Important source

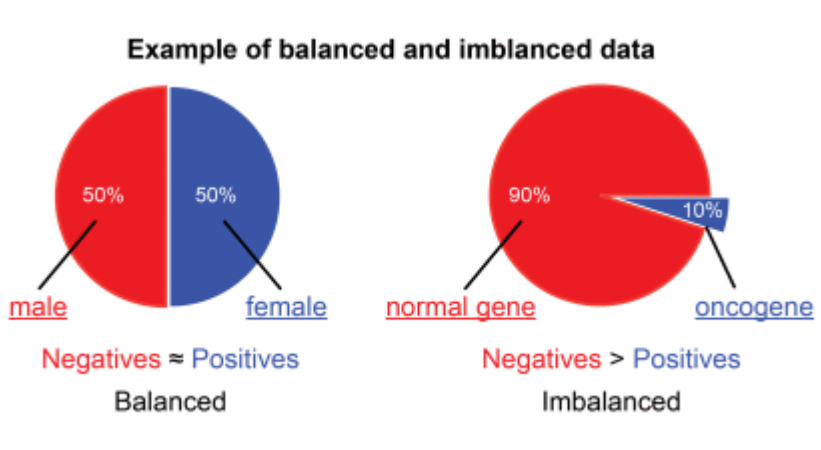
- https://christophm.github.io/interpretable-ml-book/intro.html

In [ ]:

> KNN gets effected when the data is imbalanced and column standardization is one important technique to be followed before proceeding with the algorithm.

Balanced data vs Imbalanced data

- A **balanced data** has an equal number of observations for all possible level of combinations.
- If there is a high disparity of observations for all possible level of combinations is called **imbalanced data**.

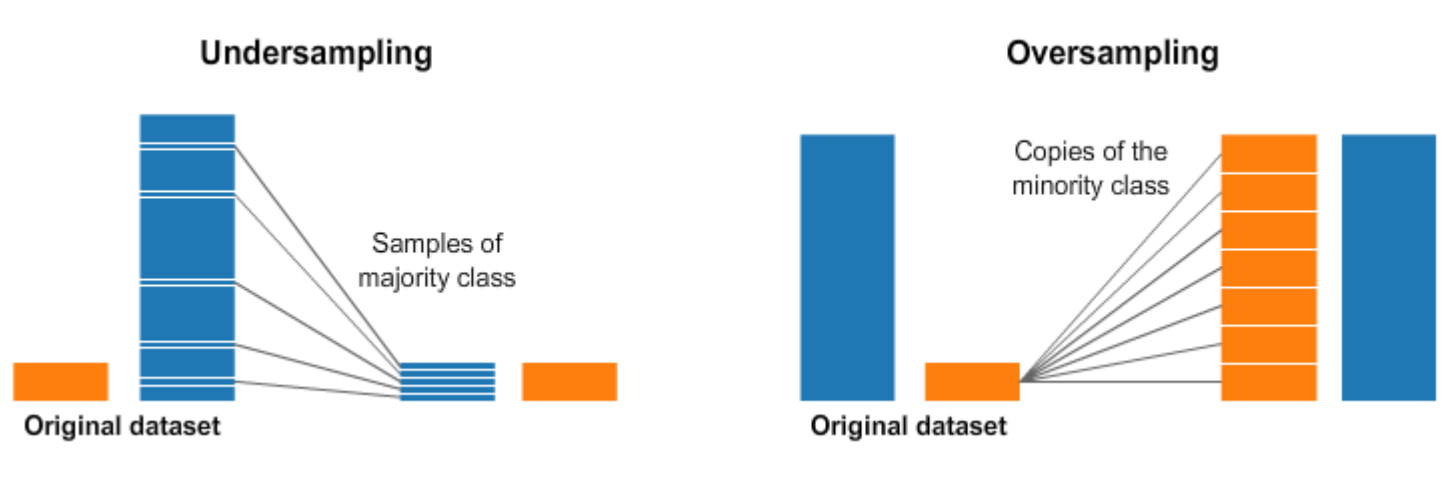


Credits - Image from Internet

In [ ]:

How to work around imbalanced data?

