Advanced Ensemble Methods

Holly Xie
2019.08.10

About Me

- Master of Mathematics in ***Statistics*** from University of Waterloo

- At RBC I built the next-generation ***credit risk*** measurement models

- Currently a **Machine Learning Scientist** at *integrate.ai*, a SaaS AI startup

Today's Contents

1. **Recap** Ensemble Machine Learning Methods: Bagging, Boosting

2. **Deep dive**: Random Forest, XGBoost, CatBoost, LightGBM, Model Stacking

3. **Imbalanced Classification**: over-sampling and under-sampling techniques

4. Hands-on Lab

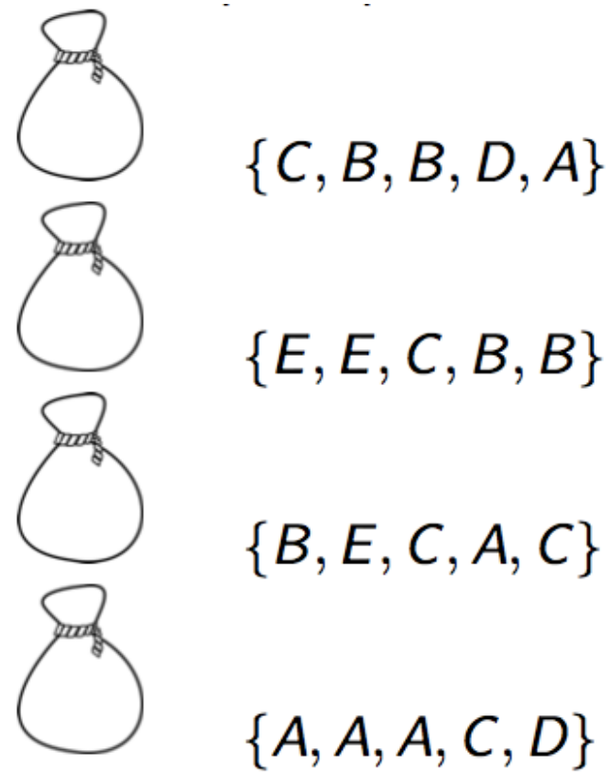
Bagging (**B**ootstrap **Agg**regat**ing**)

- Breiman, *Machine Learning* 1996

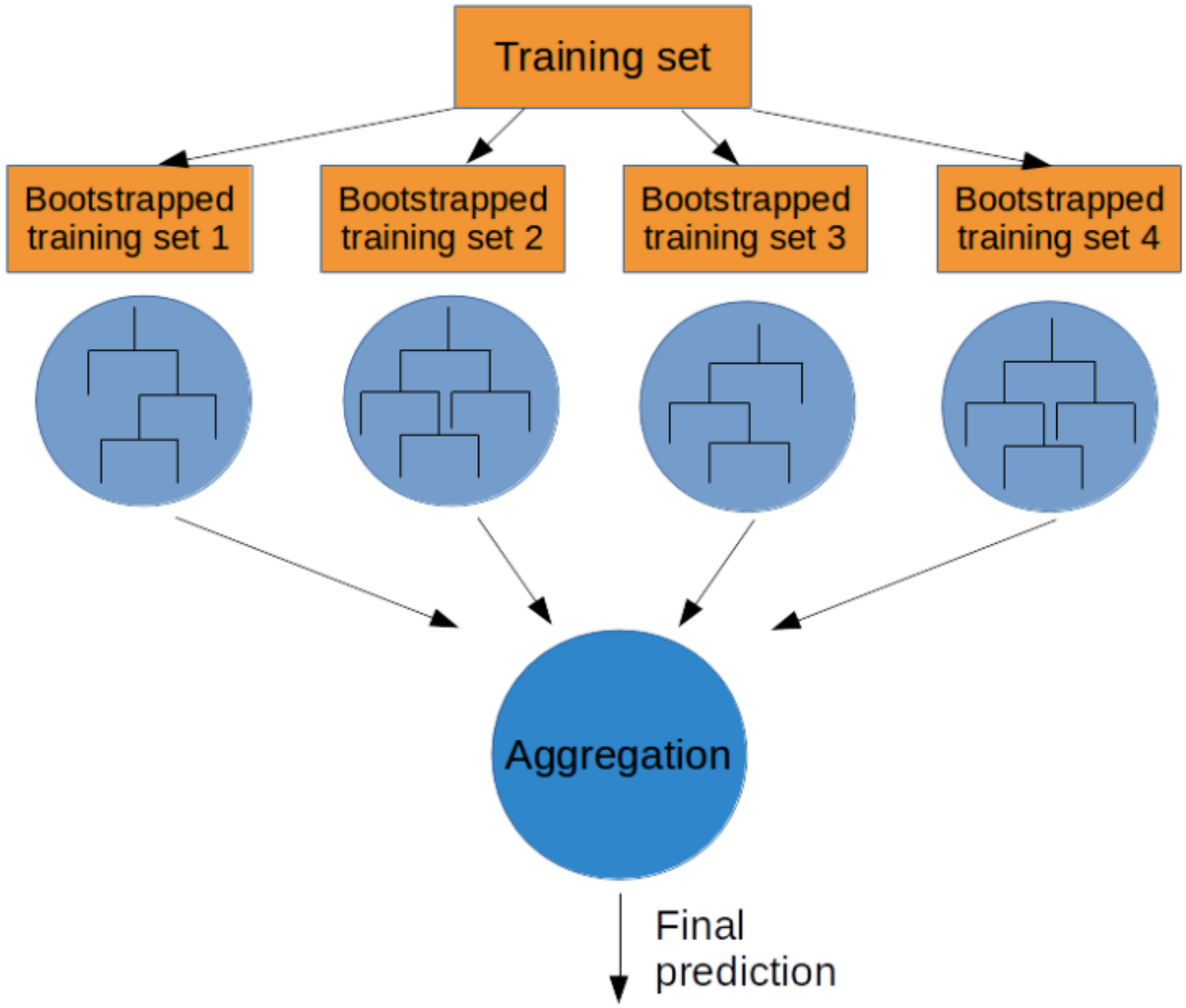
- Bootstrap: random sampling size *N* **with replacement** (end up with some **out-of-bag** samples)

