

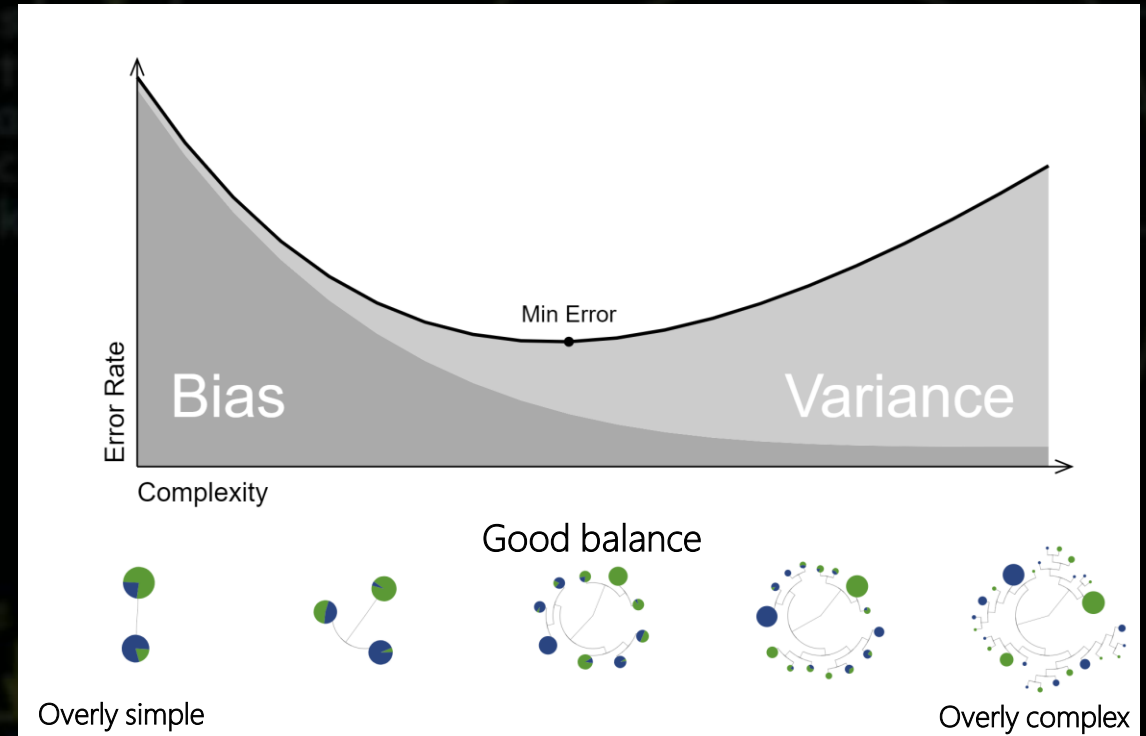
How to win a Kaggle competition

Part 4: model tuning

Chris Chen@Calgary Data Science Academy

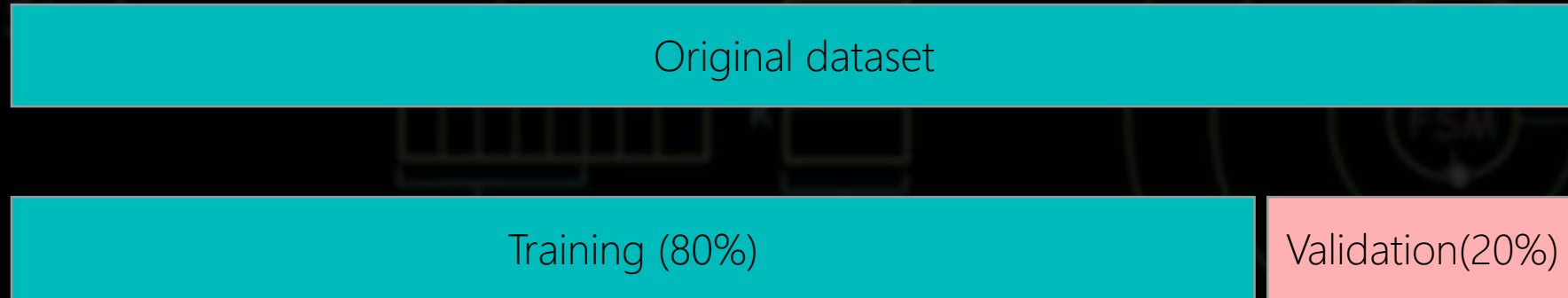
Why model tuning?

- Models approximate real-life situations by learning from limited data (training)
- The way for a model to learn from the data can be changed by adjusting the a few “settings”, which are called “hyper-parameters”, e.g. the number of nodes of a decision tree.
- Models can suffer from both overly-simple settings (under-fitting, high bias vs high variance) and overly-complex settings (over-fitting, low bias vs high variance)
- The purpose of model tuning is to find a good balance between the two.

