Lecture 2 - Module 2. K-Nearest Neighbours

COMP 551 Applied machine learning

Yue Li
Assistant Professor
School of Computer Science
McGill University

January 9, 2025

Outline

Objectives

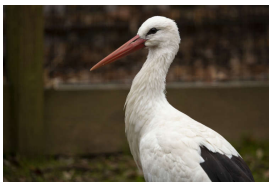
- You may ask "Why start the course with KNN among many other methods"?

  Simple to understand. It allows us to quickly focus on learning the fundamental ML concepts including hyperparameter selection and evaluation.
- Variations of k-nearest neighbours for classification and regression
  - How to train a KNN for classification
  - How to train a KNN for regression
- The choice of K as hyper-parameter
- Computational complexity
- Pros and cons of KNN

Outline

Classifying new objects based on similar objects observed before

We guess type of unseen instances based on their similarity to our past experience. For instance,



Type of bird:

(a) stork
(b) pigeon
(c) penguin

Language:

(a) South Asian
(b) African
(c) Middle eastern

"Accretropin" is a

(a) protein
(b) drug
(c) rock band

Predicting housing price based on the price of neighbour or similar houses (example of regression)

Terminologies of nearest neighbour (KNN) classification

- Training: KNN does nothing but "remembering" the training data set

$$\mathcal{D} = \{(\mathbf{x}^{(n)}, y^{(n)})\}_{n=1}^{N}$$

where

  - $\mathcal{D}$: training set
  - $\mathbf{x}^{(n)}$: a D-dimensional input feature of the n^{th} training example
  - $y^{(n)}$: a categorical output of the n^{th} training example
  - N: number of training examples (i.e., the total number of $(\mathbf{x}^{(n)}, y^{(n)})$ pairs)
  - n: index of training example

Because of that, KNN is known as examplar-based (or instance-based) learner or non-parametric method.

- Prediction: KNN predicts the label of a new data point based on labels from the K most similar examples in the training set (more details follow).

Pop quiz

1. Input dimension D? 4
2. Number of training examples N? 150
3. What's the value stored in $x_1^{(50)}$? 7.0
4. What's the value stored in $x[1, 3]$? 0.2

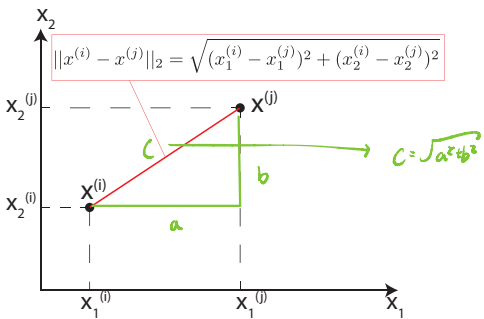
index	sepal length (x_0)	sepal width (x_1)	petal length (x_2)	petal width (x_3)	label (y)
0	5.1	3.5	1.4	0.3	Setosa
1	4.9	3.0	1.7	0.2	Setosa
⋮	⋮	⋮	⋮	⋮	⋮
50	7.0	3.2	4.7	1.4	Versicolor
⋮	⋮	⋮	⋮	⋮	⋮
149	5.9	3.0	5.1	1.8	Virginica

Pairwise distance

The key step in KNN in predicting new data point $\mathbf{x}^{(*)}$ is to find the K most similar examples in the training set. Let's first consider $K = 1$, i.e., we want to find the most similar example from the training set for the new data point $\mathbf{x}^{(*)}$. There are many ways to measure similarity. One common way is by *Euclidean distance*:

$$distance(\mathbf{x}^{(i)}, \mathbf{x}^{(j)}) = ||\mathbf{x}^{(i)} - \mathbf{x}^{(j)}||_2 = \sqrt{\sum_{d=1}^{D} (x_d^{(i)} - x_d^{(j)})^2} \quad (1)$$

Euclidean distance between any pair of data points is the length of the line segment between them. The plot illustrates the Euclidean distance between point $\mathbf{x}^{(i)}$ and point $\mathbf{x}^{(j)}$ in two dimensions.



