Bias, Variance, and K-Nearest Neighbors

CSC4601 Theory of Machine Learning

Overview

- Last week, we discussed linear vs non-linear models, and we motivated graphically what we mean by complex vs simple models.
- Each model has its own advantages and disadvantages, and each model makes certain assumptions.
- Which model to choose? Where to start?

No-Free-Lunch theorem

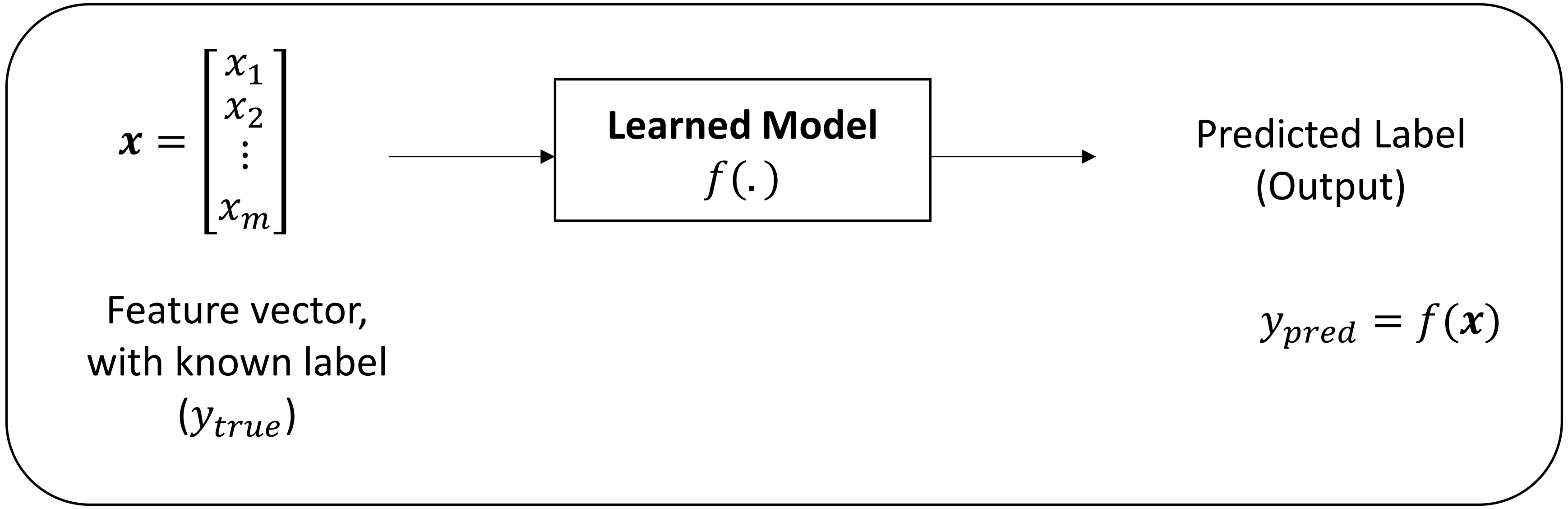
- The "*No Free Lunch*" Theorem argues that, without having prior knowledge, there is no single model that will always do better than any other model.

- Hence it is important to evaluate different approaches or techniques and select the best performing approach.

- In this lecture, we will discuss:
  - Best practices on model evaluation
  - Overfitting vs Underfitting
  - Evaluation metrics (classification)

Model Evaluation

Model Evaluation



$\boldsymbol{x} = \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_m \end{bmatrix}$
Learned Model
$f(.)$
Predicted Label (Output)
Feature vector, with known label (y_{true})
$y_{pred} = f(\boldsymbol{x})$

To evaluate the performance of a machine learning model on a dataset (consisting of feature vectors and known labels), we need quantify how close the model's predictions are to the true labels of the given dataset.

Evaluation Metrics *(you may have used)*

Regression:

- MSE (Mean Squared Error):

$$\frac{1}{n_{samples}} \sum_{i=1}^{n_{samples}} \left(y_{true}^{(i)} - y_{pred}^{(i)}\right)^2$$

- MAE (Mean Absolute Error):

$$\frac{1}{n_{samples}} \sum_{i=1}^{n_{samples}} \left|y_{true}^{(i)} - y_{pred}^{(i)}\right|$$

Classification:

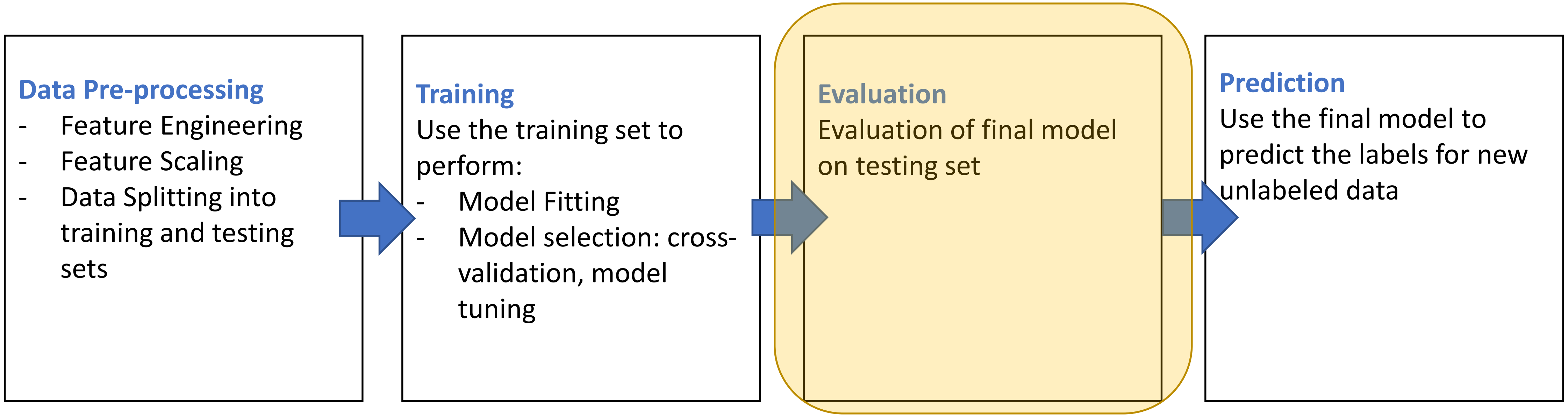
- **Accuracy**: proportion of correctly predicted labels

- Other metrics?

Model Evaluation

- When do we need to evaluate a model?
- And on which data set?

Model Evaluation – Final Model



Data Pre-processing
- Feature Engineering
- Feature Scaling
- Data Splitting into training and testing sets

Training
Use the training set to perform:
- Model Fitting
- Model selection: cross-validation, model tuning

Evaluation
Evaluation of final model on testing set

Prediction
Use the final model to predict the labels for new unlabeled data

- Data needs to be split into training and testing sets:
    - the training set should be used in the training phase to select the most promising model;
    - the testing set should be used in the evaluation phase to evaluate the final model.
- No observations are present in both sets.

Train-Test Data Splitting

Why can’t we evaluate the final model on the training set?

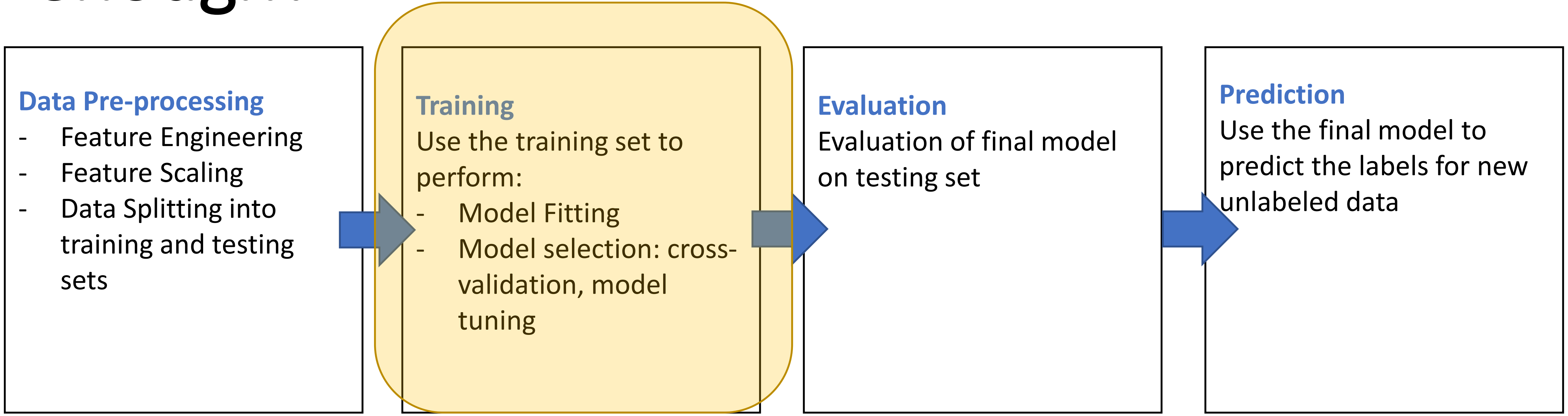
Train-Test Data Splitting

- Evaluating the final model on the training set may result in an optimistic performance: the final model might have picked up on some patterns that only exist in the training data.

- We’re interested in evaluating the performance of the model on **unseen data:** estimating the generalization error. How well will the model generalize to unseen data?

- Our experimental setup needs to match (simulate) how we intend to use the model.

Model Evaluation – Is train-test splitting enough?



