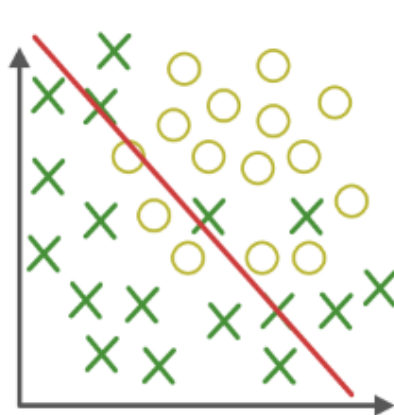


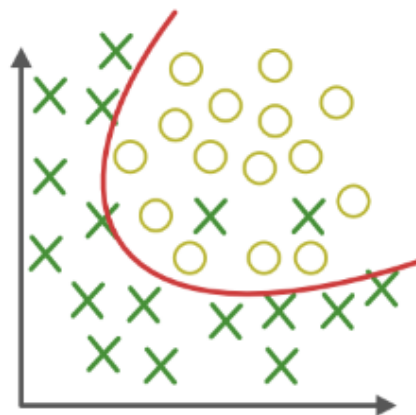
Cross Validation & Hyperparameter Tuning

Underfitting and Overfitting

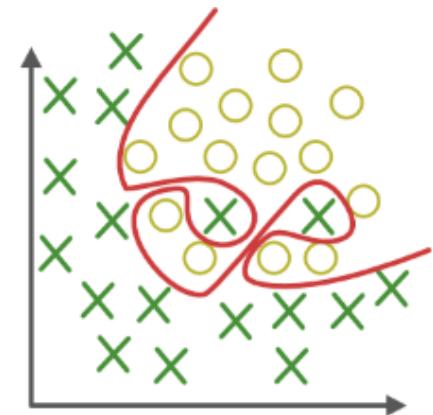
- Underfitting and overfitting are two common challenges faced in machine learning.
- The main goal of each machine learning model is **to generalize well**. Here **generalization** defines the ability of an ML model to provide a suitable output by adapting the given set of unknown input.
- It means after providing training on the dataset, it can produce reliable and accurate output.
- Hence, the underfitting and overfitting are the two terms that need to be checked for the performance of the model and whether the model is generalizing well or not.



Under-fitting



Appropriate-fitting



Over-fitting

Overfitting

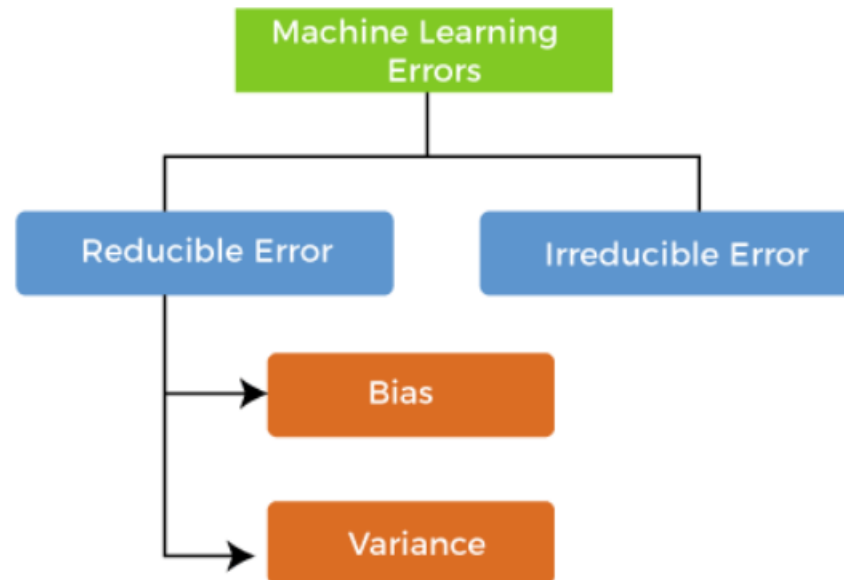
- Overfitting occurs when our machine learning model tries to cover all the data points or more than the required data points present in the given dataset.
- Because of this, the model starts caching noise and inaccurate values present in the dataset, and all these factors reduce the efficiency and accuracy of the model.
- **Reasons for Overfitting**
 1. Data used for training is not cleaned and contains noise (garbage values) in it
 2. The model has a high variance
 3. The size of the training dataset used is not enough
 4. The model is too complex
- **Ways to Tackle Overfitting**
 - Using cross-validation
 - Using Regularization techniques such as Lasso and Ridge
 - Training model with sufficient data
 - Adopting ensembling techniques

Underfitting

- Underfitting occurs when our machine learning model is not able to capture the underlying trend of the data.
- To avoid the underfitting in the model, the fed of training data can be stopped at an early stage, due to which the model may not learn enough from the training data.
- As a result, it may fail to find the best fit of the dominant trend in the data.
- **Reasons for Underfitting**
 1. Data used for training is not cleaned and contains noise (garbage values) in it
 2. The model has a high bias
 3. The size of the training dataset used is not enough
 4. The model is too simple
- **Ways to Tackle Underfitting**
 - Increase the number of features in the dataset
 - Increase model complexity
 - Reduce noise in the data
 - Increase the duration of training the data

Errors in Machine Learning?