Making prediction using KNN algorithm ($K = 1$)

Predict the label of a new data point $\mathbf{x}^{(*)}$ based on labels from the most similar example (i.e., $K = 1$) in the training set in terms of *Euclidean distance*:

sorting by distance

$$i = \underset{i \in \text{training set}}{\arg\min} \; \textit{distance}(\mathbf{x}^{(i)}, \mathbf{x}^{(*)}) = \underset{i \in \text{training set}}{\arg\min} \; ||\mathbf{x}^{(i)} - \mathbf{x}^{(j)}||_2 \quad (2)$$

$$y^{(*)} = y^{(i)} \quad (3)$$

We can also use other distance or similarity functions.

For continuous input features, we can use:

- Manhattan distance: $\sum_{d=1}^{D} |x_d^{(i)} - x_d^{(j)}|$
- Cosine similarity: $\frac{\sum_{d=1}^{D} x_d^{(i)} x_d^{(j)}}{(\sqrt{\sum_d (x_d^{(i)})^2} \sqrt{\sum_d (x_d^{(j)})^2})}$

L2 norm of each

* if the same, dot product will increase

$\|\mathbf{x}^{(i)}\|_2 \cdot \|\mathbf{x}^{(j)}\|_2$

For discrete or categorical input features, we can use:

- Hamming distance: $\sum_{d=1}^{D} \mathbb{I}(x_d^{(i)} \neq x_d^{(j)})$, where $\mathbb{I}(b) = \begin{cases} 1 & \text{if b is True} \\ 0 & \text{if b is False} \end{cases}$

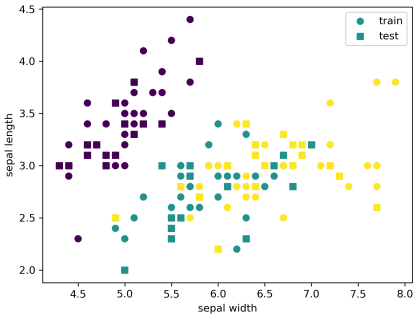
boolean indicator

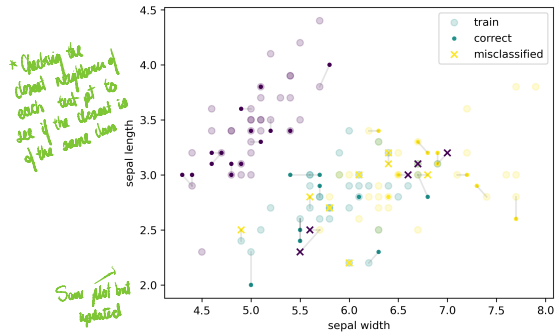
*Sum up all the features that are not the same ⇒ the lower the Hamming number, the more similar the data is

Note: For similarity, we need to choose the max: $i = \underset{i \in \text{training set}}{\arg\max} \; \textit{similarity}(\mathbf{x}^{(i)}, \mathbf{x}^{(*)})$

KNN algorithm ($K = 1$) applied to Iris dataset (See Colab for the code)

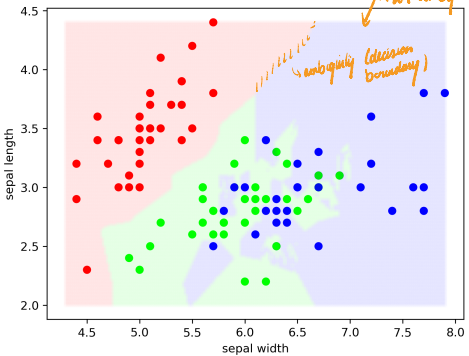
- $\{\mathbf{x}^{(n)}, y^{(n)}\}_{n=1}^{N=150}$ 2D input: sepal length (x_1) and sepal width (x_2) (i.e., $x^{(n)} \in \mathbb{R}^2$)
- 3-class output: flower classes $y \in \{setosa, versicolor, virginica\}$ (50 flowers each)
- Randomly split data into **training** (100 flowers) and **testing** (50 flowers)
- **Prediction accuracy** on the test set: $\frac{1}{50}\sum_{t=1}^{50}\mathbb{I}(\hat{y}^{(t)} = y^{(t)}) = 74\%$
- The right panel displays the testing data points connected with its nearest neighbour (big circles) from the training set. The correctly classified test data points are small circles and the incorrect predictions are crosses.