Data Pre-processing

- Feature Engineering
- Feature Scaling
- Data Splitting into training and testing sets

Training

Use the training set to perform:

- Model Fitting
- Model selection: cross-validation, model tuning

Evaluation

Evaluation of final model on testing set

Prediction

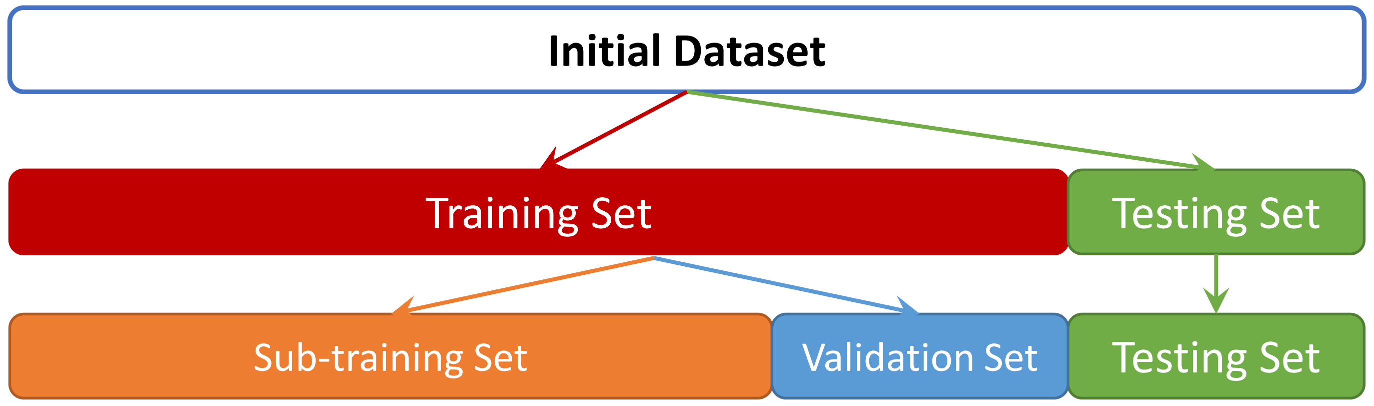
Use the final model to predict the labels for new unlabeled data

- During the training phase, we might need to try different approaches: train different models, fine-tune some of them (Model Selection).

- To select the best model, can we rely on the performance of each model on the entire training set?

Validation Set

- Training set needs to be further split into sub-training and validation sets:



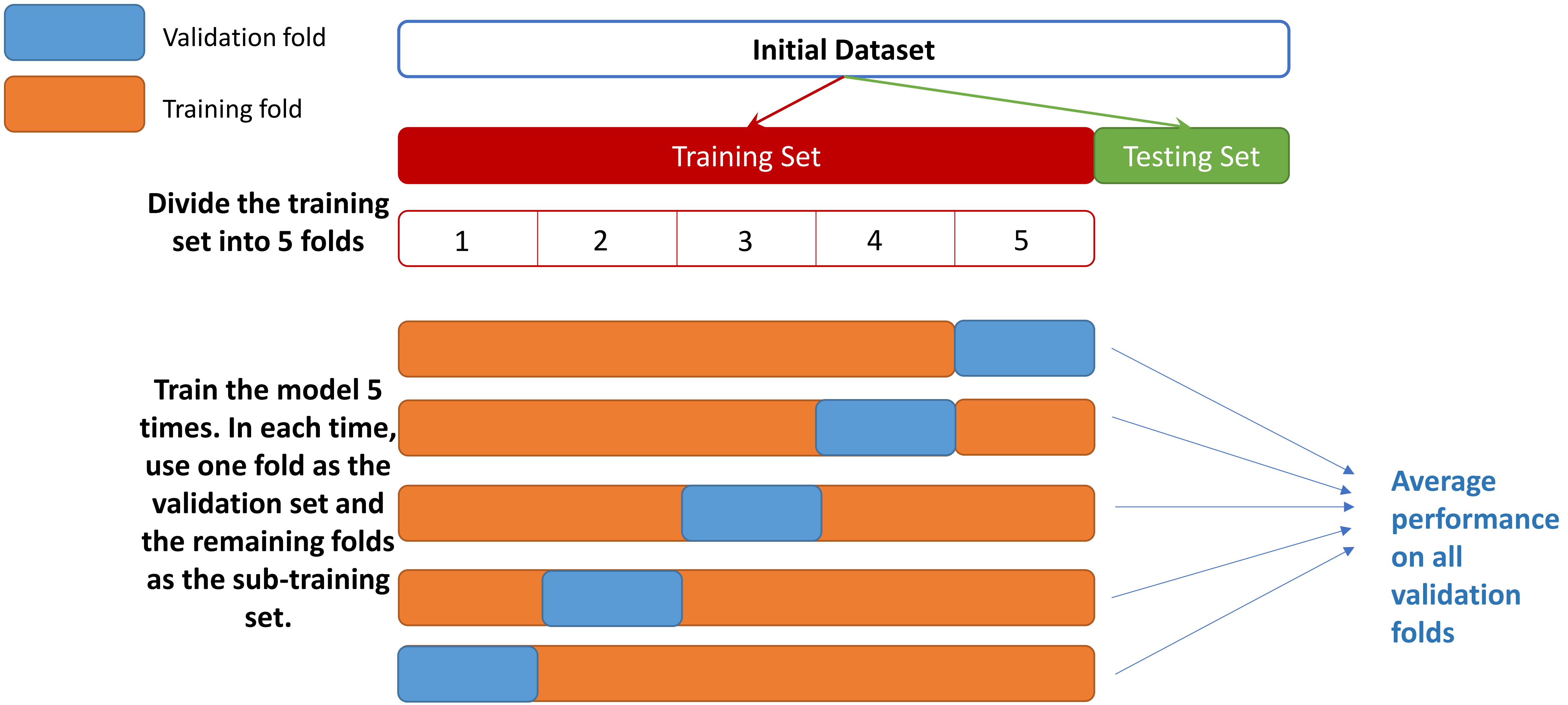
- The different models are trained using the sub-training set and then evaluated on the validation set.
- The final model with the best performance on the validation set is the one that is chosen and finally evaluated on the test set.

Validation Set

- If the size of the data is very large, partitioning the training set into sub-training and validation sets once is enough.
- Otherwise, the performance of any model might be sensitive to how we partition the data into sub-training and validation sets.

- **Solution: K-fold cross-validation**

Validation Set – K-fold cross-validation



Validation fold
Training fold
Initial Dataset
Training Set
Testing Set
Divide the training set into 5 folds
1
2
3
4
5
Train the model 5 times. In each time, use one fold as the validation set and the remaining folds as the sub-training set.
Average performance on all validation folds

Underfitting, Overfitting

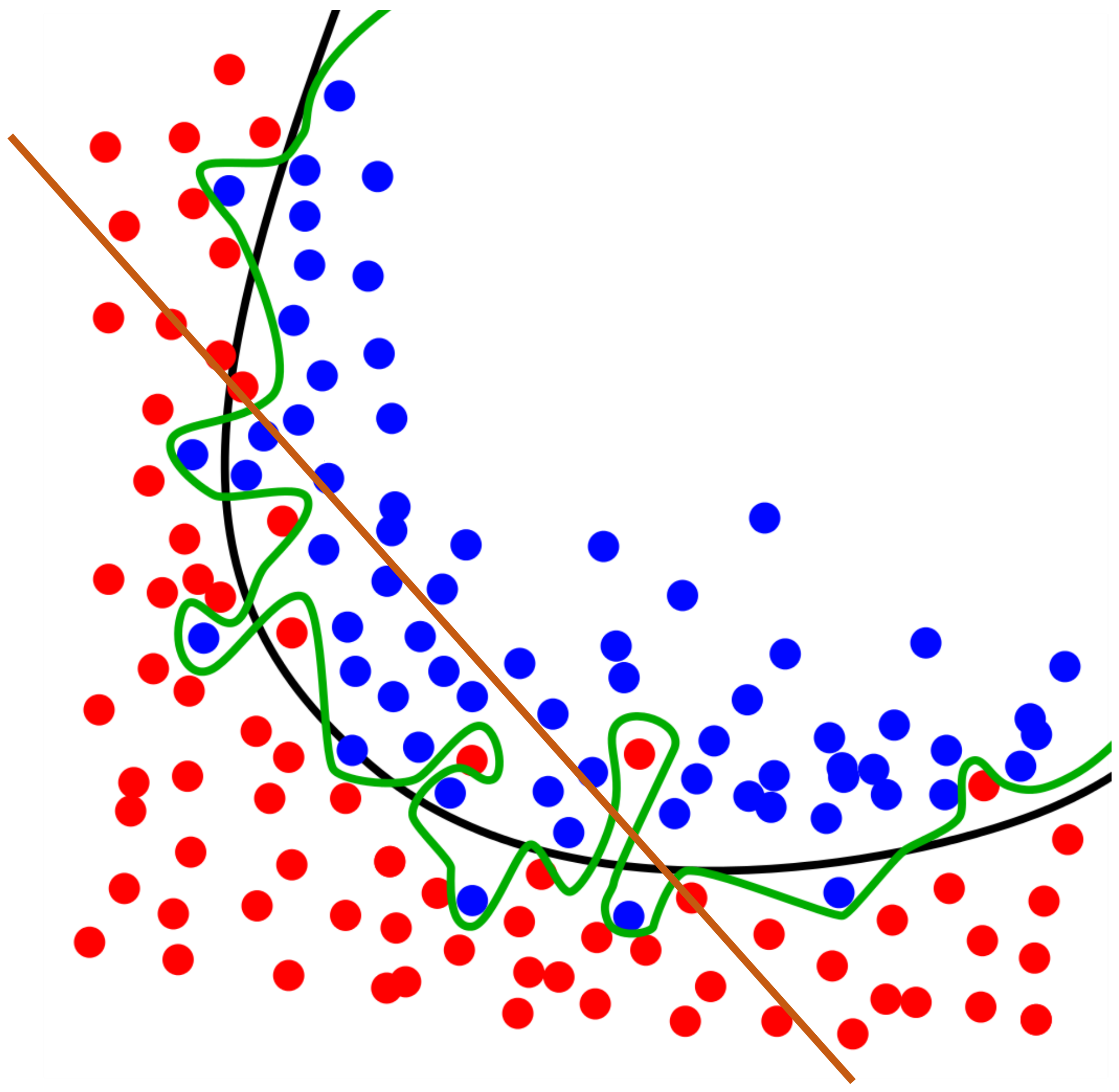
Model Evaluation

- Splitting the data into training and testing sets and performing cross-validation on the training test help us:

  - not choose an overfit or underfit model.