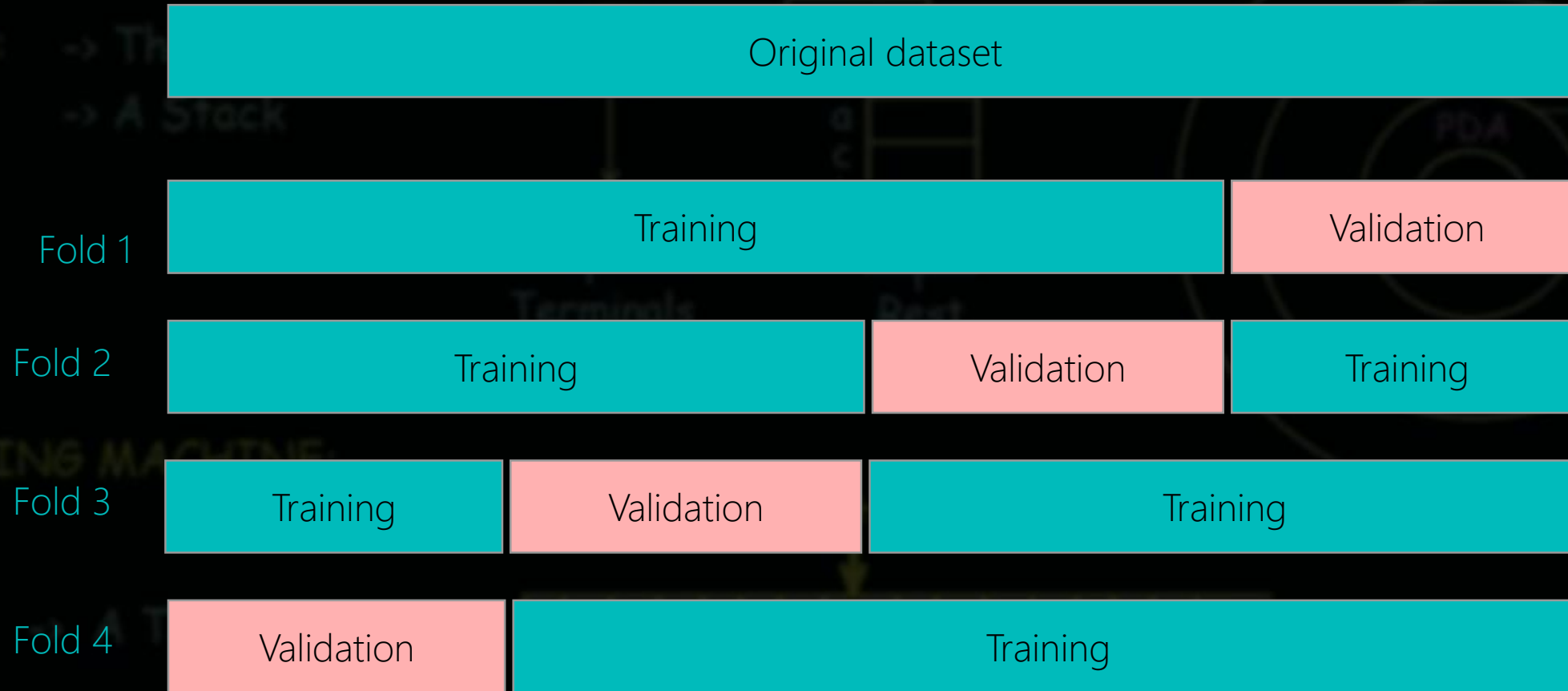


<http://www.r2d3.us/visual-intro-to-machine-learning-part-2/>

Holdout validation

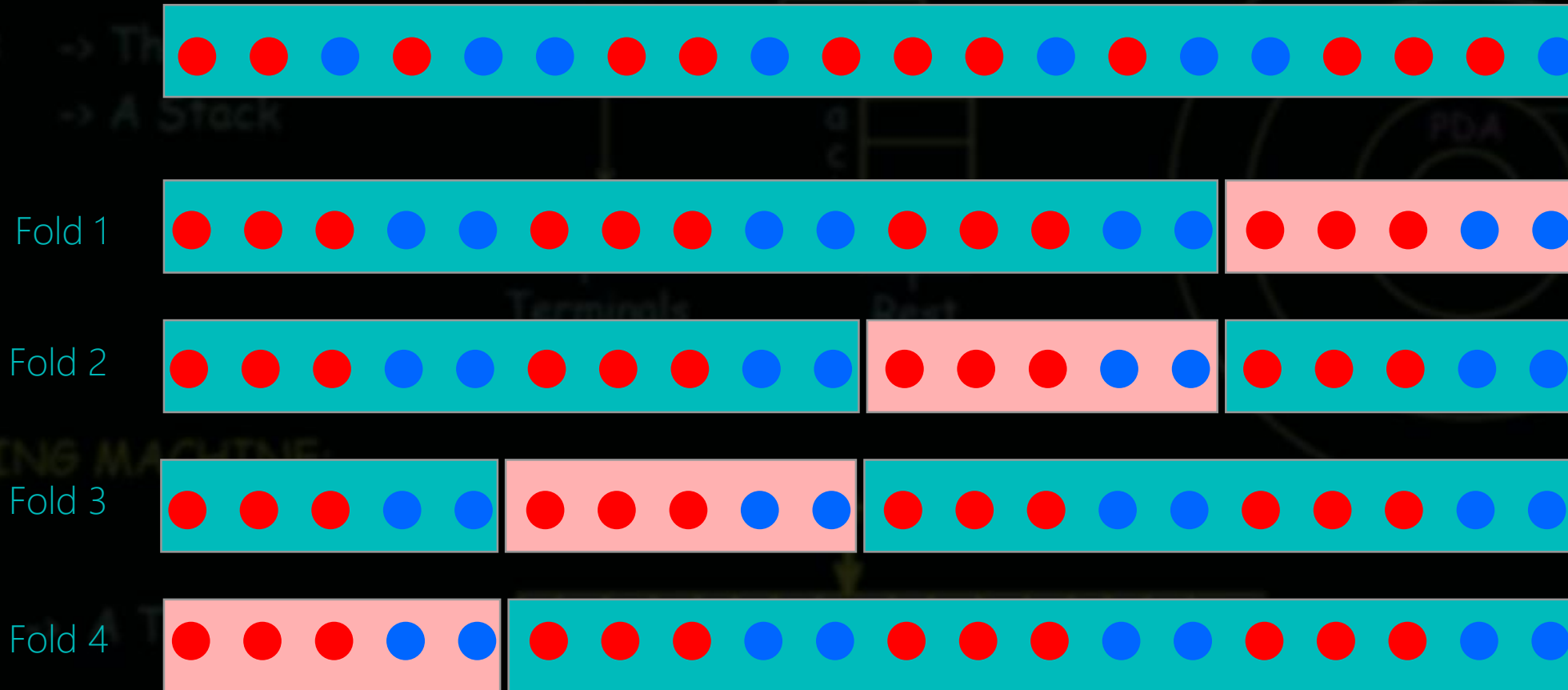


K-fold cross validation



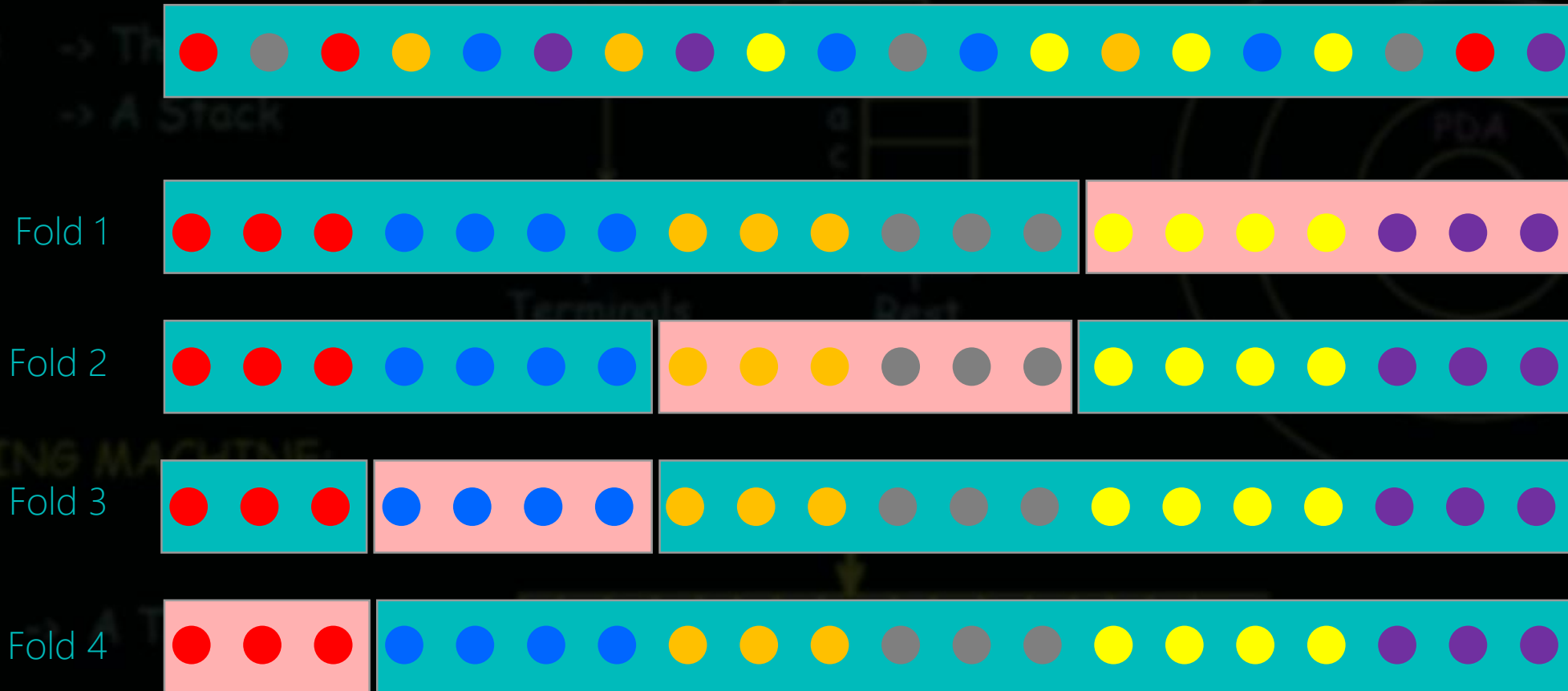
Cross validation score = mean([score of fold 1, score of fold 2, score of fold 3, score of fold 4])

K-fold stratified cross validation



- Stratified splitting works for classification problems.
- Each training and validation dataset has the same or very close proportion of classes on the target (dependent) variable.