- **Undersampling**
  - The simplest undersampling technique involves randomly selecting examples from the majority class and deleting them from the training dataset. This is referred to as random undersampling.
  - We make sure that after selecting the random sample, the size remains same to that of minority class.
  - The model will end up having very small amount of data.
- **Oversampling**
  - The simplest oversampling technique involves randomly selecting examples from minoroty class with replacesments from the training dataset. This is referred to as random oversampling.
  - We make sure that after sampling (random) with replacement, the size remains same to that of majority class.
  - The model will end up having sufficient amount of data for effective training.
  - **Extrapolation** - Creating new data by considering all the data values of a certain class. These points are called synthetic points.



Credits - Image from Internet

> **_With an imbalanced data the accuracy (metrics) can be high and here the `accuracy_score` doesn't work effectively._**

In [ ]:

Multi-class classification

- Binary Classifier → $y_i \in \{0, 1\}$
- Multi-class Classifier → $y_i \in \{0, 1, 2, \ldots, 9\} \implies \text{MNIST data set}$

> KNN can easily be extented to multi-class classifier. It uses both majority voting and probabilistic methods to classify the data.
>
> Logistic regression is used for binary classification. Given a multi-class classfication problem, can we convert it into binary classification problem such as - from $\text{f}(x) \rightarrow \{0, 1\}$ to $\text{g}(x) \rightarrow \{0, 1, 2, 3, \ldots, C-1\}$ (considering there are C classes).

- Imagine the $y_i \in \{0, 1, 2, 3, \ldots, C-1\}$, we need to develop a model in such a way that it classifies the data totally C times. Basically, we are going to train C models.
  - Divide (D_n) in two part (to replicate binary classification problem)
    - $D_n \rightarrow \{(x_i, y_i) | y_i = 0\}$
    - $D_n \rightarrow \{(x_i, y_i) | y_i \neq 0\}$
  - Divide (D_n) in two part (to replicate binary classification problem)
    - $D_n \rightarrow \{(x_i, y_i) | y_i = 1\}$
    - $D_n \rightarrow \{(x_i, y_i) | y_i \neq 1\}$
  - ...
  - Divide (D_n) in two part (to replicate binary classification problem)
    - $D_n \rightarrow \{(x_i, y_i) | y_i = C-1\}$
    - $D_n \rightarrow \{(x_i, y_i) | y_i \neq C-1\}$

> This technique is called ***1 VS Rest***.

In [ ]:

Training & Test Set differences

- When data changes over time, we need to be very careful in doing time based splitting.

Q) How to determine if the data is changing over time?

- We can do this by plotting D_{Train} and D_{Test} values.

Q) Test if D_{Train} and D_{Test} have different distributions?

- Nomally in TBS we divide D_n into two sets
  - $D_{Train} \rightarrow (x_i, y_i) \implies (x_i^1, y_i^1)$
  - $D_{Test} \rightarrow (x_i, y_i) \implies (x_i^1, y_i^1)$
- In order to do the above, we shall create a new data set D_n^1 such that
  - D_{Train}^1 will have $y_i^1 = 1; x_i^1 = \text{concat}(x_i, y_i)$ in D_{Train}
  - D_{Test}^1 will have $y_i^1 = 0; x_i^1 = \text{concat}(x_i, y_i)$ in D_{Test}
- After doing so, build a binary classifier on D_n^1
- We then ask the classifier to separate the points in two classes.
  - If they get well separated, they are highly dissimilar.
  - Else, they are sort of similar.

In [ ]:

Impact of Outliers

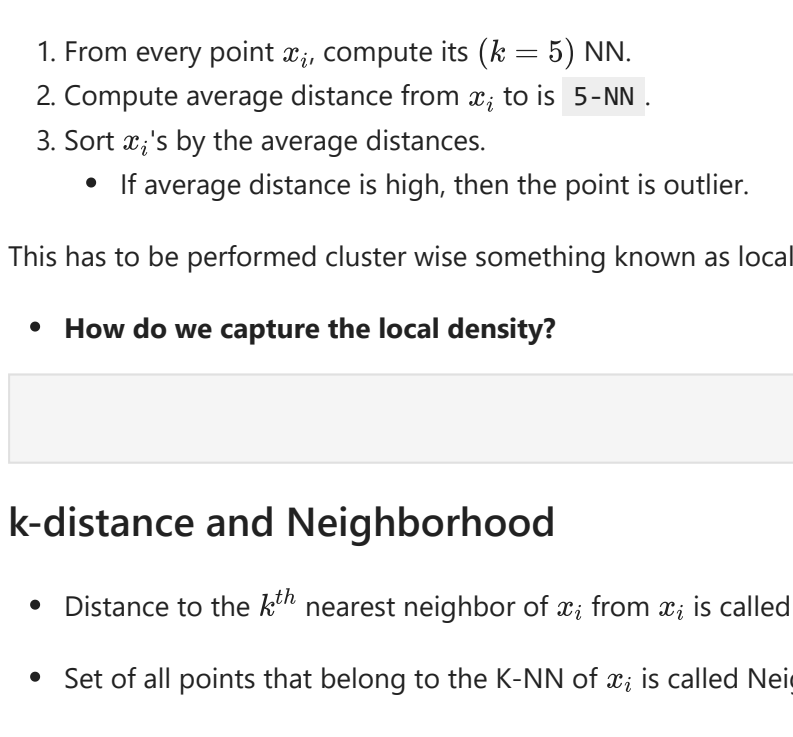
- When `k` is small, outliers can easily impact the model.
- When we have a situation where for `10 fold` cross validation we have accuracies of all the `k`'s as -
  - **k = 1 → 97%**