Decision boundaries created by the KNN (K=1) (see Colab)

- Voronoi diagram illustrates the decision boundary of KNN (K=1).
- It generates 200 values evenly spaced from min value to max value for sepal length $\in [2.0, 4.4]$ and sepal width $\in [4.3, 7.9]$.
- Therefore, there are 40,000 data points being predicted by the KNN based on the 100 training data points.
- The color shades were generated by predicting all input features



- The 40K points were colored based on the class label of their nearest neighbours from the training set.
- Each color corresponds to a flower class
- Try understand why some regions were assigned the specific color
- Is K=1 too flexible or too rigid?
- What does it look like if $K = N$?

Outline

KNN ($K \geq 1$) for discrete class predictions

We set the predicted label to be the most frequently occurring class among the K neighbours $c \in \{1, \ldots, C\}$ for a new data point $\mathbf{x}^{(*)}$:

$$\hat{y}^{(*)} = \arg\max_{c} \sum_{n \in \mathcal{N}_K(\mathbf{x}^{(*)}, \mathcal{D})} \mathbb{I}(y^{(n)} = c) \quad (4)$$

Here

- $\mathcal{N}_K(\mathbf{x}^{(*)}, \mathcal{D})$ denotes the K closest examples to $\mathbf{x}^{(*)}$ based on a distance function (e.g., Euclidean), and $\mathcal{D} = \{\mathbf{x}^{(n)}, y^{(n)}\}_{n=1}^{N}$ is the training data
- When $K = 1$, the KNN reduces to finding *the most similar* example as above

KNN ($K \geq 1$) for class probabilities

We calculate the class probabilities of each class $c \in \{1, 2, 3\}$ for a new data point $\mathbf{x}^{(*)}$:

$$p(y^{(*)} = c|\mathbf{x}^{(*)}) = \frac{1}{K} \sum_{n \in \mathcal{N}_K(\mathbf{x}^{(*)}, \mathcal{D})} \mathbb{I}(y^{(n)} = c) \tag{5}$$

*Similar to (4), but we take the fraction of it

Here

- $\mathcal{N}_K(\mathbf{x}^{(*)}, \mathcal{D})$ denotes the K closest examples to $\mathbf{x}^{(*)}$ based on a distance function (e.g., Euclidean), and $\mathcal{D} = \{\mathbf{x}^{(n)}, y^{(n)}\}_{n=1}^{N}$ is the training data
- When $K = 1$, the KNN reduces to finding *the most similar* example as above

For example, for $K = 3$,

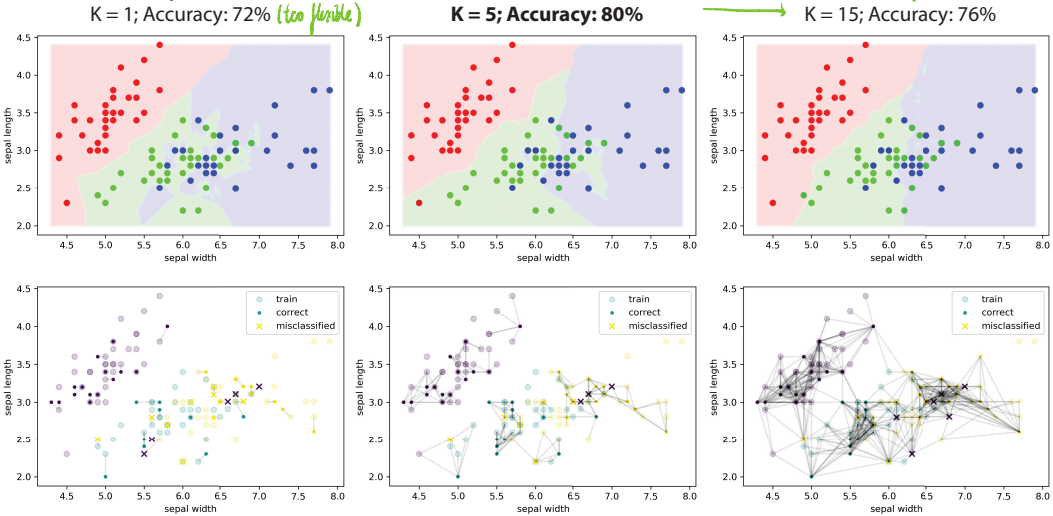
- $\mathcal{N}_K(\mathbf{x}^{(*)}, \mathcal{D}) = \{n_1, n_2, n_3\}$, which are the indices of the three most similar training examples of data point $\mathbf{x}^{(*)}$.
- Pop quiz: suppose $y^{(n_1)} = 1$, $y^{(n_2)} = 1$, $y^{(n_3)} = 3$, what are the probabilities for each class $p(y^{(*)} = c|\mathbf{x}^{(*)})$? Answer: 2/3, 0, 1/3

* $\frac{2}{3} + \frac{0}{3} + \frac{1}{3} = 1$

Outline

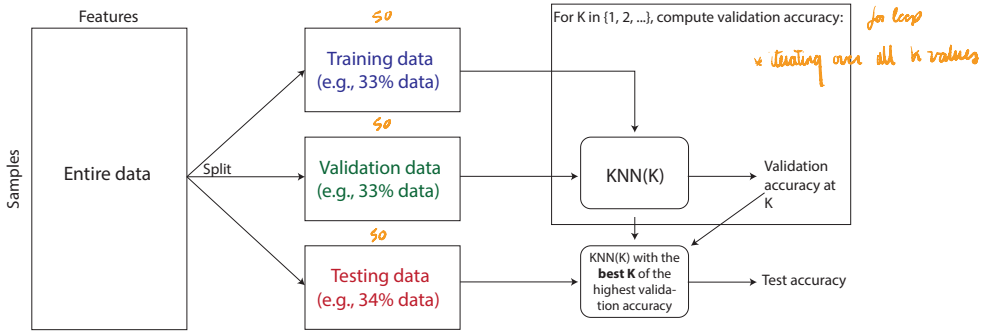
The choice of K

Choice of K is a **hyper-parameter** of the model, which are chosen empirically based on the model performance on a **held-out validation dataset**.



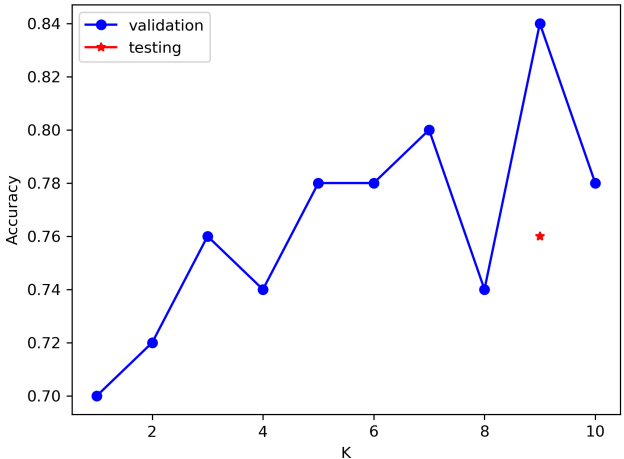
How to choose K

- The goal is to accurately predict unseen data that are not in the training set
- The accuracy can be approximated by the accuracy on the **test set**.
- To this end, we split the entire data into training, validation, and testing data.
- We use training data to "train" KNN, validation data to choose the hyperparameter from a finite set (i.e., $K \in \{1, 5, \ldots, \}$) that confers the highest validation accuracy, and finally we evaluate the chosen model on the testing data.



How to choose K using validation set (See Colab)

- To choose the best K, we further split the training data (N=100) into 50% training and 50% validation
- The red asterisk is the test accuracy based on the best $K = 9$.



