SE125 Machine Learning

K-Nearest Neighbors (KNN)

Yue Ding

School of Software, Shanghai Jiao Tong University
dingyue@sjtu.edu.cn

K-Nearest Neighbors

- 课程难度:

- 掌握程度:

References and Acknowledgement

- *Machine Learning : The Art and Science of Algorithms that Make Sense of Data By Peter Flach*
- A presentation on KNN Algorithm : West Virginia University , Published on May 22, 2015
- https://www.slideshare.net/PhuongNguyen6/text-categorization
- https://www.slideshare.net/tilanigunawardena/k-nearest-neighbors
- https://www.visiondummy.com/2014/04/curse-dimensionality-affect-classification/
- Introduction, Course Overview, Brief history of computer vision. CS294-129: Designing, Visualizing and Understanding Deep Neural Networks, https://bcourses.berkeley.edu/courses/1468734/pages/cs294-129-designing-visualizing-and-understanding-deep-neural-networks

K-Nearest Neighbors

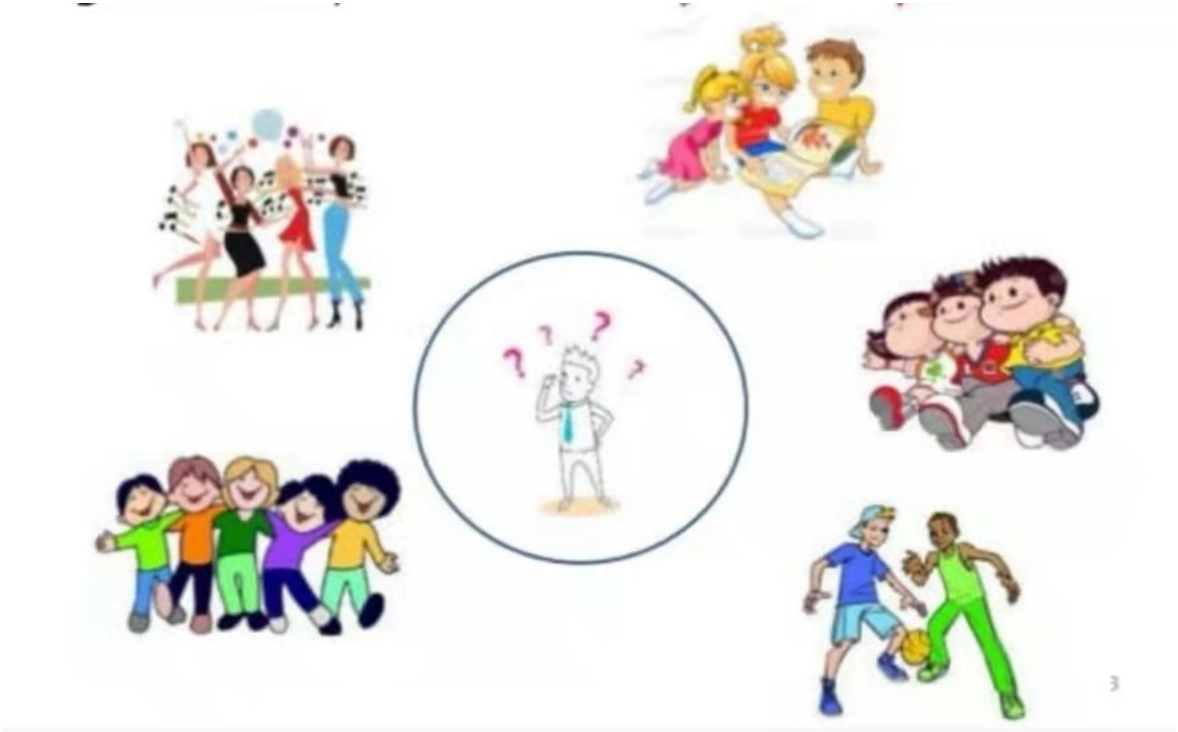
- Supervised learning:
  - Linear regression
  - Decision trees
  - Bayesian classification (Native Bayesian classifier)
  - **K-nearest neighbors**
  - Logistic regression
  - Support vector machine (SVM)
  - Neural networks (deep learning)
  - ……

K-Nearest Neighbors

- Simple, but a very powerful classification algorithm
- One of the top data mining algorithms used today
- Classifies based on a **similarity measure**

K-Nearest Neighbors

- A simple analogy…
  - Tell me about your friends (who your neighbors are).
  - Then I will tell you who you are.

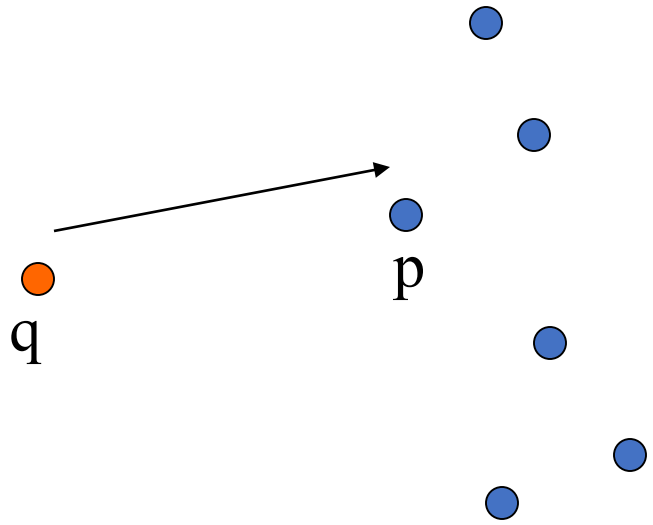


K-Nearest Neighbors

- Instance-based learning, also called lazy learning
  - Does not “learn” until the test example is given
  - Whenever we have a new data to classify, we find its K-nearest neighbors from the training data

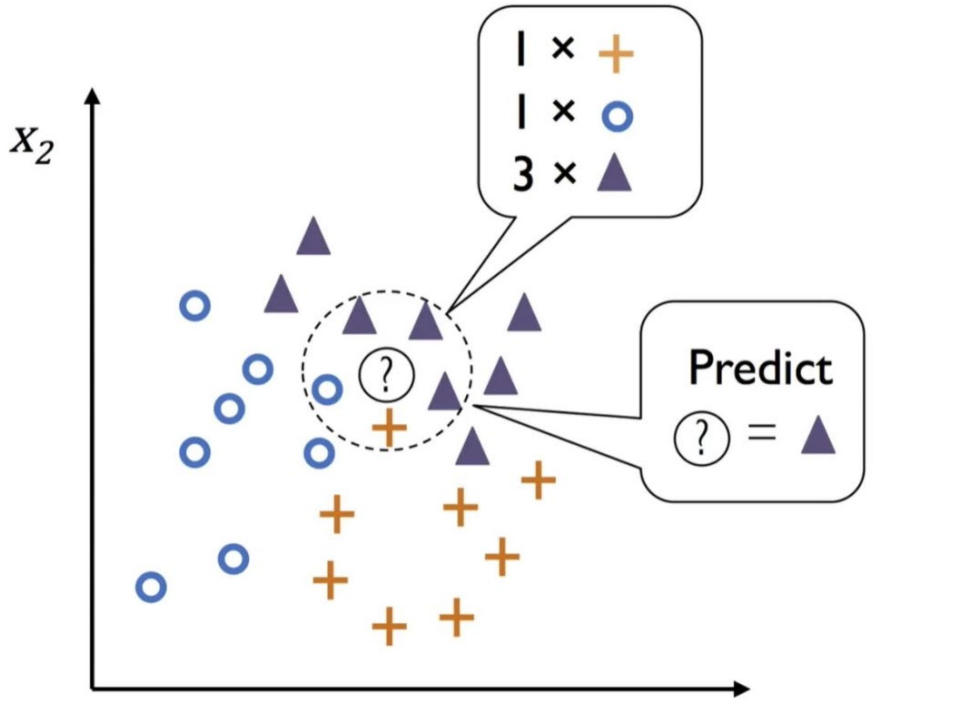
Nearest Neighbor Search

- Given: a set P of n points in R^d
- Goal: a data structure, which given a query point q, finds the *nearest neighbor* p of q in P