- What is overfitting and underfitting?

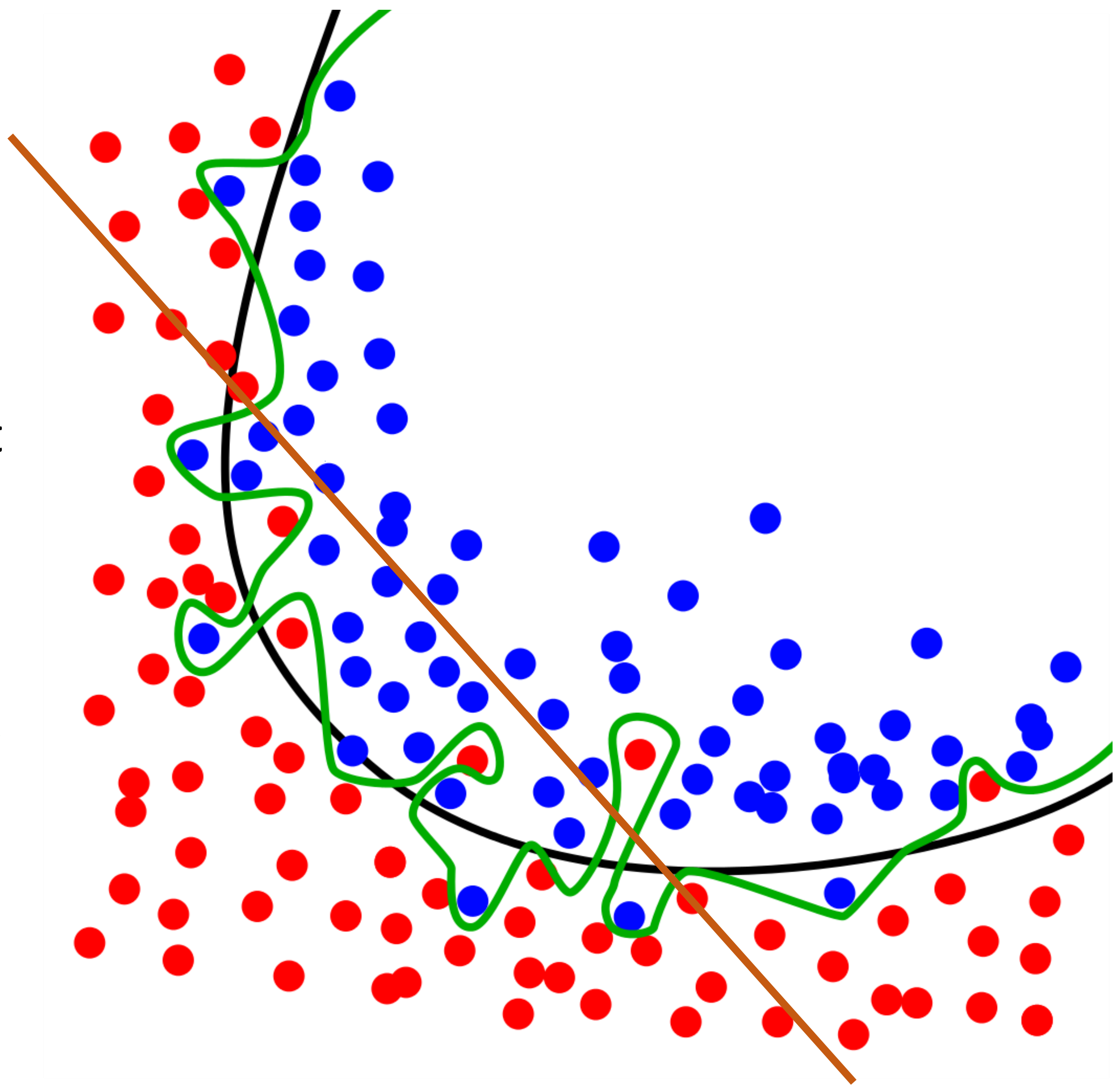
Possible Scenarios



Possible Scenarios

A model, can be bad for two different reasons:

- **Model that poorly fits the data,** is too simple and doesn't match the data well: it is said to have high bias (underfit model).
- **Model that overly fits the data,** is too complex and too specific for the data: is said to have high variance (overfit model).
- **The Goldilocks model!**



Possible Scenarios

A model's error can be expressed as the sum of three different errors:

- Bias
- Variance
- Irreducible error: Irreducible error refers to the noise that exists in the data itself. It is an aspect of the data, and we cannot beat it.

Bias

Bias:

- Bias refers to the error that is introduced due to wrong assumptions such as approximating a complicated pattern in data with a simpler model.
- A **high-bias or underfit** model is a model that fails to capture the complex patterns in the training data.
- An example is when we use a linear classification model on data that is not linearly separable.
- In this case, both the training error and the validation/test error will be high.

Variance

Variance:

- Variance captures how much a model is over-specialized to a particular training dataset.
- Variance measures the consistency or variability of the model predictions if we retrain the model on a different training set, i.e., how much the model's predictions change if it is trained on a different training set.
- A **high-variance or overfit** model is a model that performs well on the training set by memorizing its specific patterns but fails to generalize to unseen data (validation/testing errors).
- In this case, both the training error and the validation/test error will be high.

How do we know if a given trained model is underfit or overfit?

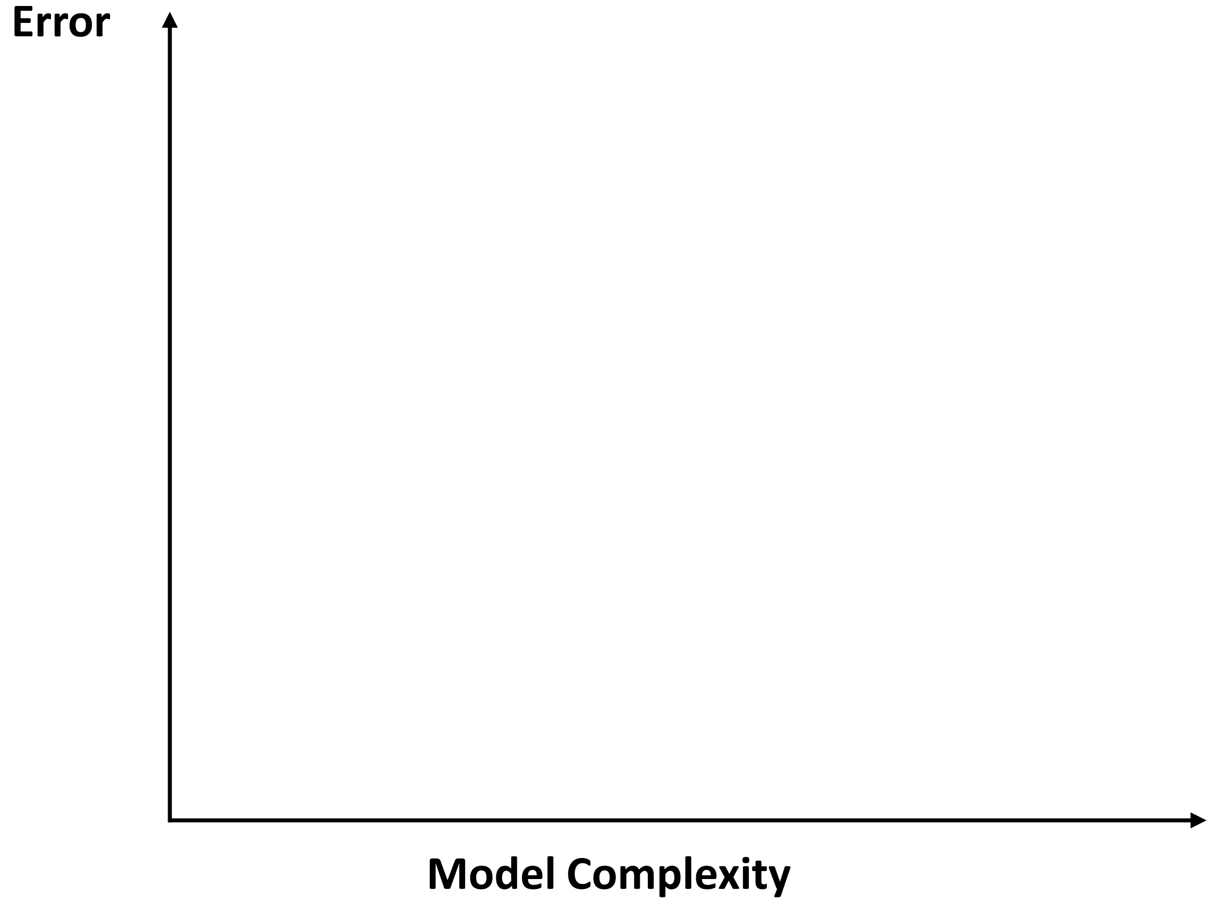
- When a model does not perform well on the training set, we know that the model underfits the training set. Both the training error and the validation error of an underfit model will be high.

- When a model performs very well on the training set, but fails to show similar performance on the validation set, we conclude that the model overfits the training set.

Bias-Variance Tradeoff

Validation Error

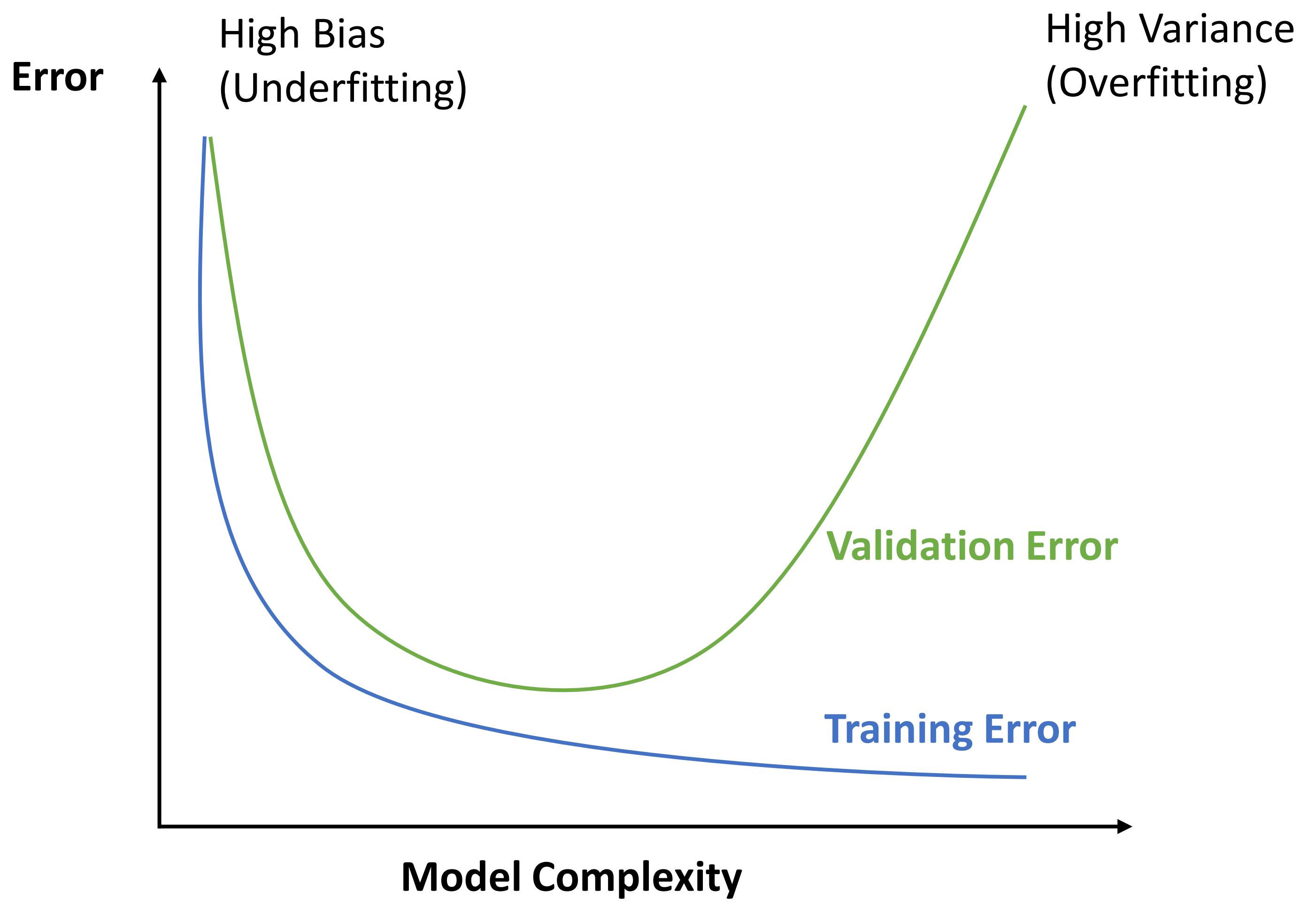
Training Error



Bias-Variance Tradeoff

Validation Error

Training Error

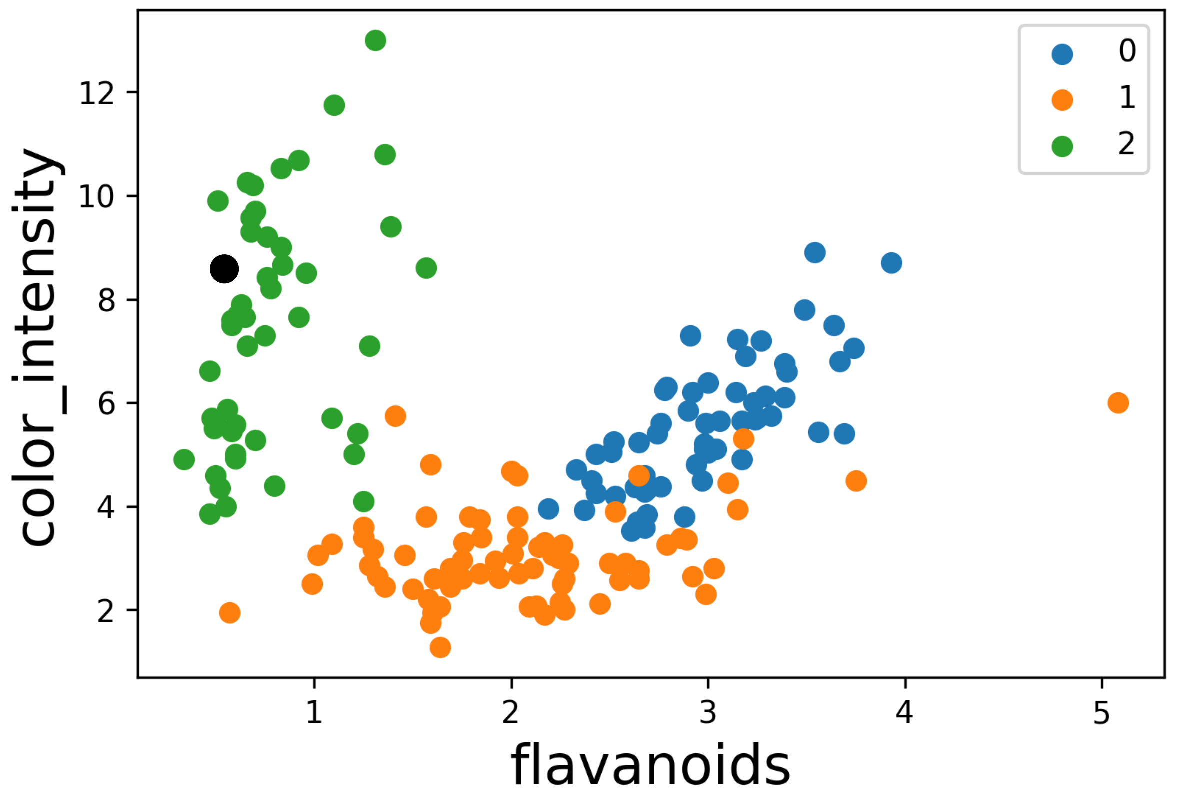


What is KNN?
Who is KNN?
Where is KNN?

K-Nearest Neighbors (KNN)

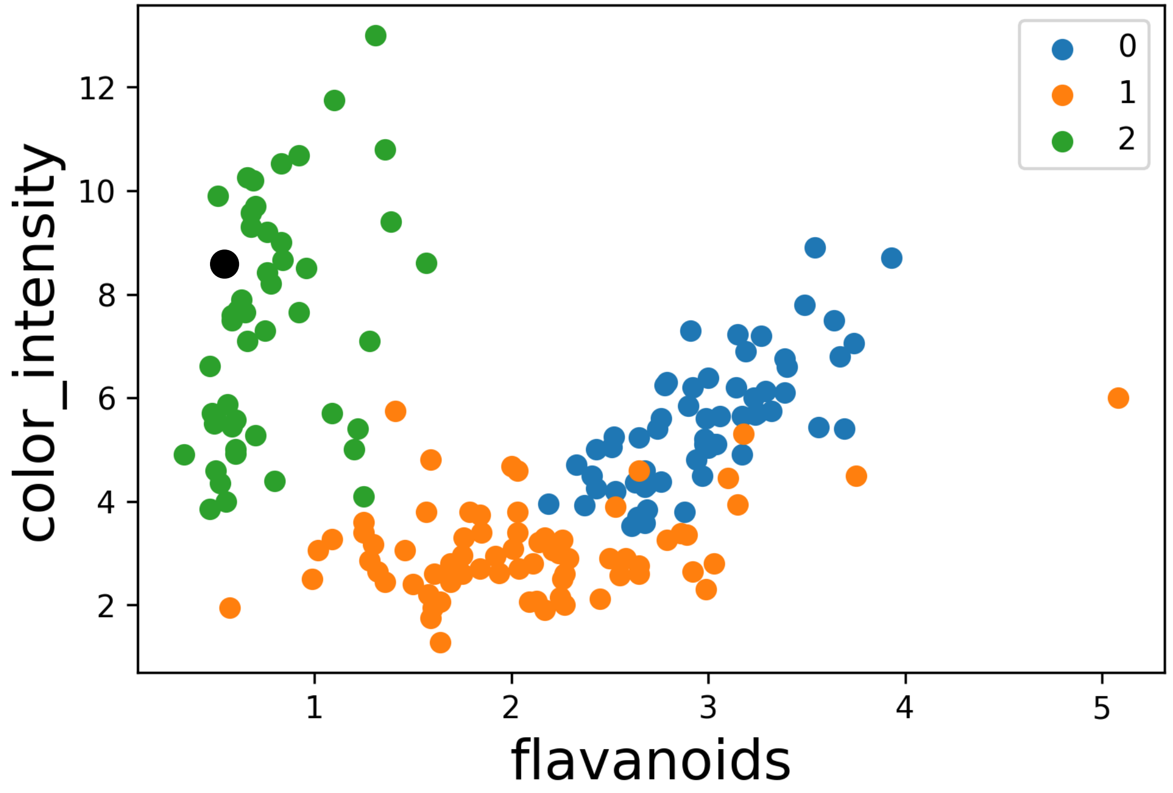
1. Overview
2. KNN Training and Prediction Phase
3. What does affect KNN’s predictive power?
   Choice of k, Feature Scaling
4. Advantages and Disadvantages

KNN



What label would you choose for the new data?