  - k = 2 → 97%
  - k = 3 → 97%
  - k = 4 → 97%
  - **k = 5 → 97%**

  - k = 6 → 95%
  - k = 7 → 92%
  - ...
  - In this scenario, it is always better to choose larger `k` i.e., `k = 5` because it is less prone to outliers, whereas `k = 1` is very prone to outliers.

Local Outlier Factor (Simple Solution)

- Let's say that we have two clusters such as C_1, C_2 and outliers such as x_1 and x_2 as shown in the below figure.



- Here red lines show the average distance between any random point x_i capturing the distances between the point and it 5 nearest neighbors. We do the same thing with outlier points.

Q) Given this, how do we detect the outliers?

1. From every point x_i, compute its $(k = 5)$ NN.
2. Compute average distance from x_i to is `5-NN`.
3. Sort x_i's by the average distances.
   - If average distance is high, then the point is outlier.

This has to be performed cluster wise something known as local density. But the ultimate question still remains unanswered-

- **How do we capture the local density?**

In [ ]:

k-distance and Neighborhood

- Distance to the k^{th} nearest neighbor of x_i from x_i is called k-distance.

- Set of all points that belong to the K-NN of x_i is called Neighborhood $N(x_i)$.

In [ ]:

Reachability-Distance(A, B)

In simple, the reachability distance is written as -

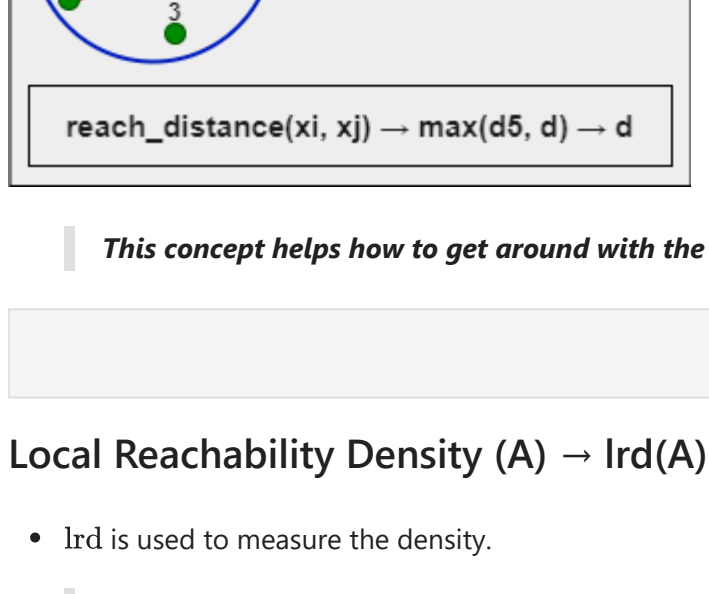
$$\text{reach\_distance}(x_i, x_j) = \max\left[\text{k\_distance}(x_j), \text{dist}(x_i, x_j)\right]$$

