# PHIL 7010: Formal Methods for AI, Data, and Algorithms

Week 2
K - Nearest Neighbors

Boris Babic,
HKU 100 Associate Professor of Data Science, Law and Philosophy

## Today

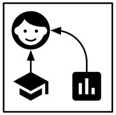
**Learning goals**

- Basics KNN Algorithm
- Bias-Variance Tradeoff
- Implement KNN using R
- Acknowledging the Challenges and Limitations

## Machine Learning Recap

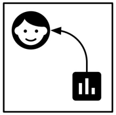
- For many problems, it's difficult to program the correct behavior by hand.
- Machine learning (ML) approach: program an algorithm to automatically learn from data, or from experience
- Three categories of ML: Supervised, Unsupervised, Reinforcement
- So far, most of the ML algorithms we've learned are supervised (Logistic Classification, Decision Tree and Random Forests, SVMs, ...)

**Supervised Learning**



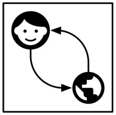
Machine is given data and examples of what to predict.

**Unsupervised Learning**



Machine is given data, but not what to predict.

**Reinforcement Learning**



Machine gets data by interacting with an environment and tries to minimize a cost.

## Recap: Classification v.s. Reinforcement Learning

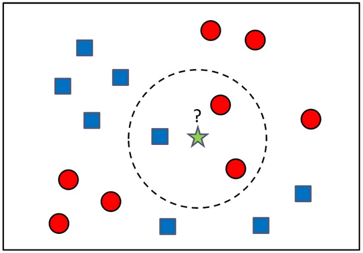
- **Classificatio**n: Classification is used for tasks where the goal is to assign a category or label to a given input. The input data is typically divided into predefined classes, and the model's objective is to correctly assign the input to one of these classes.
  - Classification is a static decision-making task.
  - Classification is often a supervised learning task, where we trained a fixed model on the labeled data.
- **Reinforcement Learning (RL)**: RL is typically used for tasks where an agent learns to interact with an environment and make a sequence of decisions to maximize cumulative rewards. It's suitable for problems where the optimal action is not known in advance, and the agent needs to explore and learn through trial and error.
  - RL receive feedback in the form of rewards or penalties based on their actions. RL models use this feedback to update their decision-making policy and improve over time.
  - Reinforcement learning often deals with Markov decision processes (MDPs), where the future state depends only on the current state and action. This introduces complexity as RL models need to consider the impact of their actions on future states.

## K-Nearest Neighbors

- Recall: A supervised machine learning algorithm is one that relies on labeled input data to learn a function that produces an appropriate output when given new unlabeled data.
- The K-nearest neighbors algorithm (KNN) is a very simple yet powerful **supervised** and **non-linear** algorithm
- It assigns a label to a new sample based on the labels of its k closest neighbors in the training set.
- KNN can be used for both classification and regression problems.

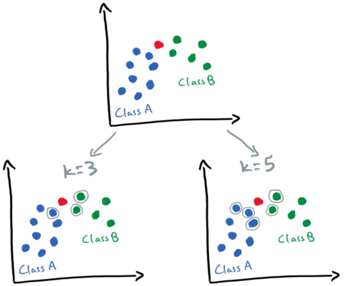
## K-Nearest Neighbors

- The idea behind the KNN classification algorithm is very simple: given a new sample, assign it to the class that is most common among its k nearest neighbors.
- The assumption behind KNN's classification rule is that similar things are near to each other.



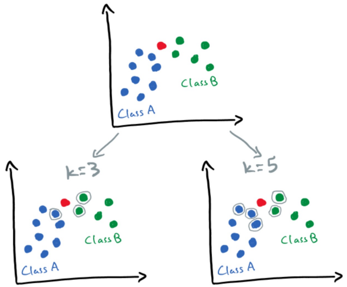
## K-Nearest Neighbors

- KNN is a supervised learning technique, so we should have a labeled dataset. Let's say we have two classes: Class A (blue points) and Class B (green points).
- A new data point (red) is given to us and we want to predict whether the new point belongs to Class A or Class B.
- Let's first try K = 3. In this case, we have to find the three closest data points (aka three nearest neighbors) to the new (red) data point.
- As can be seen from the left side, two of three closest neighbors belong to Class B (green) and one belongs to Class A (blue). So, we should assign the new point to Class B.



## K-Nearest Neighbors

- Now let's set K = 5. In this case, three out of the closest five points belong to Class A, so the new point should be classified as Class A.