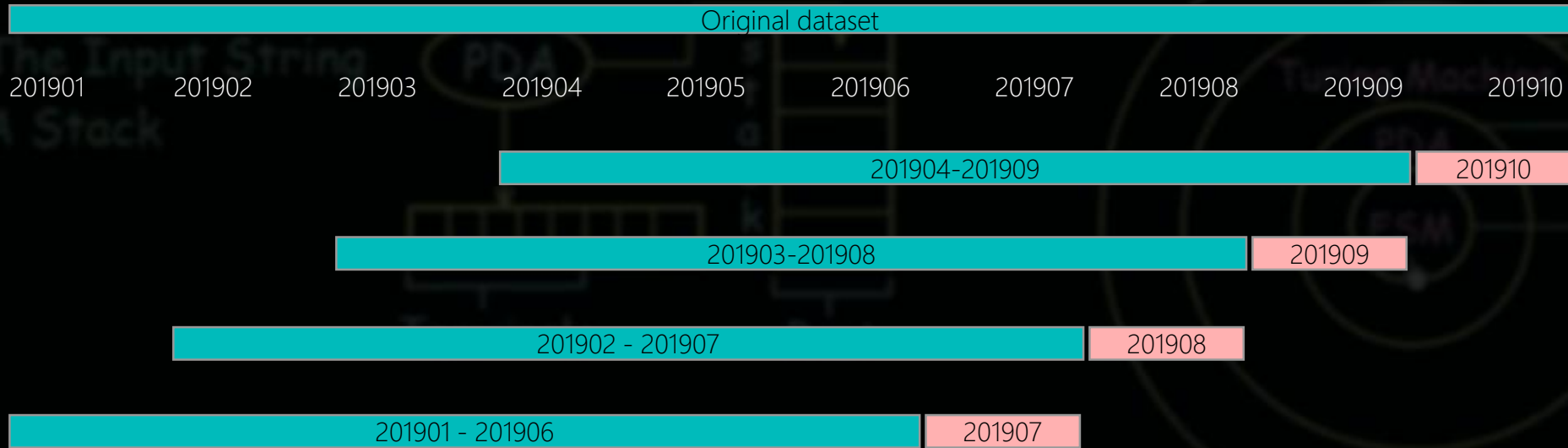
K-fold group cross validation



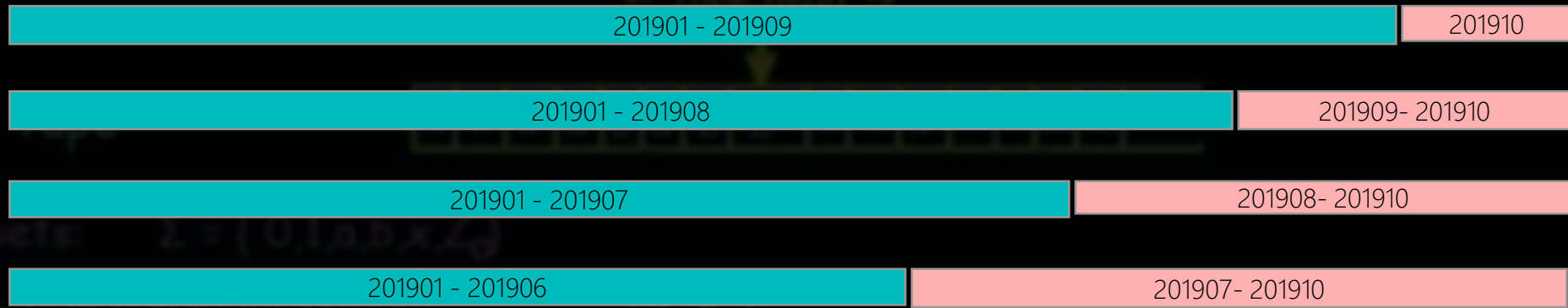
- Group splitting works for datasets that can naturally be grouped and samples within each group are inherent correlated to each other, e.g. credit card transactions grouped by card id.
- Samples from the same group stay together in either training or validation set to avoid data leakage.

Validation for time series

Method A

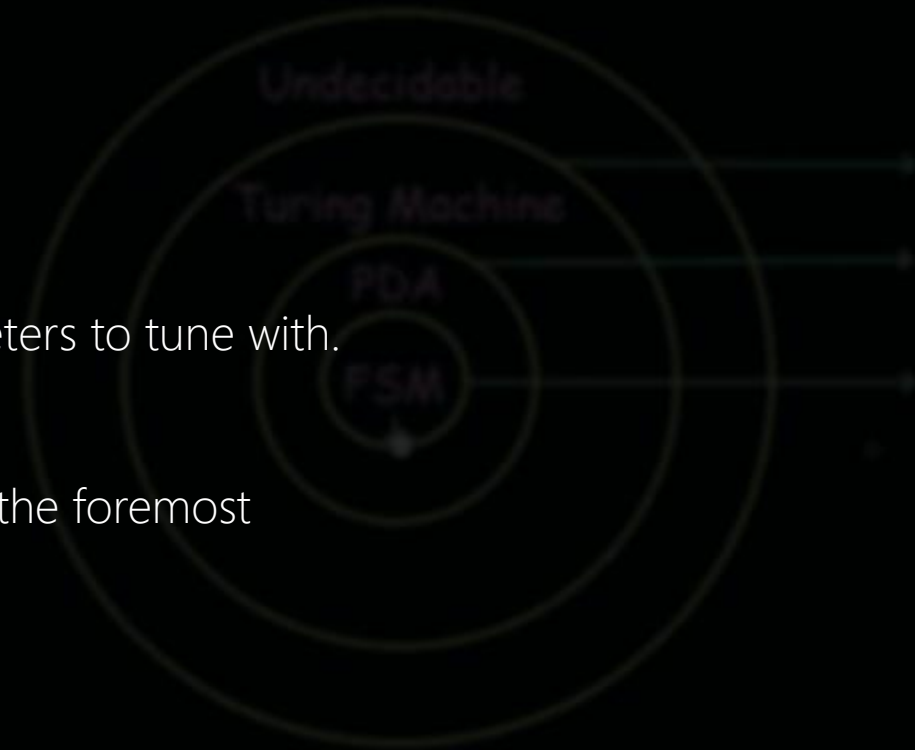


Method B



Model tuning challenge

- Model tuning is very time consuming
 - A model may have a few to a hundreds of hyper-parameters to tune with.
Exhausting all the possible combinations is impossible.
- How to efficiently find optimal values for hyper parameters is the foremost challenge for model tuning.
- Common model tuning strategies:
 - Grid search
 - Random search
 - Greedy search
 - Bayesian optimization



Greedy search

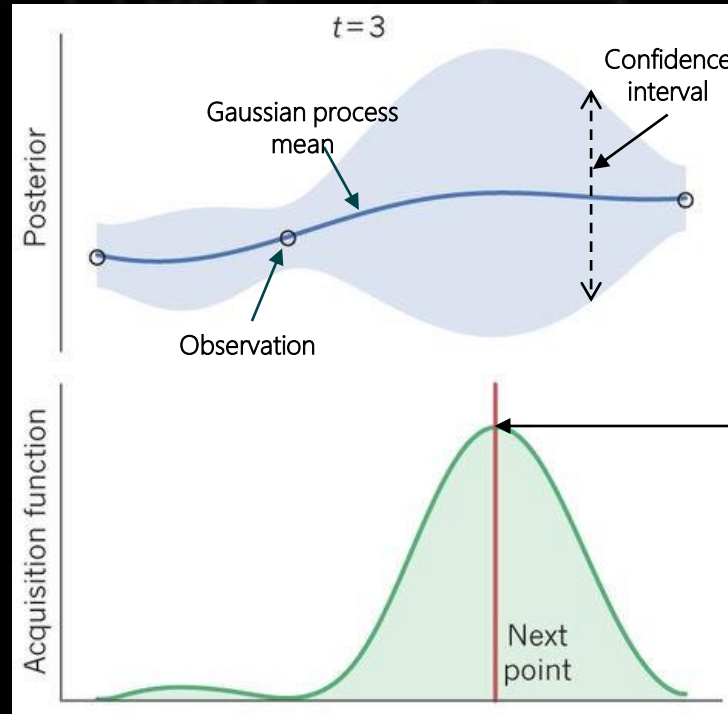
1. **Prioritize** hyper-parameters based on the arbitrary importance. Good understanding about algorithms, mechanism of each hyper-parameters and the data are required.
2. Tune one hyper-parameter at a time. Once the optimal value was found, move to the next hyper-parameter
3. Iterate the process for all hyper-parameters