- Example: {A,B,C,D,E}, Bootstrap samples of size 5

- "almost-representative" and "almost-independent"

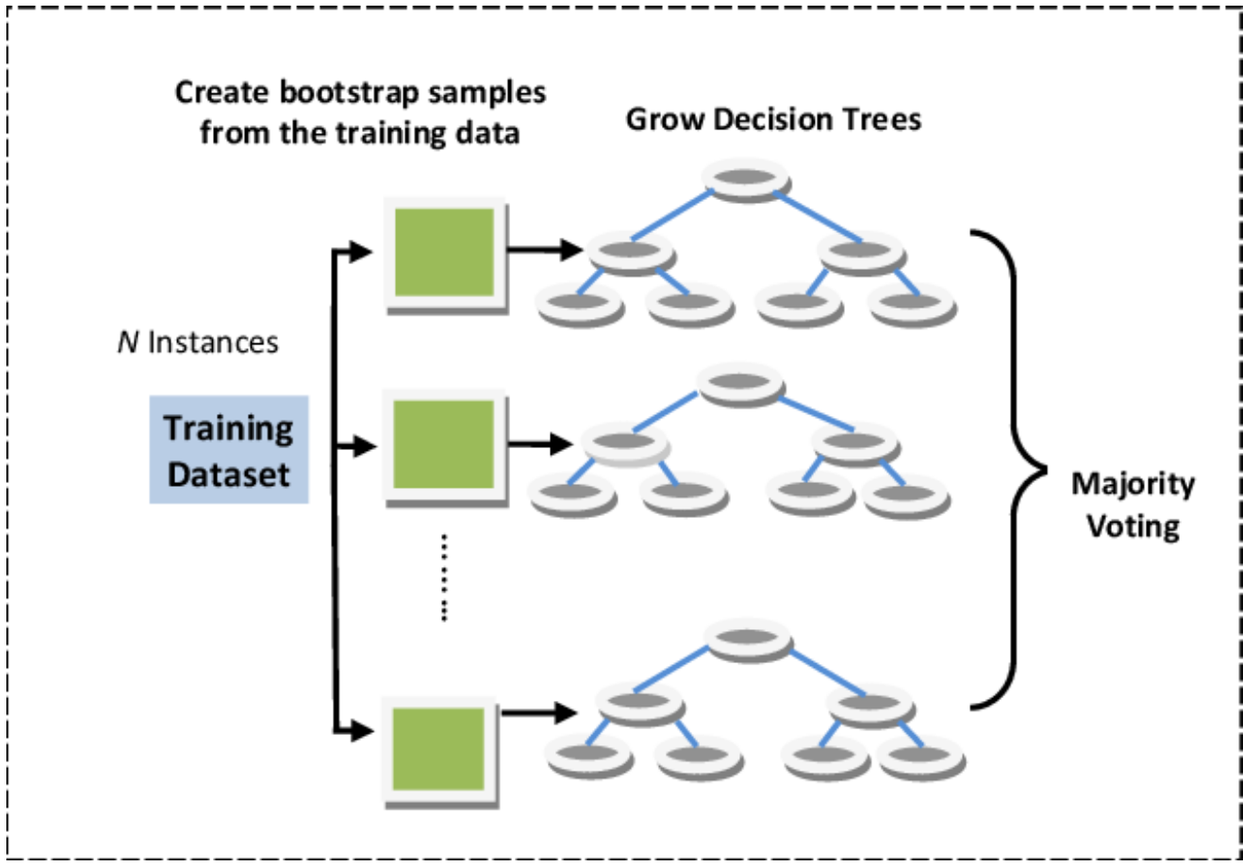


Wisdom of Crowd



Random Forest

- Random Forest is the fancier version of *bagging tree-based* model.
- RF grows **many** classification trees, i.e. a **forest.**
- Grow each tree on an **independent** bootstrap sample from the training data.



