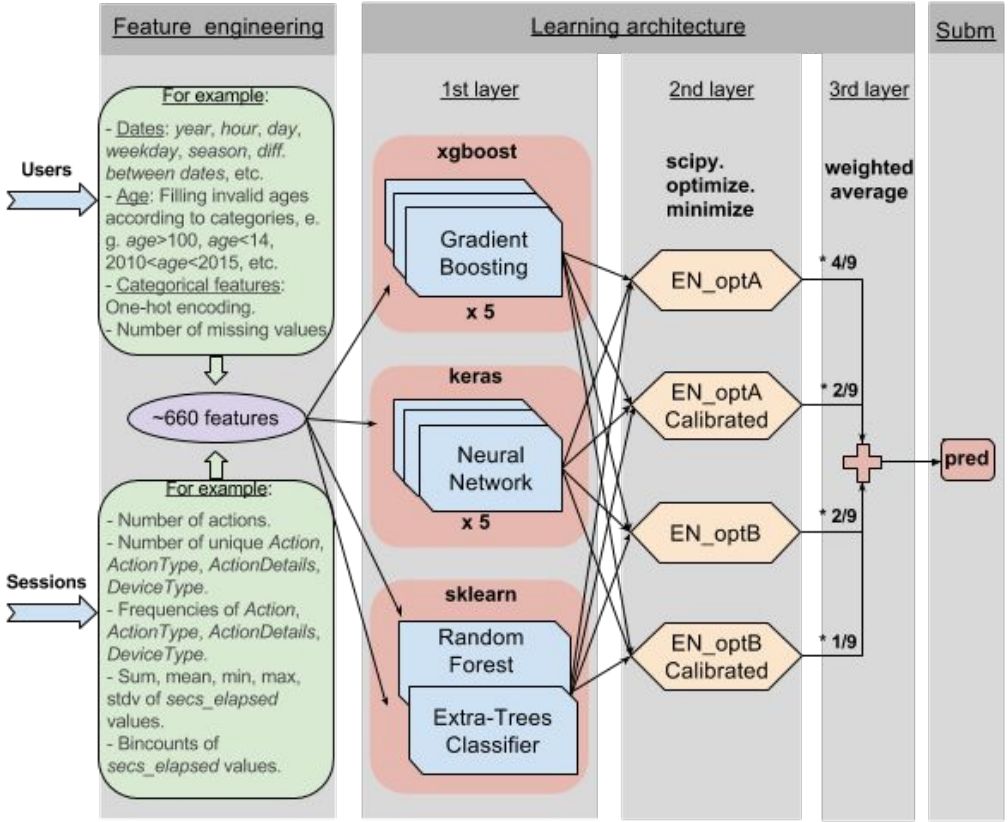
Private Leaderboard - Otto Group Product Classification Challenge

This competition has completed. This leaderboard reflects the final standings.

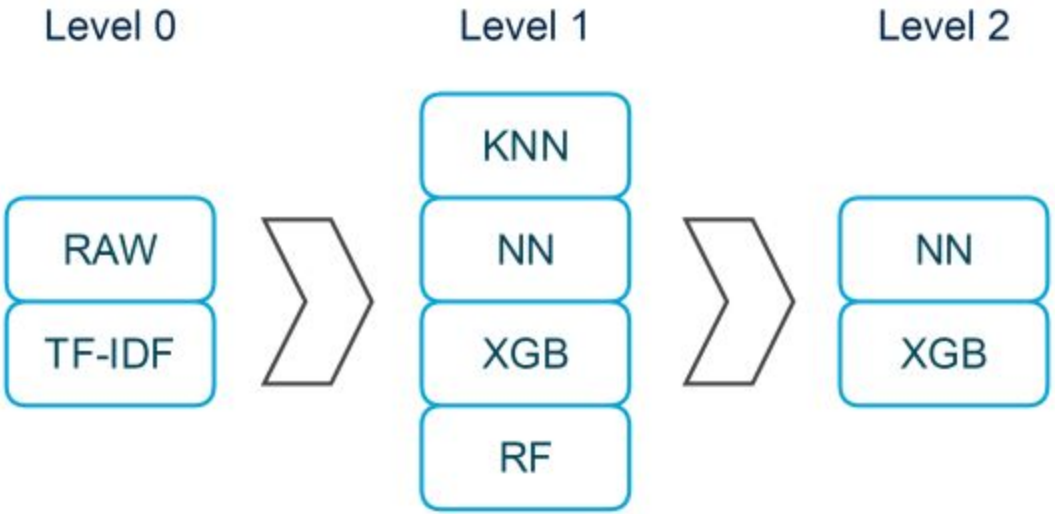
See someone using multiple accounts? Let us know.

