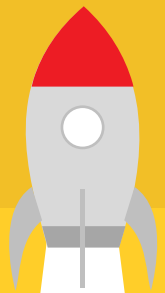
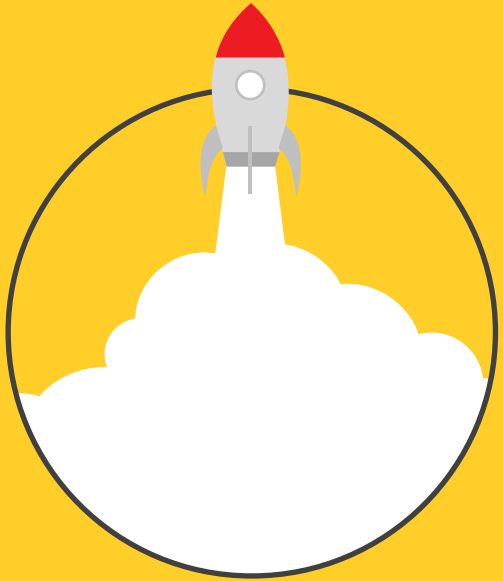
Creating powerful prediction



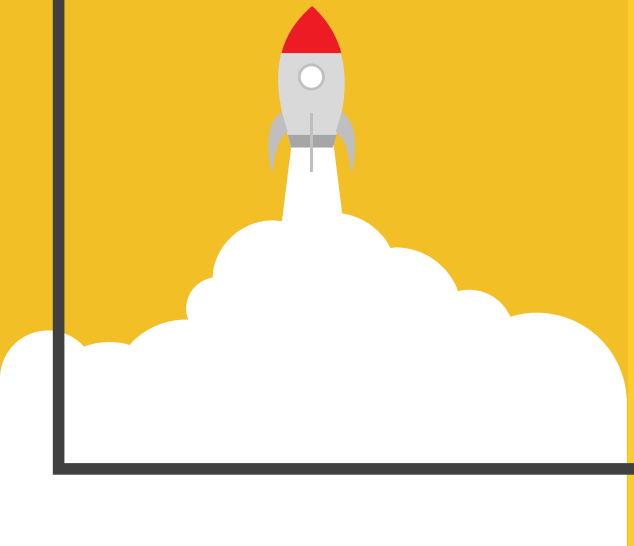
Resampling methods

By- Sudhanshu Saxena



Resampling methods

Resampling methods

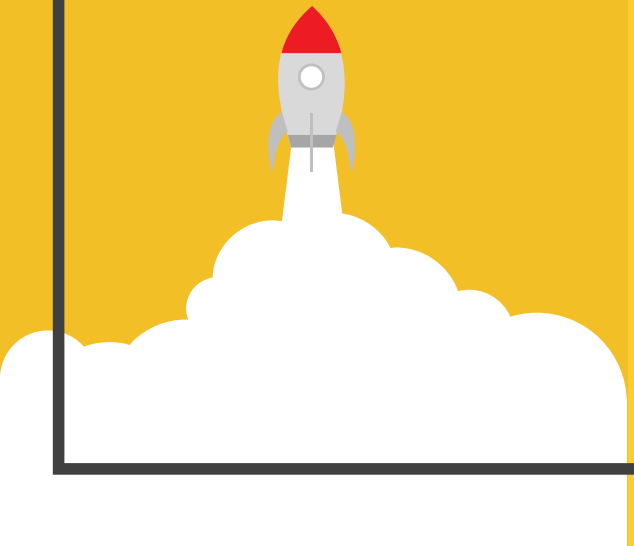


What are Resampling methods ?

involve repeatedly drawing samples from a training set and refitting a model of interest on each sample in order to obtain additional information about the fitted model.

For example, in order to estimate the variability of a linear regression fit, we can repeatedly draw different samples from the training data, fit a linear regression to each new sample, and then examine the extent to which the resulting fits differ. Such an approach may allow us to obtain information that would not be available from fitting the model only once using the original training sample.

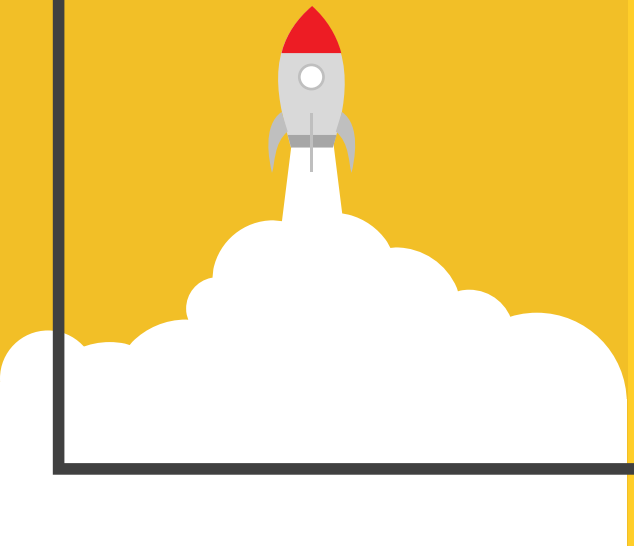
Resampling methods



Resampling approaches are:

1. Computationally expensive
2. Involve fitting the same statistical method multiple times
3. Use different subsets of the training data

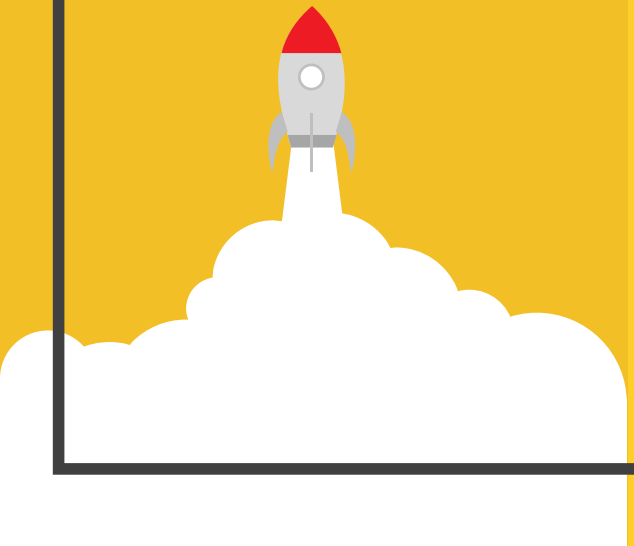
Resampling methods



Resampling approaches Majorly called as statistical resampling methods

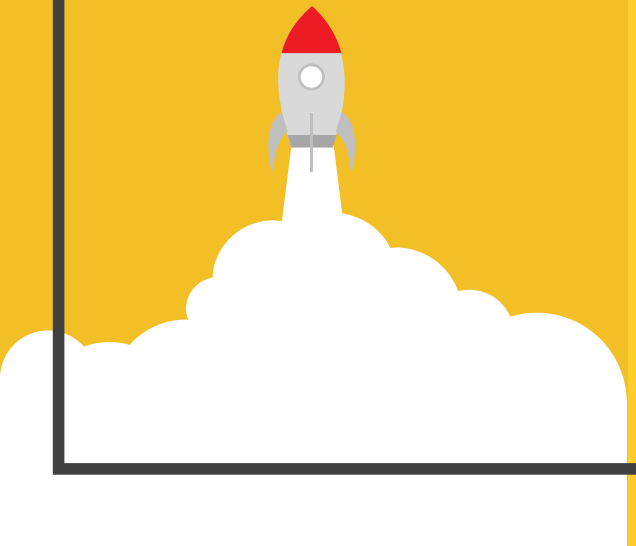
- Cross-validation
- Bootstrap

Cross-validation



Cross-validation (**CV**) is used to estimate the test error associated with a model to evaluate its performance or to select the appropriate level of flexibility. Evaluating a model's performance is usually defined as **model assessment**, and **model selection** is used for selecting the level of flexibility

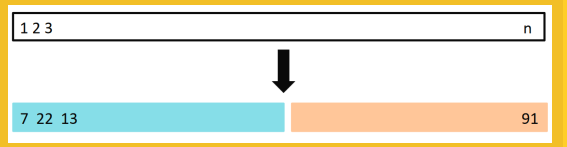
Cross-validation

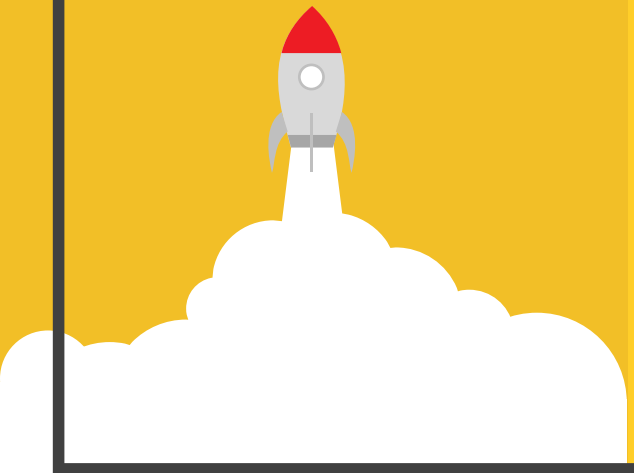


Validation set approach

This is the most basic approach. It simply involves randomly dividing the dataset into two parts:
a **training set** and a **validation set** or **hold-out set**.
The model is fit on the training set and the fitted model is used to make predictions on the validation set.