where

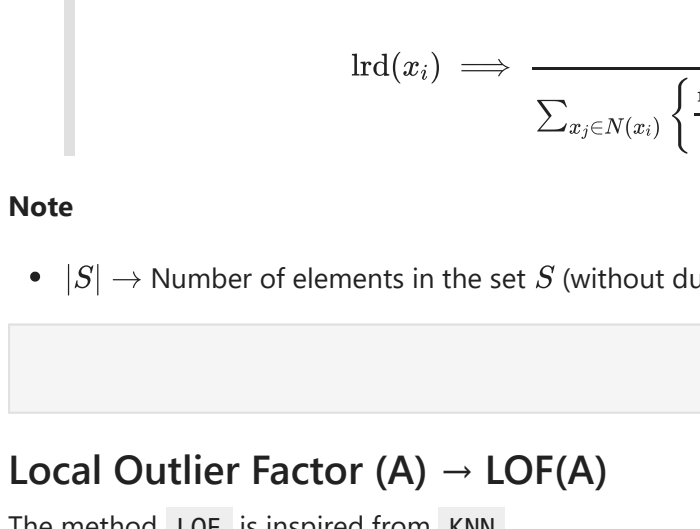
- $\text{k\_distance}(x_j) \rightarrow$ distance of K^{th}-NN of x_j from x_j
  - If $(k = 5)$, then $\text{k\_distance}(x_j)$ is the distance of the fifth farthest (nearest) point
- $\text{dist}(x_i, x_j) \rightarrow$ actual distance between x_i and x_j

Note

- If $x_i \in N(x_i)$ then $\text{reach\_distance}(x_i, x_j)$ is $\text{k\_distance}(x_j)$.



- Otherwise, $\text{reach\_distance}(x_i, x_j)$ is $\text{dist}(x_i, x_j)$.



> ***This concept helps how to get around with the problem of two clusters...***

In [ ]:

Local Reachability Density (A) → lrd(A)

- lrd is used to measure the density.

> $\text{lrd}(x_i) \implies \text{Inverse of average reachability distance of a point from its neighbors.}$
>
> $$\text{lrd}(x_i) \implies \frac{1}{\sum_{x_j \in N(x_i)} \left\{\frac{\text{reach\_dist}(x_i, x_j)}{|N(x_i)|}\right\}} = \frac{|N(x_i)|}{\sum_{x_j \in N(x_i)} \{\text{reach\_dist}(x_i, x_j)\}}$$

Note

- $|S| \rightarrow$ Number of elements in the set S (without duplicates)

In [ ]:

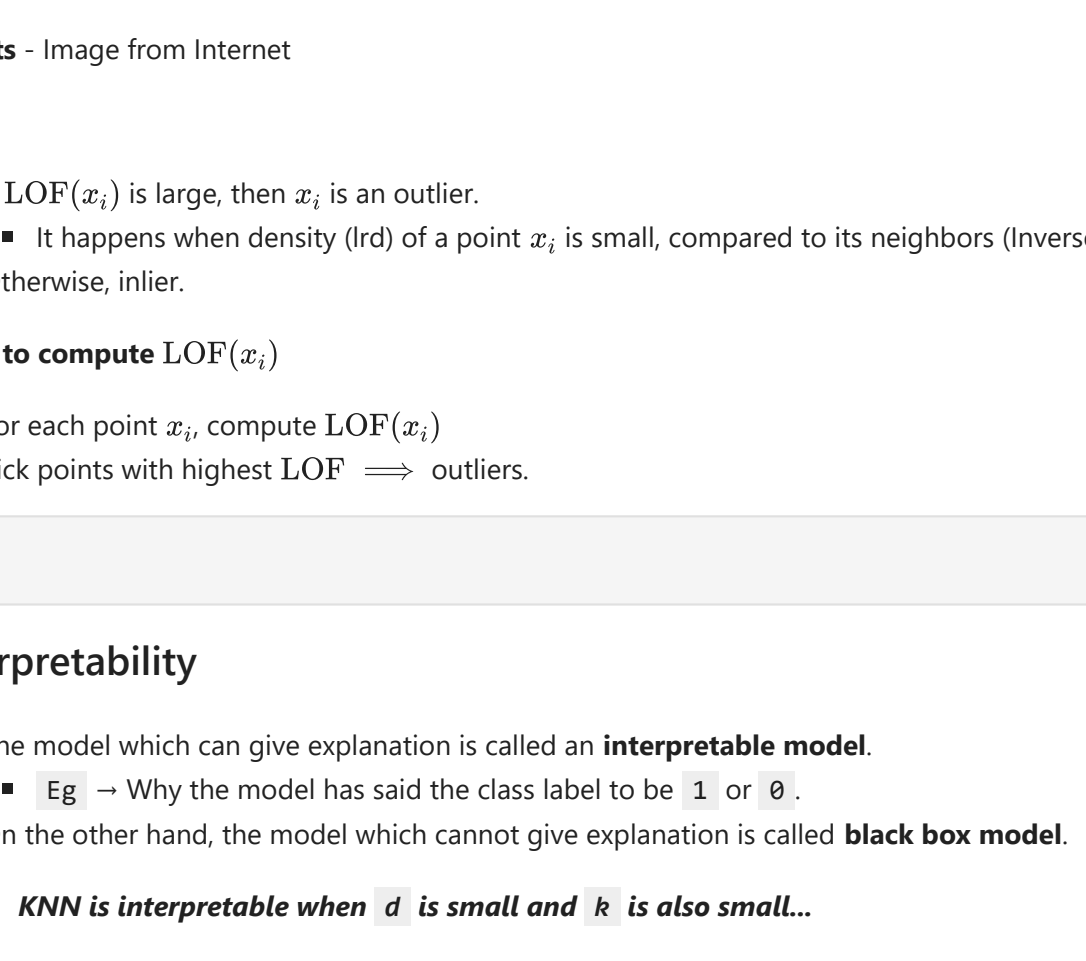
Local Outlier Factor (A) → LOF(A)