- In machine learning, an error is a measure of how accurately an algorithm can make predictions for the previously unknown dataset.
- On the basis of these errors, the machine learning model is selected that can perform best on the particular dataset.
- There are mainly two types of errors in machine learning, which are:
- **Reducible errors:** These errors can be reduced to improve the model accuracy. Such errors can further be classified into bias and Variance.
- **Irreducible errors:** These errors will always be present in the model



Bias

- In general, a machine learning model analyses the data, find patterns in it and make predictions. While training, the model learns these patterns in the dataset and applies them to test data for prediction.
- ***While making predictions, a difference occurs between prediction values made by the model and actual values/expected values, and this difference is known as bias errors or Errors due to bias.***
- It can be defined as an inability of machine learning algorithms such as Linear Regression to capture the true relationship between the data points.
- Each algorithm begins with some amount of bias because bias occurs from assumptions in the model, which makes the target function simple to learn.
- A model has either:
- **Low Bias:** A low bias model will make fewer assumptions about the form of the target function.
- **High Bias:** A model with a high bias makes more assumptions, and the model becomes unable to capture the important features of our dataset. **A high bias model also cannot perform well on new data.**

Variance

- The variance would specify the amount of variation in the prediction if the different training data was used.
- In simple words, ***variance tells that how much a random variable is different from its expected value.***
- Ideally, a model should not vary too much from one training dataset to another, which means the algorithm should be good in understanding the hidden mapping between inputs and output variables.
- Variance errors are either of **low variance or high variance.**
- **Low variance** means there is a small variation in the prediction of the target function with changes in the training data set. At the same time.
- **High variance** shows a large variation in the prediction of the target function with changes in the training dataset.
-

Different Combinations of Bias-Variance

- There are four possible combinations of bias and variances, which are given as below:
- **Low-Bias, Low-Variance:**
The combination of low bias and low variance shows an ideal machine learning model. However, it is not possible practically.
- **Low-Bias, High-Variance:** With low bias and high variance, model predictions are inconsistent and accurate on average. This case occurs when the model learns with a large number of parameters and hence leads to an **overfitting**.
- **High-Bias, Low-Variance:** With High bias and low variance, predictions are consistent but inaccurate on average. This case occurs when a model does not learn well with the training dataset or uses few numbers of the parameter. It leads to **underfitting** problems in the model.
- **High-Bias, High-Variance:**
With high bias and high variance, predictions are inconsistent and also inaccurate on average.

Bias-Variance Trade-off

- While building the machine learning model, it is really important to take care of bias and variance in order to avoid overfitting and underfitting in the model.
- If the model is very simple with fewer parameters, it may have low variance and high bias.
- Whereas, if the model has a large number of parameters, it will have high variance and low bias.
- So, it is required to make a balance between bias and variance errors, and this balance between the bias error and variance error is known as **the Bias-Variance trade-off**.

Cross Validation

- Cross-validation is a statistical method used to estimate the performance (generalization ability) of machine learning models.
- It involves partitioning the dataset into subsets, training the model on some subsets, and validating it on the remaining subsets. This technique helps in assessing how the model will perform on an independent dataset.
- **Purpose of Cross-Validation**
- **Performance Estimation:** Provides a more accurate estimate of model performance on unseen data.
- **Model Selection:** Helps in selecting the best model among different algorithms or hyperparameters.
- **Reducing Overfitting:** Ensures that the model's performance is not overly optimistic by training and testing on different subsets of data.