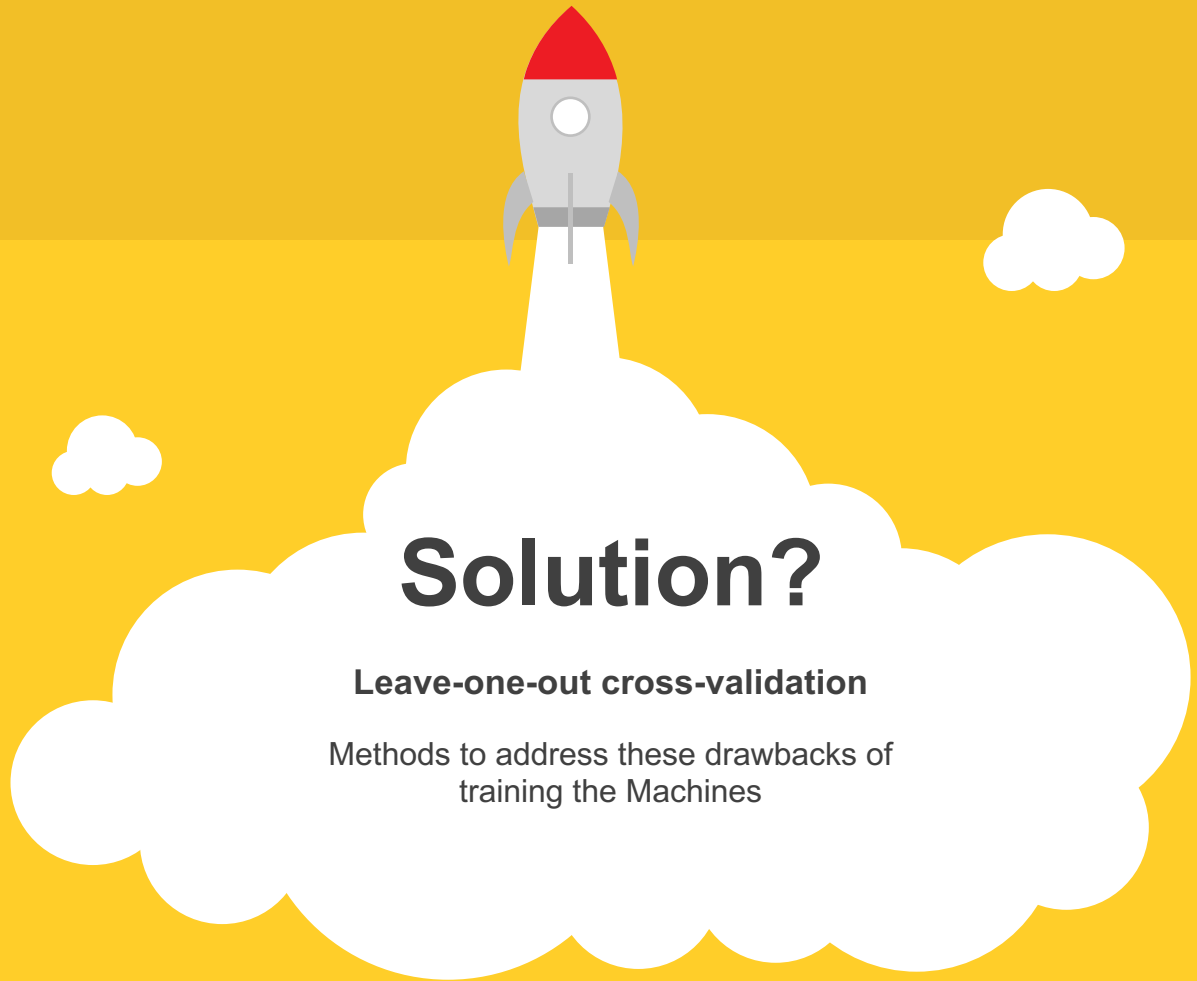
Cross-validation





Validation set approach- Drawbacks

- The validation test error rate is highly variable depending on which observations are in the training and validation set.
- Only a small subset of the observations are used to fit the model. However, we know that statistical methods tend to perform worse when trained on less data.



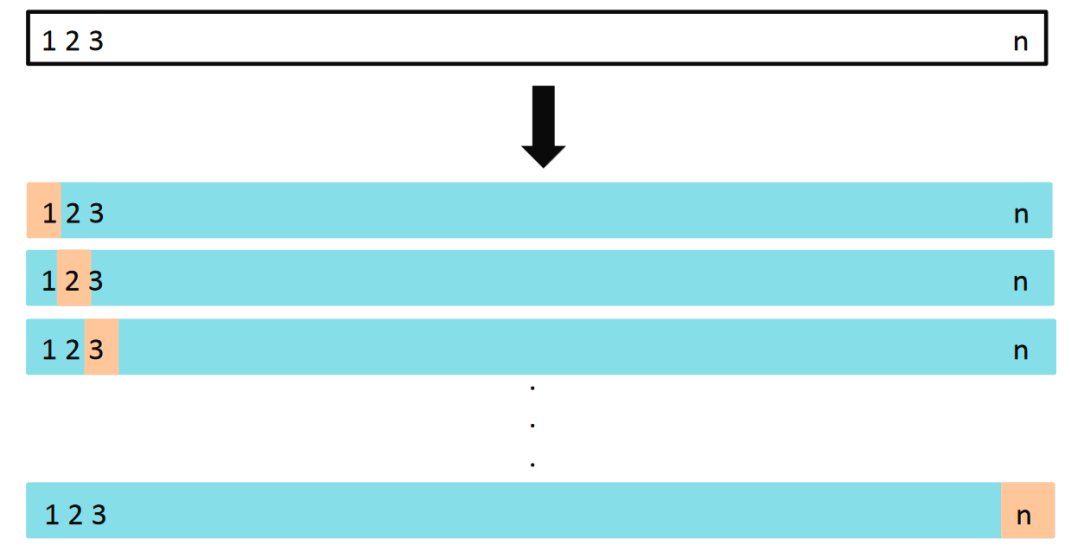
Solution?

Leave-one-out cross-validation

Methods to address these drawbacks of training the Machines

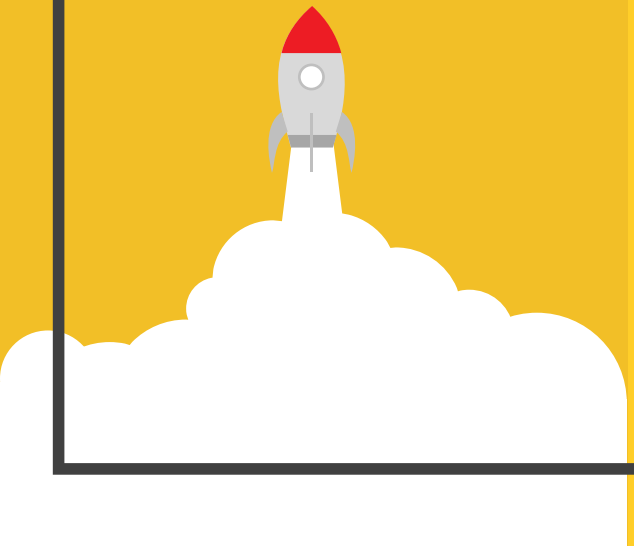
Leave-one-out cross-validation

The leave-one-out cross-validation (**LOOCV**) is a better option than the validation set approach. Instead of splitting the dataset into two subsets, only one observation is used for validation and the rest is used to fit the model.



Only one observation is used for validation and the rest is used for training. The process is then repeated multiple times.