The method `LOF` is inspired from `KNN`.

Formula

$$\text{LOF}(x_i) = \frac{\sum_{x_j \in N(x_i)} \text{lrd}(x_j)}{|N(x_i)|} * \frac{1}{\text{lrd}(x_i)}$$

Sample Image



Credits - Image from Internet

Note

- If $\text{LOF}(x_i)$ is large, then x_i is an outlier.
  - It happens when density (lrd) of a point x_i is small, compared to its neighbors (Inverse will be large).
- Otherwise, inlier.

Steps to compute $\text{LOF}(x_i)$

- For each point x_i, compute $\text{LOF}(x_i)$
- Pick points with highest $\text{LOF} \implies$ outliers.

In [ ]:

Interpretability

- The model which can give explanation is called an **interpretable model**.
  - `Eg` → Why the model has said the class label to be `1` or `0`.
- On the other hand, the model which cannot give explanation is called **black box model**.

> ***KNN is interpretable when `d` is small and `k` is also small...***

In [ ]:

Feature Importance

- Refer → https://www.analyticsvidhya.com/blog/2016/12/introduction-to-feature-selection-methods-with-an-example-or-how-to-select-the-right-variables/

- Selecting which features are most important for the **classification** or **regression** problems. Sort the features by their importance and suitability for the problem statement.
- This technique is very useful in understanding the model better and thus model interpretability increases.
- Models like **Logistic Regression**, **Decision Trees**, can be really helpful to get the features based on their importance.
  - `KNN` by default provide or get the important features.

Forward Feature Selection

- Imagine your data set is divided into two $[D_{\text{Train}}, D_{\text{Test}}]$ and the features are $\{f_1, f_2, f_3, \ldots, f_d\}$. Let's say we have a model (`KNN`) in which $[D_{\text{Train}}, D_{\text{Test}}]$ are passed.

- Now, take each feature at a time and train the model using only that (one) feature and store the accuracy.

  - Repeat this process for d times.
  - Let's say that the feature f_{10} is giving highest accuracy.
  - List of features $\implies \{f_1, f_2, \ldots, f_9, f_{11}, f_{12}, \ldots, f_d\}$
  - **Best feature** - f_{10}
- Now, take each feature at a time and train the model using only that (one) along with the best feature that is selected in the previous step (f_{10}) and store the accuracy.
  - Repeat this process for $(d-1)$ times.
  - Let's say that the feature f_5 is giving highest accuracy.
  - List of features $\implies \{f_1, f_2, f_3, f_4, f_6, \ldots, f_9, f_{11}, f_{12}, \ldots, f_d\}$
  - **Best features** - $\{f_{10}, f_5\}$
- Repeat the whole process based on the threshold that is required to have best features and is sufficient to solve the given problem.
- This technique of selecting features iteratively is called forward selection.
- Time complexity is very large since we have to train the model d times and decreases by 1 iteratively.
- **Model agnostic** - Not caring about the model and doing the process.

Backward Selection

- Takes all the features at a time and in every iteration, it removes each features and tracks the accuracy.

