**Bias-Variance Tradeoff**

* Model complexity is key factor that regulates bias and variance of the model performance.
* Our goal is to minimize generalization error by *reducing bias without introducing too much variance*.

A diagram of a normalized curve

Description automatically generated with medium confidence

1. As model becomes more complex, it tends to encounter overfitting where model can capture well on underlying pattern based on existing data but cannot perform well on generalization on unseen data.
2. As model becomes less complex, it tends to encounter underfitting where it lacks understanding on underlying pattern by introducing more bias, resulting less capacity to fit the model well and poor generalization capacity
   * + If model captures too much variance, it tends to be train to overfit the data.

A white background with black text

Description automatically generated

* + - If it captures too much bias, it tends to be trained to underfit the data.

A white background with black text

Description automatically generated

1. The main point is to seize the **sweet spot** where the model complexity can capture relatively low bias and variance to ensure it can capture the underlying data pattern and generalize to unseen data.
   * Avoid overtraining or undertraining models
     + To ensure the model can learn enough to learn the trend without overfitting to the noise.
     + If model captures too well on training data, generalization to unseen data will be poor.

**Strategies to avoid overfitting and underfitting problems**

1. *Train-test split*
   * Train model with training data.
   * Keep checking model performance on validation data.
   * The sweet spot can be detected when the validation performance is decreasing.
2. *Cross-validation: K-Fold cross-validation*
   * Arrange data into (K – 1) folds for training and K fold for testing.
   * Average across all scores in # of K folds

***K-Fold Cross Validation***

* **Splitting the Data**:
  + In k-fold CV, the dataset is randomly divided into **k subsets**, (usually 5 or 10) with equal sizes.
  + For each iteration, the model is trained on **k-1 folds (as training data)** and evaluated on the **remaining 1-fold (as validation data).** This process is repeated **k times**, with each fold being used once as the validation set.
* **Training and Evaluation Process**:
  + During each iteration, one of the folds acts as the **validation set**, while the remaining **k-1 folds** are combined to form the **training set**.
  + The model is trained on the training set and then evaluated on the validation fold.
  + The performance metric is recorded for each iteration.
* **Averaging Results**:
  + After completing all **k iterations**, the model’s performance is averaged over the **k folds** to provide a more reliable estimate of how the model is expected to perform on new, unseen data to determine the optimal hyperparameters during model selection.

A graph of a graph

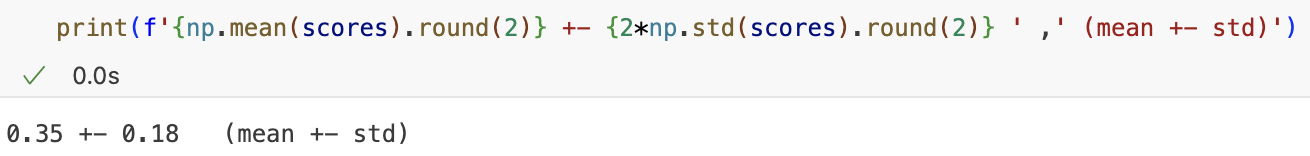
Description automatically generated with medium confidence