KNN



What label would you choose for the new data?

Key KNN – Assumptions:
Data samples that are close together (in some appropriate distance metric) are similar and will therefore have the same label.

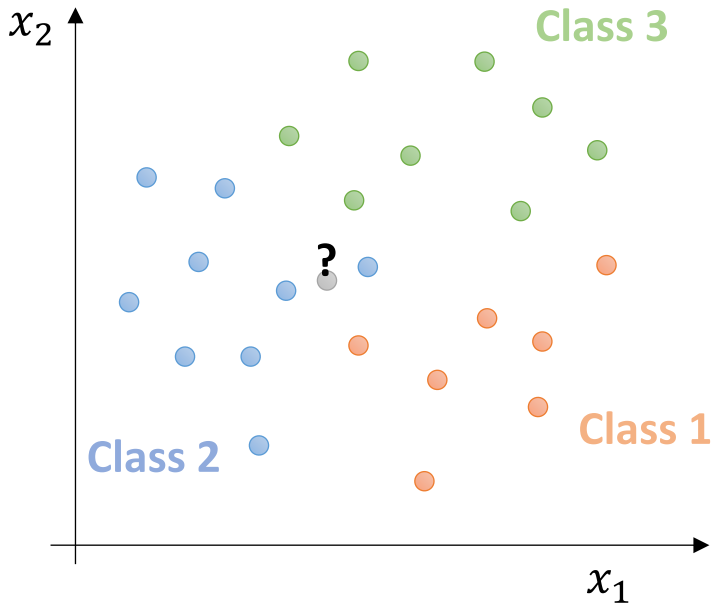
KNN

- Fundamental assumptions of KNN: Points similar to each other in feature space are likely belong to the same class
- It uses a distance metric to reflect similarity between the data points
- Distances are calculated between points to find the nearest neighbors

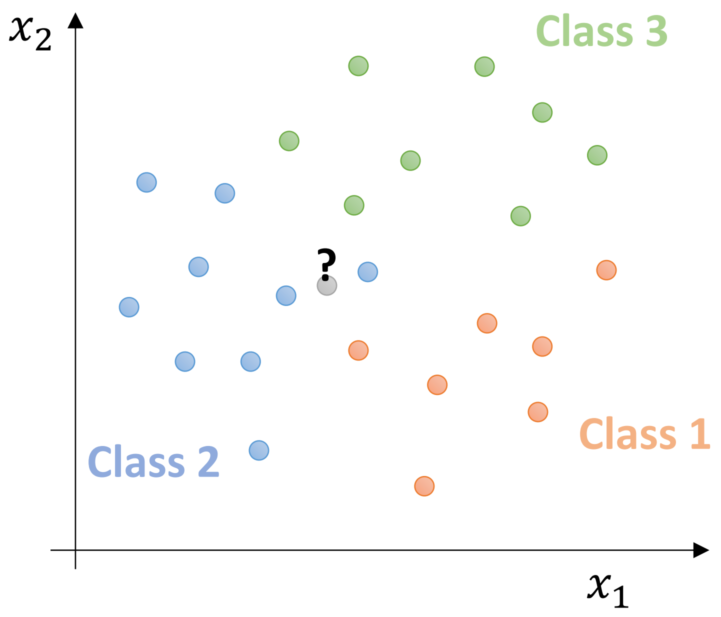
KNN

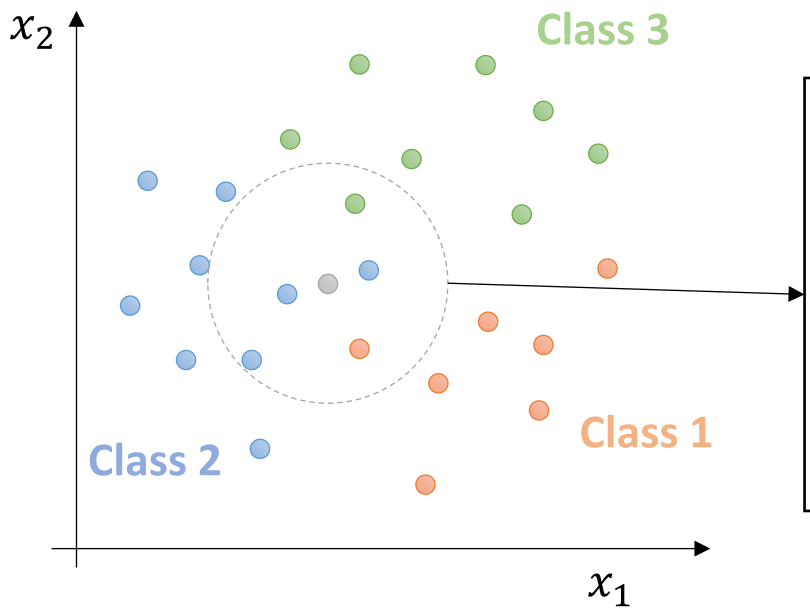
- KNN is a simple machine learning algorithm for classification and regression.
- The algorithm was first proposed in 1951 by Fix and Hodger. Then in 1967, Thomas cover provided further analytical results for the algorithm.
- KNN has been suggested for a variety of applications: recommendation systems, missing data imputation, other applications.

KNN Overview



KNN Overview





The 5 data samples that are closest to ● :
3 belong to class 2 (blue)
1 belongs to class 3 (green)
1 belongs to class 1 (orange)
Then the unlabeled data is predicted to belong to class 2 (blue).

Training Phase

Training Phase of KNN

- KNN assumes no specific functional form for the relationship between a feature vector and its label.
- There is no training phase: KNN is an example of **lazy learner**.
- The algorithm just needs to store the training data. This **memory-based learning** is also known as **instance-based learning** which consists of comparing new data instances with instances seen in training.

What are the parameters of KNN?

- The training data set itself is treated as the building blocks or the parameters for the KNN algorithm.

- KNN is an example of non-parametric model:
  - a parametric model (like linear model) summarizes the training data with a finite set of parameters (irrespective of the number of samples in the training data).
  - KNN is non-parametric: it cannot be characterized by a finite set of parameters.

Prediction phase of KNN

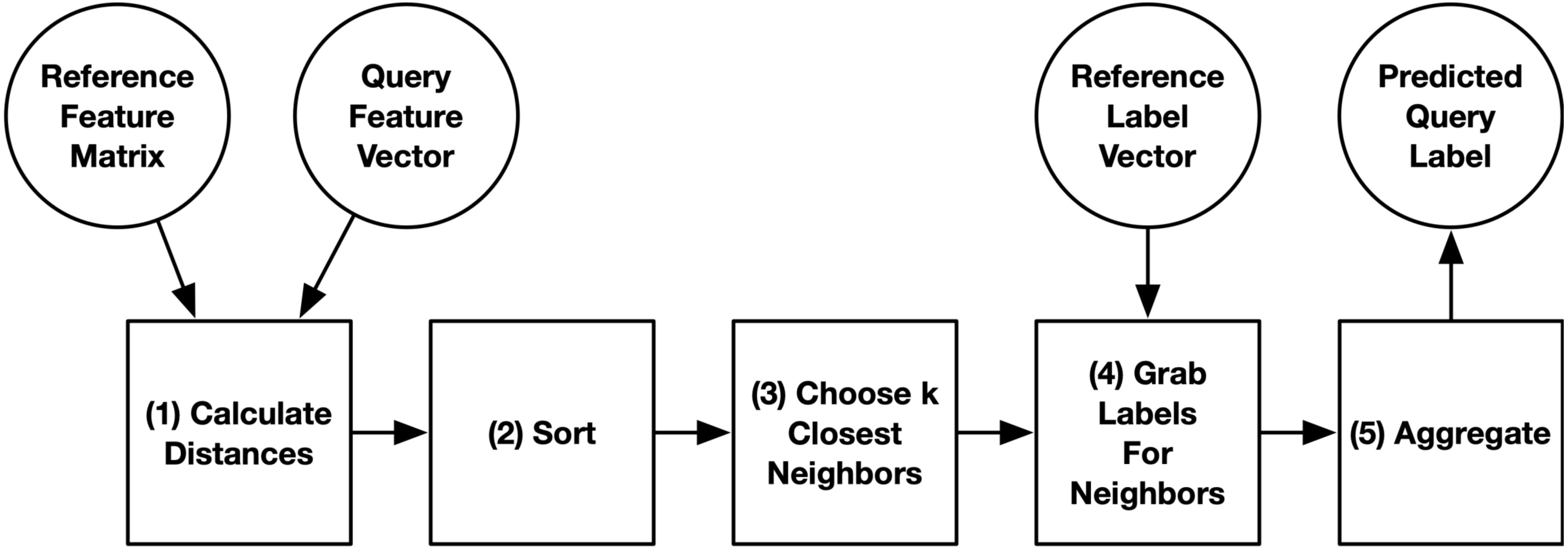
Terminology

- **Reference points**: training set data
- **Query point**: new data point we want to classify
- **Neighbors**: points near each other in some space