Outline

Time complexity

- Assume we are using Euclidean distance $\sqrt{\sum_{d=1}^{D}(x_d^{(n)} - x_d^{(*)})^2}$
- For each data point in the training set, calculate the distance in $O(D)$ for a total of $O(ND)$
- Find the K points with smallest of distances in $O(NK)$ or more precisely $\sum_{k=0}^{K-1} N - k$ (i.e., first find the smallest, then the second smallest, and so on)
- The computational complexity for a single test query: $O(ND + NK)$

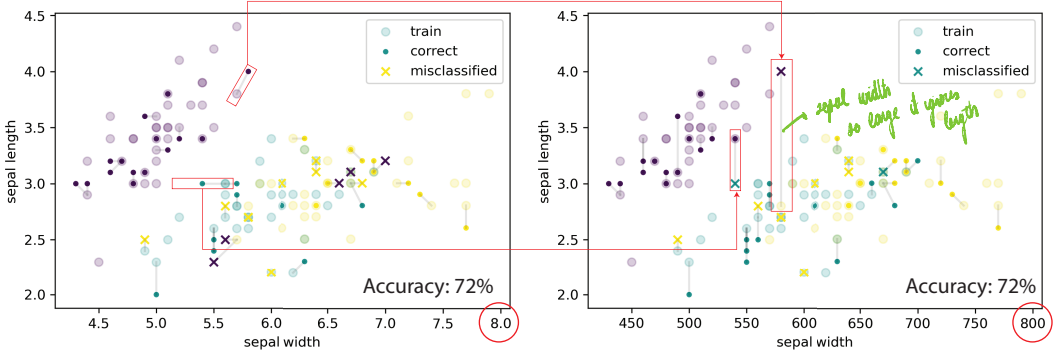
Improving the nearest neighbour search by k-d tree (optional)

Rough idea:

- We improve search of K nearest points by first building a tree by recursively clustering the training sets.
- To search the K nearest neighbours of a query point, we then go through the tree without looking at all of the N training points. $O(2^D + \log N)$

KNN is sensitive to feature scaling

- Euclidean distance: $distance(\mathbf{x}^{(i)}, \mathbf{x}^{(j)}) = ||\mathbf{x}^{(i)} - \mathbf{x}^{(j)}||_2 = \sqrt{\sum_{d=1}^{D}(x_d^{(i)} - x_d^{(j)})^2}$
- Here we multiplied the sepal width by 100.
- As a result, the highlighted point was misclassified since the sepal width is almost the same and the sepal length difference is ignored due to much lower scale – it becomes negligible in the overall distance, and width becomes the main factor contributing to the distance between any two points



Standardizing features before running KNN prediction

make sure same scale

We can standardize each feature d as follows:

$$x_d^{(n)} = \frac{x_d^{(n)} - \bar{x}_d}{sd(x_d)}$$

where

- $\bar{x}_d$ is the mean of feature d: $\bar{x}_d = \frac{1}{N}\sum_n x_d^{(n)}$
- $sd(x_d)$ is the standard deviation of feature d: $sd(x_d) = \sqrt{\frac{1}{N}\sum_n (x_d^{(n)} - \bar{x}_d)^2}$

$\frac{1}{n}\sum_1^N \left(x_d^{(n)} - \bar{x}_d\right) = \frac{1}{N}\sum_1^N x_d^{(n)} - \frac{1}{N}\sum_1^N \bar{x}_d$ (constant)

$= \bar{x}_d - \bar{x}_d$

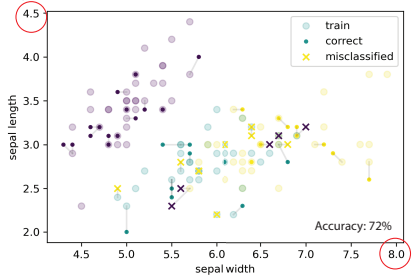
$= 0$

τ scaling (divide each feature by standard dev.)

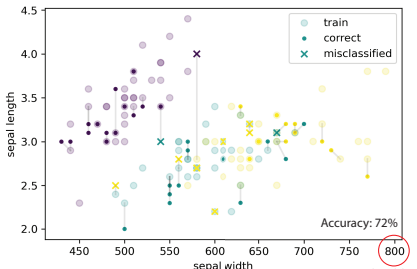
$s_d^2(x_d^*) = \frac{1}{N}\sum_n \left(\frac{x_d^{(n)}}{\sigma_d}\right) = \frac{\frac{1}{N}\sum_n (x_d^{(n)})^2}{\sigma_d^2} = 1$

Standardizing features before running KNN prediction

Original feature scale



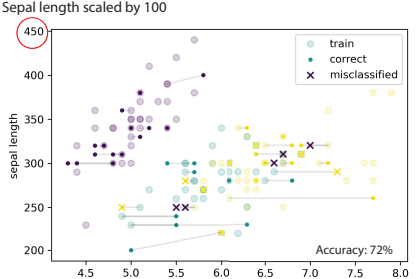
Sepal width scaled by 100



length spread

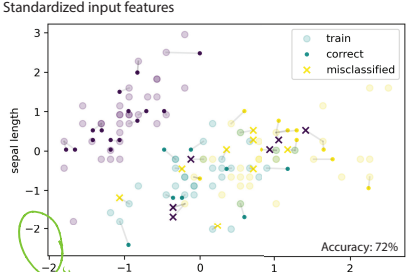
Sepal widthscaled by 100

Sepal length scaled by 100



width spread

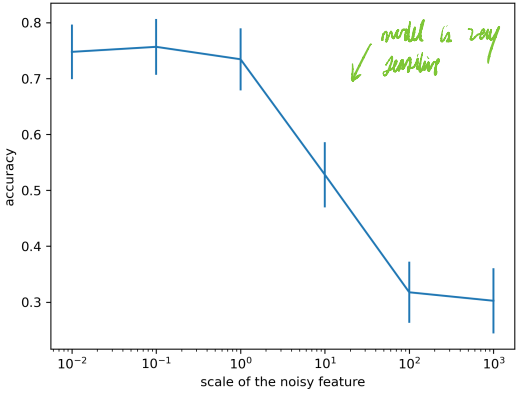
Standardized input features



normal that you have neg values → min = 0

KNN is sensitive to random class-irrelevant features

- Here we create a random noise feature sampled from a standard normal distribution $x_\epsilon^{(n)} \sim \mathcal{N}(0, 1)$
- Scale the noise feature by a factor of $s \in [.01, .1, 1, 10, 100, 1000]$
- Add the scaled noise feature to the two input features: $\mathbf{x}^{(n)} = [x_{\text{sepal length}}^{(n)}, x_{\text{sepal width}}^{(n)}, s \times x_\epsilon^{(n)}]$

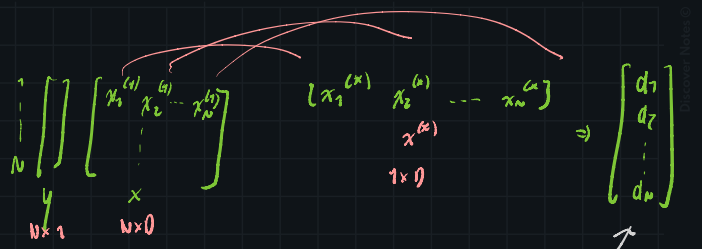


- As shown on the left, the larger the scaling factor s, the lower the prediction accuracy of the KNN (K=3).
- It is because our current KNN implementation is unable to distinguish feature importance
- Pop quiz: Can you think of a way to fix that? Answer: remove features with low correlation with the label.

How to determine the weight ?

* 0 → not important feature
  1 → important feature

- Best way : use **cosine similarity** ⇒ relate to correlation



for $d = 1 \dots D$, $\frac{y^T x_d}{\|y\|_2 \|x_d\|_2}$

$\|x_d\|_2 = \sqrt{\sum_n (x_d^{(n)})}$

$= N$

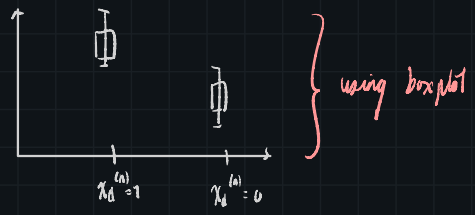
↳ $\frac{y^T x}{N} = [1 \dots 0]$

- Another way, but more expensive : Cross section ⇒ try all possibilities

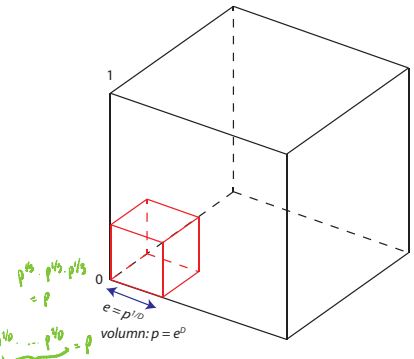
$= \left(\frac{1}{N_1} \left(\sum_{n=1}^{N} 1\left[x_d^{(n)} = 1\right] y^{(n)} \right) - \frac{1}{N_0} \left(\sum_{n=1}^{N} 1\left[x_d^{(n)} = 0\right] y^{(n)} \right) \right)^2$ ← Here the features are binary