Code example with K-Fold CV in decision tree model

A screen shot of a computer code

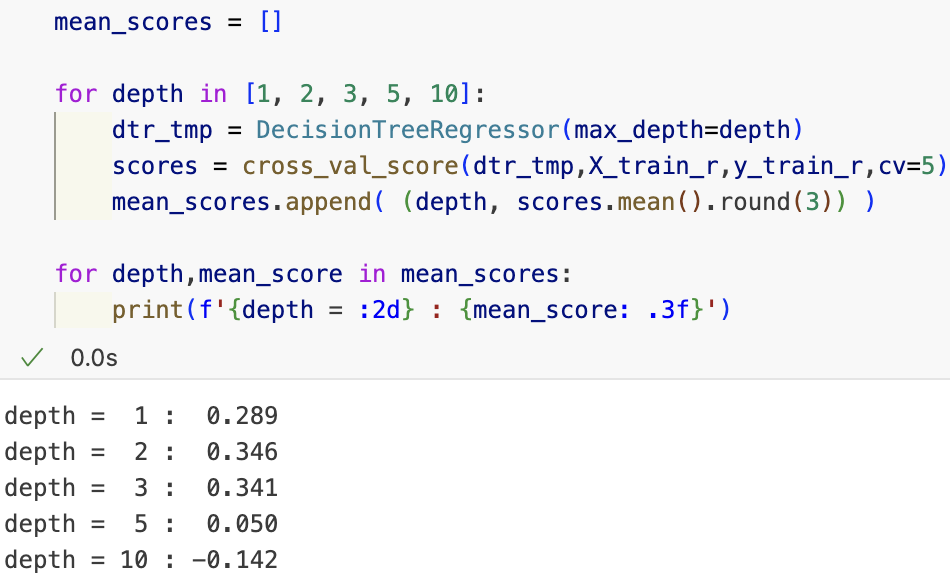
Description automatically generated

* We use ***cross\_val\_score ()*** as model training strategy to regulate overfit and underfit.
* DecisionTreeRegressor(max\_depth = 2) indicates we take 2 decision tree as model training process for regression purpose.
  + We only use 2 trees to prevent the overfitting.
* X\_train\_r and y\_train\_r are used for training purposes with Cross validation.
* cv = 5 indicates that we take training dataset by dividing into 5 different subsets each of which we take 1-fold as testing and 4-folds as training and repeat such step iteratively under 5 subsets.
  + At the end, we will obtain 5 accuracy score in all 5 subsets, where each subset has performed cross validation tasks by taking K fold for testing and K-1 fold for training.



* We take the average of the total 5 subsets performance with standard deviation.
* At the end, we train the same model (decision tree regressor) 5 times into 5 subsets where each subset we train 4 folds and test 1-fold and take the average of the accuracy performance of each subset.

**Hyperparameter tuning with cross validation**



* The # of hyperparameter indicates # of models to be trained.
* **Purpose**: we need to set a list of hyperparameter so that we can know which hyperparameter can offer the optimal performance by training model with K-Fold Cross validation.
  + Since we define 5 hyperparameters for training decision tree model, there will be 5 models to be trained where each decision tree model will perform with each hyperparameter.
  + Since we apply cross validation for training model strategies, we will train model 25 times by dividing 5 times from total 5 models (5 decision tress models with different hyperparameters).
* For each model with different hyperparameter, we can get the average cross validation score for each hyperparameter for comparison.
  + As a result, we can determine decision tree with setting hyperparameter as 2 can give the best model performance after 5-fold cross validation.

Validation curve in graph

* We can use validation curve to find a sweet spot by model score between training score and validation score

A diagram of a training and model

Description automatically generated

* + By increase model complexity, we can train the model to fit perfectly with training set, but it will lead to the poor performance on validation set.
  + By Elbow rule, we need to find a point where validation score is going to decline.

Code example

A screenshot of a computer

Description automatically generated

* We deploy 6 hyperparameters to the decision tree model to training the model with X\_train\_r and y\_train\_r and use 3-fold cross validation strategy to train each model.
* In the end, we trained models 18 times with 6 decision tree models with different hyperparameter and 3-fold cross validation for each model.

**Training performance**

A screenshot of a computer

Description automatically generated 🡺 averaged CV score for 6 models with each hyperparameters among training data

**Testing performance**

A screenshot of a test

Description automatically generated 🡺 averaged CV score for 6 models with each hyperparameters among testing data

Average training and testing performance for 3 subsets of each model.

A screenshot of a computer program

Description automatically generated

**Plotting the training-validation performance** under each hyperparameter

A graph with a red line and blue line

Description automatically generated

* The validation performance starts to drop when we have 2 or 3 trees in the model.
* The performance is optimal to prevent overfitting and underfitting when the trees are 2 or 3.

**Grid Search**

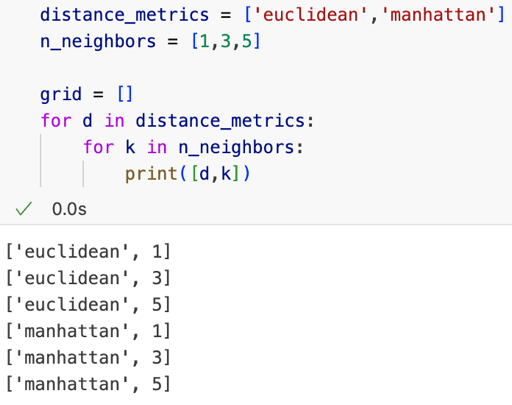
When the hyperparameter is more than one in a set and is hard to find which one is the best set of hyperparameter, we can use Grid Search cross validation (GridSearchCV) to find the optimal set of hyperparameter from 1 dimensional to n-dimensional space.

* **Grid Search** (uniformed search strategy) 🡺 parallel
  1. It involves an exhaustive search over a manually specified subset of hyperparameter space.

A screenshot of a computer

Description automatically generated🡺 The total number of combinations in this grid is 3 \* 3 \* 2 = 18. Grid search will try each of these 18 combinations, train the model, and use cross-validation to evaluate performance for each.

* 1. The main goal of grid search is to find the best combination of hyperparameters that maximizes the model’s performance.



* When there are 2 hyperparameters for distance metrics and 3 hyperparameters for number of neighbors for kNN model, we can determine the # of hyperparameters by multiplying # of the hyperparameters in each set (2 X 3 = 6).

A screenshot of a computer code

Description automatically generated

* We set the hyperparameter set into dictionary since it contains more than 1 sets. In grid search, it will contain 5 X 2 = 10 hyperparameters in total for kNN regressor model to train.
* For each model with different set of hyperparameters, it will be trained with 3 cross validation sets, indicating that the model will be trained 10 X 3 times where there will be 10 kNN model with different hypermeters and will be trained with 3 subsets under each model.

A screenshot of a computer code

Description automatically generated

* We will use X\_train\_r and y\_train\_r to train the model 30 times with different sets of hyperparameters.
* Use .best\_params\_ to get the best set of hyperparameter for later training.

A screen shot of a computer

Description automatically generated

* With Euclidean metric and 5 neighbors as the best set of hyperparameters, we can use this parameter (.best\_params\_) once again to deploy to the kNN model with 5-fold cross validation and give the average mean score of the model performance.
* In the end, there will be **2 X 5 X 3 + 1 = 31 times** for model training for it will extract the best set of hyperparameter for final training.

**Classification tasks with wine dataset**

A screen shot of a computer

Description automatically generated 🡺 class will the target variable for classification

A screenshot of a computer

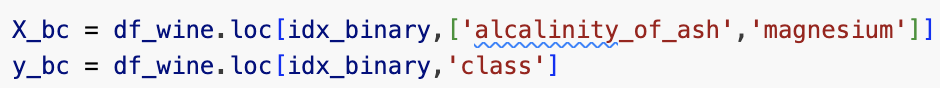
Description automatically generated 🡺 those 2 variables are used as IVs to perform the task

* We will convert the predicted outcome as binary classification tasks

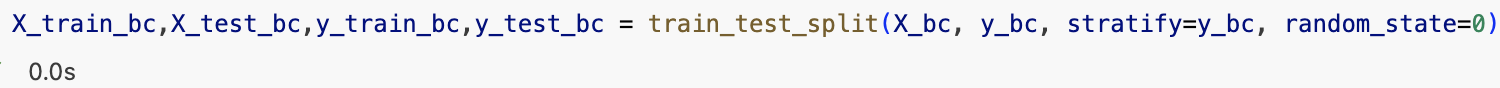
A screen shot of a computer code

Description automatically generated

* We need to select the features and target that correspond to each other



* + **Purpose**: Extracts the features and target for the filtered rows.
    - X\_bc: Contains only the alcalinity\_of\_ash and magnesium columns for rows marked True and False in idx\_binary.
    - y\_bc: Contains the class column (target variable) for the filtered rows.
  + **Result**: A subset of the dataset (X\_bc and y\_bc) focusing on binary classification.
* Data split into train and test set with stratified splitting strategy



* + stratify=y\_bc: Ensures the class proportions in the training and test sets match those in the original data to preserve the proportion of each class
* Count class distribution across train and test split

A screen shot of a computer

Description automatically generated

* + Use dataframe in Pandas to display the distribution of each class frequency under each split.