Bayesian Optimization

- It's similar to the supervised learning where a model tries to fit a function $y=f(w, x)$, where x is the data and w is the parameters of the model. Gradient decent is used to find the optimal values for w that minimizes training errors.
- The way a model fits the data is controlled by a set of hyper parameters h , so a model's performance can be expressed as a function $e=g(h, x)$, where x is the data and h is the hyper parameters. But can we use gradient decent for finding the optimal values for h ?
 - The answer is no, because the performance function $g(h, x)$ is:
 - a black box (we don't know the form)
 - derivative-free.
 - expensive to compute.
 - noisy
- Bayesian optimization to the rescue!
 - It is a sequential design strategy for global optimization of black-box functions[1] that doesn't require derivatives.

Bayesian Optimization

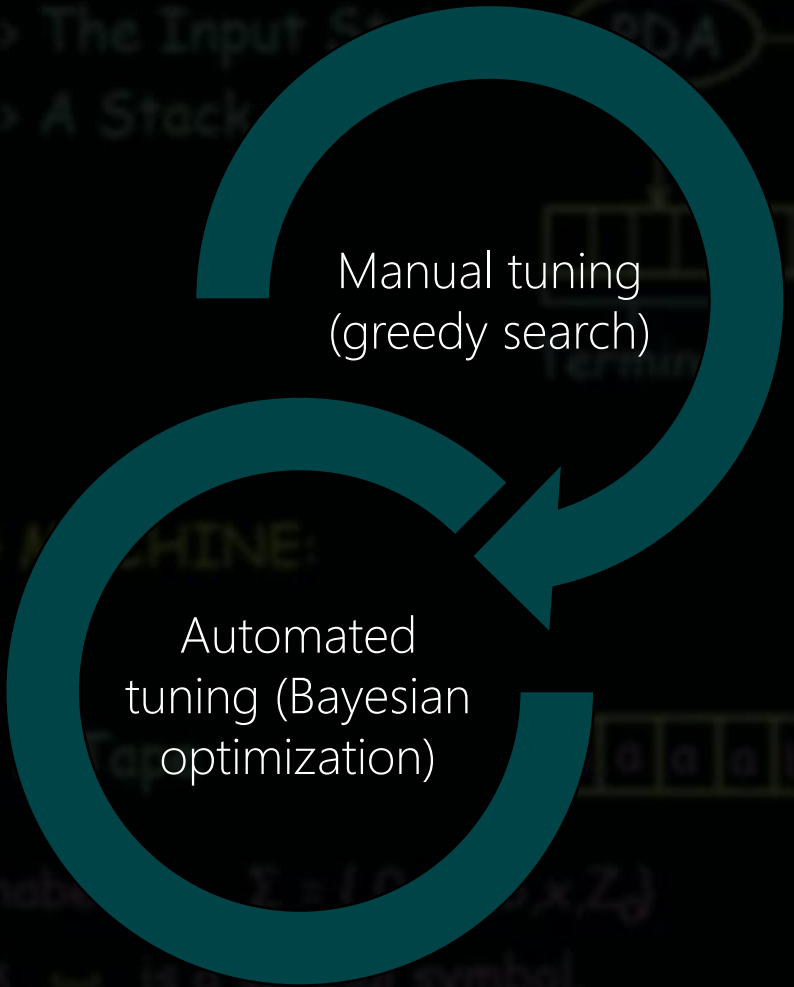
- Horizontal axis: value of a hyper-parameter
- Vertical axis: value of model's performance, e.g. accuracy
- Dots: observations. In the context of model tuning, each observation is a validation based on specific hyper-parameters. E.g. when tree depth = 3, model accuracy = 0.75
- Solid line: gaussian process mean estimated based on existing observations
- Blue shadow: confidence interval estimated based on existing observations.
- Green line: acquisition function.



<https://towardsdatascience.com/shallow-understanding-on-bayesian-optimization-324b6c1f7083>

- When 3 random observations (validations) were calculated, we can use gaussian process to estimate the values between each two observations (blue line), as well as the confidence interval(blue shadow).
- The question now becomes "how can we choose the next point for observation?"
- The next point should have a high mean (exploitation) and high variance (exploration).
- An acquisition function is used to help choose next observation as it takes account of both exploitation and exploration and makes a good balance between the two.

A proposed hybrid approach



Manual tuning
(greedy search)

Automated
tuning (Bayesian
optimization)

Manual tuning:

- Quick and not-so-bad result
- Estimate the range of each hyper parameter
- Need to prioritize hyper parameters

Automated tuning:

- Less intervention
- Typically better result than manual tuning
- Can leverage the range of hyper parameters estimated by manual tuning
- Iterate the process to fine-tune the model

What is XGBoost?

XGBoost stands for eXtreme Gradient Boosting. It is an implementation of **gradient boosting machines** created by Tianqi Chen, now with contributions from many developers. It belongs to a broader collection of tools under the umbrella of the Distributed Machine Learning Community or DMLC who are also the creators of the popular mxnet deep learning library.