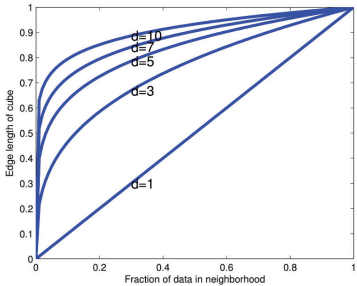
↑ bool — values

⇒ The bigger the value, the more different they are



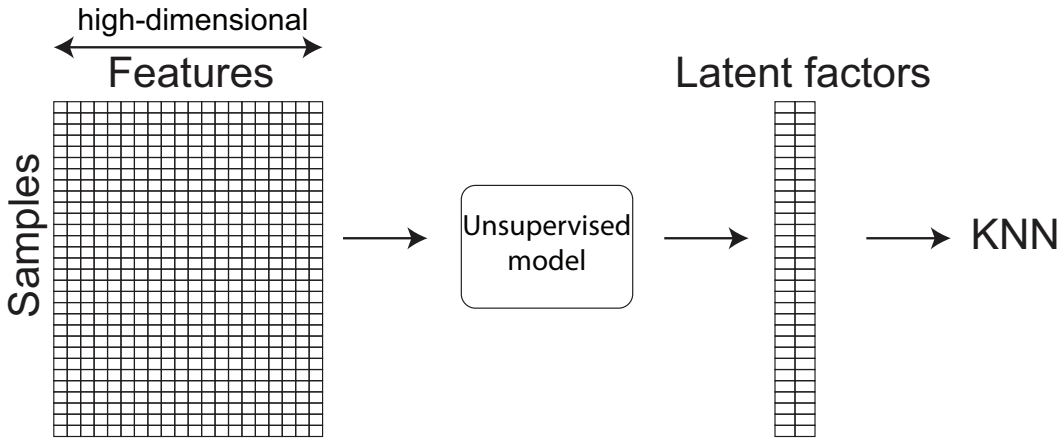
Curse of dimensionality (Murphy22 16.1.2)





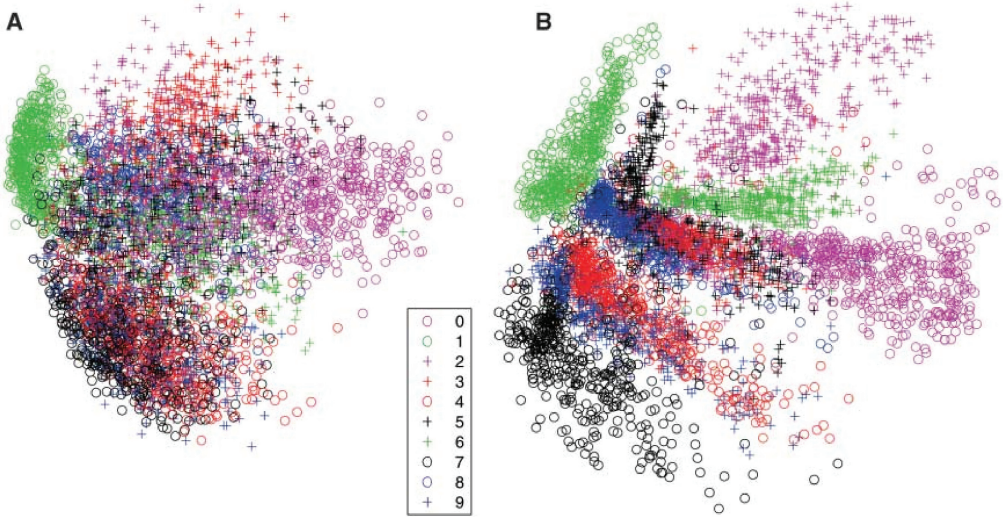
- Basic idea: the volume of space grows exponentially fast with dimension, so you might have to look quite far away in space to find your nearest neighbour.
- If $D = 10$ and we use $p = 10\%$ of the data points, the edge length $0.1^{1/10} = 0.8$. Even if we use 1% of the data, we still need an edge length of $0.01^{1/10} = 0.63$.
- Data points are sparsely distributed with high D and most pair of data points seem far from each other, which makes it difficult to find meaningful nearest neighbors.