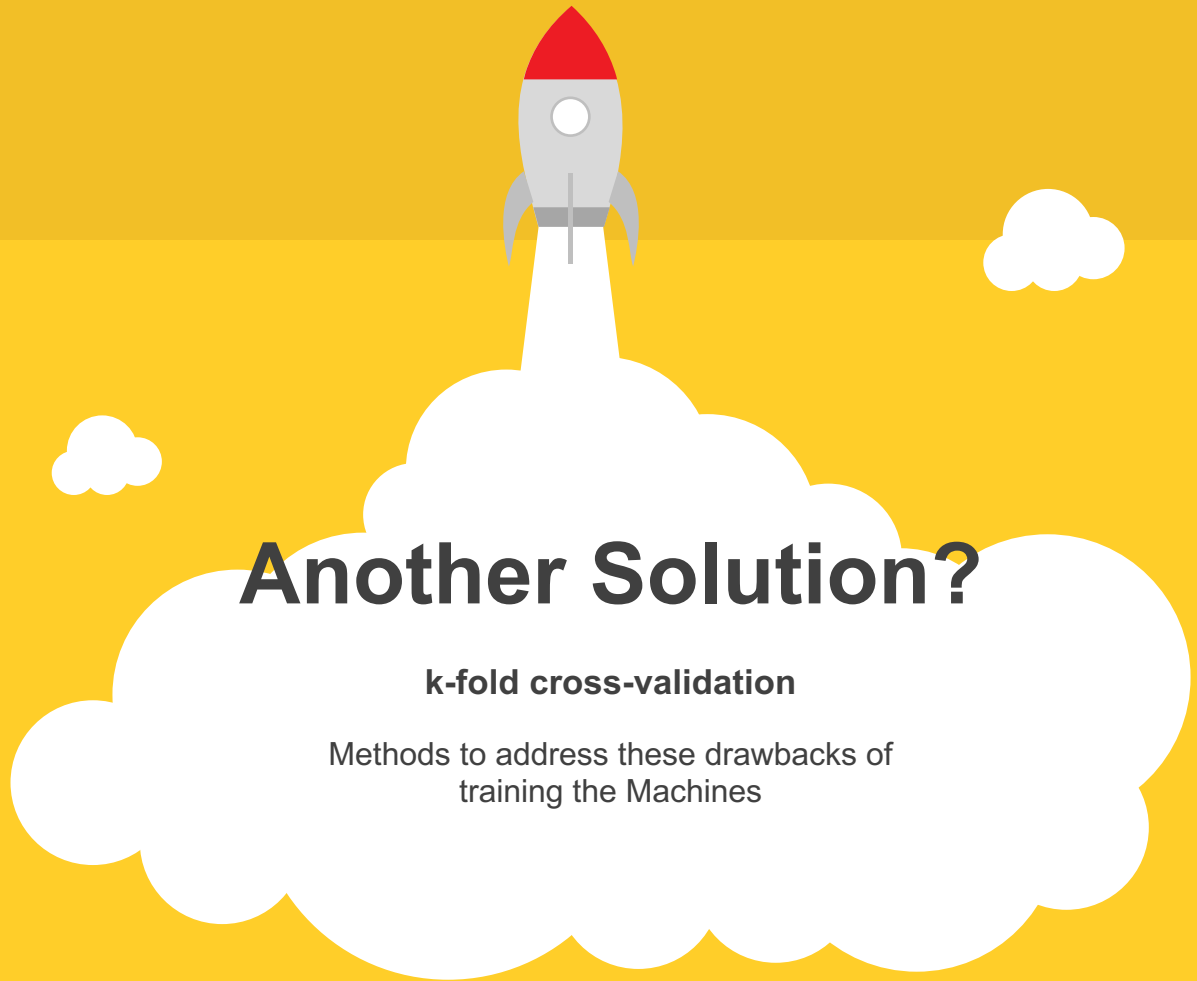
Leave-one-out cross-validation



After multiple runs, the error is estimated as:

$$CV_n = \frac{1}{n}\sum_{i=1}^{n} error_i$$

Which is simply the mean of the errors of each run.



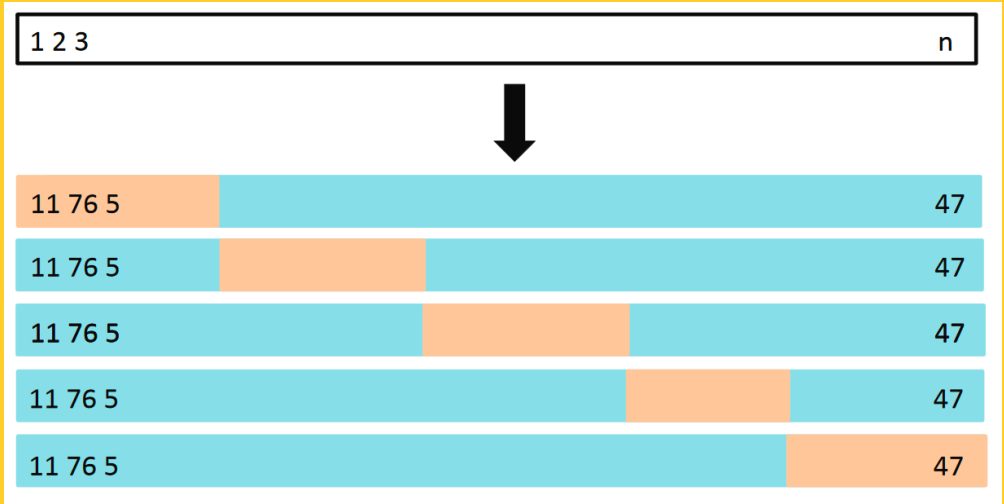
Another Solution?

k-fold cross-validation

Methods to address these drawbacks of training the Machines

k-fold cross-validation

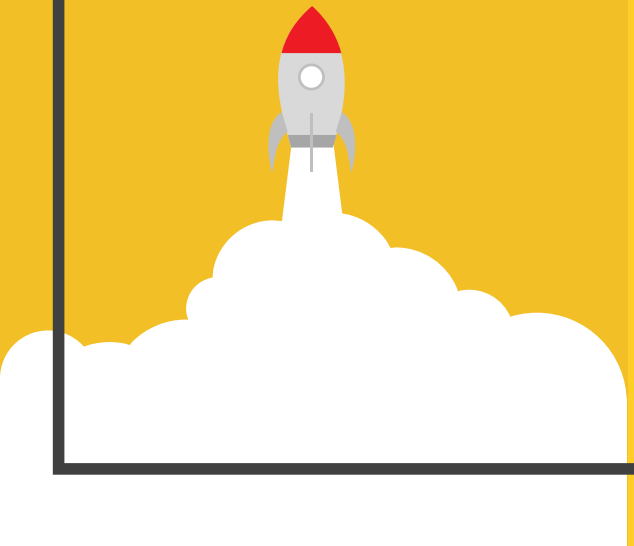
This approach involves randomly dividing the set of observations into *k*-groups or **folds** of approximately equal size. The first fold is treated as a validation set and the model is fit on the remaining folds. The procedure is then repeated *k* times, where a different group is treated as the validation set.

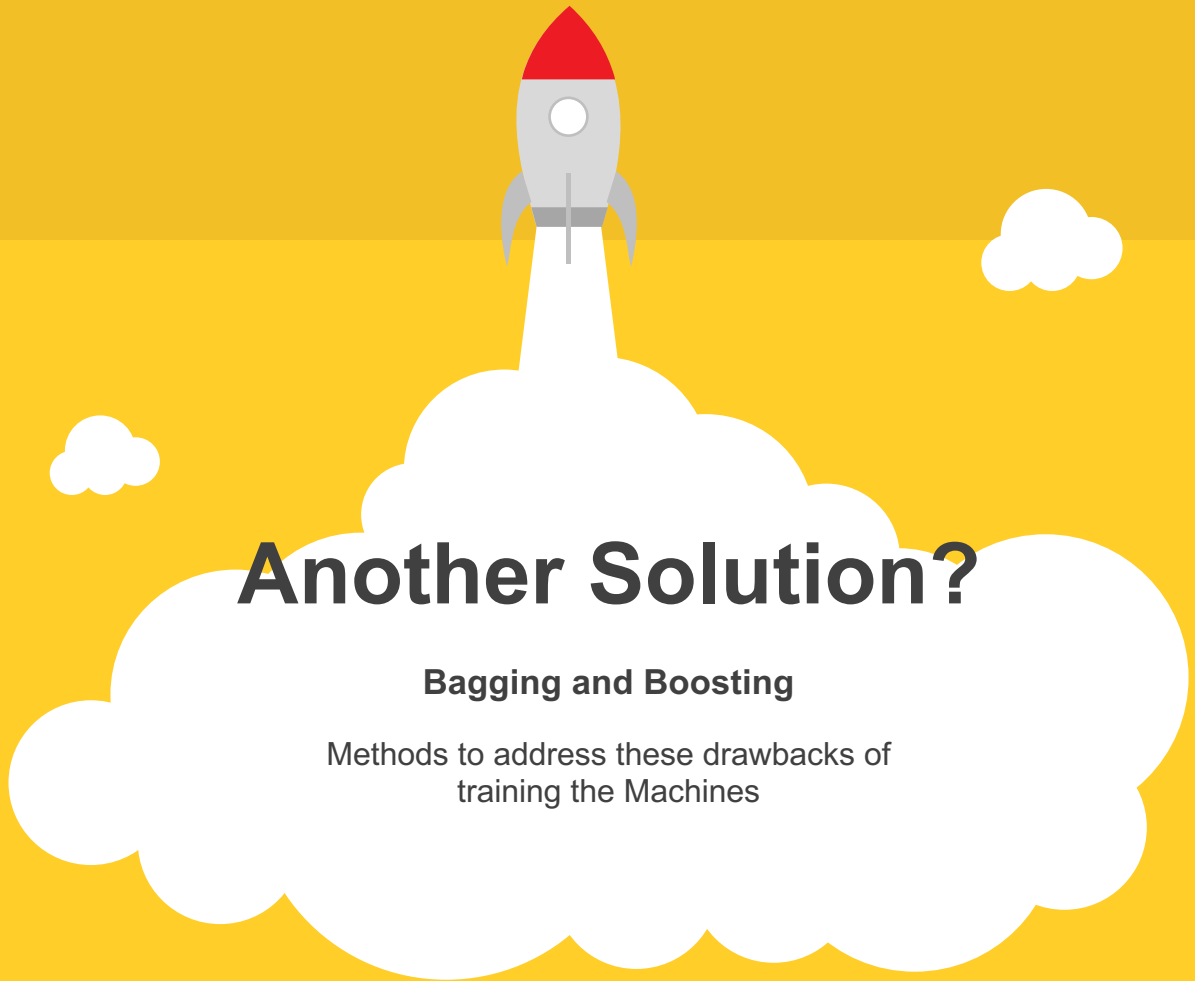


k-fold cross-validation

LOOCV is a special case of k-fold cross validation where k is equal to total number of observations n. However, it is common to set k equal to 5 or 10.

Whereas LOOCV is computationally intensive for large datasets, k-fold is more general and it can be used with any model. In addition, it often gives more accurate estimates of test error than does LOOCV.



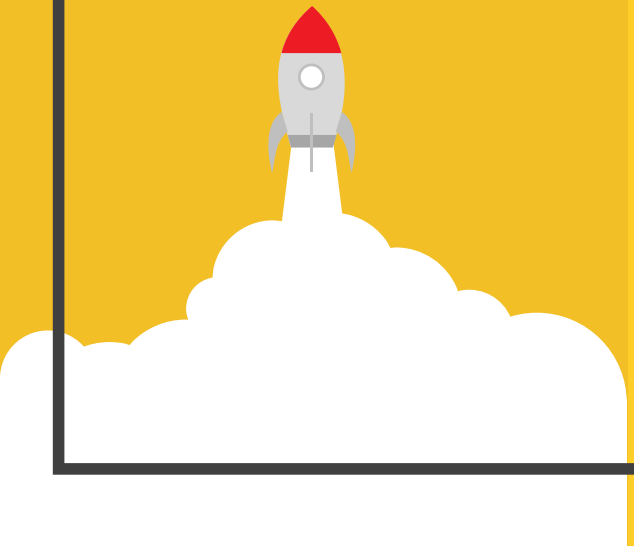


Another Solution?

Bagging and Boosting

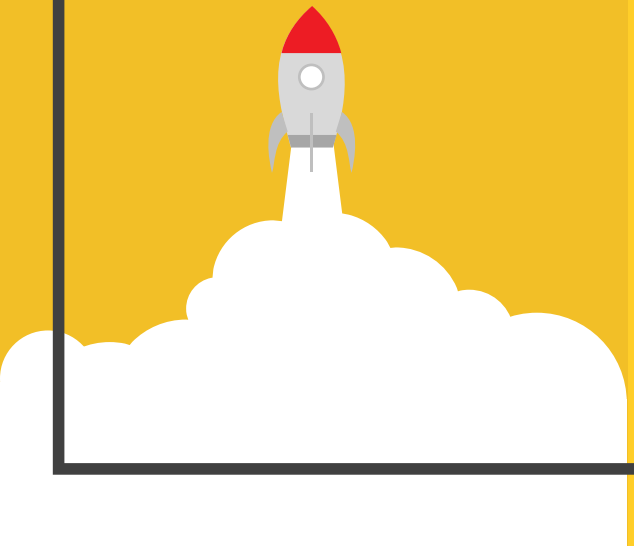
Methods to address these drawbacks of training the Machines

Bagging and Boosting



Ensemble methods combine several decision trees classifiers to produce better predictive performance than a single decision tree classifier. The main principle behind the ensemble model is that a group of weak learners come together to form a strong learner, thus increasing the accuracy of the model. When we try to predict the target variable using any machine learning technique, the main causes of difference in actual and predicted values are **noise, variance, and bias**. Ensemble helps to reduce these factors (except noise, which is irreducible error).

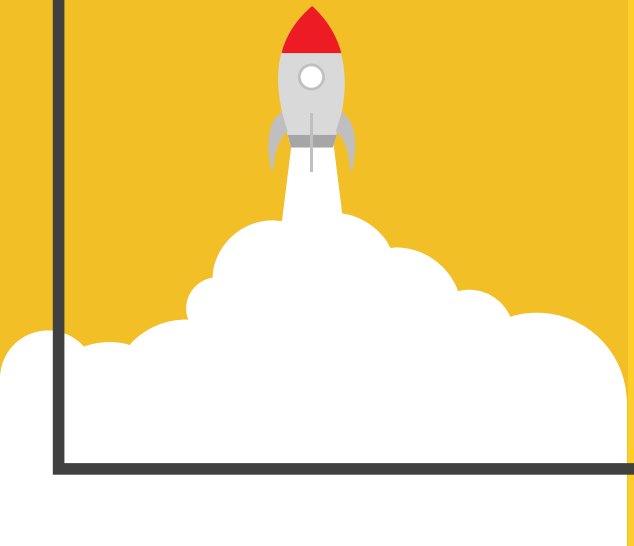
Noise



Noisy data is **data** with a large amount of additional meaningless information in it called **noise**. This includes **data** corruption and the term is often used as a synonym for corrupt **data**. It also includes any **data** that a user system cannot understand and interpreted correctly.

Statistically noise is that part of the ***residual*** which is in-feasible to model by any other means than a purely statistical description.

Residual and error



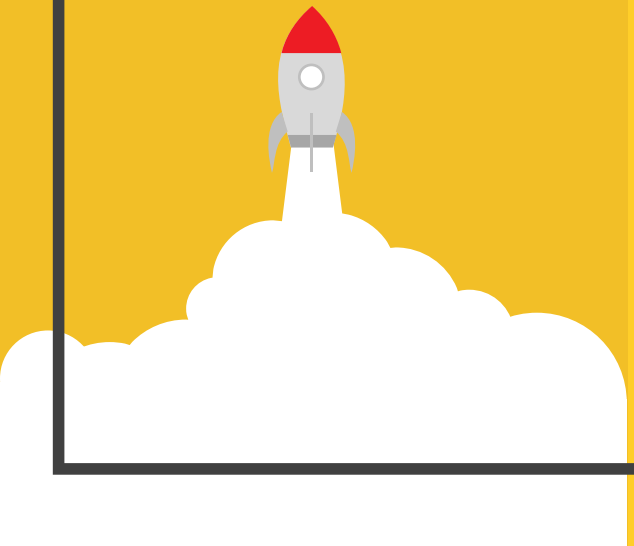
The ***residual*** is the difference between the true phenomenon being studied and the model being employed to describe it.

Error is that component of the ***residual*** that remains after accounting for the noise.

Noise and ***error*** are uncorrelated

Residual may be reduced by either reducing ***noise*** or by reducing ***error***

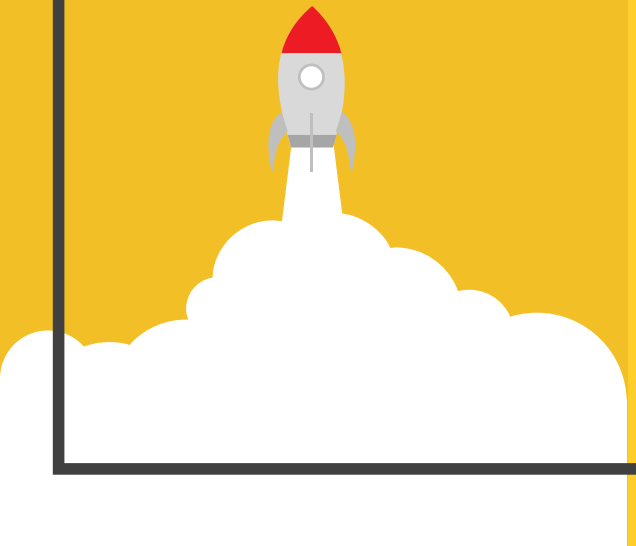
Bias and Variance



How do we know if a model is performing well?

A machine learning model's performance is considered good based on it prediction and how well it generalizes on an independent test dataset. Based on the performance of different models we choose the model which ranks highest in performance.

Bias and Variance



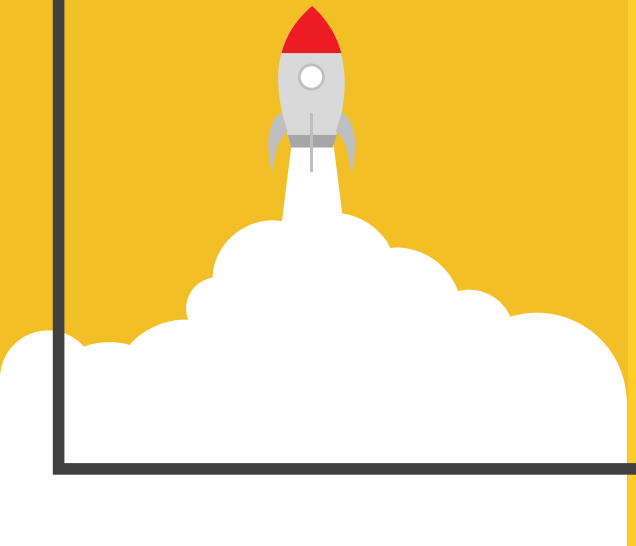
When we have **an input *x* and we apply a function *f* on the input *x* to predict an output *y***. Difference between the actual output and predicted output is the error. Our goal with machine learning algorithm is to generate a model which minimizes the error of the test dataset.

Models are assessed based on the prediction error on a new test dataset.

$$L(x, y) = \sum_{i=1}^{N} \big(y - f(x)\big)$$

Error = sum of all (Actual output – Predicted Output)

Bias and Variance



Error in our model is summation of reducible and irreducible error.

$$\text{Error} = \text{Reducible Error} + \text{Irreducible Error}$$

$$\text{Reducible Error} = \text{Bias}^2 + \text{Variance}$$

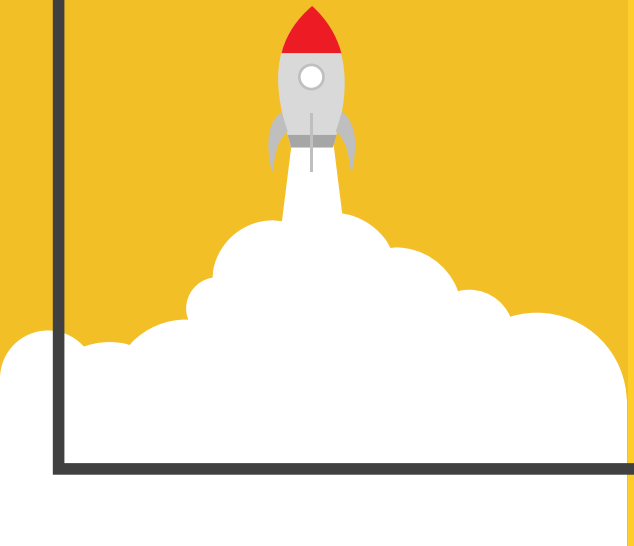
Irreducible Error

Errors that cannot be reduced no matter what algorithm you apply is called an irreducible error. It is usually caused by unknown variables that may be having an influence on the output variable.

Reducible Error has two components — **bias and variance.**

Presence of bias or variance causes overfitting or underfitting of data.

Bias and Variance

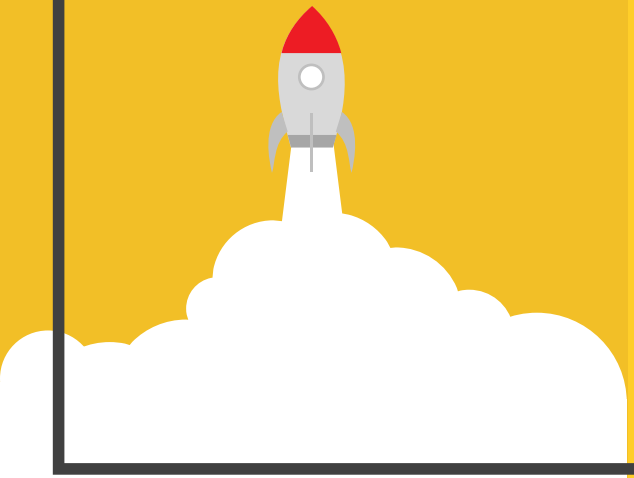


Bias

Bias is how far are the predicted values from the actual values. If the **average predicted values are far off from the actual values then the bias is high**.

High bias causes algorithm to miss relevant relationship between input and output variable. When a model has a high bias then it implies that the model is too simple and does not capture the complexity of data
thus **underfitting the data**.

Bias and Variance



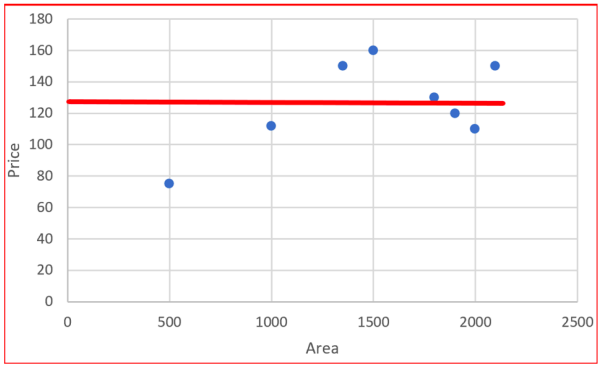
Variance

Variance occurs when the model performs good on the trained dataset but does not do well on a dataset that it is not trained on, like a test dataset or validation dataset. **Variance tells us how scattered are the predicted value from the actual value**.

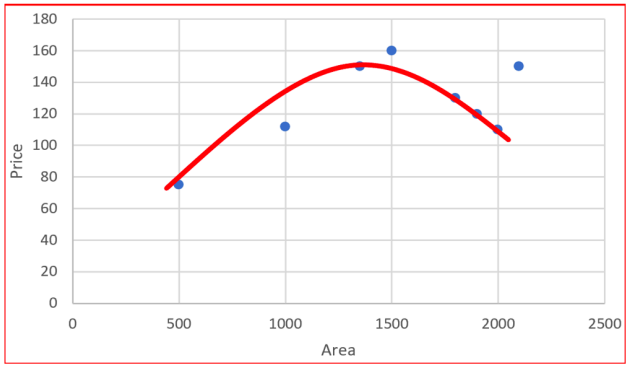
High variance causes overfitting that implies that the algorithm models random noise present in the training data.

Bias and Variance

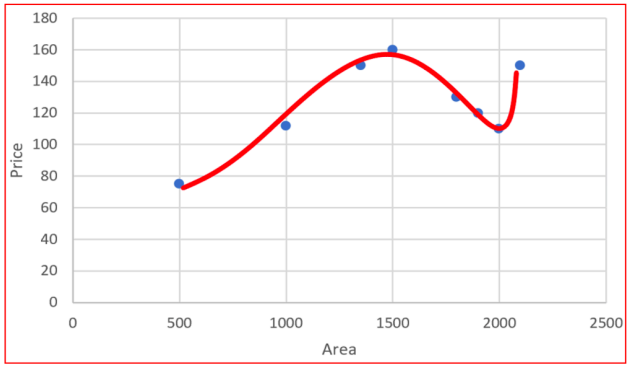
When a model has a high variance then the model becomes very flexible and tune itself to the data points of the training set. when a high variance model encounters a different data point that it has not learnt then it cannot make right prediction.