Why is it called a fancier version?

At each node:

1. If there are **M** input variables, a number **m<<M** is specified such that at each node, **m** variables are selected at random out of the **M** (independently for each node) and the best split on these **m** is used to split the node. The value of **m** is held constant during the forest growing.

2. Find the best split on the selected **m** variables, same as ordinary decision tree. Grow every single tree to some pre-determined depth.

   To determine the final class, each tree votes for one class. The forest chooses the classification having the most votes over all the tress in the forest.

- Randomness come from the parts:

1. Bootstrap samples (so each tree use different training data to grow)

2. Select m variables randomly (m << M, so each tree is considered as “weak learner”)

Question: Why do we need to introduce much of randomness in the model? (Hint: Bias-variance trade-off)

Tune the parameter *m*

- Random forest error rate depends on the following two things:

1. The correlation between any two trees in the forest. (+)

2. The strength of each individual tree in the forest. (-)

Reducing **m** reduces both the correlation and the strength. Increasing it increase both. Optimal **m** can be found through **OOB (out-of-bag)** rate or Cross Validation.

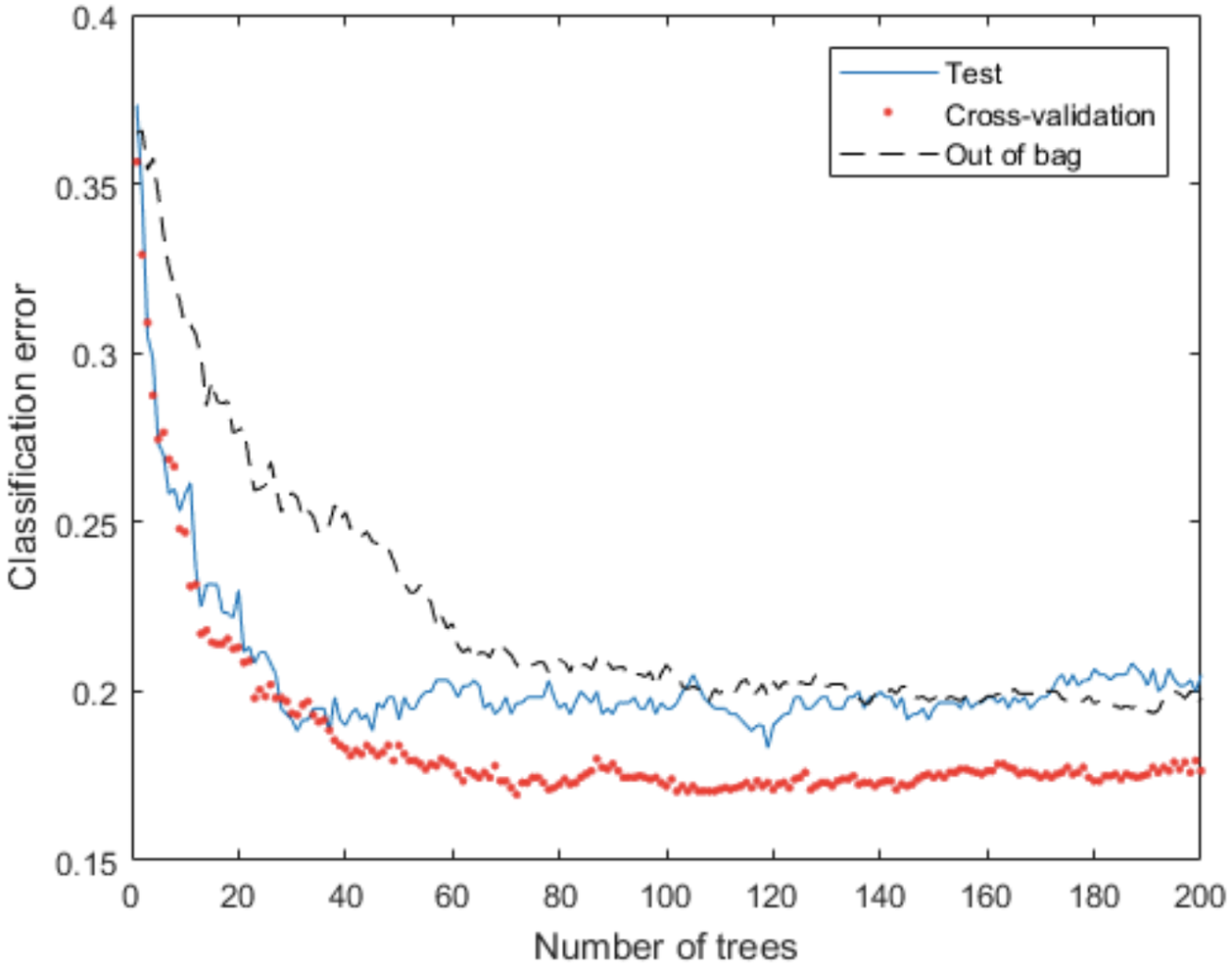
Default is $\sqrt{M}$ in classification case.