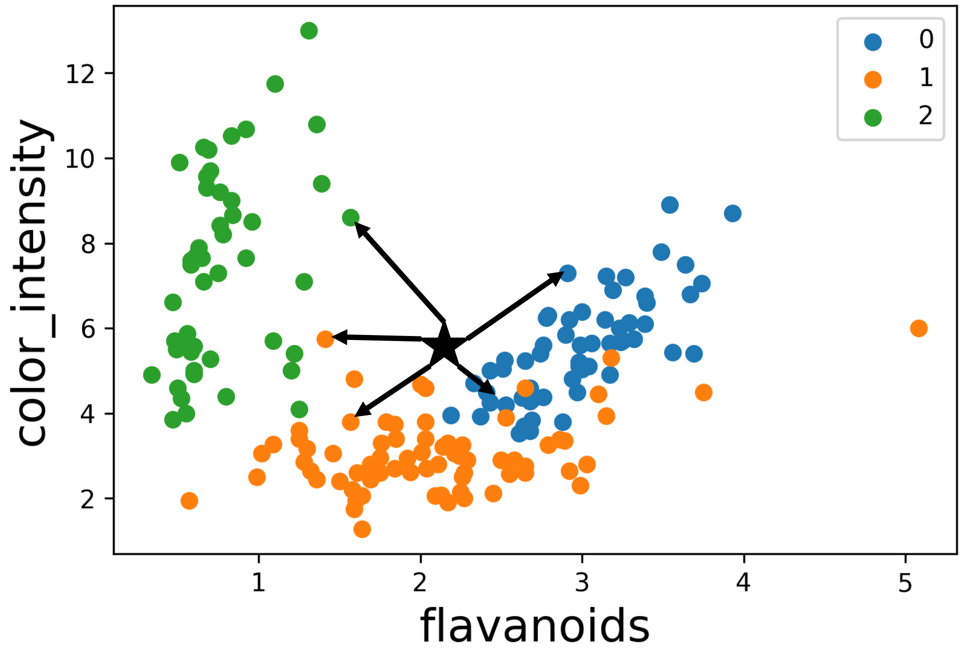
KNN Algorithm Flowchart

Training data (data with answers)

New data point without label

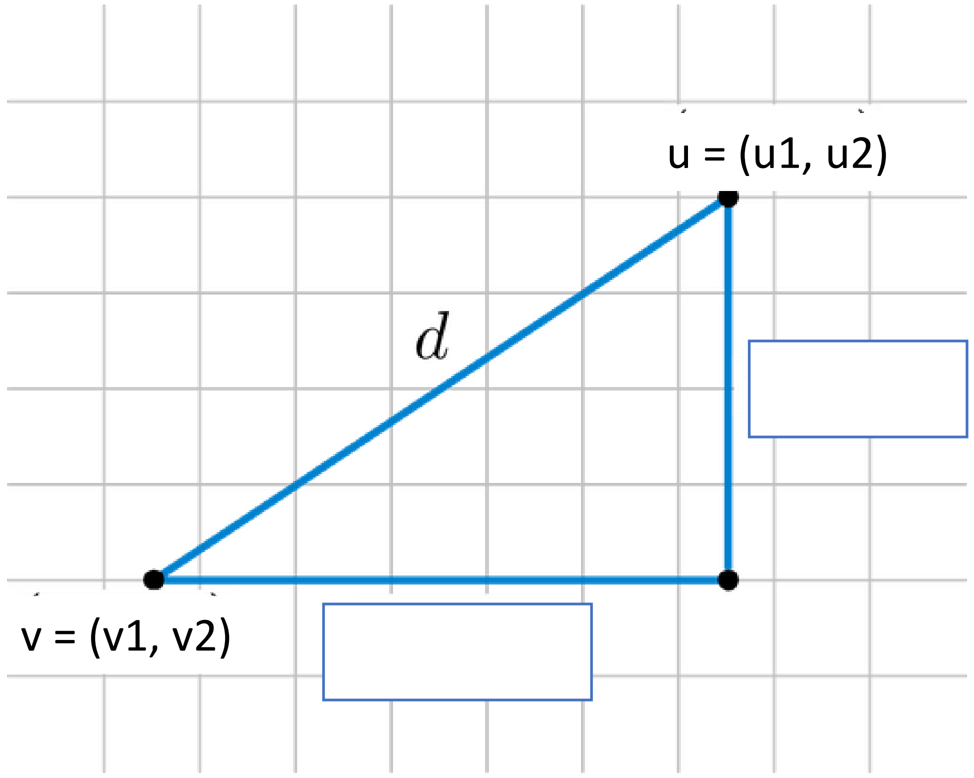


Step 1: Calculate Distances



- We calculate the distance between the query point (star) and every reference point
- Note that we do this in feature space!
  - Meaning, we calculate the distances between each pair of points using their feature vectors

Step 1: Calculate Distances



The Euclidean distance of points represented as the vectors u and v can be calculated by:

$$d = \|\boldsymbol{u} - \boldsymbol{v}\|$$

http://rosalind.info/glossary/euclidean-distance/

Step 1: Calculate Distances

```
[[ 1.5186  1.0348  0.2517  1.013   1.8479]
 [ 0.2463  0.7336 -0.2933  0.9652  1.1134]
 [ 0.1969  1.2155  0.269   1.3951  0.7886]
 [ 1.6915  1.4665  1.1861  2.3346  1.1841]
 [ 0.2957  0.6634 -0.3193 -0.0379  0.4496]
 [ 1.4816  1.3661  0.7319  2.239   0.3366]
 [ 1.7163  0.4927  0.083   1.7295  1.3677]
 [ 1.3086  0.4826 -0.0035  1.7454  1.3677]
 [ 2.2598  0.9545  0.0614  0.9493  0.3366]
 [ 1.0616  1.1252  0.9352  0.9493  1.3253]]
```

```
[ 0.2957  0.6634 -0.3193 -0.0379  0.4496]
```

Step 1: Calculate Distances

Index of reference data point in the training set	Distance to the query point
0	2.2404
1	1.2062
2	1.6820
3	3.3208
4	0.0001

We create a table of the index of every reference point (row in the feature matrix) and its distance to the query point

Step 2: Sort by Distance

Index of reference data point in the training set	**Distance to the query point**
0	2.2404
1	1.2062
2	1.6820
3	3.3208
4	0.0001

Sort the reference points by their distance from the query point

Step 2: Sort by Distance

Index of reference data point in the training set	**Distance to the query point**
4	0.0001
1	1.2062
2	1.6820
0	2.2404
3	3.3208

Sort the reference points by their distance from the query point

Step 3: Find the k-closest neighbors

Index of reference data point in the training set	Distance to the query point
4	0.0001
1	1.2062
2	1.6820
0	2.2404
3	3.3208

The k closest neighbors are kept

Step 3: Find the k-closest neighbors

Index of reference data point in the training set	**Distance to the query point**
4	0.0001
1	1.2062
2	1.6820

The k closest neighbors are kept

Step 3: Find the k-closest neighbors

Index of reference data point in the training set	**Distance**
4	
1	
2	

We no longer use the distances

We can just drop them

Step 4: Grab Labels for Neighbors

Index of reference data point in the training set	**Labels**
4	
1	
2	

```
[[1]
 [2]
 [0]
 [1]
 [0]
 [0]
 [0]
 [0]
 [1]
 [1]
 [2]]
```

Step 4: Grab Labels for Neighbors

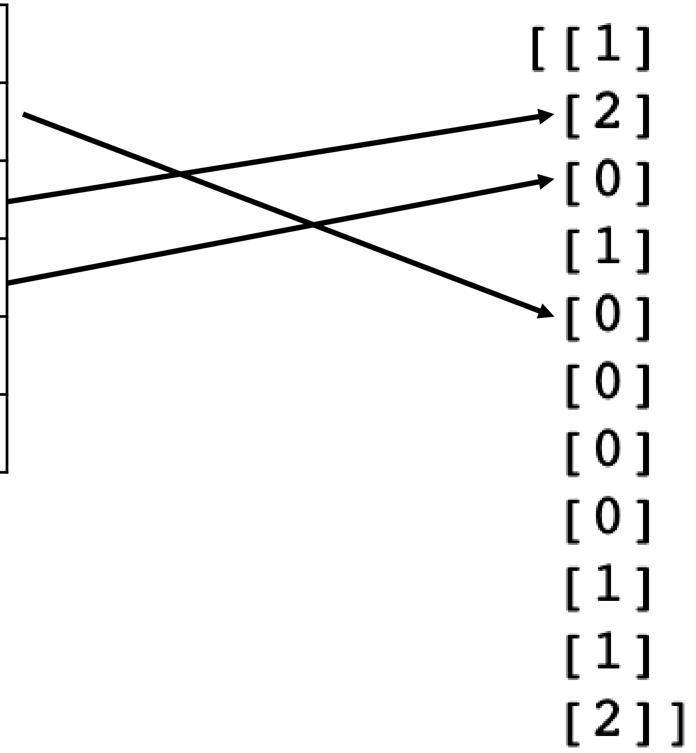
Index	Labels
4	
1	
2	

Label vector for the reference points. Stores the class for each point as an integer. Indices match rows of feature matrix.

```
[[1]
 [2]
 [0]
 [1]
 [0]
 [0]
 [0]
 [0]
 [1]
 [1]
 [2]]
```

Step 4: Grab Labels for Neighbors

Index	Labels
4	
1	
2	

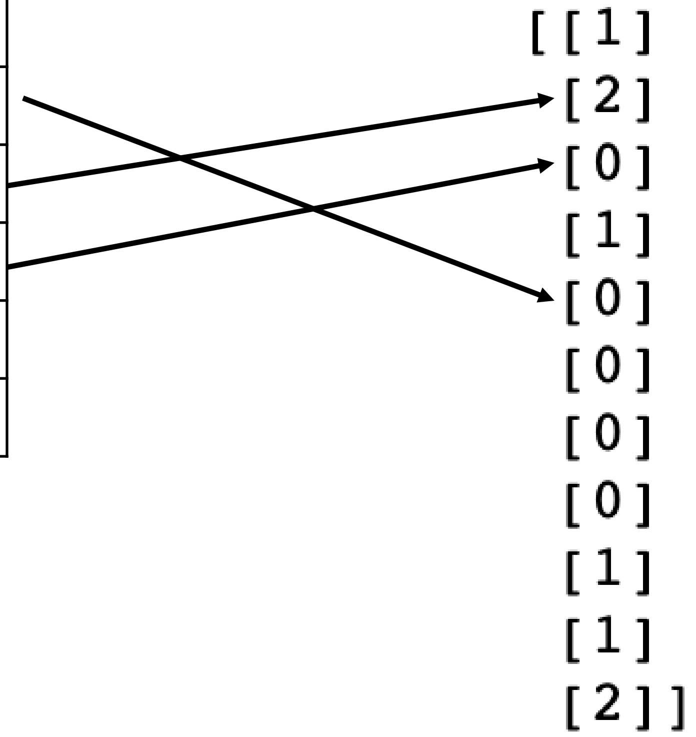


```
[[1]
 [2]
 [0]
 [1]
 [0]
 [0]
 [0]
 [0]
 [1]
 [1]
 [2]]
```