Dimensionality reduction + KNN



We will discuss unsupervised learning in Module 7.

Real-world data are **not** uniformly distributed (Hinton *Science* 2006)



A. 500 MNIST digits of each class clustered by the first two PCs
B. Same digits clustered by the output from an autoencoder of architecture 784-1000-500-250-2

Weighted KNN

Weighted KNN weighs the contributions of each example by their distance from the test data:

$$p(y^{(*)} = c|\mathbf{x}^{(*)}) = \frac{1}{W_K(\mathbf{x}^{(*)}, \mathcal{D}))} \sum_{n \in \mathcal{N}_K(\mathbf{x}^{(*)}, \mathcal{D})} similarity(\mathbf{x}^{(n)}, \mathbf{x}^{(*)}) \mathbb{I}(y^{(n)} = c) \quad (6)$$

where

- similarity is defined as either the inverse of the Euclidean distance or cosine similarity
- the normalization factor is defined as:

$$W_K(\mathbf{x}^{(*)}, \mathcal{D})) = \sum_{n \in \mathcal{N}_K(\mathbf{x}^{(*)}, \mathcal{D})} similarity(\mathbf{x}^{(n)}, \mathbf{x}^{(*)})$$

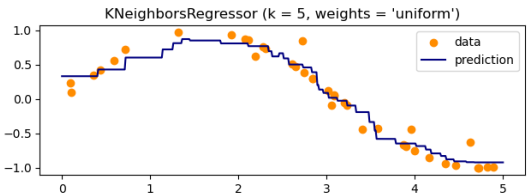
If $similarity(\mathbf{x}^{(n)}, \mathbf{x}^{(*)}) = 1 \forall n \in \mathcal{N}_K(\mathbf{x}^{(*)}, \mathcal{D})$, weighted KNN reduces to the unweighted KNN.

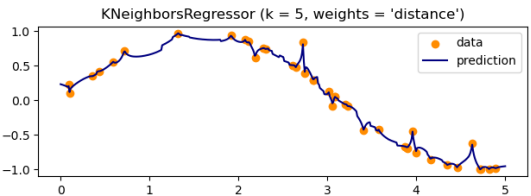
KNN regression

Adapting a classification KNN for regression is quite straightforward.

$$y^{(*)} = \frac{1}{K} \sum_{n \in \mathcal{N}_K(\mathbf{x}^{(*)}, \mathcal{D})} y^{(n)} \qquad \text{(Unweighted)}$$

$$y^{(*)} = \frac{1}{W_K(\mathbf{x}^{(*)}, \mathcal{D}))} \sum_{n \in \mathcal{N}_K(\mathbf{x}^{(*)}, \mathcal{D})} similarity(\mathbf{x}^{(n)}, \mathbf{x}^{(*)}) y^{(n)} \qquad \text{(Weighted)}$$





Predicting $y = sine(x)$ from a 1D feature x using Scikit-learn KNN

Summary

KNN performs classification/regression by finding similar instances in the training set:

- need a notion of distance, performance improves a lot with a better similarity measure e.g. see here
- Optimal number of neighbours (i.e., K) is chosen by the validation performance
- Weighted KNN weighs the neighbours' contributions by their distance from the test data point

KNN is a non-parametric method and a lazy learner: ← not trained

- non-parametric: our model has no parameters (in fact the training data points are model parameters)
- Lazy, because we don't do anything during the training
  - test-time complexity grows with the size of the training data, as well as space complexity (store all data)
  - good performance when we have lots of data, see here
  - KNN is sensitive to curse of dimensionality, feature scaling and noise