"*m* is the only adjustable parameter to which random forest is somewhat sensitive. " (Breiman)

Out-of-Bag Error Rate

- A case in the training data is not in the bootstrap sample for about 1/3 of the trees (i.e. out-of-bag).
- Vote the prediction of these trees give the RF predictors.
- The OOB error rate is the error rate of the RF predictors *across over all samples.*

Example: suppose we fit 10 trees, and a sample is out-of-bag in 3 of them, of which: 2 trees say “non-default” 1 tree says “default”, so the RF predictor is “non-default”, however we can observe the true label for this sample is “**default**”; therefore the error rate for this particular sample is 2/3 = 67%



OOB error rate has proven to be *unbiased* of test dataset error in many tests.

- Depending on your sample size and the value of m, more trees may bring more benefits. However, the marginal benefit is decreasing.
- Growing large trees will not overfit.
- Readily parallelizable

Variable Importance

- In *sci-kit learn* Random Forest, the ***default*** is ***"mean decrease impurity"***

- Every node in the decision tree is a condition on a single feature, designed to split the dataset into two so that similar response values end up in the same set. Recap: *Gini impurity* and *information gain/entropy*

- ***Permutation feature importance:*** random re-shuffling, preserving the distribution of the variable

  - Train the baseline model and record the score (any metric of importance) by passing the OOB set.

  - Re-shuffle values from one feature in the selected dataset, pass the dataset to the model again to obtain predictions and calculate the metric for this modified dataset. The feature importance is the difference between the benchmark score and the one from the modified (permuted) dataset.

  - Repeat until done with all features in the dataset.

Raw importance score for variable *j* = average of this number over all tress in the forest

Default feature importance

- **Pros:** fast calculation; easy to retrieve — one command
- **Cons:** Biased as it has a tendency to inflate the importance of continuous features or high-cardinality categorical variables

Permutation feature importance

- **Pros:** reasonably efficient; reliable technique; no need to retrain the model at each modification of the dataset

- **Cons:** more computationally expensive; permutation importance overestimates the importance of correlated predictors

Proximities

- One of the most useful tools in random forest which can be used in *replacing missing data, locating outliers*, and producing illuminating low-dimensional views of the data.

- **Proximity** of two cases is the proportion of the time that they end up in the same terminal node.

- **How to compute?** The proximities originally formed a *N*N* matrix. After a tress is grown, put all of the data, both training and OOB, down the tree. If two samples are in the same node increase their proximity by one. At the end, normalize the proximities by dividing by the number of tress.

- The proximities don't just measure similarity of the two cases - they also take into account the importance of the variables.

Missing Values in RF

- The fast way: imputation by median (continuous) or the most frequent non-missing value (categorical)

- Proximities-weighted way: (more computationally expensive):

1. Start with the fast way

2. Run a forest and compute proximities. (Note that no missing values are allowed in growing a forest!)

3. Average (or frequency) over the non-missing values weighted by **proximities** between the sample and the non-missing-value samples.

4. Repeat steps 2 and 3 a few times.

Boosting

- A technique that consists in fitting ***sequentially*** multiple weak learners in a very adaptive way

- Each model in the sequence is fitted giving more importance to observations in the dataset that were badly handled by the previous models in the sequence

- Commonly implemented with ***trees***

- Successful implementations: Gradient Boosting

Tree Ensemble

- Very widely used, almost half of data mining competitions are won by using some variants of tree ensemble methods
- Invariant to scaling of inputs, so you do not need to do careful features normalization
- Learn higher order interaction between features.
- Can be scalable, and are used in industry

Bagging v.s. Boosting

	Bagging	**Boosting**
Base models	deep tree	shallow tree
Trade-off	Low bias, high variance —-> reduce variance	High bias, low variance —> reduce bias
Computations	parallelizable —> fast	Not readily parallelizable —> slow

Gradient Boosting

- Original paper: *Greedy Function Approximation: A Gradient Boosting Machine*, by Friedman

- **XGBoost**: an optimized distributed gradient boosting library designed to be highly efficient, flexible and portable. It is consistently used to win machine learning competitions on Kaggle.