- The above is just a simple illustration, how is the KNN algorithm being computed on the computer?

## K-Nearest Neighbors

- The training phase of the algorithm consists of only storing the training samples and their labels, i.e., no model is built from the data.
- In the prediction phase, KNN compute the distances between the test (query) point x and all the stored samples.
- It finds the k samples with the smallest computed distances, and assigns the majority label of these samples to x.
  - For classification: assign the majority class label (majority voting)
  - For regression: assign the average response
- Thus, the algorithm requires:
  - **Distance function**: To compute the similarities between examples
  - **Parameter K**: number of nearest neighbors to look for

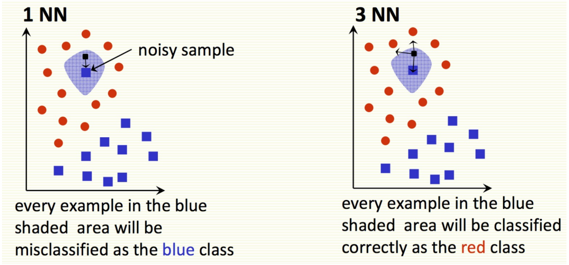
## Distance function

- Several ways to compute distances
- The choice depends on the type of the features in the data
- Real-valued features: Euclidean distance is commonly used

$$d(\mathbf{x}_i, \mathbf{x}_j) = \sqrt{\sum_{m=1}^{D} (x_{im} - x_{jm})^2} = \sqrt{||\mathbf{x}_i||^2 + ||\mathbf{x}_j||^2 - 2\mathbf{x}_i^T \mathbf{x}_j}$$

## Choosing K

- Nearest neighbors sensitive to noise or mis-labeled data (“class noise”).
- Solution? Smooth by having larger k nearest neighbors vote

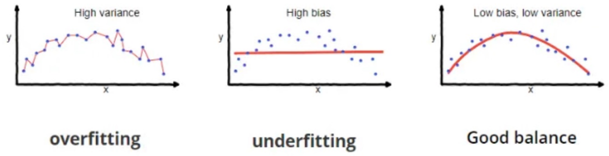

1 NN
noisy sample
every example in the blue shaded area will be misclassified as the blue class
3 NN
every example in the blue shaded area will be classified correctly as the red class

## Bias-Variance Tradeoff

Tradeoffs in choosing k?

- Small k
  - Good at capturing fine-grained patterns
  - May be sensitive to random idiosyncrasies, in the training data, we call this **overfitting**.
- Large k
  - Makes stable predictions by averaging over lots of examples
  - May fail to capture important regularities, we call this **underfitting**.

Increasing k reduces variance but increases bias, we call it a Bias-Variance Tradeoff problem.

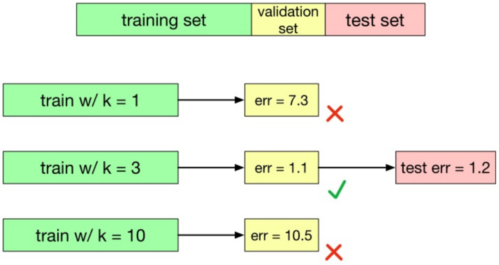


## Generalization error for k-NN

- We would like our algorithm to generalize to data it hasn't seen before.
- How can we measure the generalization error (error rate on new examples)?
- Maybe we can use the test set to pick k? However, once we use the test set to pick k, it is no longer an unbiased way of measuring of how well we will do on **unseen** data.
- k is an example of a **hyperparameter**, something we can't fit as part of the learning algorithm itself.

## Validation sets

- We can tune hyperparameters using a validation set, which is a third, unseen set of input-label pairs.
- The test set is used only at the very end, to measure the generalization performance of the final configuration.



## The Curse of Dimensionaility

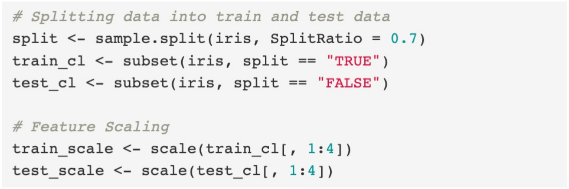
- Every machine learning algorithm needs a dense data set in order to accurately predict over the entire data space.
- The special challenge with k-nearest neighbors is that it requires a point to be close in every single dimension.
- Some algorithms can create regressions based on single dimensions, and only need points to be close together along that axis.
- KNN needs all points to be close along every axis in the data space. And each new axis added, by adding a new dimension, makes it harder and harder for two specific points to be close to each other in every axis.
- Low-dimensional visualizations are misleading. In high dimensions, “most” points are far apart.