**Classification Metrics in Accuracy**

**Accuracy: out of all observations, what percentage did I get it right?**

**Dummy classifier:**

* Use dummy classifier as mean of making predictions using simple rules rather than learning patterns from the data. Its purpose is to provide a benchmark against which more complex models can be compared.

1. Set the dummy classifier as baseline for later comparison

A screenshot of a computer code

Description automatically generated

* + Use (strategy = ‘prior’) to assign the dummy model to predict majority of class.
  + Even though we use dummy classifier to fit the model with training and testing data, the model doesn’t learn from the data, it tends to predict the majority of class assigned by strategy.

**Decision Tree Classifier:**

1. Set decision tree classifier to learn from train and test data

A screenshot of a computer code

Description automatically generated

* + Set 2 decision trees to learn from the data and make the prediction.
  + Decision tree model will learn training data and make the prediction to the test data.
  + The performance is much better based on the accuracy score than dummy model.

**Classification model evaluation metrics**

* **Threshold-based metrics** (define *threshold* to determine point is classified one way or the other) 🡺 **1 or 0**
  + Classification accuracy
  + Precision, Recall & F1-Score
* **Ranking-based metrics** (use *score for ranking* the specific range of samples) 🡺 **top 1,000**
  + Average Precision (AP)
  + Area Under Receiver Operating Curve (AUROC)

For **Threshold-based metrics**, we use Confusion matrix to determine the metrics

A diagram of different negatives

Description automatically generated

* **TP:** Correctly predicted positive instances.
* **TN**: Correctly predicted negative instances.
* **FP**: Incorrectly predicted positive instances (actual label is negative).
* **FN**: Incorrectly predicted negative instances (actual label is positive).
  + **Strategy to interpretation**
    - **Positive (TP or FP): positive always refers to prediction** (not with respect to original label) **🡺 prediction is positive, and that prediction is either true (TP) or false (FP)**
    - **Negative (TN or FN): negative always refers to prediction** 🡺 **prediction is negative, and that prediction is either true (TN) or false (FN)**.

**Accuracy**:

A black and white text

Description automatically generated

* It calculates as the ratio of correctly predicted instances to the total number of instances.

A diagram of a scale

Description automatically generated with medium confidence

Limitations of Classification Accuracy

1. **Misleading in Imbalanced Datasets**:

* Accuracy can be deceptive when classes are imbalanced when a model can achieve high accuracy simply by always predicting the majority class, even though it fails to identify the minority class correctly.
* For instance, in a dataset where 95% of samples belong to one class, a model that always predicts this majority class will still achieve 95% accuracy without actually understanding the underlying data patterns.

1. **Accuracy Paradox**:

* The accuracy paradox refers to situations where high accuracy does not necessarily reflect good performance where the model’s cost of false positives and false negatives is asymmetric
* For example, in fraud detection, misclassifying a fraudulent transaction as legitimate (a false negative) has a much higher cost than the opposite. Relying solely on accuracy in such cases can lead to misleading conclusions about the model’s effectiveness.

**Code for visualizing confusion matrix**

A screen shot of a computer

Description automatically generated

* we can use target variables predicted by X trained and tested to visualize for confusion matrix.
* We actually need apply test predicted result and visualize in mlxtend.

A screenshot of a computer screen

Description automatically generated

* + Confusion matrix by accuracy can provide more visual convenience to measure model performance on binary outcome.
  + We need to use metrics to quantify how well the model perform.
    - In this case, accuracy may not be best model since the data is imbalanced since it will take the majority class in account for biased prediction.

**Precision:** (When the model predicts positive, how often is it correct?)

*Out of observations I predicted positive (TP + FP), how many are truly positive (TP)?*

A black background with white text

Description automatically generated 🡺 focus on predictive positives

* It measures proportion of true positive predictions out of all positive predictions made by the model.
* It indicates how many of the predicted positive cases were actually positive.
* **Precision is always with respect to predicted values.**
* **Precision is true positive by predicted positive**

Precision interpretation

1. **High Precision**:
   1. Indicates that when the model predicts a positive result, it is usually correct, meaning there are few false positives.
2. **Low Precision**:
   1. Implies that the model has a high rate of false positives, meaning it incorrectly predicts negative cases as positive.