High Bias - underfit



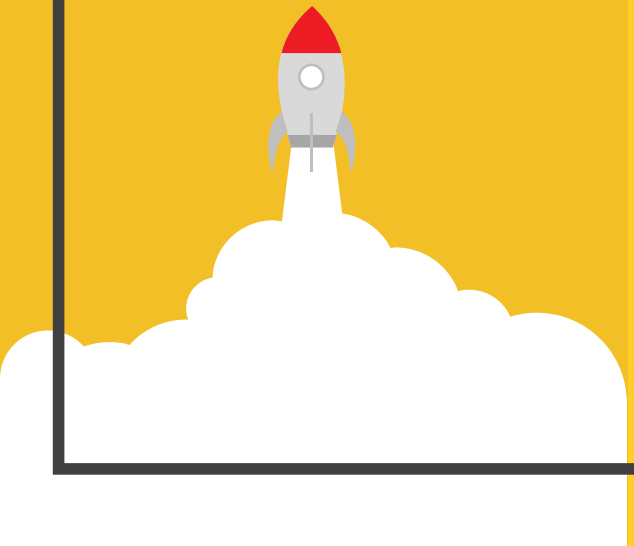
Just Fit



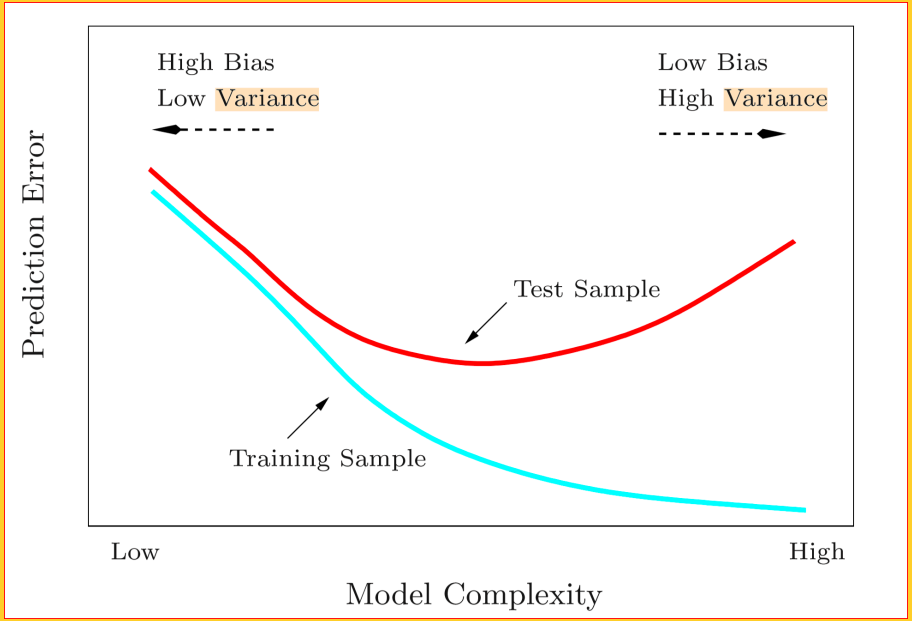
High Variance – overfit

A model with high bias looks very simple. A model with high variance tries to fit most of the data points making the model complex and difficult to model.

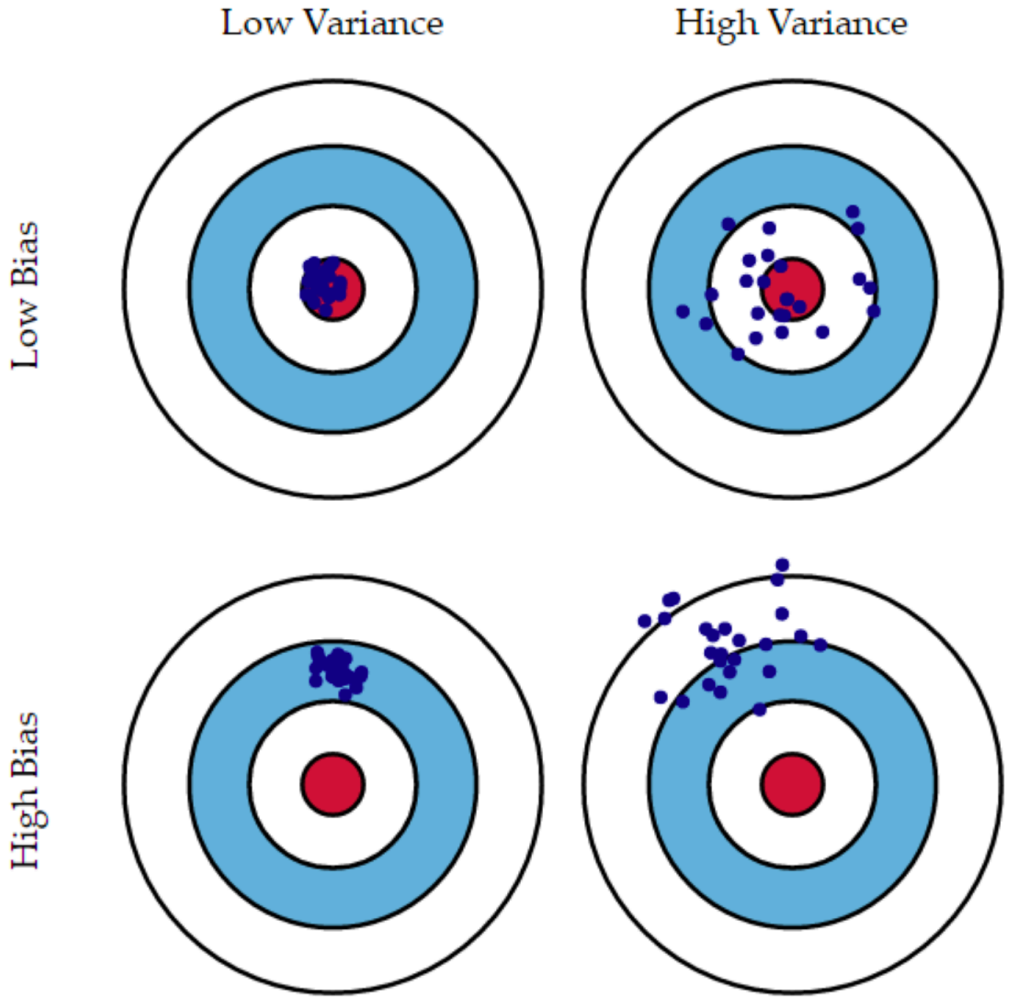
Model Complexity



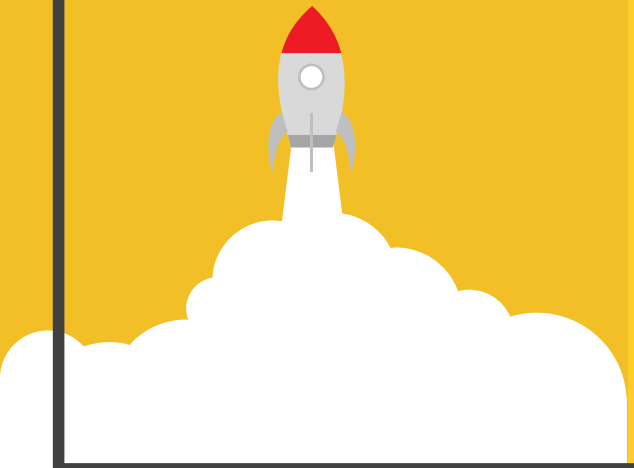
Plot between test and training prediction error as a function of model complexity.



We would like to have a model complexity that trades bias off with variance so that we minimize the test error and would make our model perform better. This is illustrated as the the bias variance trade off .



Bias Variance trade off



High Bias Low Variance: Models are consistent but inaccurate on average
High Bias High Variance : Models are inaccurate and also inconsistent on average

Low Bias High variance: Models are somewhat accurate but inconsistent on averages. A small change in the data can cause a large error.

Low Bias Low Variance: Models are accurate and consistent on averages. We strive for this in our model

Bias or a high Variance?

Is there a way to find when we have a high bias or a high variance?

High Bias can be identified when we have

- High training error
- Validation error or test error is same as training error

High Variance can be identified when

- Low training error
- High validation error or high test error

How do we fix high bias or high variance in the data set?

High bias is due to a simple model and we also see a high training error. To fix that we can do following things

- Add more input features
- Add more complexity by introducing polynomial features
- Decrease Regularization term

How do we fix high bias or high variance in the data set?

High variance is due to a model that tries to fit most of the training dataset points and hence gets more complex. To resolve high variance issue we need to work on

- Getting more training data
- Reduce input features
- Increase Regularization term

Regularization

Regularization is a technique where we penalize the loss function for a complex model which is very flexible. This helps with overfitting. It does this by penalizing the different parameters or weights to reduce the noise of the training data and generalize well on the test data

Regularization

L1 regularization does feature selection. It does this by assigning insignificant input features with zero weight and useful features with a non zero weight.

L1 regularization is also referred as L1 norm or Lasso.

$$L(x, y) \equiv \sum_{i=1}^{n} (y_i - h_\theta(x_i))^2 + \lambda \sum_{i=1}^{n} |\theta_i|$$

Regularization

L1 regularization does feature selection. It does this by assigning insignificant input features with zero weight and useful features with a non zero weight.

$$L(x, y) \equiv \sum_{i=1}^{n} (y_i - h_\theta(x_i))^2 + \lambda \sum_{i=1}^{n} |\theta_i|$$

In L1 regularization we penalize the absolute value of the weights. L1 regularization term is highlighted in the red box.

Regularization

$$L(x, y) \equiv \sum_{i=1}^{n} (y_i - h_\theta(x_i))^2 + \lambda \sum_{i=1}^{n} \theta_i^2$$

L2 Regularization or Ridge Regularization

In L2 regularization, regularization term is the sum of square of all feature weights as shown above in the equation.

L2 regularization forces the weights to be small but does not make them zero and does non sparse solution.