"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

"As the winner of an increasing amount of Kaggle competitions, XGBoost showed us again to be a great all-round algorithm worth having in your toolbox."

— Dato Winners' Interview: 1st place, Mad Professors

"When in doubt, use xgboost."

— Avito Winner's Interview: 1st place, Owen Zhang

"I love single models that do well, and my best single model was an XGBoost that could get the 10th place by itself."

— Caterpillar Winners' Interview: 1st place

"I only used XGBoost."

— Liberty Mutual Property Inspection, Winner's Interview: 1st place, Qingchen Wang

"The only supervised learning method I used was gradient boosting, as implemented in the excellent xgboost package."

— Recruit Coupon Purchase Winner's Interview: 2nd place, Halla Yang

XGBoost features

- Using Taylor expansion for loss function approximation

$$x(i+1) = x(i) - \nabla f(x(i))$$

- Allows the model to gather more information (second order derivative) to decide which direction to go.
- Enables customization of loss function as long as the loss function is second-order derivable.

- Regularized learning objective

$$\text{Obj}(\Theta) = L(\Theta) + \Omega(\Theta)$$

where Ω is the regularization term which is not included by most GBDT algorithms.

- Parallel learning enabled by column block

- Data is pre-sorted and stored in “blocks” so they won’t have to be sorted again for new trees.
- Parallelization is done when XGBoost tries to find optimal splitting point for each feature. As a comparison, Random Forest parallizes the process for constructing trees.

- Sparse Aware implementation with automatic handling of missing data values.
- Out-of-Core Computing for very large datasets that don’t fit into memory.
- Handles missing value gracefully

Tree complexity

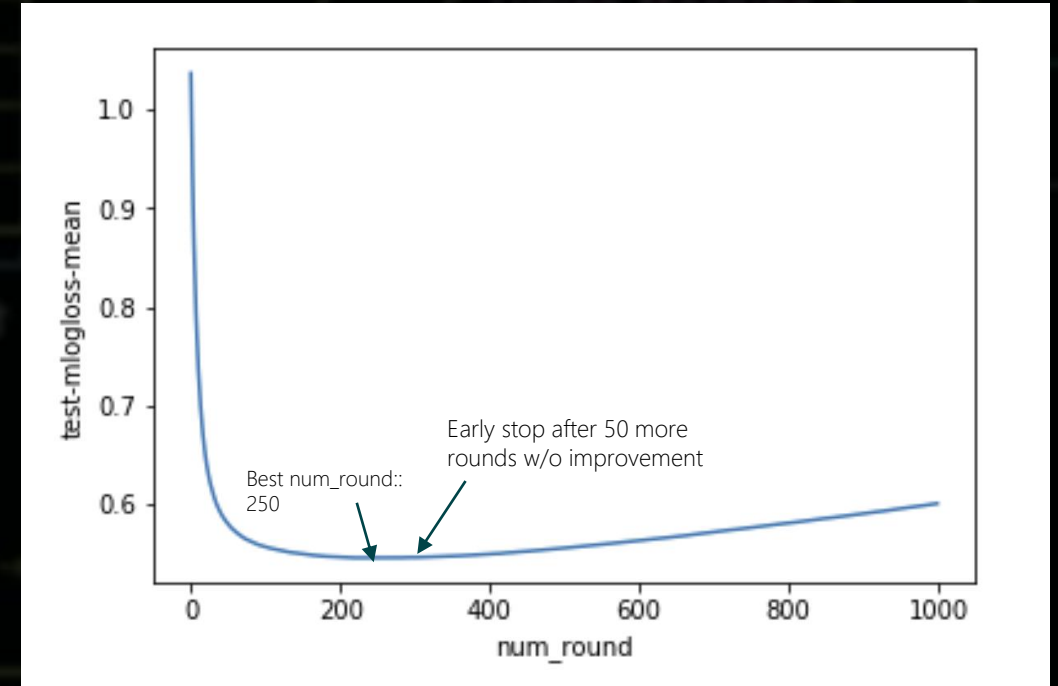
- `max_depth` [default=6]
 - Maximum depth of a tree. Increasing this value will make the model more complex and more likely to overfit. 0 is only accepted in lossguided growing policy when `tree_method` is set as `hist` and it indicates no limit on depth. Beware that XGBoost aggressively consumes memory when training a deep tree.
 - Range: $[0, \infty]$ (0 is only accepted in lossguided growing policy when `tree_method` is set as `hist`)
 - Typical value: 3-10
 - Question:
 - What is the typical value for `max_depth` in Random Forest? What about naïve decision tree?
 - Why the difference?
- `min_child_weight` [default=1]
 - Minimum sum of instance weight (hessian) needed in a child. If the tree partition step results in a leaf node with the sum of instance weight less than `min_child_weight`, then the building process will give up further partitioning. In linear regression task, this simply corresponds to minimum number of instances needed to be in each node. The larger `min_child_weight` is, the more conservative the algorithm will be.
 - Range: $[0, \infty]$
 - Typical value: 10-100
 - Question:
 - If the value was tuned on a dataset of 1M samples, would it still work for production dataset which has 100M samples?
- `gamma` [default=0, alias: `min_split_loss`]
 - Minimum loss reduction required to make a further partition on a leaf node of the tree. The larger `gamma` is, the more conservative the algorithm will be.
 - range: $[0, \infty]$

Stochasticity

- `subsample` [default=1]
 - Subsample ratio of the training instances. Setting it to 0.5 means that XGBoost would randomly sample half of the training data prior to growing trees. and this will prevent overfitting. Subsampling will occur once in every boosting iteration.
 - Range: (0,1]
 - Typical value: 0.6 – 1
 - Question:
 - In the case the tuned value is 0.3 for subsample, what does it indicate?
- `colsample_bytree`
 - Subsample ratio of columns when constructing each tree. Subsampling occurs once for every tree constructed.
 - Range: (0,1]
 - Typical value: 0.1 – 0.9
 - Question:
 - In the case the tuned value is 0.1 for subsample, what does it indicate?
 - If the dataset contains lots of sparse features, what could be the optimal value for `colsample_bytree`?