Gradient Boosting

- Recall that random forest and boosted trees are not different in terms of model, the difference is how we train them.
- Gradient Boosting = Gradient descent + Boosting

XGBoost: e**X**teme **G**radient **Boost**ing

- XGBoost is developed with both deep consideration in terms of **systems optimization** and **principles in machine learning**.
- System Optimization:
    - parallelization
    - “depth-first” tree pruning
    - hardware optimization
- Algorithmic Enhancements:
    - Sparsity-aware for handling sparse data
    - Regularization to avoid overfitting
    - A theoretically justified ***weighted quantile sketch*** to effectively find the optimal split points among weighted datasets

Boosted Trees Algorithm

- Additive training
- Minimize the *regularized* objective
- The **scoring function**, which is used in practice for evaluating the split candidates, is different from Gini impurity and the Information gain
- See the **objective function** derivation on the whiteboard

Prevent overfitting in XGBoost

- Two additional **techniques** are used in XGBoost to further prevent **overfitting**:

    - **Shrinkage**: scales newly added weights by a factor η after each step of tree boosting. Similar to a learning rate in stochastic optimization, shrinkage reduces the influence of each individual tree and leaves space for future trees to improve the model.

    - **Column Subsampling**: also used in Random Forest

Sparsity-aware algorithm in XGBoost

- When a value is missing in the sparse matrix x, the instance is classified into the default direction.
- The optimal default directions are learnt from the data.

Split Finding Algorithms

Basic Exact Greedy Algorithm

- enumerates over all the possible splits on all the features
- most existing single machine tree boosting implementations, such as *scikit-learn* as well as the single machine version of XGBoost
- computationally demanding

Approximate Algorithm

- The algorithm first proposes candidate splitting points according to percentiles of feature distribution, then maps the continuous features into buckets split by these candidate points, aggregates the statistics and finds the best solution among proposals based on the aggregated statistics
- Hyper-parameter: *sketch_eps*. This roughly translates into 1/*sketch_eps* number of bins

Parameter Tuning

- Control overfitting: *max_depth, min_child_weight, min_split_loss, subsample, colsample_bytree*

- Handle **Imbalanced dataset**: if care about the model overall performance (AUC), considering *scale_pos_weight (control the balance of positive and negative weights, useful for unbalanced classes, default: 1).* If care about predicting the right probabilities, considering setting the parameter *max_delta_step (maximum delta step we allow each leaf output to be, default: 0)* to a finite number between 1-10 to help convergence

- Official Parameters Page

- Parameter tuning is a ***dark art*** in machine learning, the optimal parameters of a model can depend on many scenarios. So it is ***nearly impossible*** to create a complete and comprehensive guide for doing so.

Break

Model Stacking

- Stacked Generalization (stacking) is an ensemble learning technique, which learns an ensemble classifier based on the output of multiple base classifiers.

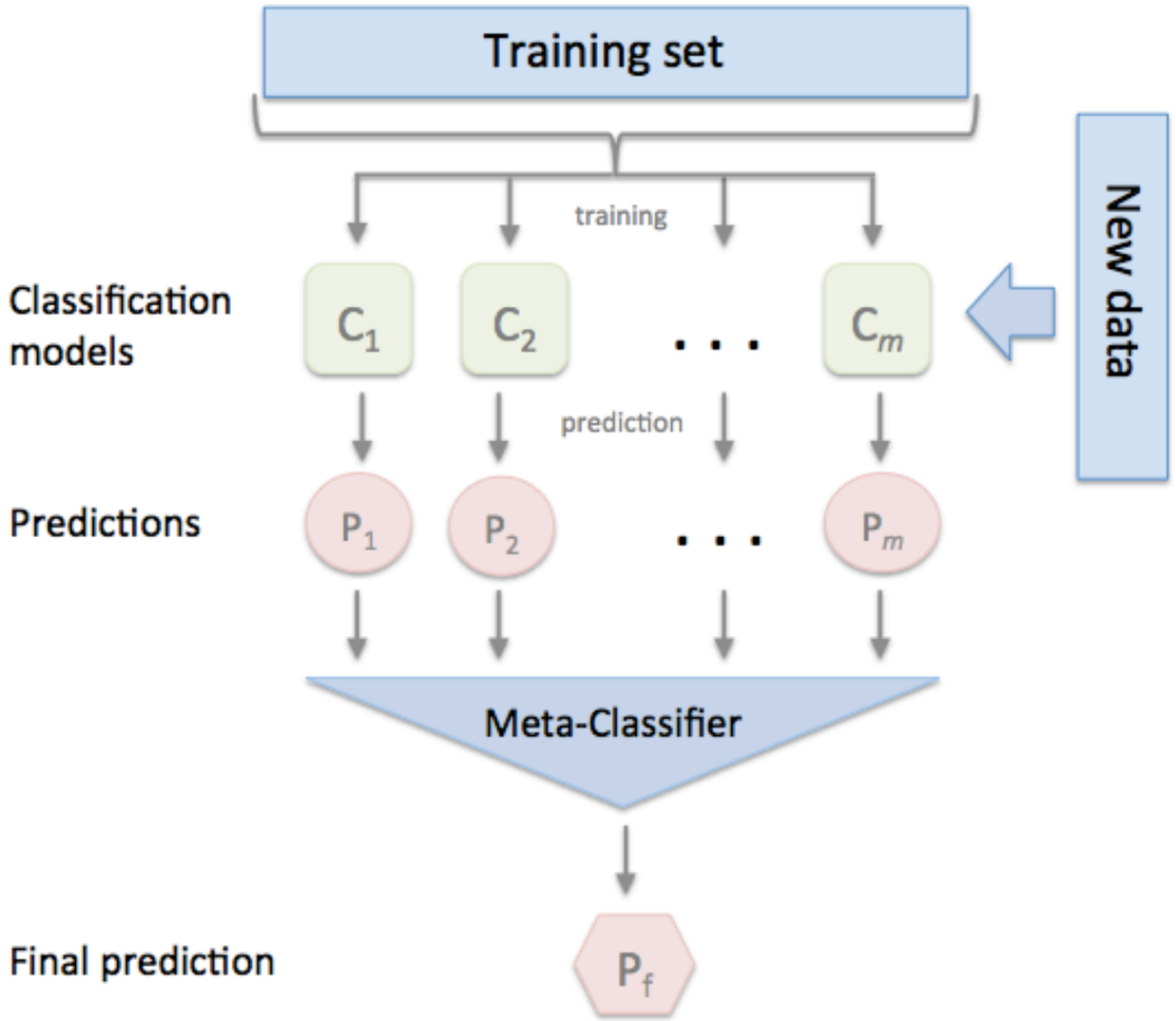
- Meta-learning method in which the base classifiers are called first-level classifiers and a second-level classifier is learnt to combine the first-level classifiers