A screenshot of a computer program

Description automatically generated

* Measure precision by training dummy classifier and decision classifier with 5-fold cross validation

**Recall:** (How well does the model capture all positive original cases?)

*Out of truly positive (TP + FN), how many observations did I predict positive (TP)?*

A black background with white text

Description automatically generated 🡺 focus on actual positives

* It measures proportion of actual positive cases that the model correctly identified.
* It shows how well the model captures all the true positive instances.
* **Recall is always with respect to original values**
* **Recall is true positive by actual positive**

Recall interpretation

1. **High Recall**: Indicates that the model captures most of the actual positive cases, with few false negatives.
2. **Low Recall**: Suggests the model misses many actual positive cases, resulting in a high rate of false negatives.

A screenshot of a computer code

Description automatically generated

* Measure recall by training dummy and decision tree classifier with 5-fold CV.

Comparison over metrics to determine which one is the best

* From accuracy, recall, and precision, we can determine which model is the best based on the purpose of the problem.
  + If the data is **highly imbalanced**, we can turn to **precision or recall** instead of accuracy to determine the best model.
  + Since dummy model tends to **focus on majority class as positive**, the recall will be 1 since it only predicts the majority of class as positive without learning from the data 🡺 recall of dummy = 1.

Note: Minority class in imbalanced dataset

* It is always good practice to treat minority class as positive class (label) when defining precision, recall and f-1 score.

A screenshot of a computer code

Description automatically generated 🡺 use minority class can easily determine the performance efficiency where each metric other than accuracy is 0 🡺 indicating model is poor.

**Setting threshold for precision and recall**

**Threshold**: The cutoff value for deciding whether a prediction belongs to the positive class

* Probability score of 0.5 is a common default threshold.
  + 0.5 is the line that segment 2 different classes.
* Lower thresholds 🡺 increase recall but decrease precision.
* Higher thresholds 🡺 increase precision but decrease recall.

**F-1 Score**: (used for highly imbalanced dataset)

A black and white sign with white text

Description automatically generated

* It measures harmonic mean of precision and recall, balancing the two metrics by penalizing extreme values.
  + **F1 = 1**: Perfect balance between precision and recall (0 FP and 0 FN)
  + **F1 = 0**: Either precision or recall is zero
* It’s useful when we need a single metric to assess models on imbalanced datasets.

A computer screen shot of a computer code

Description automatically generated

* Decision tree is not outperformed since its score is pretty closed to the dummy.

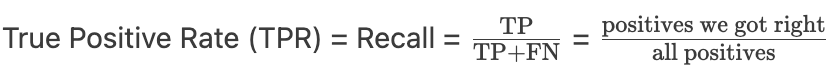
Note for choosing right metric

1. If the model can accurately classify the actual positives, then we use recall.
2. If the model can provide as less negative as possible, then we use precision.
3. If the data is highly imbalanced, we can use F-1.

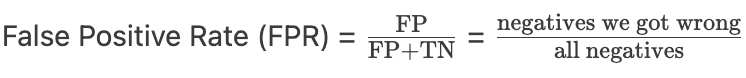
**Receiver Operating Curve (ROC)**

ROC curve plots the relationship between:

* **True Positive Rate (TPR)**, also known as **Recall** or **Sensitivity**, on the y-axis:



* + This indicates the proportion of actual positives that the model correctly identifies.
* **False Positive Rate (FPR)** on the x-axis:



* + This measures the proportion of actual negatives that are incorrectly classified as positive.
* The ROC curve is generated by varying the decision threshold (from 0 to 1) that the model uses to classify a prediction as positive or negative, and plotting TPR against FPR at each threshold.

A white rectangular box with black text

Description automatically generated

* Similarly to the PR curve, each point in the ROC is the threshold for each confusion matrix, generating to TPR and FPR. (PR is depicting precision and recall threshold).

A graph with a line

Description automatically generated🡺 ROC is used to determine one model performance.