Learning rate vs number of round

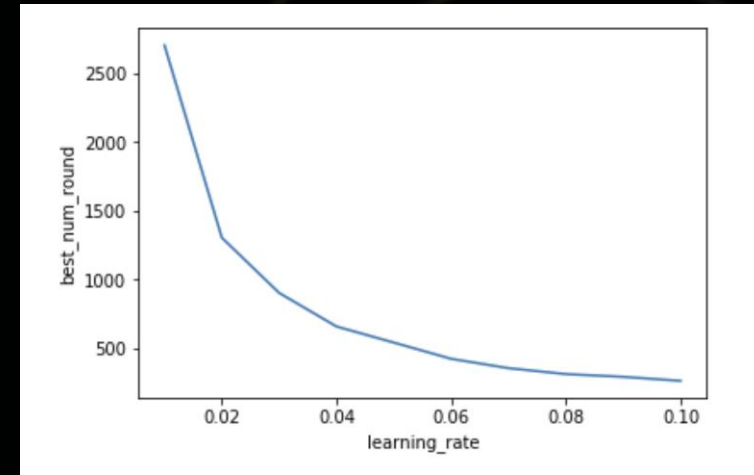
- `num_round` (alias: `n_estimators`, `num_of_round`)
 - The number of rounds (or number of decision trees) for boosting.
 - Range: $[0, \infty]$
 - An XGBoost would typically need many rounds to converge (reach out to the optimal). However, if too many rounds were used to train the model it may overfit.
 - The optimal value can be tuned using early stopping schema which checks metrics validated on the validation set after each new tree was trained. The the model didn't get improved after a specific number of new rounds(trees), XGBoost will stop training.
 - In practice, we may want to use a **larger learning rate** (0.1) to tune other parameters and once optimal values were identified we can switch gear to a **smaller learning rate** (0.01) for better performance.



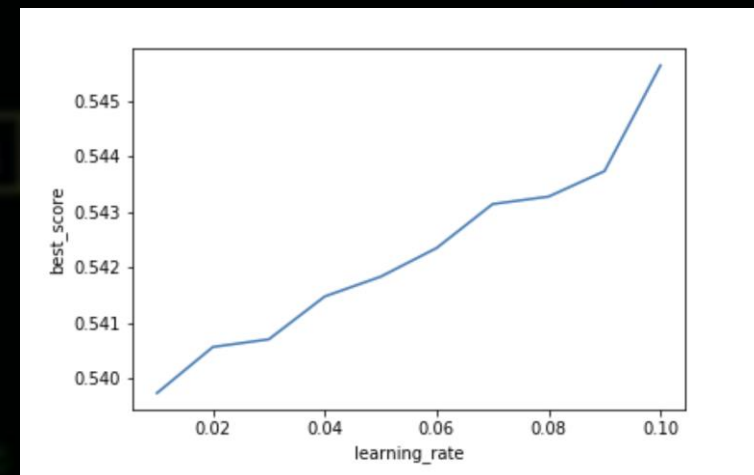
Learning rate vs number of round

- `eta` [default=0.3, alias: `learning_rate`]
 - Step size shrinkage used in update to prevents overfitting. After each boosting step, we can directly get the weights of new features, and `eta` shrinks the feature weights to make the boosting process more conservative.
 - Range: [0,1]
 - The larger the learning rate is, the fewer rounds will be needed for the model to converge, however, the accuracy may suffer from underfitting, and vice versa.
 - In practice, we can use a **larger learning rate** (0.1) to tune other parameters. Once other parameters were tuned we can switch gear to a **smaller learning rate** (0.01) to train the actual model for better performance.

of rounds needed by learning rate

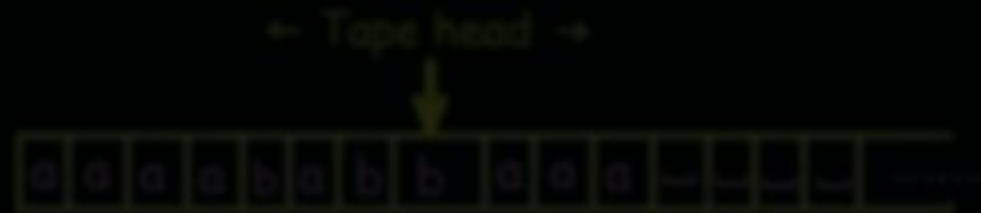


CV logloss by learning rate



LightGBM overview

- Best alternative to XGBoost (my new power horse)
- 4 – 7x faster than XGBoost (with comparable settings)
- Subproject of Microsoft's DMTK project.
- Developed by the winning team of Didi Data Science Competition 2017