- Introduced in 1992 by Wolpert. Shown great success in the Netflix competition. The winning team adopted stacking to combine hundreds of different models.

Standard Model Stacking

- The individual classification models are trained based on the complete training set (can also apply Bootstrap sampling); then, the meta-classifier is fitted based on the outputs of the individual classification models in the ensemble.

- Tend to overfitting due to data leakage



CV Model Stacking

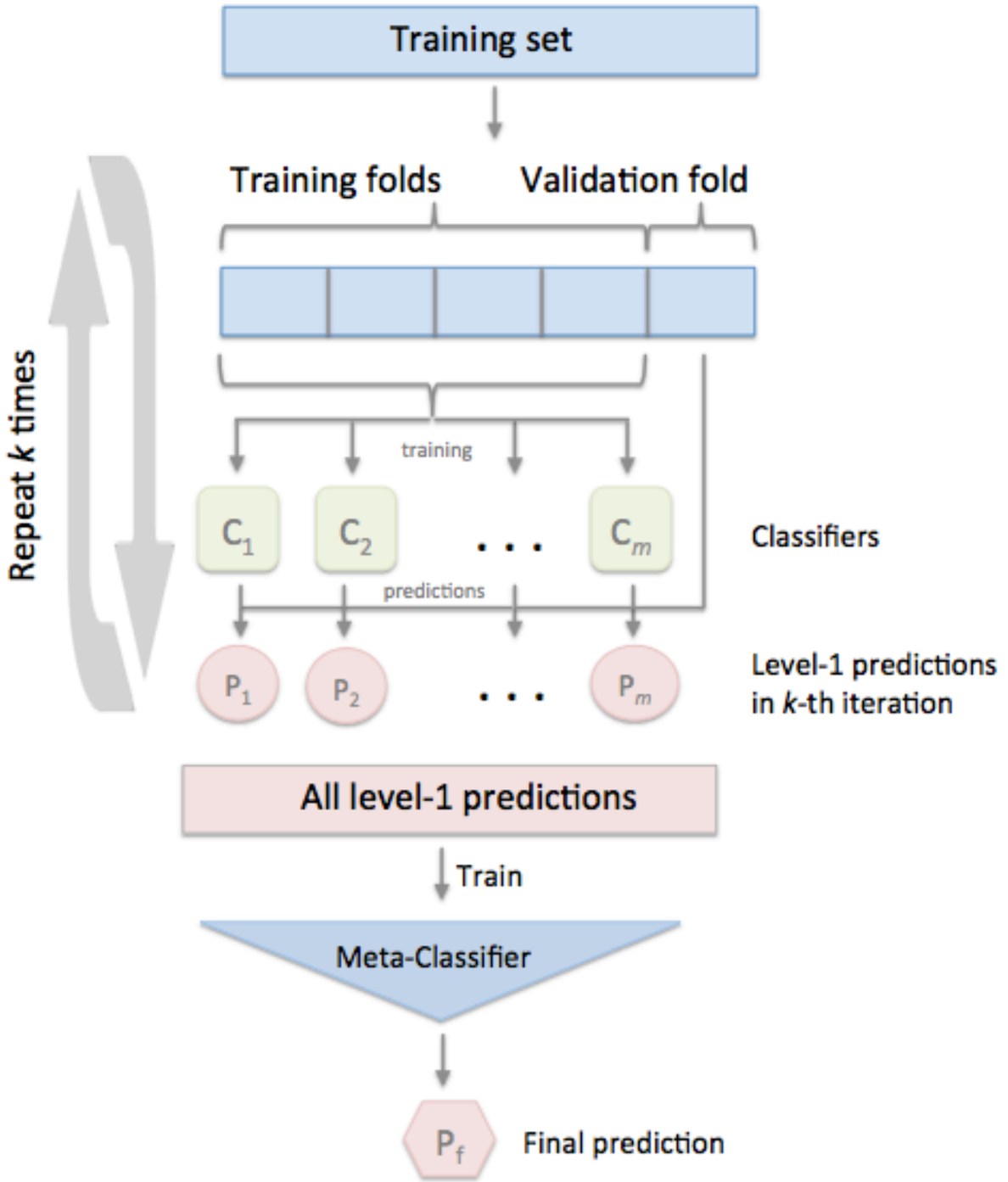
- The dataset is split into *k* disjoint folds and run the learning algorithm *k* times. *k-1* folds are used to fit the first level classifier; in each round, the first-level classifiers are then applied to the remaining 1 subset. The resulting predictions are then stacked and provided -- as input data -- to the second-level classifier.