Use the indices to get labels for
reference points from label vector

Step 4: Grab Labels for Neighbors

Index	Labels
4	0
1	2
2	0



```
[[1]
 [2]
 [0]
 [1]
 [0]
 [0]
 [0]
 [0]
 [1]
 [1]
 [2]]
```

Use the indices to get labels for reference points from label vector

Step 5: Aggregate the Output Values

Index	**Labels**
4	0
1	2
2	0
Predicted Value	

- Classification:
  - Mode – most common value (majority vote)
- Regression:
  - Mean
  - Median

Step 5: Aggregate the Output Values

Index	Labels
4	0
1	2
2	0
Predicted Value	

- Classification:
  - Mode – most common value
- Regression:
  - Mean
  - Median

Step 5: Aggregate the Output Values

Index	**Labels**
4	0
1	2
2	0
Predicted Value	0

- Classification:
  - Mode – most common value
- Regression:
  - Mean
  - Median

Run time and Space Complexity

Run Time Complexity

Assume there are **n** reference points, 1 query point, and **m** features that you will use the **k** closest neighbors

- Step 1: Compute distances:
- Step 2: Sorting the neighbor list (using a heap):
- Step 3: Get known labels:
- Step 4: Aggregation:

Run Time Complexity

Assume there are **n** reference points, 1 query point, and **m** features that you will use the **k** closest neighbors:

- Step 1: Compute distances: O(nm)
- Step 2: Sorting the neighbor list (using a heap): O(n log k) (if using max-heap)
- Step 3: Get known labels: O(k)
- Step 4: Aggregation: O(k)

Total run time: O(nm + n log k + k) = O(nm)

Space Complexity

Assume there are n reference points, 1 query point, and m features that you will use the k closest neighbors

The "training" step of KNN merely involves storing the n x m feature matrix and n-length label for the reference points.

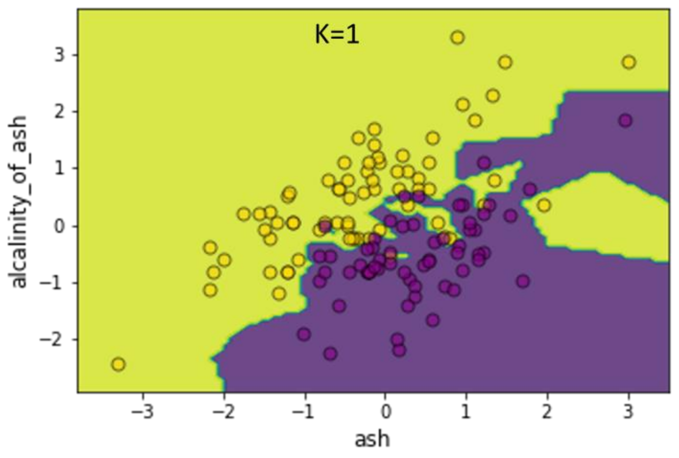
Space complexity: O(nm + n) = O(nm)

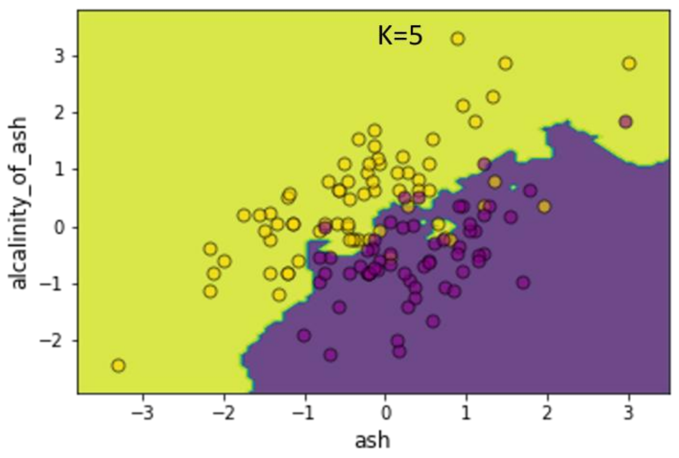
How do we choose k?

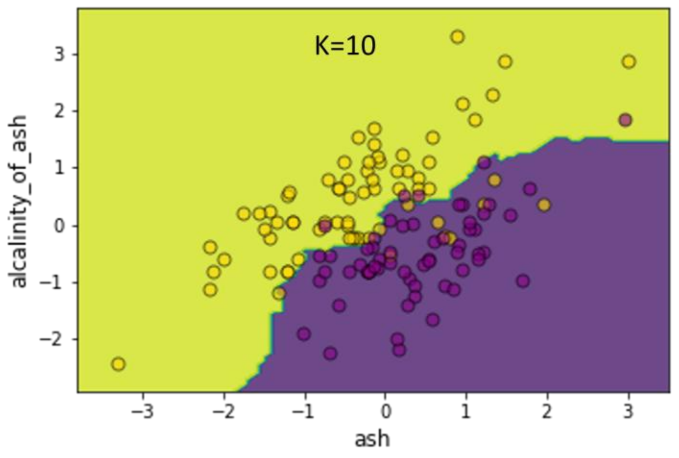
Choice of K

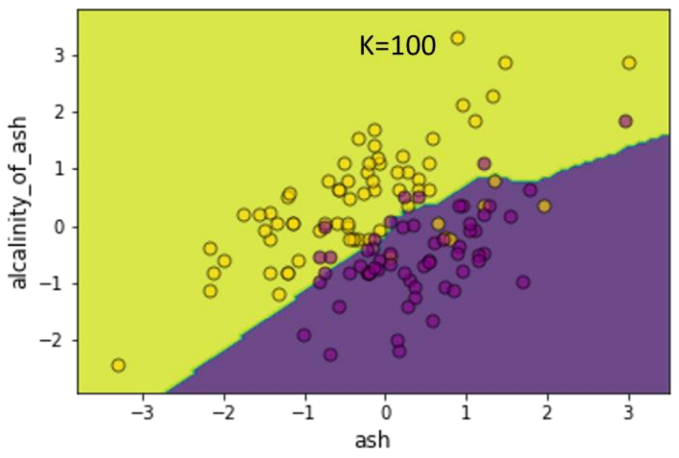
- Choosing the right k is about finding a good balance between overfitting and underfitting.

- Note: k is a hyperparameter
  - a hyperparameter is a setting in your model that you need to adjust (using cross-validation): model tuning, hyperparameter tuning
  - parameters are what define a model (building blocks). The goal of model training is to find the parameters while fixing its hyperparameters.