## R Example

- Now, let's build a KNN algorithm in R using the package 'class'.
- Again, we will use the Iris dataset. Recall that it contains measurements of four features: sepal length, sepal width, petal length, and petal width.

```
## Rows: 150
## Columns: 5
## $ Sepal.Length <dbl> 5.1, 4.9, 4.7, 4.6, 5.0, 5.4, 4.6, 5.0, 4.4, 4.9, 5.4, 4.…
## $ Sepal.Width  <dbl> 3.5, 3.0, 3.2, 3.1, 3.6, 3.9, 3.4, 3.4, 2.9, 3.1, 3.7, 3.…
## $ Petal.Length <dbl> 1.4, 1.4, 1.3, 1.5, 1.4, 1.7, 1.4, 1.5, 1.4, 1.5, 1.5, 1.…
## $ Petal.Width  <dbl> 0.2, 0.2, 0.2, 0.2, 0.2, 0.4, 0.3, 0.2, 0.2, 0.1, 0.2, 0.…
## $ Species      <fct> setosa, setosa, setosa, setosa, setosa, setosa, setosa, s…
```

## R Example

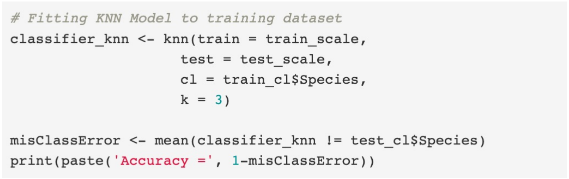


```r
# Splitting data into train and test data
split <- sample.split(iris, SplitRatio = 0.7)
train_cl <- subset(iris, split == "TRUE")
test_cl <- subset(iris, split == "FALSE")

# Feature Scaling
train_scale <- scale(train_cl[, 1:4])
test_scale <- scale(test_cl[, 1:4])
```

- We allocate 70% of the data for training and the rest for testing. For the sake of simplicity, we will not use a validation set here.
- For KNN, we also applied the 'scale' function for feature scaling.
- In KNN, the distance between data points is a crucial factor in predicting the class of a test point. If one feature has a much larger range of values than others, it can dominate the distance calculation, leading to biased results. Feature scaling ensures that each feature contributes approximately proportionately to the final distance.

## R Example



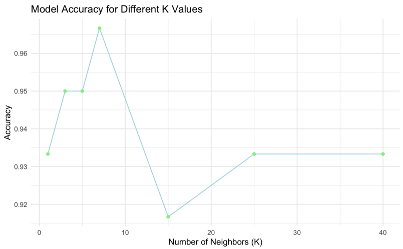
```r
# Fitting KNN Model to training dataset
classifier_knn <- knn(train = train_scale,
                      test = test_scale,
                      cl = train_cl$Species,
                      k = 3)

misClassError <- mean(classifier_knn != test_cl$Species)
print(paste('Accuracy =', 1-misClassError))
```

- We use the function 'knn()' to fit the KNN model used to fit the KNN model
- 'cl = train_cl$Species' specifies the class labels for the training data.
- k = 3 sets the number of neighbors to consider in the KNN algorithm.
- After fitting the model, 'classifier_knn' holds the predicted species for each instance in the test set.
- 'misClassError' calculates the mean misclassification error by comparing these predictions to the actual species in 'test_cl'.
- Finally, '1 - misClassError' gives the accuracy of the model, which is printed.

## R Example

Let's play around with the hyperparameter k and see how the model's accuracy changes.



The highest accuracy is achieved when k is around 7. As k increases, there's a sharp decline in accuracy, indicating that including more neighbors is detrimental to the model's performance. This could be because adding more neighbors includes points that are further away from the query point, potentially introducing noise into the prediction.

## KNN - Wrap-up

- Advantages
  - Easy to program
  - No optimization or training required
  - Classification accuracy can be very good; can outperform more complex models
- Disadvantages
  - Store all the training data in memory even at test time
  - Expensive at test time
  - May perform badly in high dimensions (curse of dimensionality)

## Today

**Learning goals**

- Basics KNN Algorithm
- Bias-Variance Tradeoff
- Implement KNN using R
- Acknowledging the Challenges and Limitations