KNN Classification

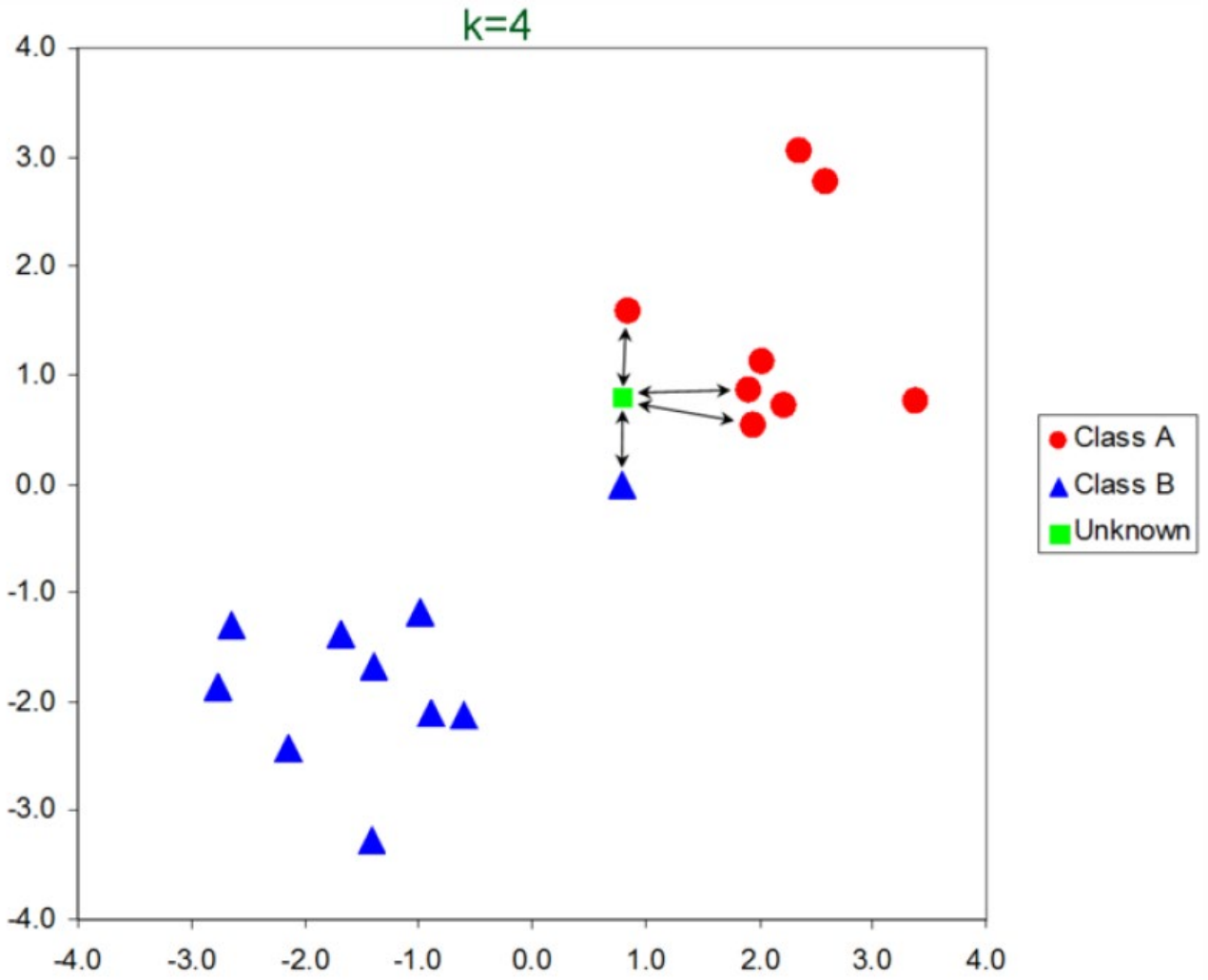
- Classified by "**MAJORITY VOTES**" for its neighbor classes
  - Assigned to the most common class amongst its *K*-nearest neighbors (by measuring "distance" between data)



KNN: Pseudocode

- Step 1: Determine parameter *K* = number of nearest neighbors
- Step 2: Calculate the distance between the query-instance and all the training examples.
- Step 3: Sort the distance and determine nearest neighbors based on the *k*-th minimum distance.
- Step 4: Gather the category Y of the nearest neighbors.
- Step 5: Use simple majority of the category of nearest neighbors as the prediction value of the query instance.

KNN: Example


k=4
4.0
3.0
2.0
1.0
0.0
-1.0
-2.0
-3.0
-4.0
-4.0
-3.0
-2.0
-1.0
0.0
1.0
2.0
3.0
4.0
Class A
Class B
Unknown

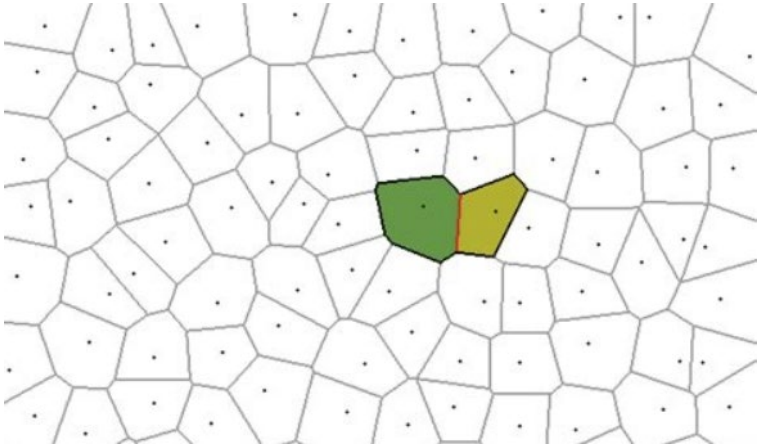
KNN: Euclidean Distance Matrix

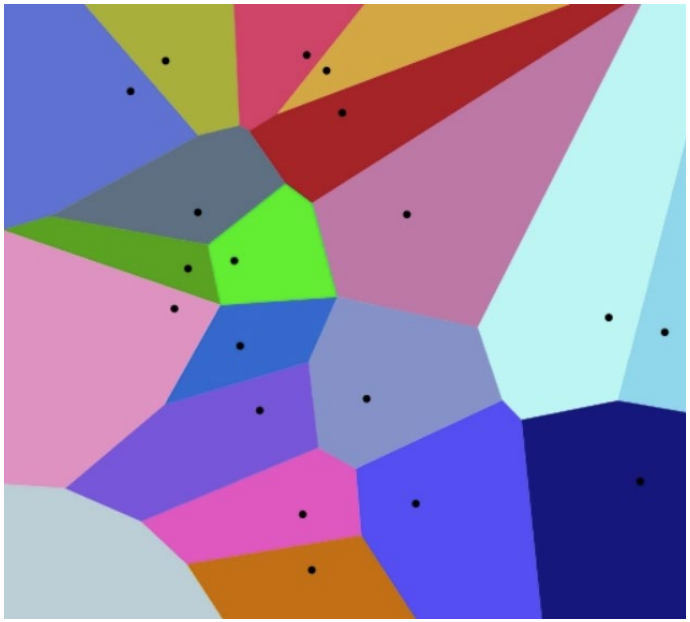
- Euclidean distance matrix D listing all possible pairwise Euclidean distances between 19 samples.

	x_1	x_2	x_3	x_4	x_5	x_6	x_7	x_8	x_9	x_{10}	x_{11}	x_{12}	x_{13}	x_{14}	x_{15}	x_{16}	x_{17}	x_{18}
x_2	1.5																	
x_3	1.4	1.6																
x_4	1.6	1.4	1.3															
x_5	1.7	1.4	1.5	1.5														
x_6	1.3	1.4	1.4	1.5	1.4													
x_7	1.6	1.3	1.4	1.4	1.5	1.8												
x_8	1.5	1.4	1.6	1.3	1.7	1.6	1.4											
x_9	1.4	1.3	1.4	1.5	1.2	1.4	1.3	1.5										
x_{10}	2.3	2.4	2.5	2.3	2.6	2.7	2.8	2.7	3.1									
x_{11}	2.9	2.8	2.9	3.0	2.9	3.1	2.9	3.1	3.0	1.5								
x_{12}	3.2	3.3	3.2	3.1	3.3	3.4	3.3	3.4	3.5	3.3	1.6							
x_{13}	3.3	3.4	3.2	3.2	3.3	3.4	3.2	3.3	3.5	3.6	1.4	1.7						
x_{14}	3.4	3.2	3.5	3.4	3.7	3.5	3.6	3.3	3.5	3.6	1.5	1.8	0.5					
x_{15}	4.2	4.1	4.1	4.1	4.1	4.1	4.1	4.1	4.1	4.1	1.7	1.6	0.3	0.5				
x_{16}	4.1	4.1	4.1	4.1	4.1	4.1	4.1	4.1	4.1	4.1	1.6	1.5	0.4	0.5	0.4			
x_{17}	5.9	6.2	6.2	5.8	6.1	6.0	6.1	5.9	5.8	6.0	2.3	2.3	2.5	2.3	2.4	2.5		
x_{18}	6.1	6.3	6.2	5.8	6.1	6.0	6.1	5.9	5.8	6.0	3.1	2.7	2.6	2.3	2.5	2.6	3.0	
x_{19}	6.0	6.1	6.2	5.8	6.1	6.0	6.1	5.9	5.8	6.0	3.0	2.9	2.7	2.4	2.5	2.8	3.1	0.4

Decision Boundaries

- Voronoi diagram
  - Describes the areas that are nearest to any given point, given a set of data
  - Each line segment is equidistant between two points of opposite class



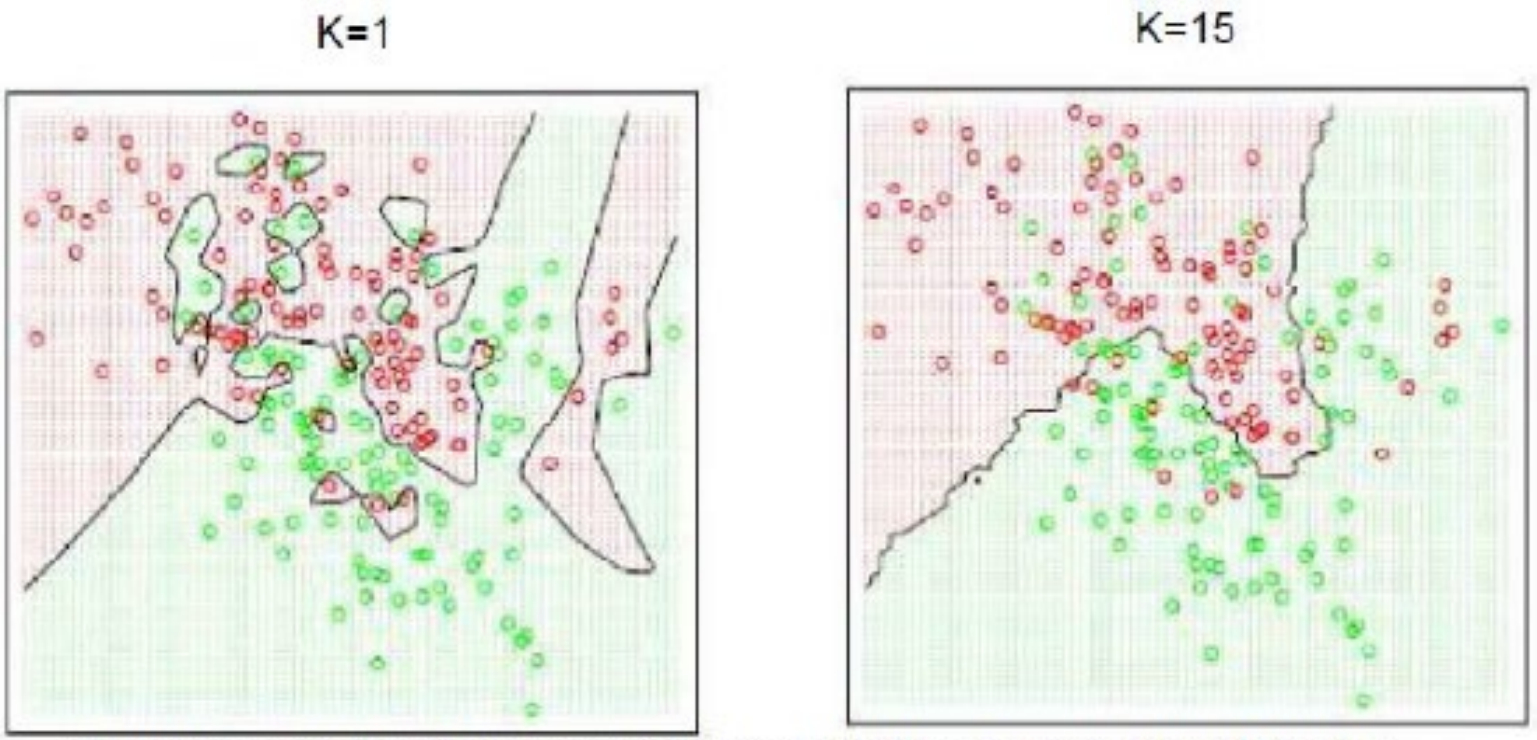


Decision Boundaries

- With large number of examples and possible noise in the labels, the decision boundary can become nasty!

Effect of K

- Larger *K* produces smoother boundary effect
- When *K*==*N*, always predict the majority class



Figures from Hastie, Tibshirani and Friedman (Elements of Statistical Learning)

Distances

- Distance
  - Numerical measure of how alike two data objects are
  - Is higher when objects are more alike.
  - Often falls in the range [0,1]

Distances

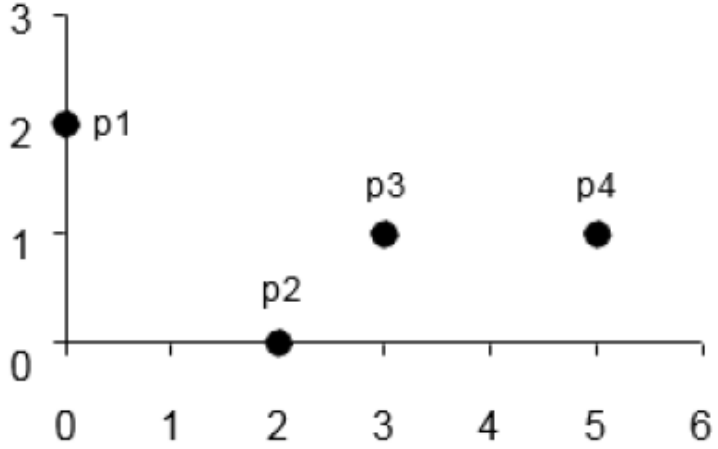
- Euclidean Distance

$$dist = \sqrt{\sum_{k=1}^{p}(a_k - b_k)^2}$$

Where *p* is the number of dimensions (attributes)

a_k and b_k are the k-th attributes of data objects a and b, respectively.