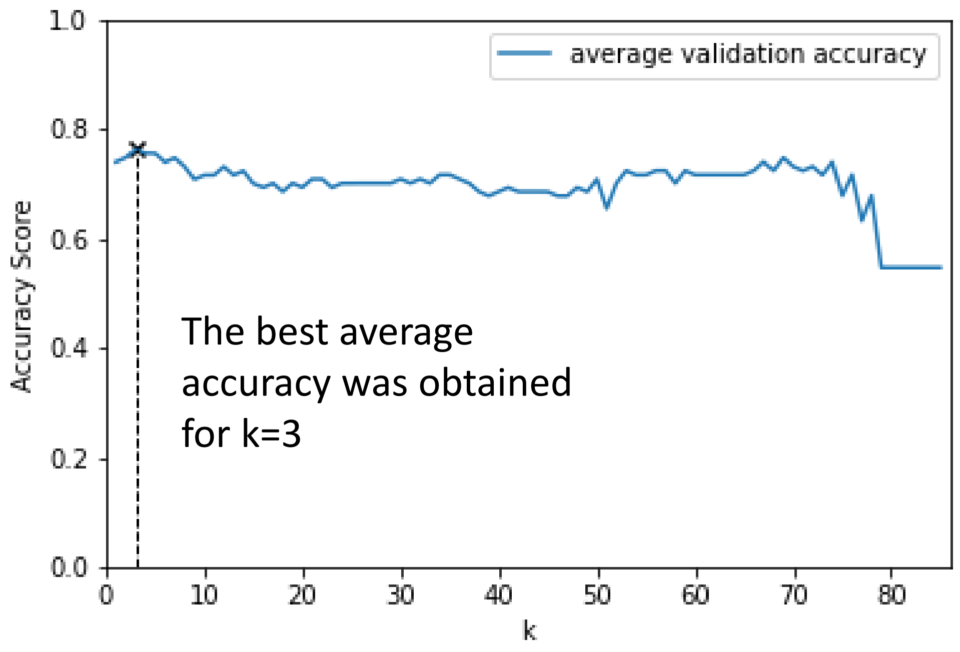
Choice of K

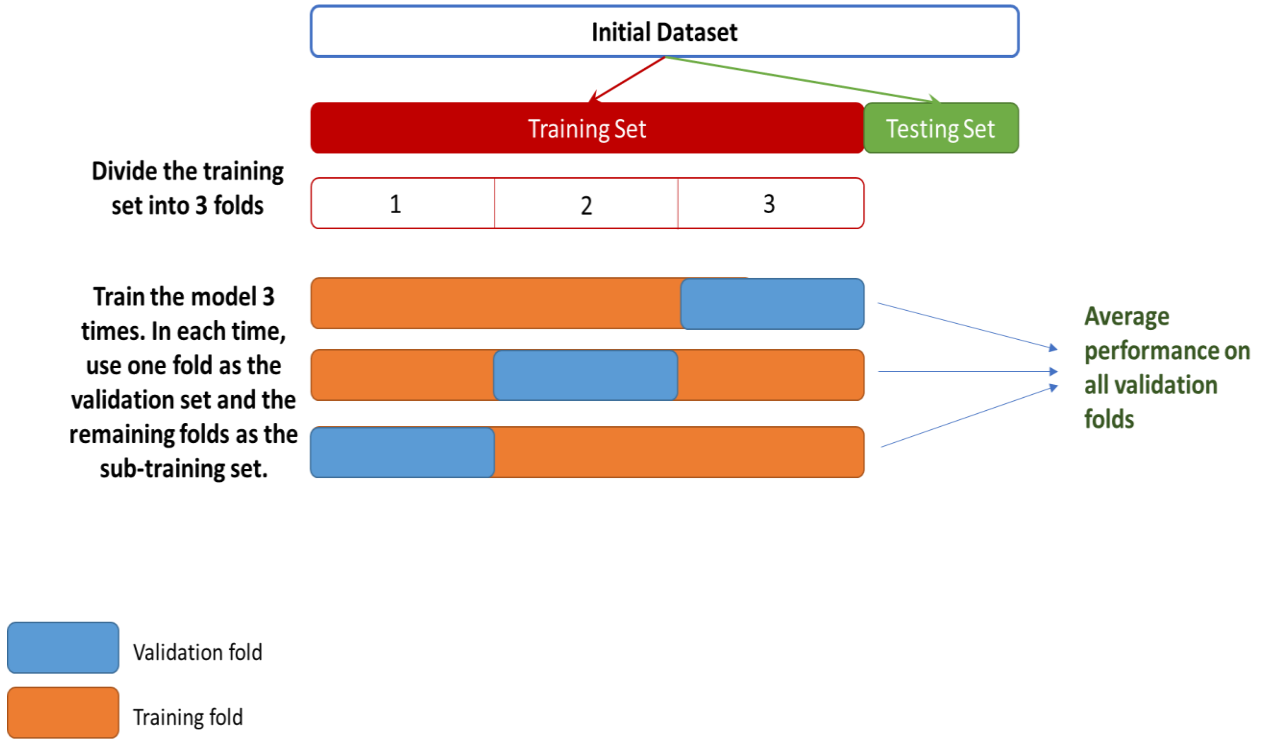
- For very small values of k, the model can be sensitive to data noise, and thus can have high variance and can be prone to overfitting.

- For very large values of k, the model considers far points as neighbors which will lead to a model with high errors, and thus high bias or underfitting.

Choice of K : use k-fold cross validation

- For each K, we compute the model's response using cross validation.

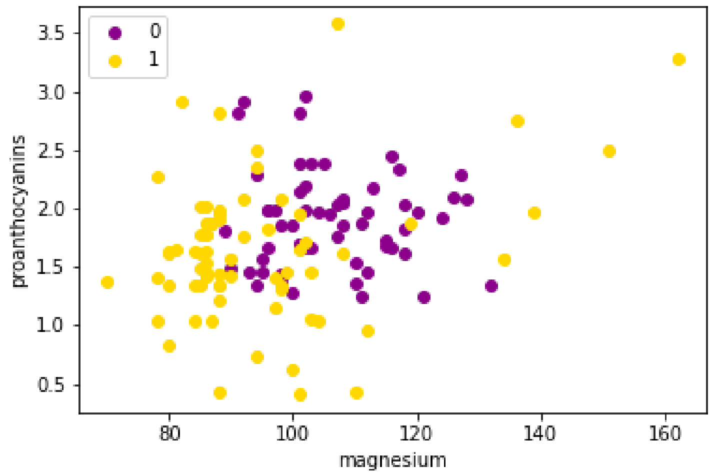


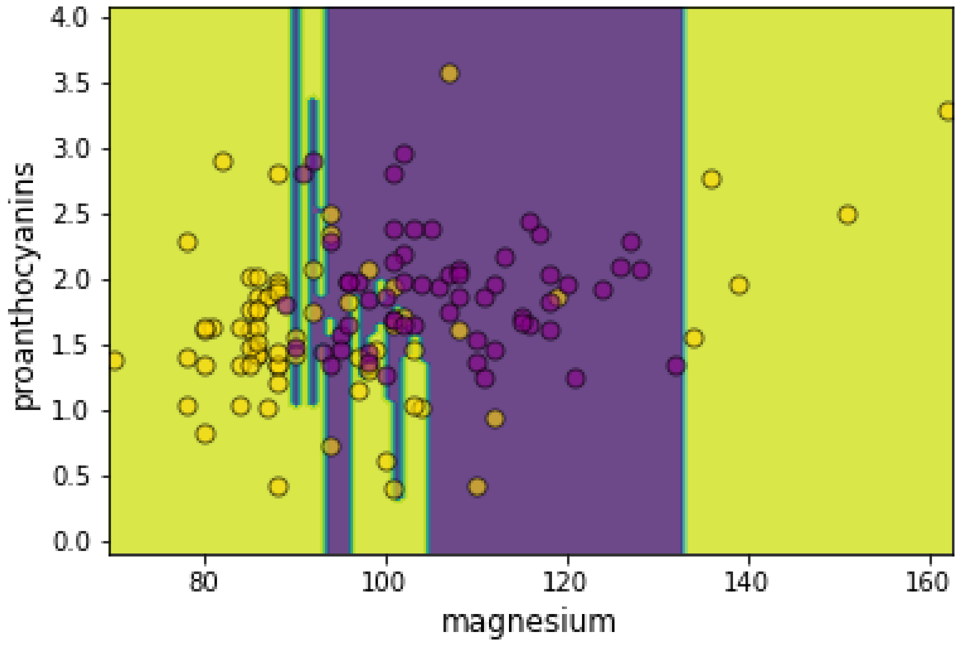


Initial Dataset
Training Set
Testing Set
Divide the training set into 3 folds
1
2
3
Train the model 3 times. In each time, use one fold as the validation set and the remaining folds as the sub-training set.
Average performance on all validation folds
Validation fold
Training fold

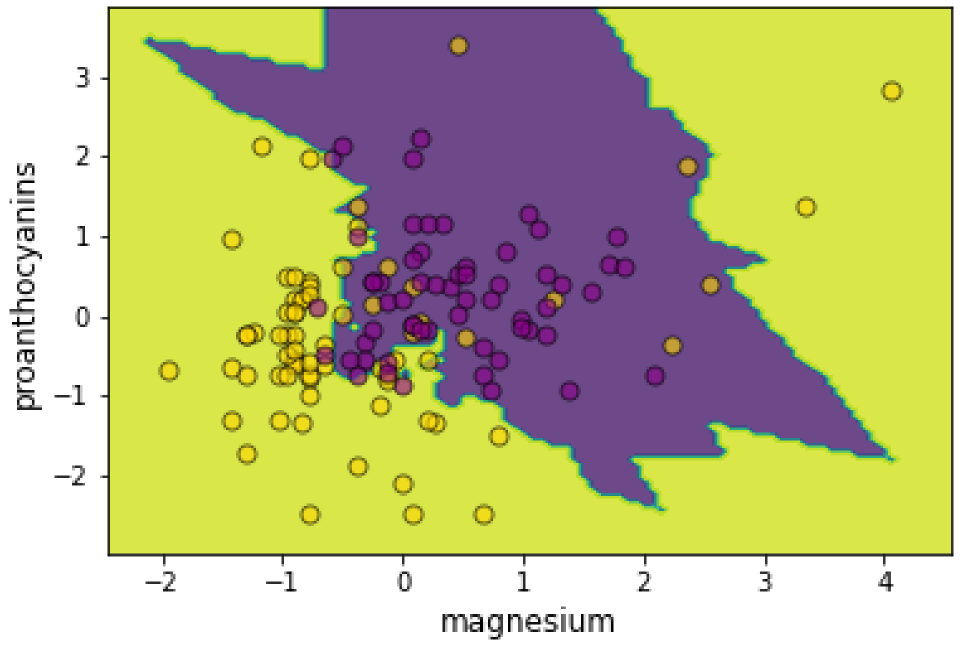
Importance of Scaling

Feature Scaling





No Scaling



After Scaling

Feature Scaling

- If one feature has much bigger range of values than others, this feature will end up contributing to the total distance more than the other features.
- Euclidean distance squares the difference between the values of features, so that variations in the values of the features with greater range of values will be magnified more than those of other features.
- This means that small variations in the feature with greater range of values will have more effect on the distance than the large variations in other features. The model will end up choosing the wrong neighbors.

KNN Advantages & Limitations

KNN Advantages

- **Simple model**: The model has no training step and the prediction phase of the algorithm is easy to implement. The model requires few hyperparameters: choice of k and distance metric.
- **Non-linear model**: The model does not assume any specific functional form, which results in more flexible decision boundaries (classification) or model’s response (regression) than those of linear models.
- **Adapts easily to new training samples**: Since the training sample is the model itself, the model adjusts easily to new training samples. The new samples needs to be added to the old training samples and both stored in memory.
- **Extends easily to multi-class problem**: The prediction algorithm of KNN works no matter how any classes to classify, it does not only work for binary classification. When predicting the label of a query point, we choose the majority vote label from its nearest neighbors.

KNN Disadvantages

- **Memory and computation costly**: Since KNN is a lazy learning, the training data needs to be entirely stored, which might not be efficient from memory and storage perspective. Moreover, the prediction run time is slower than many other models.
- **Curse of dimensionality:** In high-dimensional feature space, the data points become more sparse or scattered (especially if we don’t have lots of training samples), so that the nearest points to a given query point might be too far.
- **Sensitive to feature scaling**: The performance of KNN is also affected if features have different scales, this is why it is important to make sure that features are scaled before applying KNN algorithm.
- **Interpretability**: Linear models are considered interpretable model because the weights can give us insight into the contribution of each feature to the model. However, KNN is not able to do this. When computing the distance between two data points, all features are considered