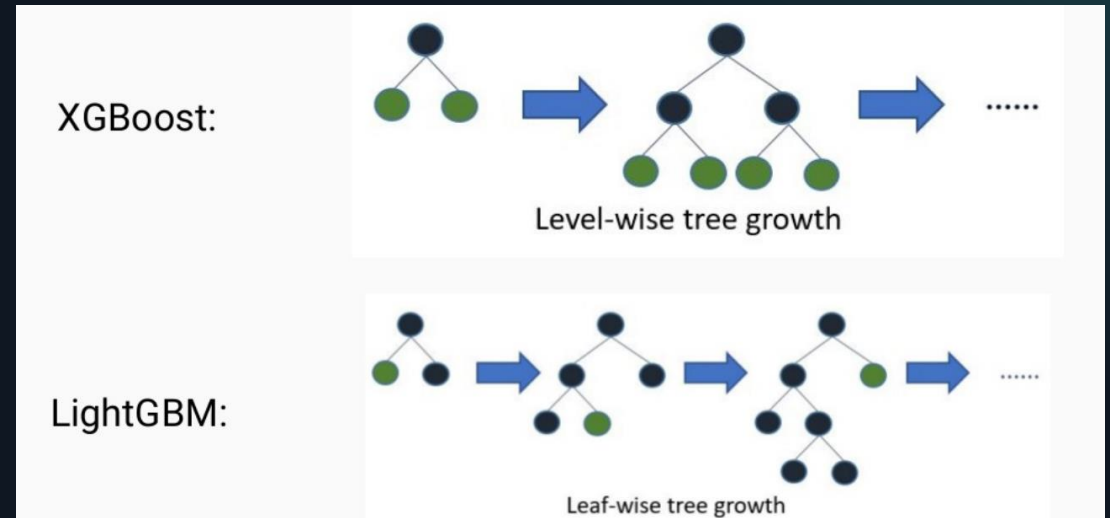
Tape Alphabets: $\Sigma = \{0, 1, a, b, x, Z_0\}$

The blank $_$ is a special symbol. $_ \notin \Sigma$

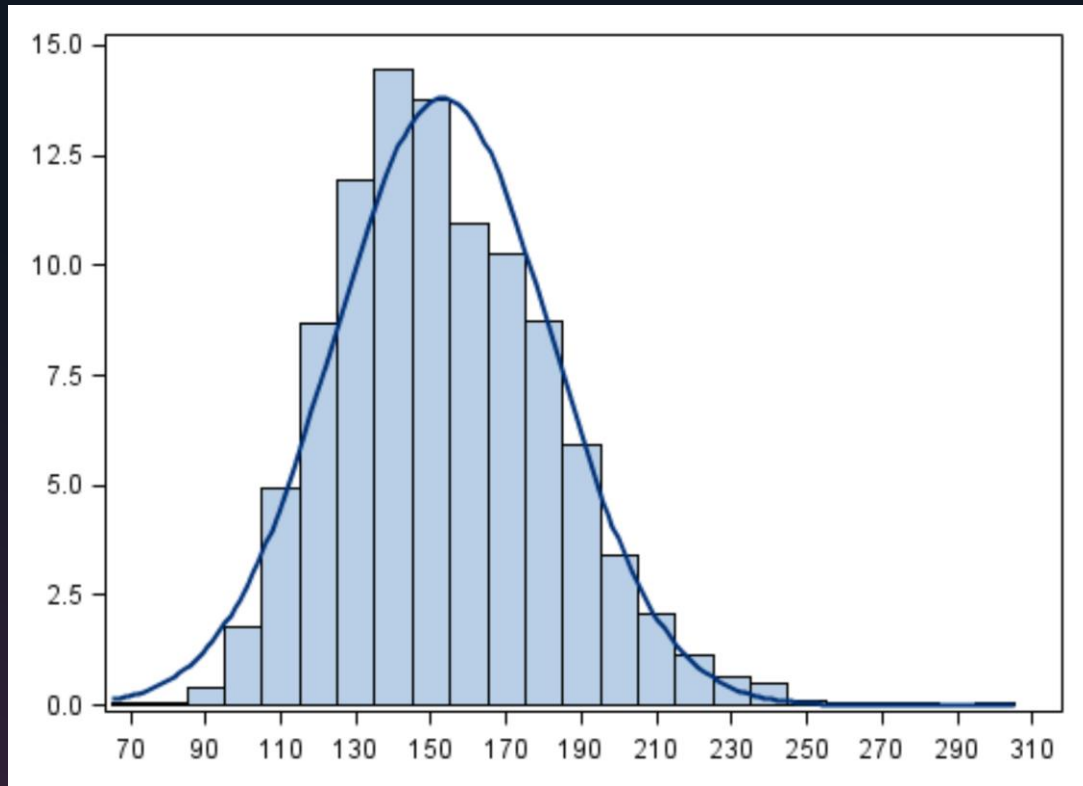
The blank is a special symbol used to fill the infinite tape

Leaf-wise growth vs depth-wise growth

- XGBoost uses level-wise tree growth and `max_depth` is the foremost hyper parameter to tune.
- LightGBM uses leaf-wise growth and `num_of_leaves` is the foremost hyper parameter to tune.



Finding optimal splitting point: exact vs histogram



- Decision tree/ GBDT: no process on the data. Use exact greedy algorithm for optimal splitting point
- XGBoost: data is pre-sorted and stored in column blocks for speed. However, when parameter `'tree_method'` is set as `"exact"` XGBoost will enumerate all split candidates to find the optimal splitting point.
- LightGBM and XGBoost with `"hist"` or `"gpu_hist"` as value for parameter `"tree_method"`: approximate greedy algorithm on histogram optimized data.
- Using histogram approximation instead of the exact value makes LightGBM significantly faster than XGBoost.

Leader board progress

Feature engineering	logloss	Would have ranked in this competition
Basic feature engineering	0.5324	Top 25%
+ target encoding	0.5296	Top 24%
+ manager performance	0.5287	Top 23%
+ location	0.5251	Top 21%
+ item2vec	0.5250	Top 20%
+ magic feature	0.5124	Top 10% (Bronze medal)
Manual tuning (greedy search)	0.5102	Top 8%
Automated tuning(BO, lr=0.05)	0.5095	Top 6%
Automated tuning(BO, lr=0.01)	0.5052	Top 5% (Silver medal)

Recap

- The purpose of model tuning is to find out the optimal values for hyper parameters, so that the model can be of good balance between underfitting and overfitting.
- Choosing an appropriate validation strategy is the key for any practical machine learning projects.
- Automated tuning (Bayesian optimization) is a powerful method but it can be more powerful by combining prior human being knowledge through manual tuning.
- XGBoost and LightGBM are the state-of-art algorithms for most tabular dataset except for texts and images.