Regularization

$$L(x,y) \equiv \sum_{i=1}^{n} (y_i - h_\theta(x_i))^2 + \lambda \sum_{i=1}^{n} \theta_i^2$$

L2 Regularization or Ridge Regularization

L2 is not robust to outliers as square terms blows up the error differences of the outliers and the regularization term tries to fix it by penalizing the weights

Ridge regression performs better when all the input features influence the output and all with weights are of roughly equal size

Bootstrap Aggregation - Bagging

Definition:
Bagging is used when the goal is to reduce the variance of a decision tree classifier. **Here the objective is to create several subsets of data from training sample chosen randomly with replacement. Each collection of subset data is used to train their decision trees.** As a result, we get an ensemble of different models. Average of all the predictions from different trees are used which is more robust than a single decision tree classifier.

Bootstrap Aggregation - Bagging

Bagging Steps:

- Suppose there are N observations and M features in training data set. A sample from training data set is taken randomly with replacement.
- A subset of M features are selected randomly and whichever feature gives the best split is used to split the node iteratively.
- The tree is grown to the largest.
- Above steps are repeated n times and prediction is given based on the aggregation of predictions from n number of trees.

Bootstrap Aggregation - Bagging

Advantages:

- Reduces over-fitting of the model.
- Handles higher dimensionality data very well.
- Maintains accuracy for missing data.

Disadvantages:

- Since final prediction is based on the mean predictions from subset trees, it won't give precise values for the classification and regression model.

Bootstrap Aggregation - Bagging

Python Syntax:

- rfm = RandomForestClassifier(n_estimators=80, oob_score=True, n_jobs=-1, random_state=101, max_features = 0.50, min_samples_leaf = 5)
- fit(x_train, y_train)
- predicted = rfm.predict_proba(x_test)

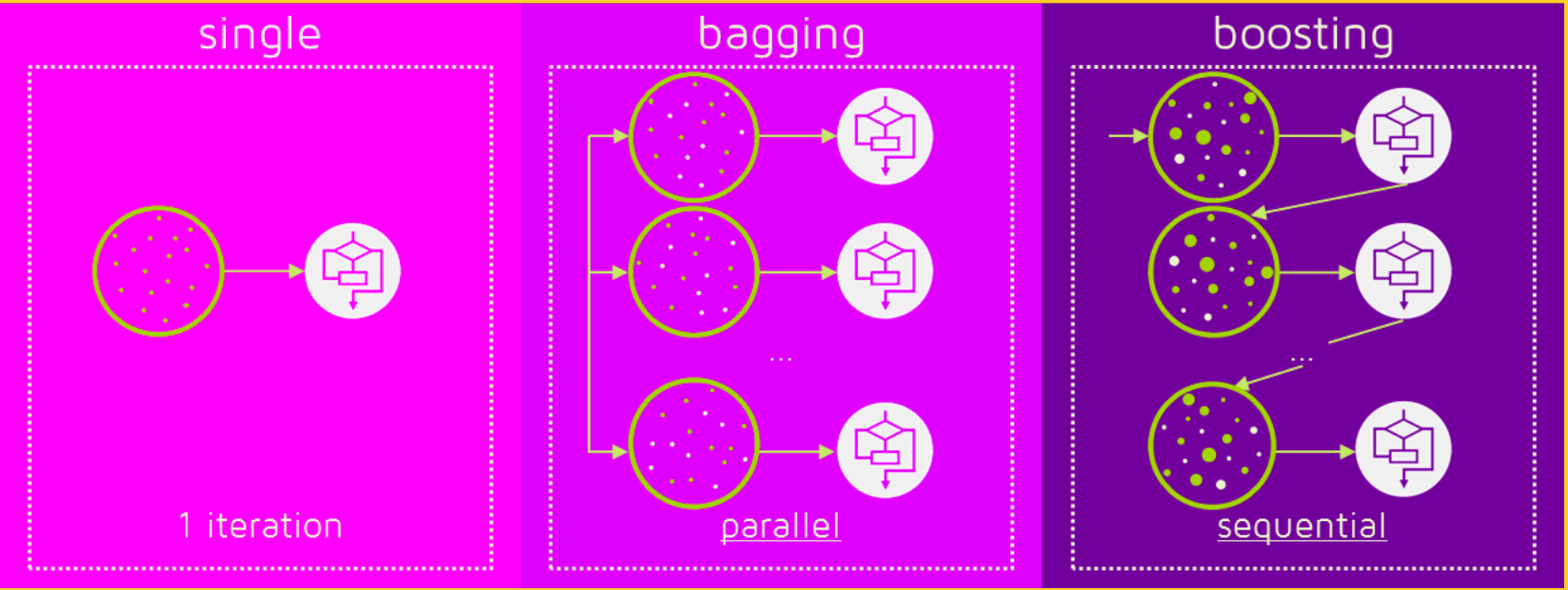
Boosting

Definition:

Boosting is used to create a collection of predictors. **In this technique, learners are learned sequentially with early learners fitting simple models to the data and then analysing data for errors. Consecutive trees (random sample) are fit and at every step, the goal is to improve the accuracy from the prior tree.** When an input is misclassified by a hypothesis, its weight is increased so that next hypothesis is more likely to classify it correctly. This process converts weak learners into better performing model.

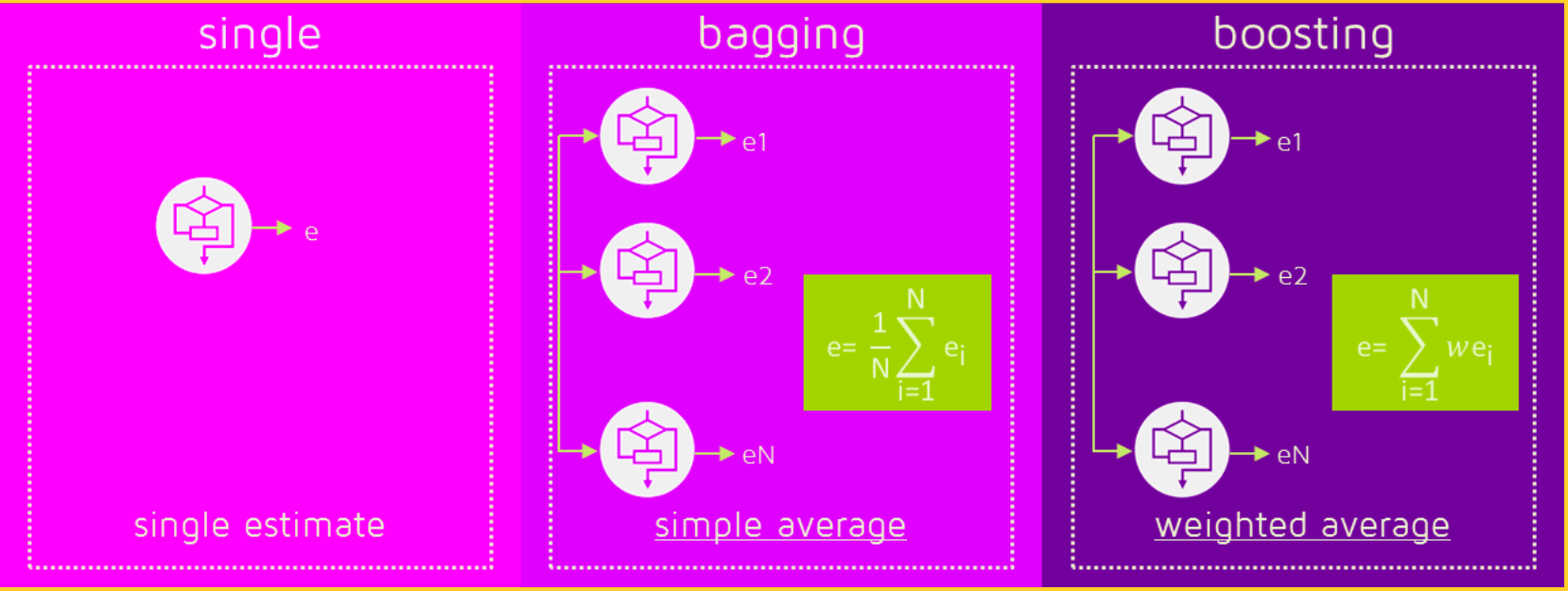
Boosting

In Boosting algorithms each classifier is trained on data, taking into account the previous classifiers' success. After each training step, the weights are redistributed. M**isclassified data increases its weights** to emphasise the most difficult cases. In this way, subsequent learners will focus on them during their training.