* + The ideal ROC curve moves sharply up to the top-left corner, indicating high true positive rates with low false positive rates.
  + This corner signifies high sensitivity and specificity, which are desirable in classification tasks.

**AUROC**

* The goal is to find the largest area under the ROC curve so that we can be sure the model provides the optimal performance for classification.

A screenshot of a computer code

Description automatically generated

* + Dummy model serves as a baseline as .5 portion of the area.

**Ways to choose a model**

1. Use train-test split from the original data.
2. Determine which metric to use for model evaluation
3. Set a baseline model for performance comparison
4. Use K-Fold cross validation to fit hyperparameters and choose model.
   * Use training set for training the model
   * Use the validation set to determine the best set of hyperparameters
   * Use the best hyperparameter to deploy test set and make final evaluation.
5. Evaluation chosen model based on train-test split dataset.

A diagram of a machine learning algorithm

Description automatically generated

**Regularization to prevent overfitting**

Ways to reduce complexity of linear model by minimizing weights.

* Since there is not restriction on the coefficient weight for each feature in linear model, it will easily make the model overfitting.
* Among 100 feature variables, we can use technique to restrict the weights for some of irrelevant features so that we can make the model more interpretable.

Assign penalty to the model

* **Ridge regression**
  + It tends to keep coefficient relatively small.
  + It drives coefficient of uninformative feature to be smaller

A black text with a white background

Description automatically generated

* **Lasso regression**
  + It keeps all coefficient relatively small.
  + It drives coefficient of unhelpful features to be exactly zero.

A black text with a white background

Description automatically generated

* **Elastic-Net regulation**
  + It is mixture of ridge and lasso
  + It will leverage 2 hyperparameters (L1 and L2)

A black text with black letters

Description automatically generated with medium confidence

* + We need to balance which hyperparameter should be put with more weight

A black text with a white background

Description automatically generated

**Ridge regression (L-2 norm)**

A computer screen shot of a code

Description automatically generated

* Use L-2 penalty as hyperparameter with a set of weights to train the X and y in logistic regression model, which tends to make the weight smaller near 0 as C = .001

**Lasso Regression (L-1 norm)**

A screen shot of a computer code

Description automatically generated

* Use L-1 penalty as hyperparameter with a set of weights to train X and y in logistic regression model, and some coefficients are shrunk to exactly 0 when C = .001.

**GridSearchCV with Regularization**

We can determine the best performance hyperparameters with L-1 and L-2 in the logistic

regression model and then use the best parameter to see the accuracy performance between with

regularization and without regularization.

* Determine the best set of hyperparameter by Grid search

A screenshot of a computer code

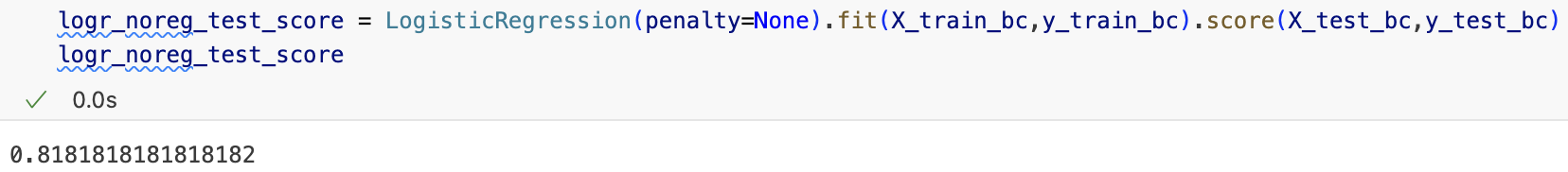
Description automatically generated

* + We found the best hyperparameter set with the best weights for training set.
* Determine the model testing score by comparing against model without regularization
  + Logistic regression with regularization

A screenshot of a computer

Description automatically generated

* + Logistic regression without regularization



* + Even though there is no difference between with or without the penalty for accuracy performance, the accuracy remains consistent to the training set.

Combine different model for compare accuracy performance

A screenshot of a computer program

Description automatically generated

* Elastic-net outperform on testing tasks after being determined by the best set of hyperparameters.