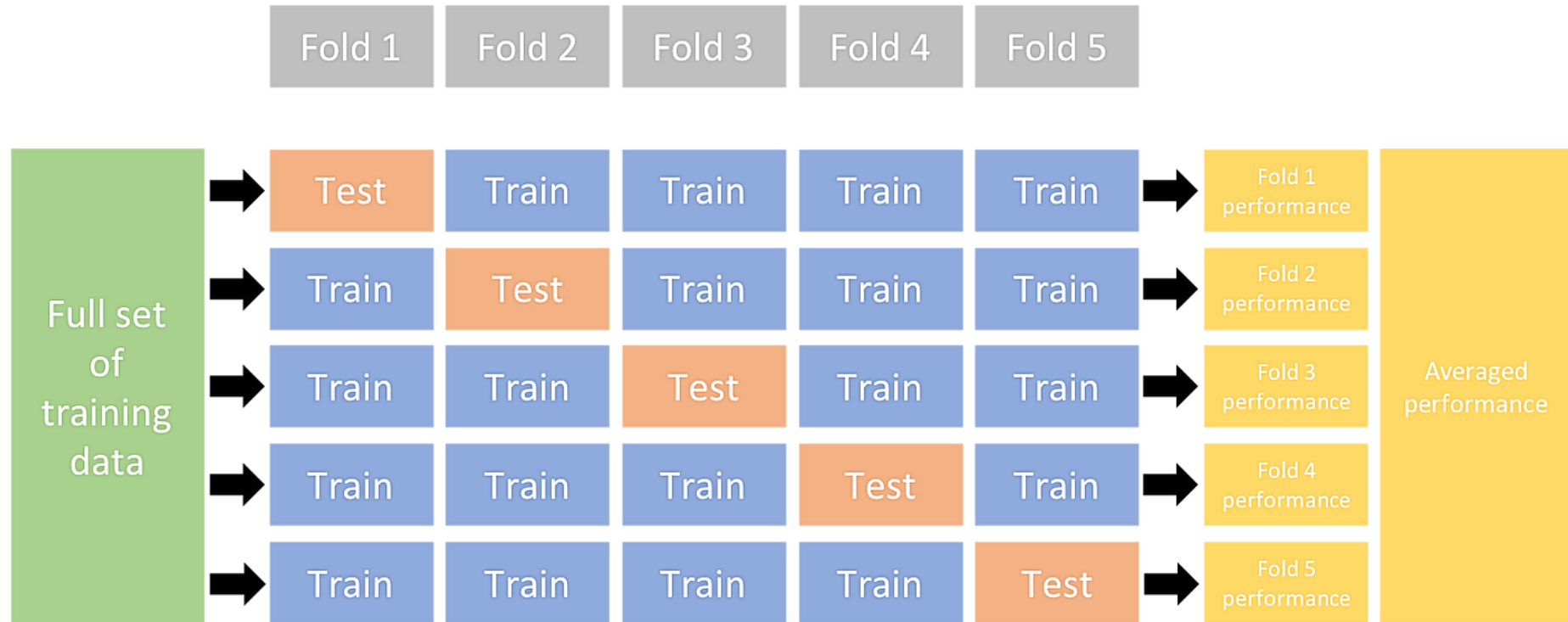
K-Fold Cross-Validation

- K-Fold Cross-Validation is a robust method for evaluating the performance of a machine learning model.
- It involves splitting the dataset into k equally sized (or nearly equal) subsets or "folds". The model is trained and validated k times, with each fold being used once as the validation set while the remaining $k-1$ folds serve as the training set.
- This approach ensures that every data point is used for both training and validation, providing a comprehensive performance estimate.

Process

1. **Data Splitting:** The dataset is randomly divided into k equally sized folds.
2. **Training and Validation:** For each fold:
 - Use $k-1$ folds for training the model.
 - Use the remaining fold for validating the model.
3. **Repeat:** Repeat the process k times, each time using a different fold as the validation set.
4. **Performance Aggregation:** Aggregate the performance metrics (e.g., accuracy, precision, recall, F1-score) from all k iterations to get the final estimate.

K-Fold Cross-Validation



Hyperparameter Tuning techniques

Models can have many hyperparameters and finding the best combination of parameters can be treated as a search problem.

The two best strategies for Hyperparameter tuning are:

1. GridSearchCV
2. RandomizedSearchCV

1. GridSearchCV:

- One traditional and popular way to perform hyperparameter tuning is by using an Exhaustive Grid Search from Scikit learn.
- This method tries every possible combination of each set of hyper-parameters.
- Using this method, we can find the best set of values in the parameter search space.
- This usually uses more computational power and takes a long time to run since this method needs to try every combination in the grid size.
-

Define the model

```
model = RandomForestClassifier(random_state=42)
```

Define the hyperparameter grid

```
param_grid = {  
    'n_estimators': [10, 50, 100],  
    'max_depth': [None, 10, 20, 30],  
    'min_samples_split': [2, 5, 10]  
}
```

Set up GridSearchCV

```
grid_search = GridSearchCV(estimator=model, param_grid=param_grid, cv=5,  
scoring='accuracy', verbose=2, n_jobs=-1)
```

- **Parameter List**

**cv=5 typically refers to using 5-fold cross-validation*

** scoring='accuracy' it means that the accuracy metric is used to evaluate the performance of the model.*

2. RandomizedSearchCV

- The main difference in the RandomizedSearch CV, when compared with GridCV, is that instead of trying every possible combination, this chooses the hyperparameter sample combinations randomly from grid space.
- Because of this reason, there is no guarantee that we will find the best result like Grid Search. But, this search can be extremely effective in practice as computational time is very less.
- The computational time and model performs mainly depends on the n_iter value.
- Because this value specifies how many times the model should search for parameters. If this value is high, there is a better chance of getting better accuracy, but also this comes with more computational power.

Example:

- **# Set up RandomizedSearchCV**
random_search = RandomizedSearchCV(estimator=model,
param_distributions=param_dist, n_iter=100, cv=5, scoring='accuracy',
verbose=2, n_jobs=-1, random_state=42)
- **# Fit the model**
random_search.fit(X_train, y_train)