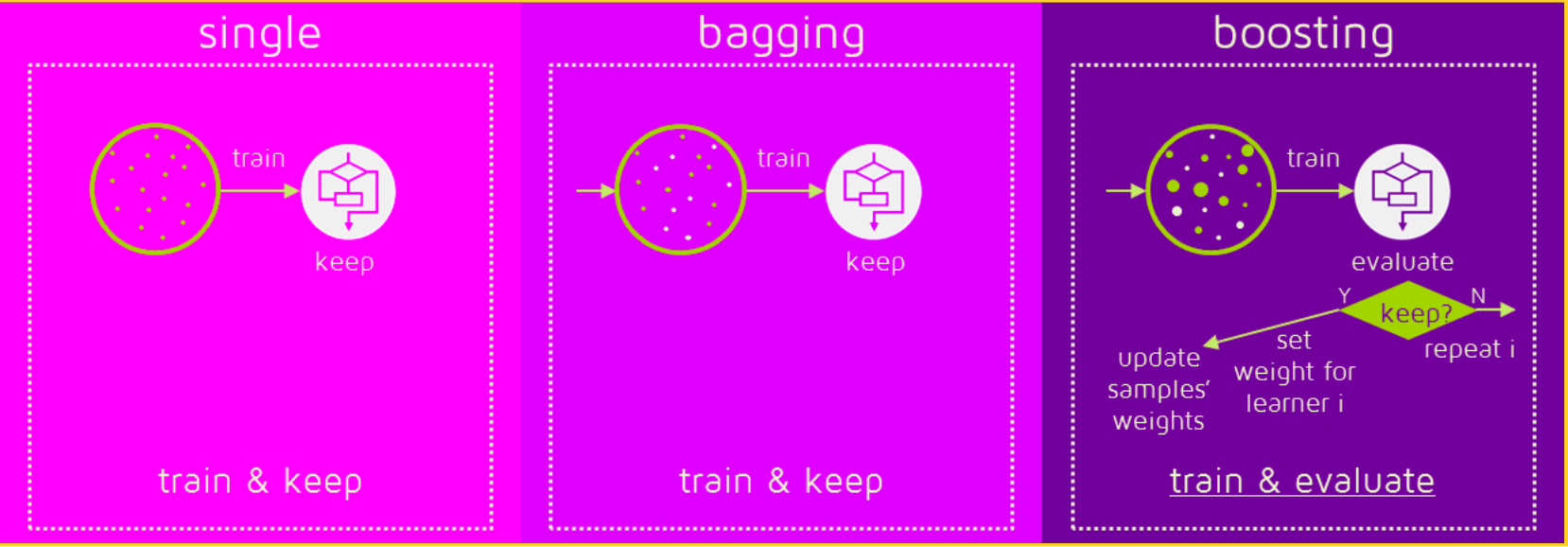
How does the classification stage work?

To predict the class of new data we only need to **apply the N learners to the new observations.** In Bagging the result is obtained by averaging the responses of the N learners (or majority vote). However, Boosting assigns a second set of weights, this time for the N classifiers, in order to take a **weighted average** of their estimates.



How does the classification stage work?

In the Boosting training stage, the algorithm allocates weights to each resulting model. A learner with good a classification result on the training data will be assigned a higher weight than a poor one. So when evaluating a new learner, Boosting needs to keep track of learners' errors, too. Let's see the differences in the procedures:



Boosting

Boosting Steps:

- Draw a random subset of training samples d1 without replacement from the training set D to train a weak learner C1
- Draw second random training subset d2 without replacement from the training set and add 50 percent of the samples that were previously falsely classified/misclassified to train a weak learner C2
- Find the training samples d3 in the training set D on which C1 and C2 disagree to train a third weak learner C3
- Combine all the weak learners via majority voting.

Boosting

Advantages:

•Supports different loss function (we have used ‘binary:logistic’ for this example).

•Works well with interactions.

Disadvantages:

•Prone to over-fitting.

•Requires careful tuning of different hyper-parameters.

Boosting

Python Syntax:

- from xgboost import XGBClassifier
- xgb = XGBClassifier(objective='binary:logistic', n_estimators=70, seed=101)
- fit(x_train, y_train)
- predicted = xgb.predict_proba(x_test)

Classification Report

Precision – What percent of your predictions were correct?
Precision is the ability of a classifier not to label an instance positive that is actually negative. For each class it is defined as the ratio of true positives to the sum of true and false positives.

TP – True Positives
FP – False Positives
Precision – Accuracy of positive predictions.
Precision = TP/(TP + FP)

Classification Report

Recall – What percent of the positive cases did you catch?
Recall is the ability of a classifier to find all positive instances. For each class it is defined as the ratio of true positives to the sum of true positives and false negatives.

FN – False Negatives
Recall: Fraction of positives that were correctly identified.
Recall = TP/(TP+FN)

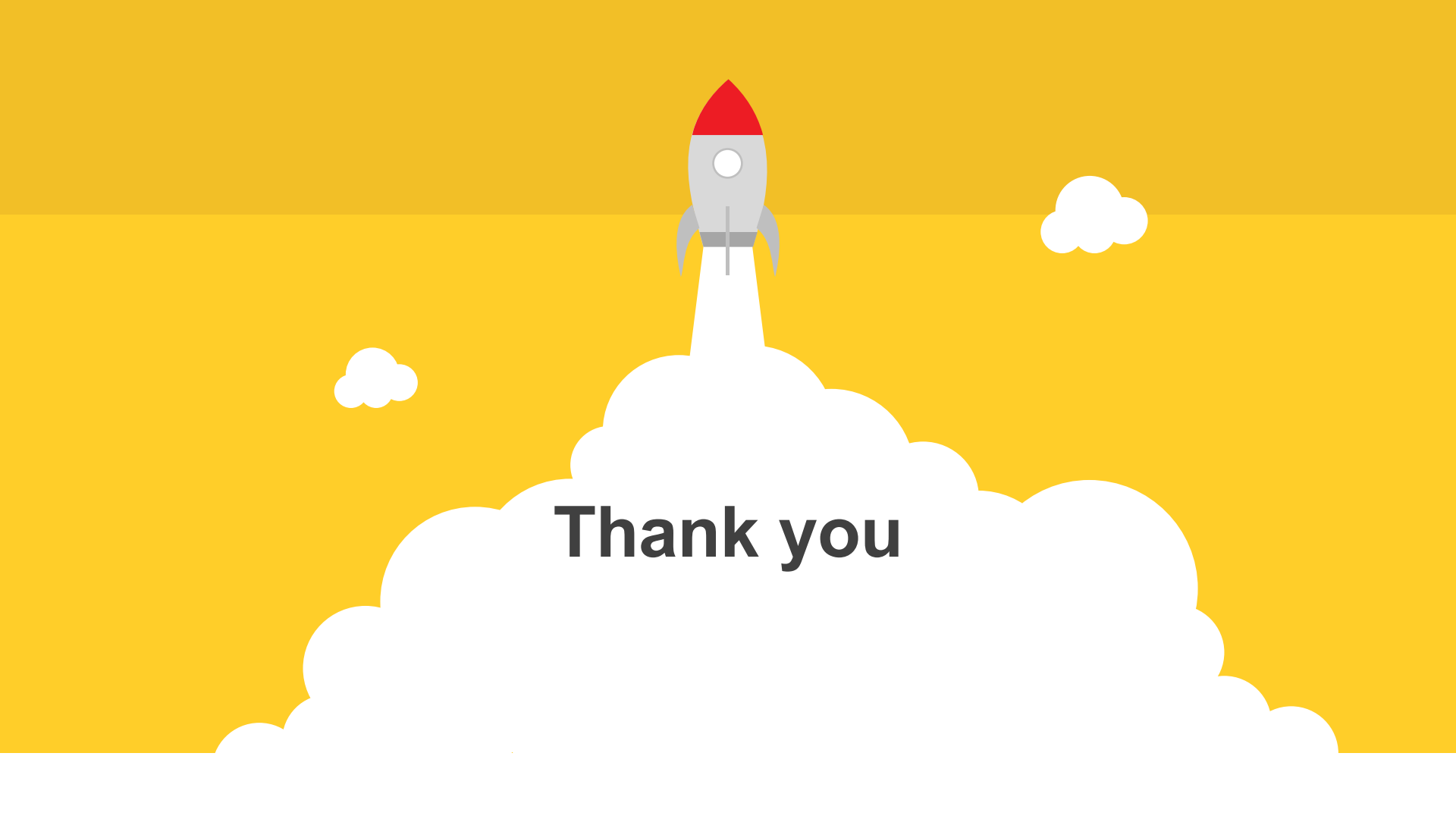
Classification Report

F1 score – What percent of positive predictions were correct?

The F_1 score is a weighted harmonic mean of precision and recall such that the best score is 1.0 and the worst is 0.0. Generally speaking, F_1 scores are lower than accuracy measures as they embed precision and recall into their computation. As a rule of thumb, the weighted average of F_1 should be used to compare classifier models, not global accuracy.

F1 Score = 2*(Recall * Precision) / (Recall + Precision)

The **support** is the number of samples of the true response that lie in that class.


Thank you