- How to predict on new data?

  - First generate predictions from the base learners

  - Feed those predictions into the meta learner to generate the ensemble prediction

- Reference: https://github.com/rasbt/mlxtend



CatBoost

- Successfully handles categorical features
- There are two ways to deal with categorical features: (1) One Hot Encoding (2) Computing some statistics using the label values of the examples
- Overfitting problem associated with the latter
- Consider using it when the dataset has a lot of categorical features

LightGBM

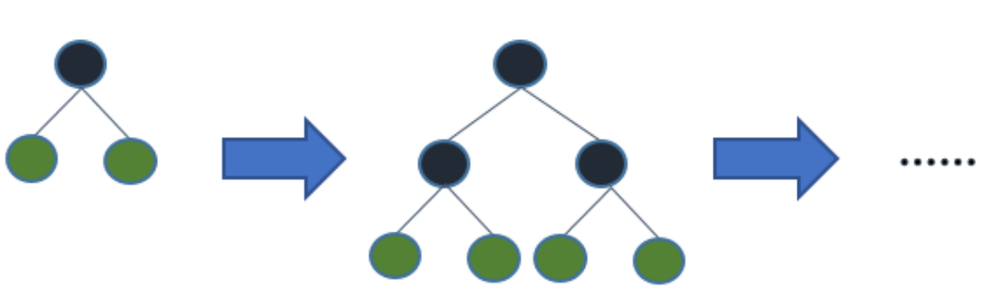
- Designed to deal with large number of data instances and large number of features respectively

- It produces much more complex trees by following leaf-wise split approach rather than a level-wise approach which is the main factor in achieving higher accuracy. However, it can sometimes lead to ***overfitting*** which can be avoided by setting the *max_depth* parameter.

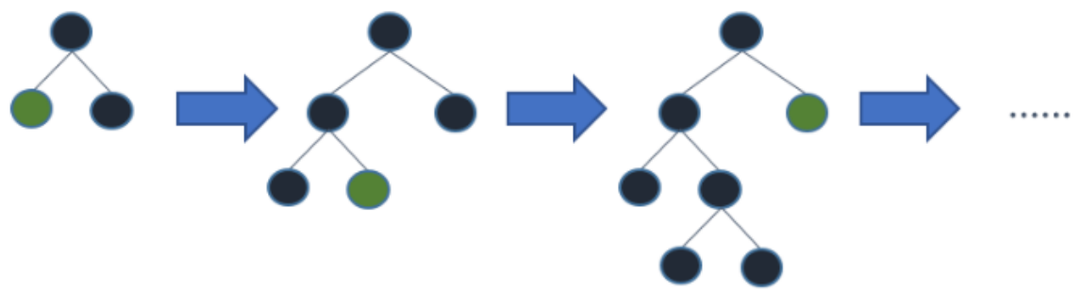
- The experiments on multiple public datasets show that LightGBM can significantly outperform XGBoost in terms of computational speed and memory consumption

Light GBM v.s. XGBoost and CatBoost

LightGBM differs from XGBoost and CatBoost in how it prioritizes which nodes to split. LightGBM decides on splits leaf-wise, i.e., it splits the leaf node that maximizes the information gain, even when this leads to unbalanced trees. In contrast, XGBoost and CatBoost expand all nodes depth-wise and first split all nodes at a given depth before adding more levels. The two approaches expand nodes in a different order and will produce different results except for complete trees.