#	Δrank	Team Name ‡ model uploaded * in the money	Score	Entries	Last Submission UTC (Best – Last Submission)
1	–	Gilberto Titericz & Stanislav Semenov ‡ *	0.38243	90	Mon, 18 May 2015 23:42:43 (-6.6h)
2	↑1	¯_(ツ)_/¯ ‡ *	0.38656	26	Mon, 18 May 2015 23:31:27 (-1.5h)
3	↓1	i dont know *	0.38667	103	Mon, 18 May 2015 23:53:10 (-8.6h)
4	–	Dmitry & Davut ‡	0.39027	168	Mon, 18 May 2015 21:26:53 (-0.2h)
5	–	Mikhail Trofimov	0.39263	27	Mon, 18 May 2015 21:31:14

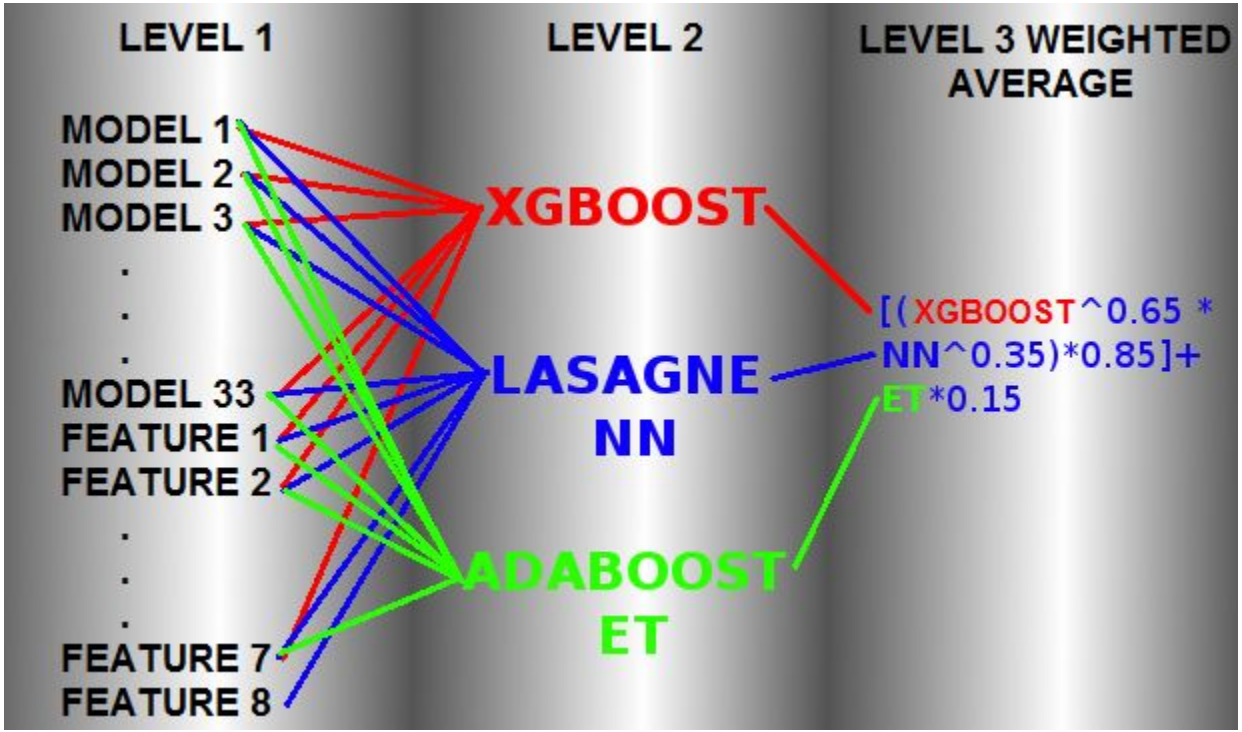
The 3rd place solution



The 2nd place solution



The 1st place solution



Ensemble software

- **R**
  - **caretEnsemble**
  - **h2oEnsemble**
  - **...**
- **Python**
  - **sklearn.Ensemble**
  - **...**

Links and references

- http://sli.ics.uci.edu/Classes/2012F-273a?action=download&upname=10-ensembles.pdf
- http://mlwave.com/kaggle-ensembling-guide/
- http://www.slideshare.net/MoscowDataFest/df1-dmc-trophimov-tips-tricks-and-usecases-of-ensembling-in-practice
- http://mlwave.com/human-ensemble-learning/
- https://github.com/vkantor/MIPT_Data_mining_in_action_2015/blob/master/Slides/Competitions6_Stacking.pdf
- https://github.com/vkantor/MIPT_Data_mining_in_action_2015/blob/master/Slides/Lecture3_Part1_EnsembleModels.pdf
- http://blog.kaggle.com/