Distances



point	x	y
p1	0	2
p2	2	0
p3	3	1
p4	5	1

	p1	p2	p3	p4
p1	0	2.828	3.162	5.099
p2	2.828	0	1.414	3.162
p3	3.162	1.414	0	2
p4	5.099	3.162	2	0

Feature Scaling

- Standardize the range of independent variables (features of data)
- A.k.a Normalization or Standardization

Standardization

- Standardization or Z-score normalization
  - Rescale the data so that the mean is zero and the standard deviation from the mean (standard scores) is one.

$$x_{norm} = \frac{x - \mu}{\sigma}$$

  - μ is mean, σ is a standard deviation from the mean.

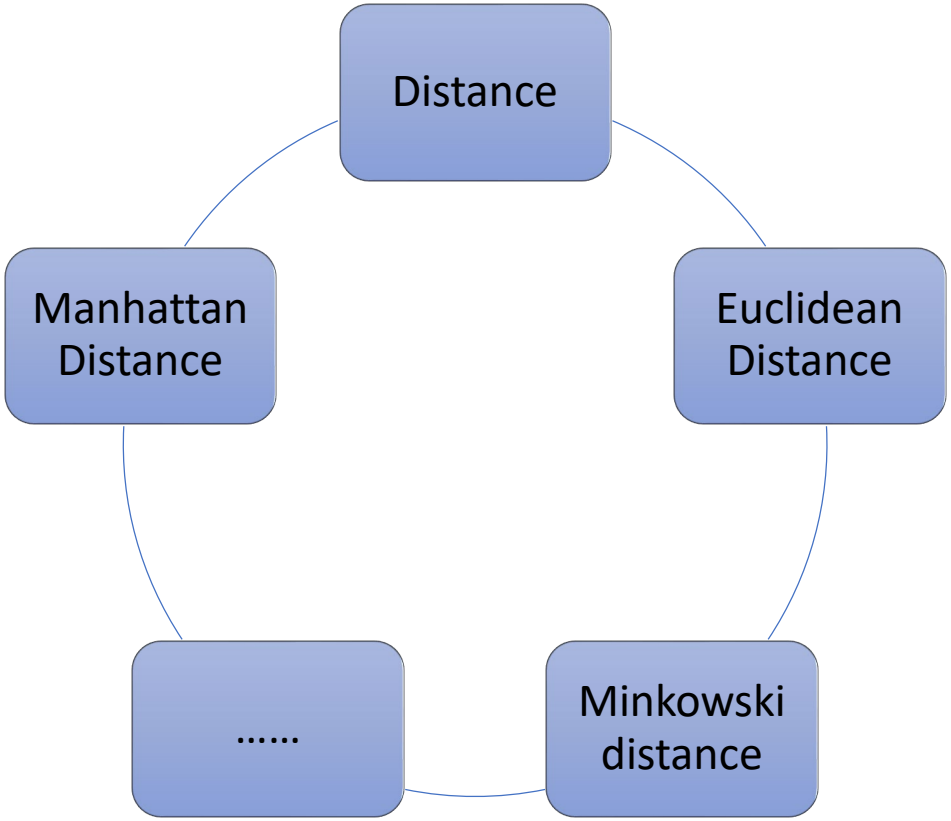
Min-Max Scaling

- Scale the data to a fixed range between 0 and 1.

$$x_{morm} = \frac{x - x_{min}}{x_{max} - x_{min}}$$

Distances

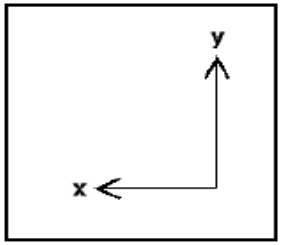
- Distance are used to measure similarity
- There are many ways to measure the distances between two instances



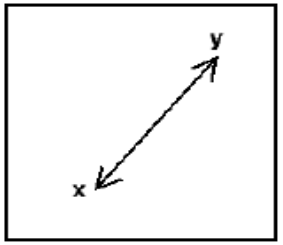
Distances

- Manhattan Distance

$$|x_1 - x_2| + |y_1 - y_2|$$



Manhattan



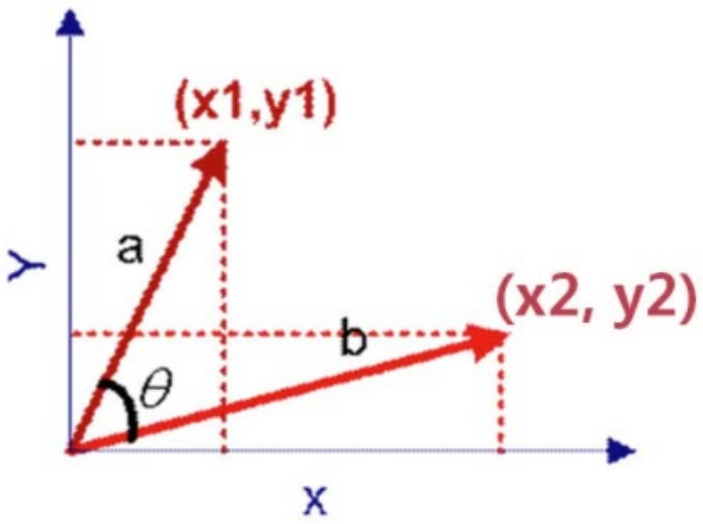
Euclidean

- Minkowski Distance

$$D(x, y) = \left(\sum_{u=1}^{n} |x_u - y_u|^p \right)^{\frac{1}{p}}$$

Distances

- Cosine Similarity



$$\cos(\theta) = \frac{\mathrm{a} \bullet \mathrm{b}}{||\ \mathrm{a}\ || \times ||\ \mathrm{b}\ ||}$$

$$= \frac{(x_1, y_1) \bullet (x_2, y_2)}{\sqrt{x_1^2 + y_1^2} \times \sqrt{x_2^2 + y_2^2}}$$

$$= \frac{x_1 x_2 + y_1 y_2}{\sqrt{x_1^2 + y_1^2} \times \sqrt{x_2^2 + y_2^2}}$$

Distances

- Example:

$$\cos(d_1, d_2) = (d_1 \cdot d_2) / ||d_1|| \, ||d_2||$$

d_1 = 3 2 0 5 0 0 0 2 0 0

d_2 = 1 0 0 0 0 0 0 1 0 2

$d_1 \cdot d_2$ = 3*1 + 2*0 + 0*0 + 5*0 + 0*0 + 0*0 + 0*0 + 2*1 + 0*0 + 0*2 = 5

$||d_1||$ = (3*3+2*2+0*0+5*5+0*0+0*0+0*0+2*2+0*0+0*0)**0.5**= (42)^**0.5**= 6.481

$||d_1||$ = (1*1+0*0+0*0+0*0+0*0+0*0+0*0+1*1+0*0+2*2)**0.5**= (6)^**0.5**= 2.245

cos(d1, d2) = .3150

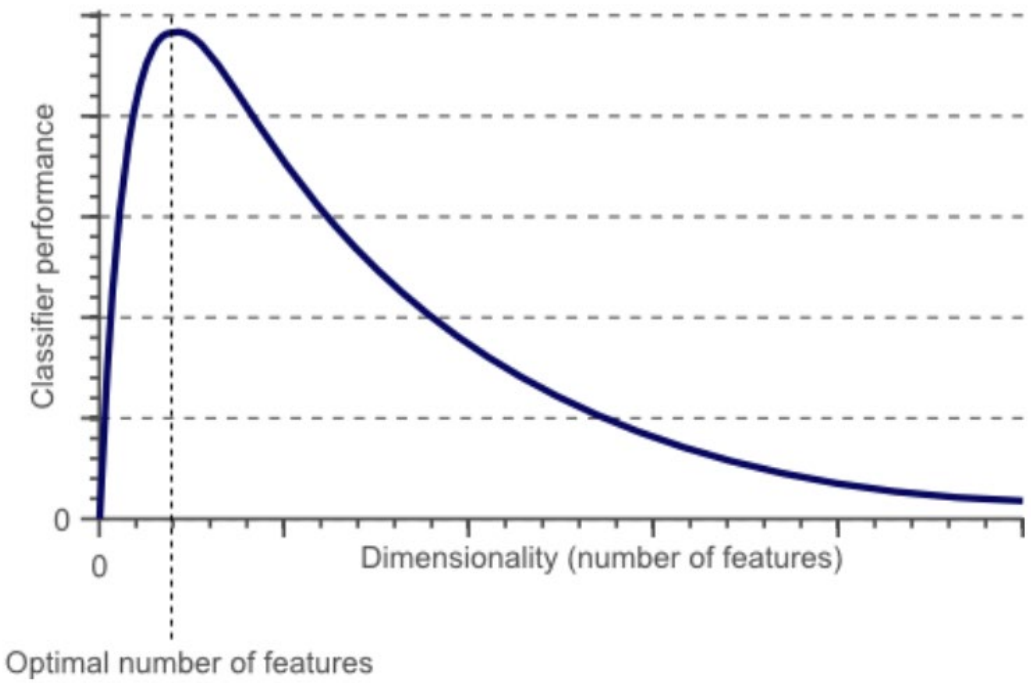
$$\cos(d_1, d_2) = \begin{cases} 1: \text{exactly the same} \\ 0: \text{orthogonal} \\ -1: \text{exactly opposite} \end{cases}$$

Distances

- Euclidean distance is not a good distance in high dimensions
- Distance measures start losing their effectiveness to measure dissimilarity in highly dimensional spaces

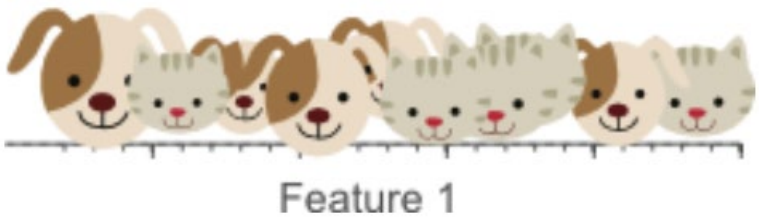
Curse of Dimensionality

- As the dimensionality increases, the classifier's performance increases until the optimal number of features is reached. Further increasing the dimensionality without increasing the number of training samples results in a decrease in classifier performance.



Curse of Dimensionality

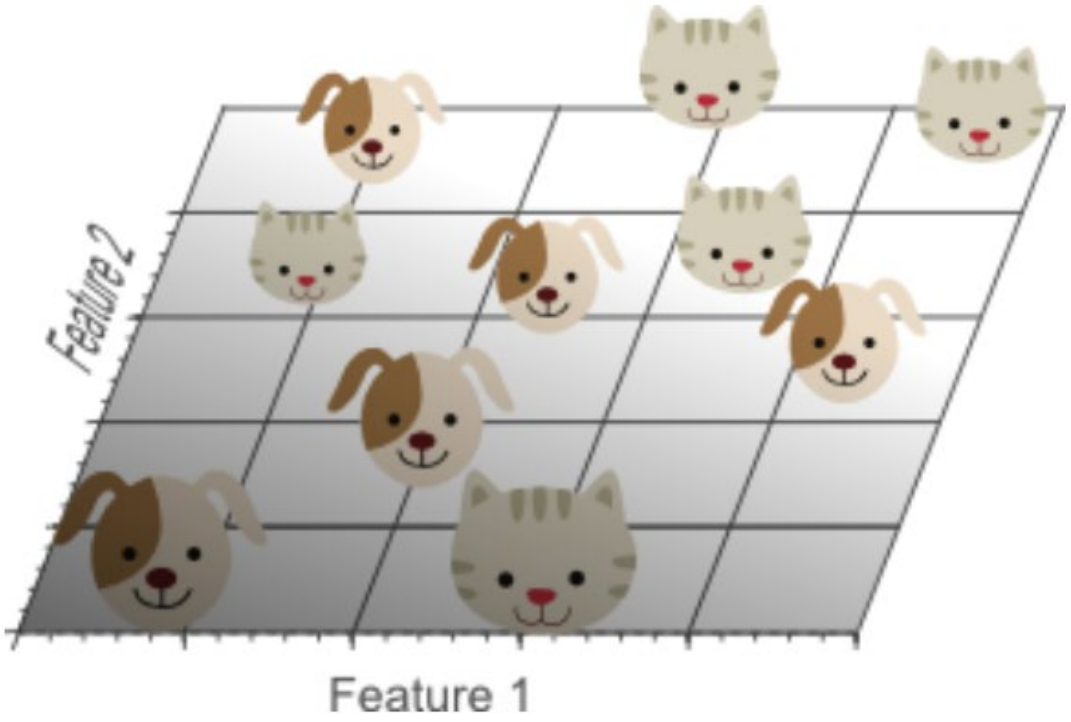
- Given 10 pictures of cats and dogs, we want to train a classifier



  - A single feature does not result in a perfect separation

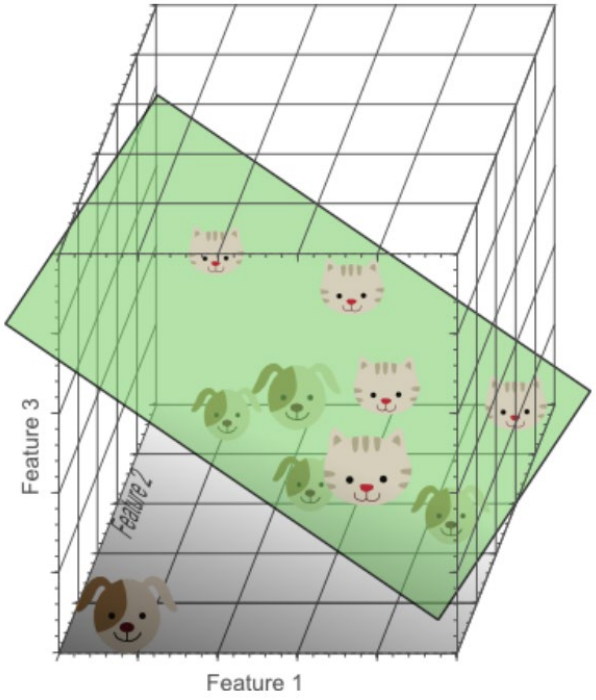
Curse of Dimensionality

- Adding a second feature still does not result in a linearly separable classification problem: No single line can separate all cats from all dogs in this example



Curse of Dimensionality

- Adding a third feature results in a linearly separable classification problem in our example



Curse of Dimensionality

- It might seem to suggest that increasing the number of features is the best way to train a classifier.

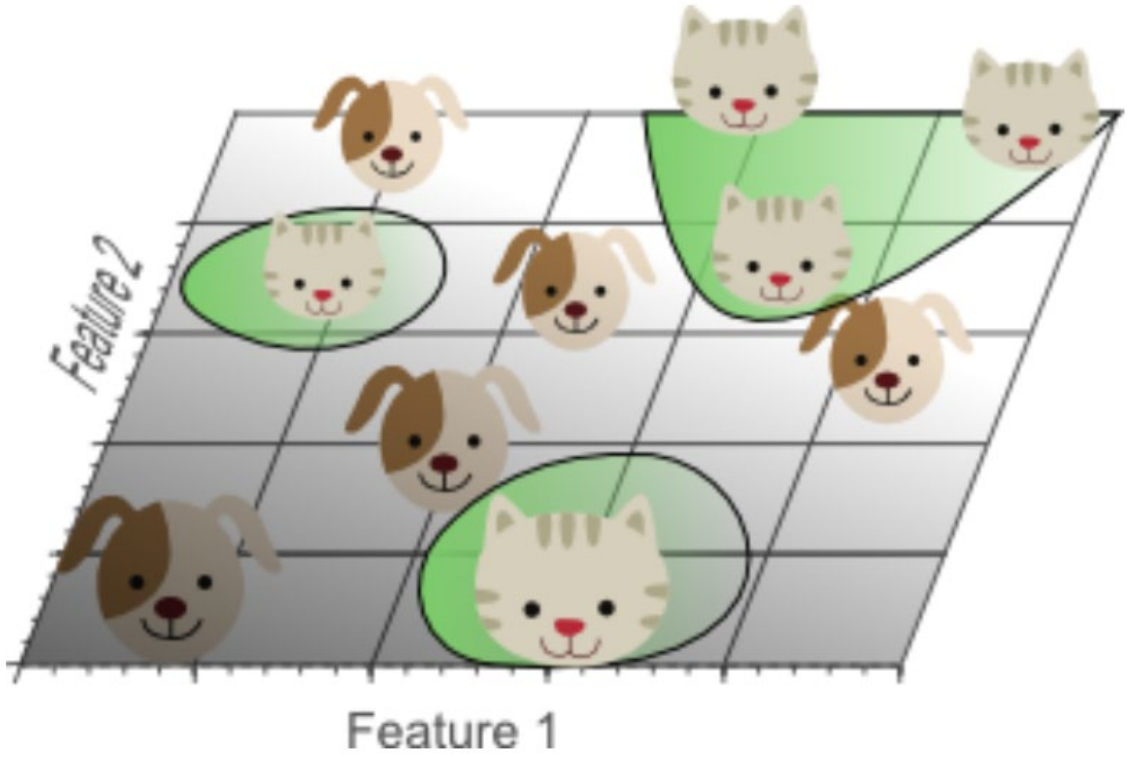
- However, the density of the training samples decreased exponentially.

Curse of Dimensionality

- In the 1D case, the width is 5 unit intervals, the sample density was 10/5=2 samples/interval
- In the 2D case, 10 training instances cover an area of 5×5=25 unit squares, the sample density is 10/25 = 0.4 samples/interval
- In the 3D case, the 10 samples have to cover a feature space volume of 5x5x5=125 unit cubes, the sample density is 10/125 = 0.08 samples/interval

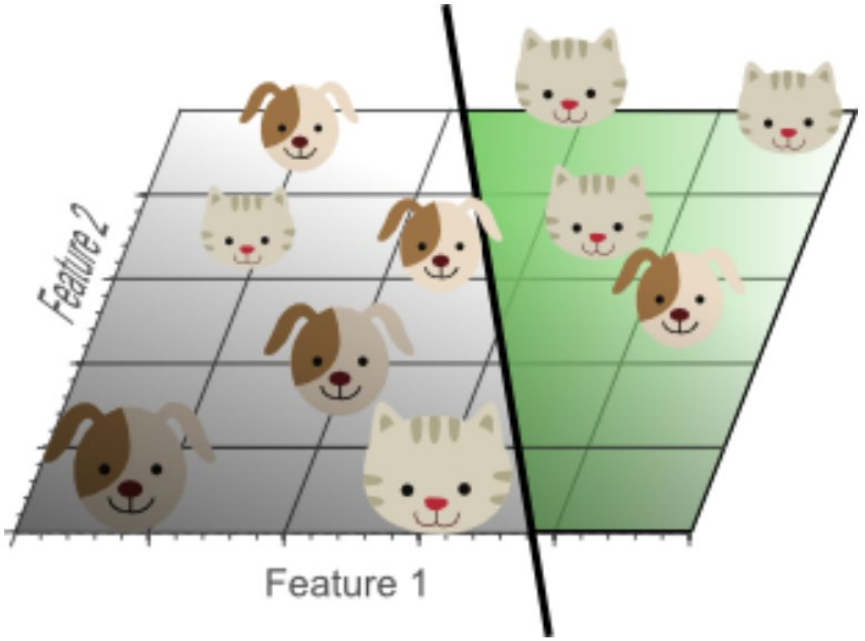
Curse of Dimensionality

- Using too many features results in overfitting. The classifier starts learning exceptions that are specific to the training data and do not generalize well when new data is encountered.



Curse of Dimensionality

- This concept is called overfitting and is a direct result of the curse of dimensionality.
- Although the training data is not classified perfectly, this classifier achieves better results on unseen data



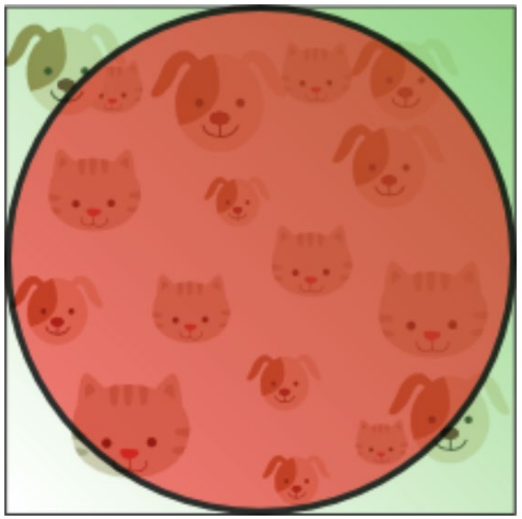
Curse of Dimensionality

- The curse of dimensionality introduces sparseness of the training data.

- This sparseness is not uniformly distributed over the search space.

The curse of dimensionality

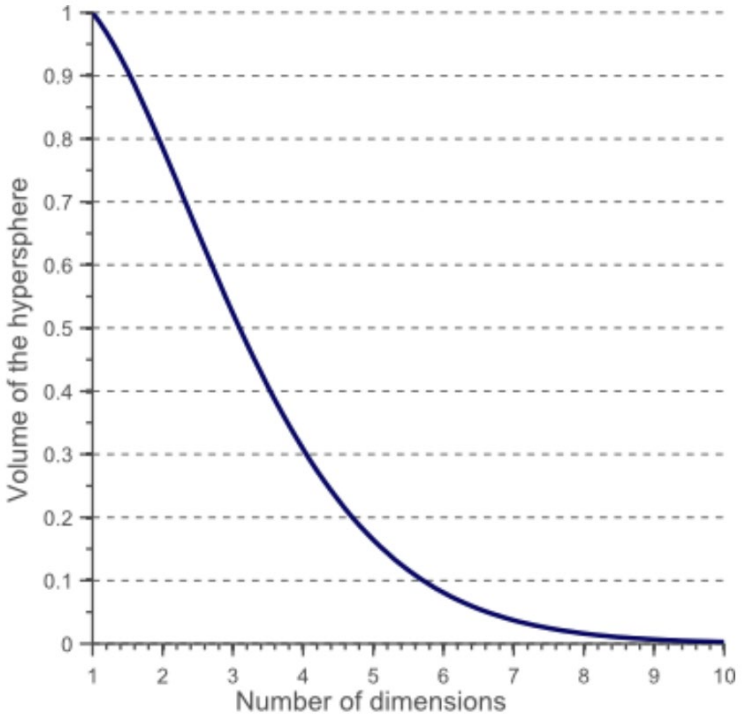
- Imagine a unit square that represents the 2D feature space. The average of the feature space is the center of the unit square.
- Training samples that fall outside the unit circle are in the corners of the feature space and are more difficult to classify than samples near the center of the feature space.



Curse of Dimensionality

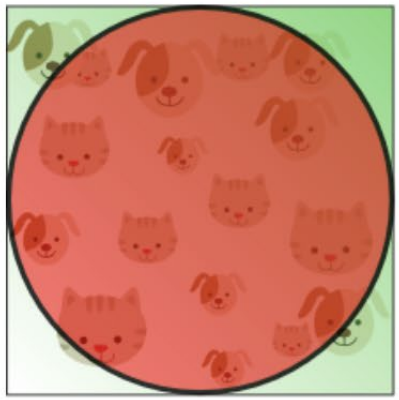
- The volume of the hypersphere tends to zero as the dimensionality increases.

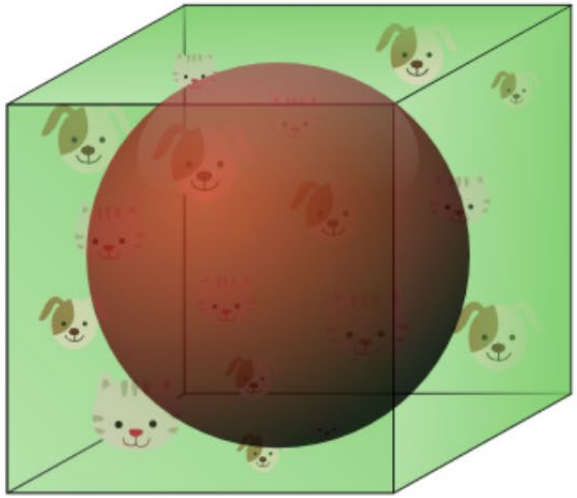
$$\lim_{d \to +\infty} \frac{V_{hypersphere}}{V_{hypercube}} = 0$$



Curse of Dimensionality

$$\lim_{d \to \infty} \frac{\text{dist}_{\text{max}} - \text{dist}_{\text{min}}}{\text{dist}_{\text{min}}} \to 0$$





The curse of dimensionality

- How to avoid the curse of dimensionality?
  - what 'too large' means?
  - how overfitting can be avoided?

- There is no fixed rule that defines how many feature should be used in a classification problem. In fact, this depends on:
  - the amount of training data available
  - the complexity of the decision boundaries
  - the type of classifier used

Curse of Dimensionality

- The smaller the size of the training data, the less features should be used.

- Classifiers that tend to ***model non-linear decision boundaries*** very accurately (e.g. neural networks, KNN classifiers, decision trees) do not generalize well and are **prone to overfitting**.
  - If a classifier is used that ***generalizes easily*** (e.g. naive Bayesian, linear classifier), then **the number of used features can be higher** since the classifier itself is less expressive

Curse of Dimensionality

- The variance of a parameter estimate increases if the **number of parameters** to be estimated increases (and if the bias of the estimate and the amount of training data are kept constant). This means that the quality of our parameter estimates decreases if the dimensionality goes up, due to the increase of variance. **An increase of classifier variance corresponds to overfitting.**

Curse of Dimensionality

- Another interesting question is which features should be used.
  - Given a set of N features; how do we select an optimal subset of M features such that M<N?
  - Another approach would be to replace the set of N features by a set of M features, each of which is a combination of the original feature values。
    - PCA

Curse of Dimensionality

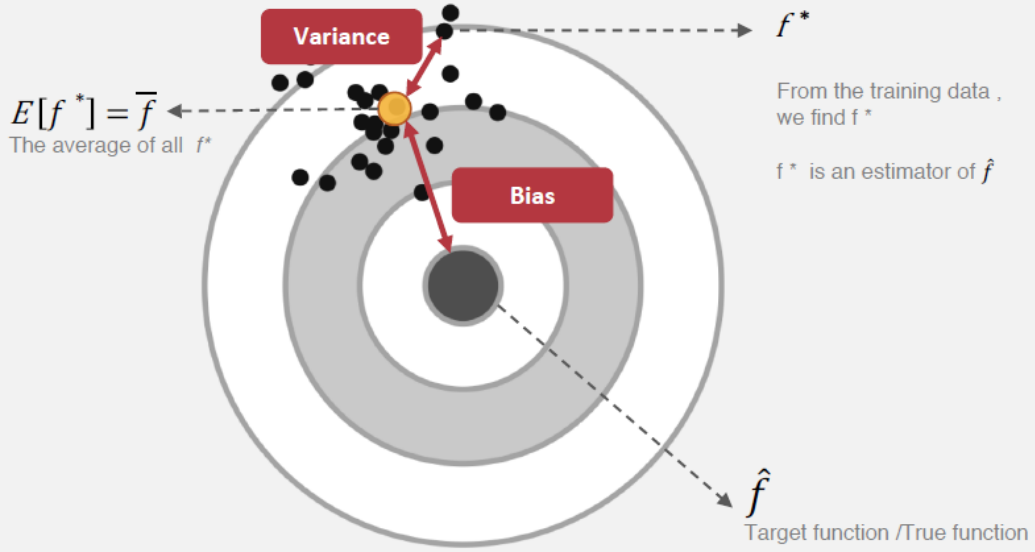
- An invaluable technique used to detect and avoid overfitting during classifier training is cross-validation.

KNN hyperparameter tuning

What is the best **distance** to use?
What is the best value of **k** to use?

i.e. how do we set the **hyperparameters**?

Bias and Variance



- **The variance** of the learning method represents how much the learning method will move around its mean; The variance is error from sensitivity to small fluctuations in the training set[3].
- **The Bias** of learning method represents the difference between the expected (or average) prediction of our model and the correct value which we are trying to predict.

Expected predicted error = **Bias**^{2} + **Variance** + Irreducible error

The prediction errors can be decomposed into two main subcomponents we care about:

- error due to "bias"
- error due to "variance" [1]

Irreducible error, is the noise term that cannot fundamentally be reduced by any model. Given the true model and infinite data to calibrate it, we should be able to reduce both the bias and variance terms to 0. However, in a world with imperfect models and finite data, there is a tradeoff between minimizing the bias and minimizing the variance[2].

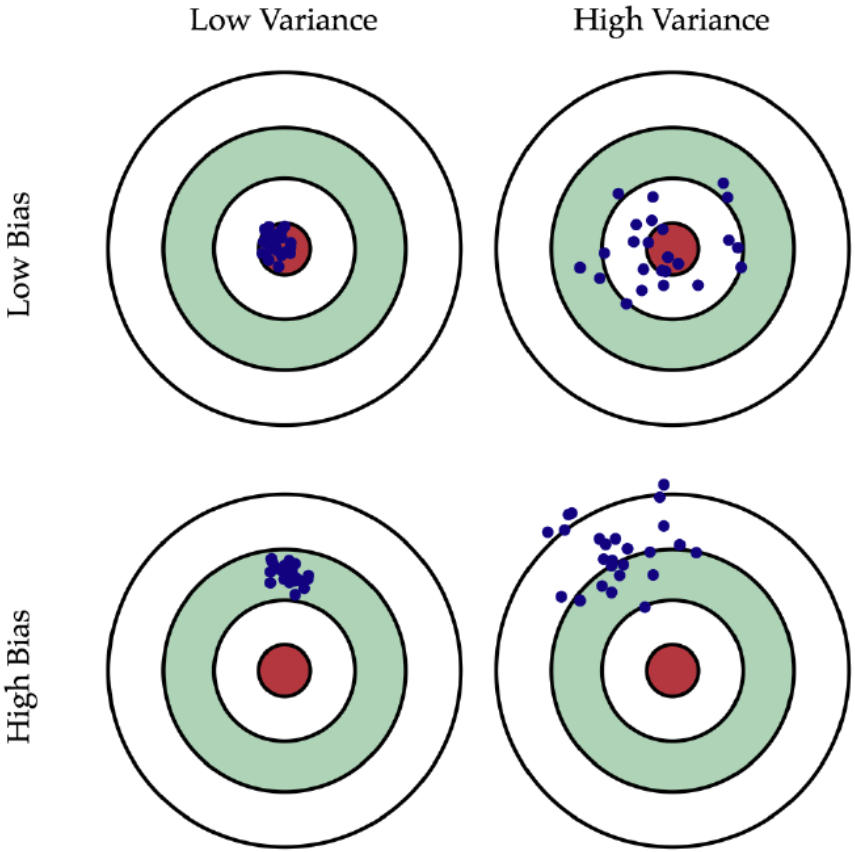
Bias and Variance

- Bias are the simplifying assumptions made by a model to make the target function easier to learn.
  - **Low Bias**: Suggests less assumptions about the form of the target function.
  - **High-Bias**: Suggests more assumptions about the form of the target function.

Bias and Variance

- Variance is the amount that the estimate of the target function will change if different training data was used.
  - **Low Variance**: Suggests small changes to the estimate of the target function with changes to the training dataset.
  - **High Variance**: Suggests large changes to the estimate of the target function with changes to the training dataset.

Bias and Variance



There are 4 cases representing combinations of both high and low bias and variance.

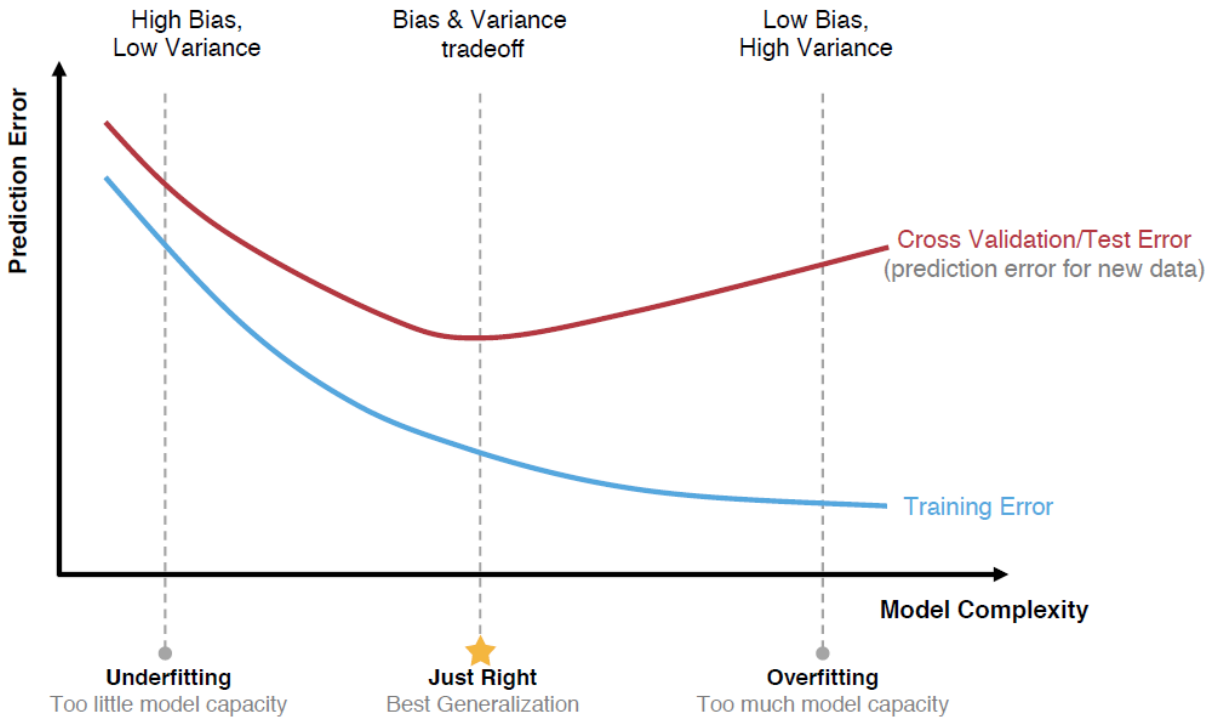
- Low Bias & Low Variance → Good case
- Low Bias & High Variance
- High Bias & Low Variance
- High Bias & High Variance → Bad case

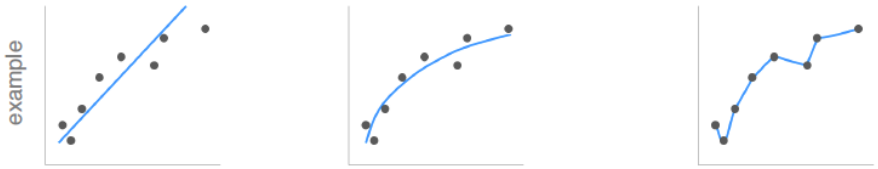
Ideally, one wants to choose a model that both accurately captures the regularities in its training data, but also **generalize**s well to unseen data (i.e. new data). Unfortunately, it is typically impossible to do both simultaneously[1].

Model Selection

- There is usually a trade-off between Bias and Variance. So normally reducing one tends to increase the other
- Select a model that balances two kinds of error to minimize the total error

Bias-Variance Trade-Off





Underfitting = Bias problem

Overfitting = Variance problem

The simplest way to determine if your model is suffering more from bias or from variance is the following rule of thumb:

- If your model is performing really well on the training set, but much poorer on the hold-out set, then it's suffering from high variance[1].
- On the other hand if your model is performing poorly on both training and test data sets, it is suffering from high bias[2].

Bias-Variance Trade-Off

- The general trend :
  - **Linear** machine learning algorithms often have a high bias but a low variance.
  - **Nonlinear** machine learning algorithms often have a low bias but a high variance.
- The parameterization of machine learning algorithms is often a battle to balance out bias and variance.
  - Increasing the bias will decrease the variance.
  - Increasing the variance will decrease the bias.

Bias-Variance Trade-Off

- The k-nearest neighbors algorithm has low bias and high variance, but the ***trade-off can be changed by increasing the value of k*** which increases the number of neighbors that contribute the prediction and in turn increases the bias of the model.

Pros and Cons of Using KNN

- Pros:
  - Since the KNN algorithm requires no training before making predictions, new data can be added seamlessly, which will not impact the accuracy of the algorithm.
  - KNN is very easy to implement. There are only two parameters required to implement KNN—the value of K and the distance function (e.g. Euclidean, Manhattan, etc.)

Pros and Cons of Using KNN

- Cons:
  - The KNN algorithm does not work well with large datasets. The cost of calculating the distance between the new point and each existing point is huge, which degrades performance.
  - Feature scaling (standardization and normalization) is required before applying the KNN algorithm to any dataset. Otherwise, KNN may generate wrong predictions.