Level-wise tree growth



Leaf-wise tree growth

Imbalanced Classification

- Commonly seen in real-world
- Extremely imbalanced classification example: default rate can be as low as 0.01%

- Question: Why it is an issue that we need to concern about?

Imbalanced Classification

- **Problem:** tend to provide a severely imbalanced degree of accuracy.

- **Solution:** imbalanced learning algorithm!

1. **Sampling methods:** down-sampling majority class or over-sampling minority or both.

2. **Cost Sensitive Methods:** assigning a high cost (i.e. weight) to *misclassification of minority class* and minimizing the overall error rate.

Over-sampling

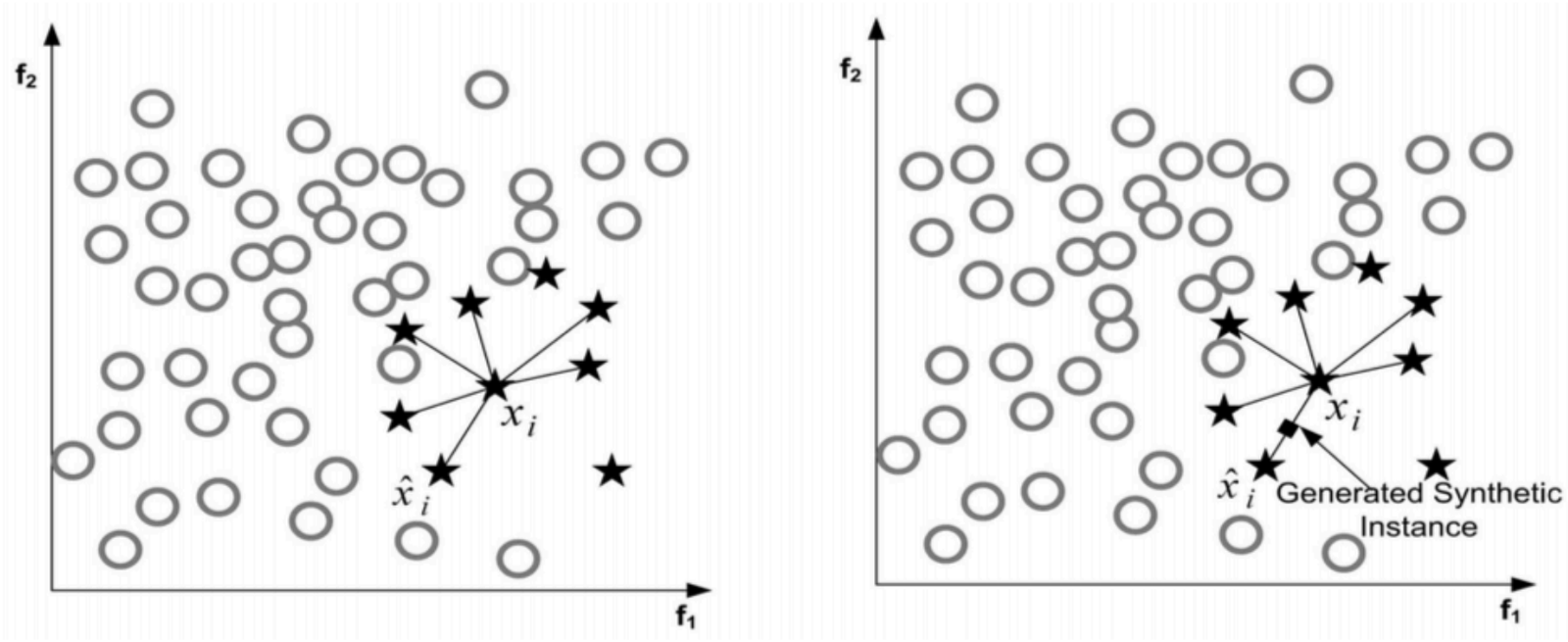
- For a set of randomly selected *E* sample from the minority class, augment the original set *S* by replicating selected minority samples.
- **Problem:** over-fitting; large variance
- Commonly used implementation: ***SMOTE***

SMOTE

- **S**ynthetic **M**inority **O**ver-sampling **TE**chnique

- Idea: create artificial data based on the feature space similarities between existing minority samples.

- SMOTE thinks from the perspective of existing minority instances and synthesizes new instances at some distance from them towards one of their neighbours. For example K = 5 shown below



$$x_{new} = x_i + (\hat{x}_i - x_i) \times \delta \qquad x_i \in S_{\min}$$

Down-sampling

- The majority class is under-sampled by randomly removing samples from the majority class population until the minority class becomes some specified percentage of the majority class.
  Example: #of non-defaulters: #of defaulters = 1:1

- **Problem:** loss of information

- Usually worse performance than Over-sampling

Stay in touch!

Email: xhonglei2007@gmail.com
LinkedIn: http://www.linkedin.com/in/hongleixie
My Personal Blog: http://hongleixie.github.io
Github: http://github.com/HongleiXie

Lab