In [ ]:

Handling Categorical & Numerical Features

- Refer
  - https://contactsunny.medium.com/label-encoder-vs-one-hot-encoder-in-machine-learning-3fc273365621
  - https://towardsdatascience.com/choosing-the-right-encoding-method-label-vs-onehot-encoder-a4434493149b

Different ways of converting categorical data into numeric features

1. Simply convert the categorical data into number by replacing random numbers.

2. Best solution for this is to do `one-hot-encoding` rather than giving artificial numbers.

   - Imagine I have data such as `['black', 'brown', 'green', 'red']`. For this we can create five vectors.
     - `black` → [1, 0, 0, 0]
     - `brown` → [0, 1, 0, 0]
     - `green` → [0, 0, 1, 0]
     - `red` → [0, 0, 0, 1]
3. If the number of distinct values for a categorical feature is large then `one-hot-encoding` can create sparse and large vectors.

   - In that case, we can take mean value of `target` for each category and replace it in the entire data set. This is called `mean-replacement`.
4. Use domain knowledge and convert.

In [ ]:

Handling `NaN` Values

Missing values in the data occur due to various reasons. Some of them are -

- Data Corruption
- Collection Error

Q) How can we featurize the missing data?

- Strategies to fill the missing values

  - Imputation

    - Mean replacement
    - Median replacement
    - Mode replacement
    - (If certain feature is categorical, then compute any of the above strategies based on class label or target variable)
  - Create a new feature of missing values after imputing. Generally, this new feature will comprise binary values -

    - `1` → representing the value is missing
    - `0` → representing the value is not missing.
  - Model based imputation
    - Create whichever column has missing as a `target` column and specifically in the test set. Use a machine learning model to predict the missing value.
    - `KNN` is often used for model based imputation because of the neighborhood concept.

In [ ]:

Curse of Dimensionality

- Refer → https://en.wikipedia.org/wiki/Curse_of_dimensionality

Generally when the dimensions of the data increases the problem arises.

- In **Machine Learning**

  - As the dimensions increase, the total number of data points increase exponentially.
  - Let's say we have got binary (0, 1) features (3) → $f_1, f_2, f_3 \implies 2^3 = 8$
  - Similarly if we have 10 features → $f_1, f_2, \ldots, f_{10} \implies 2^{10} = 1024$
  - With a fixed number of traning samples, the prediction power of a model decreases as the dimensions increase. This is called **Hughes Phenomenon**.
- In **Distance Functions** (euclidean distance)

  - The intuition of distances in `3D` is not valid in higher dimension spaces.
    - In `1D`, `2D`, and `3D` the equation $\frac{\text{dist\_max}(x_i) - \text{dist\_min}(x_i)}{\text{dist\_min}(x_i)} > 0$
    - As `d` increases the above equation tends to 0. This can be proved by limit concepts in mathematics.
  - **Twist**
    - If the data is **high dimensional** and **dense** → impact of dimensionality is high.
    - If the data is **high dimensional** and **sparse** → impact of dimensionality is low.
- In **Overfitting** & **Underfitting**

  - As `d` increases, the chances of model getting overfitted in high.
  - To solve this, we can make use of -
    - **Forward Feature Selection** to pick the most useful subset of features.
    - **PCA** and **T-SNE** can also be used to reduce the dimensions.

Intuitional Explanation

- Let's say you have a straight line 100 yards long and you dropped a penny somewhere on it. It wouldn't be too hard to find. You walk along the line and it takes two minutes.

- Now let's say you have a square 100 yards on each side and you dropped a penny somewhere on it. It would be pretty hard, like searching across two football fields stuck together. It could take days.

- Now a cube 100 yards across. That's like searching a 30-story building the size of a football stadium. Ugh.

- The difficulty of searching through the space gets a *lot* harder as you have more dimensions. You might not realize this intuitively when it's just stated in mathematical formulas, since they all have the same "width". That's the curse of dimensionality. It gets to have a name because it is unintuitive, useful, and yet simple.

In [ ]:

Bias Variance Tradeoff

$$\text{Generalization Error (future data error)} = \text{Bias}^2 + \text{Var} + \text{Irreducible Error}$$

- **Bias** - Errors due to simplifying assumptions.
  - If the bias is high, it means the model will underfit, otherwise not.
- **Variance** - Represents, how much the model changes as training data changes.
  - High variance model always leads to overfitting.
- **Irreducible Errors** - Errors which cannot be further reduced.

We have to tradeoff to find the moderate solution in which there is not much bias and less variance. This balancing of the model performance is always encouraged to apply.

In [ ]: