Machine Learning

Lecture 08

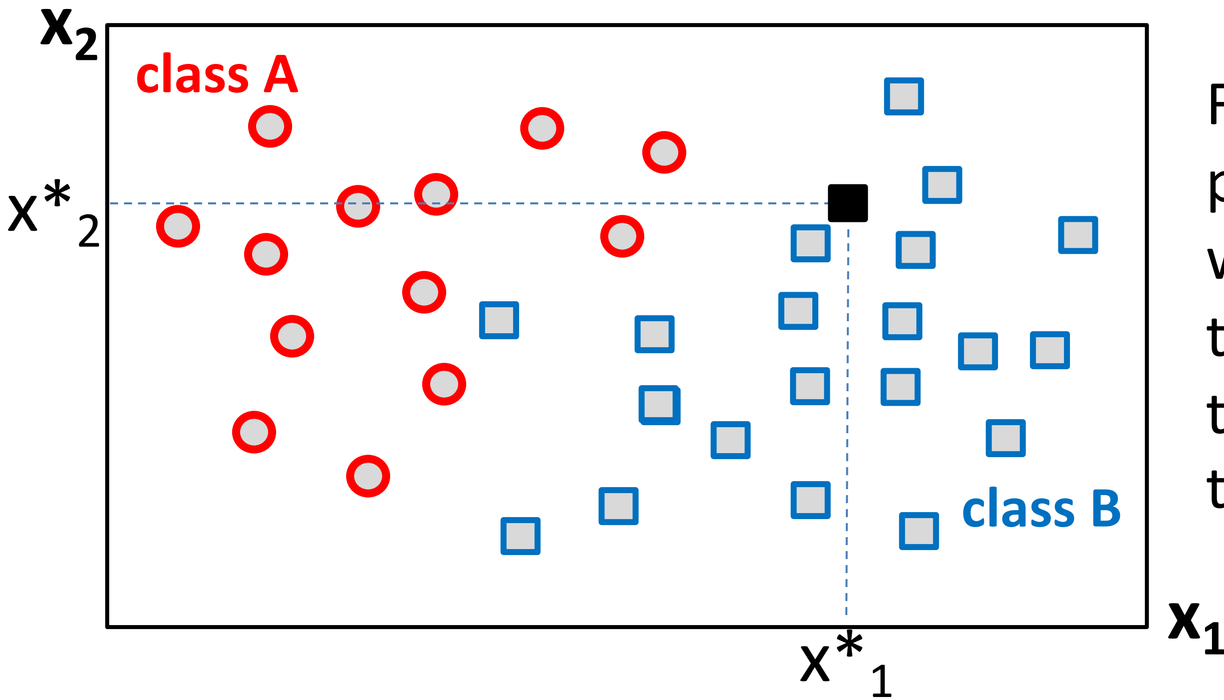
k Nearest Neighbors

Characteristics of kNN

- k-Nearest Neighbors (**kNN**) for predictive modeling
  - A simple, supervised, **instance-based** algorithm
  - **Non-linear** and **non-parametric**
  - A lazy learner (no training)
  - Very intuitive (high interpretability)
- Can handle:
  - Mixed data types (continuous and categorical, but careful there!)
  - Linearly inseparable data
  - Multi-class problems
- When to consider: Recommender systems, fault and intrusion detection, low-dimensional datasets, document categorization, target marketing, great for remote sensing data.

Intuition on kNN

- Which class does ■ belong to?

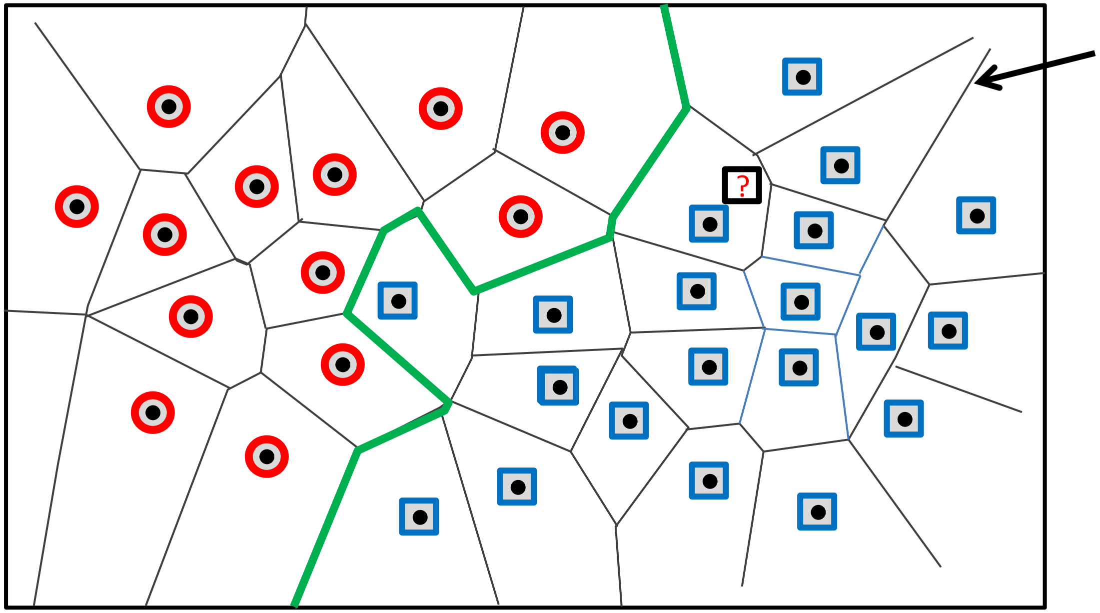


For a given set of points (x_1*,x_2*), which one of the two classes does this point belong to?

- Key point: **nearby points => same class**
- Based on the idea that closer two objects are in space, the more similar they are
- Need a good distance function to establish similarity

kNN Classification

- Voronoi Diagram



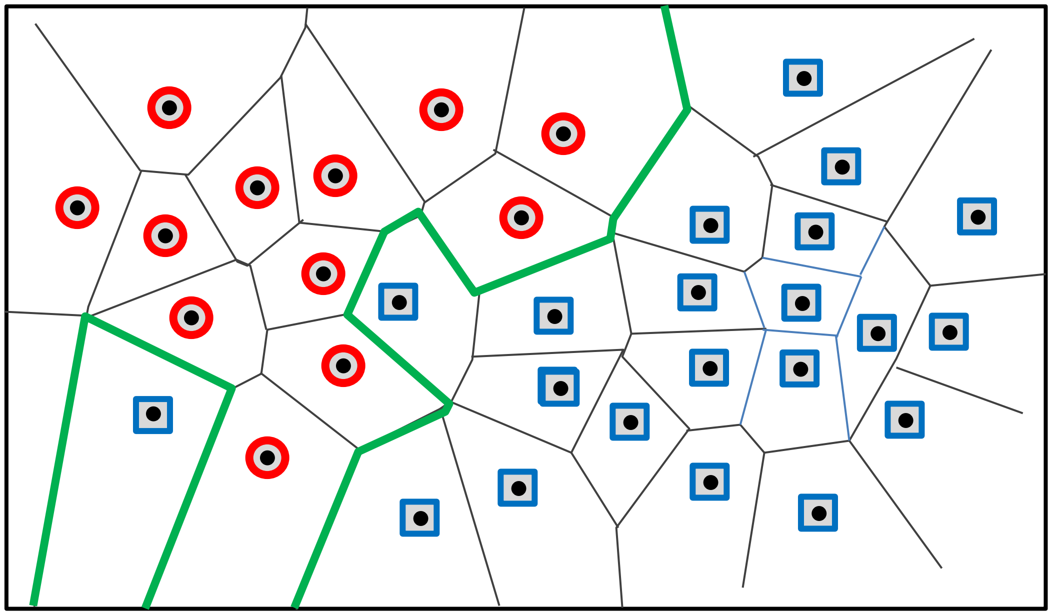
Boundary is located at the same distance from both neighboring data points

Decision Boundary

- Partitions the whole space into cells and finds the most similar training data point to □ and predicts class
- A complicated decision boundary (nonlinear—composed of broken straight lines) which separates the classes well

Sensitivity to outliers

- What happens when you change the class label of a single data point?



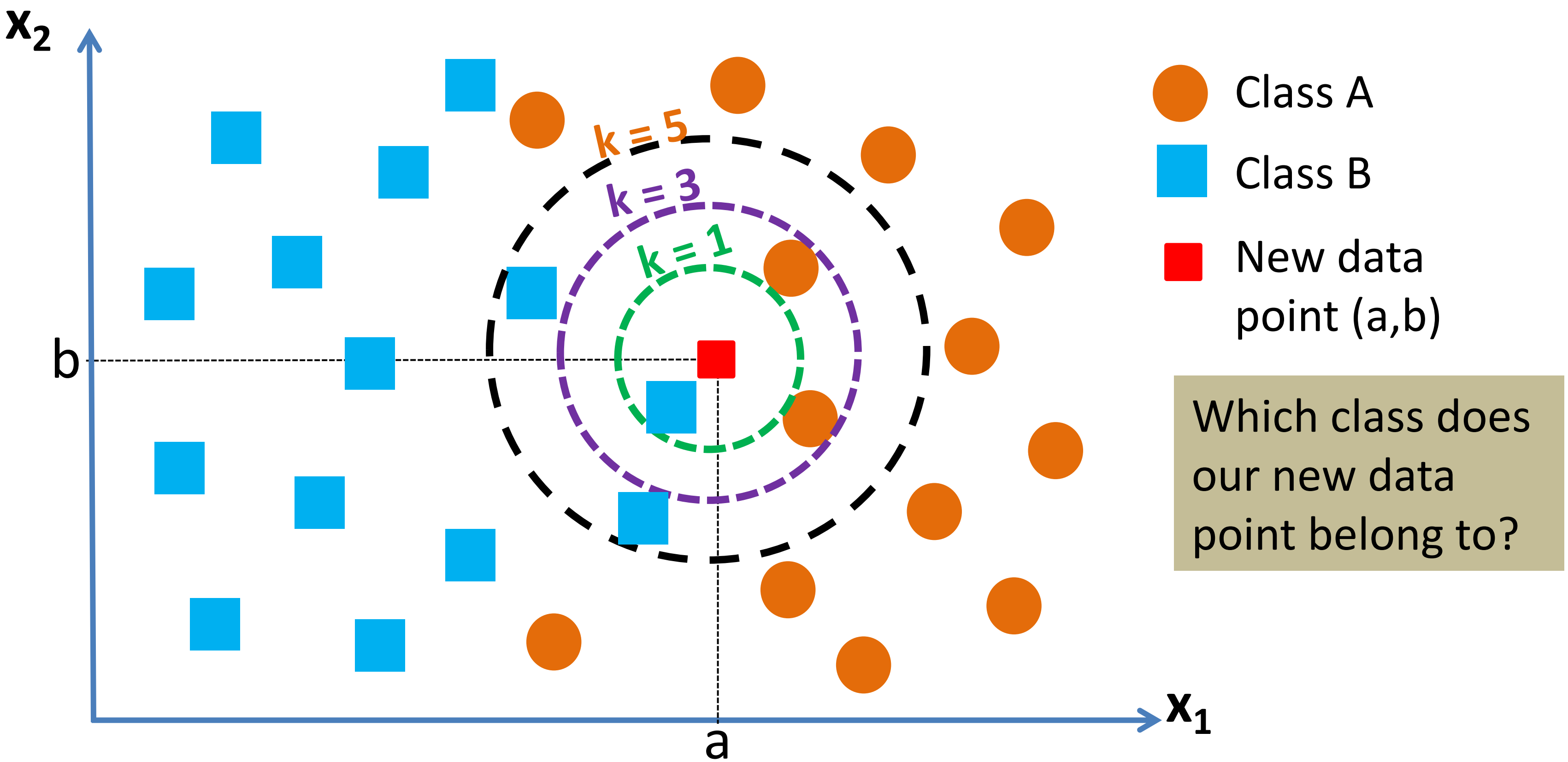
Creates a dramatic change in the decision boundary which is not desirable.

We don't want small changes in the data to have large effects on the prediction.

- Mislabeled training data have a compound effect on the class prediction. Why?
- No conditional probability in the form of p(y|x), i.e., it's not sensitive to class prior.
- Is there a way to make kNN sensitive to data counts?

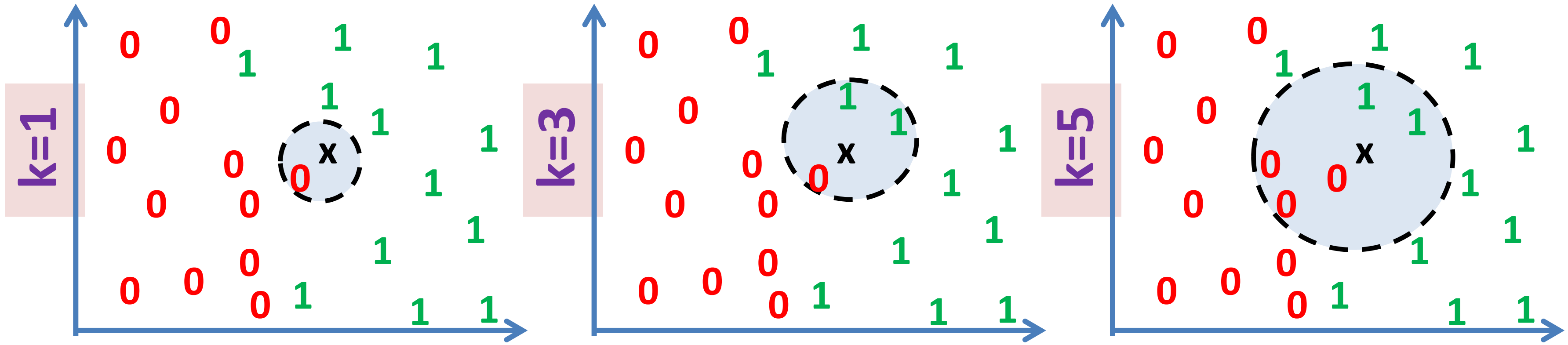
Nearest Neighbors algorithm

- How about using more than one nearest neighbor for class prediction?
- Spot the "**k**" most similar neighbors to the new data and count class labels
- Metric for quantifying similarity : "**distance**"



How many neighbors?

x : New data point



Predicted class is 0 (No) | Predicted class is 1 (Yes) | Predicted class is 0 (No)

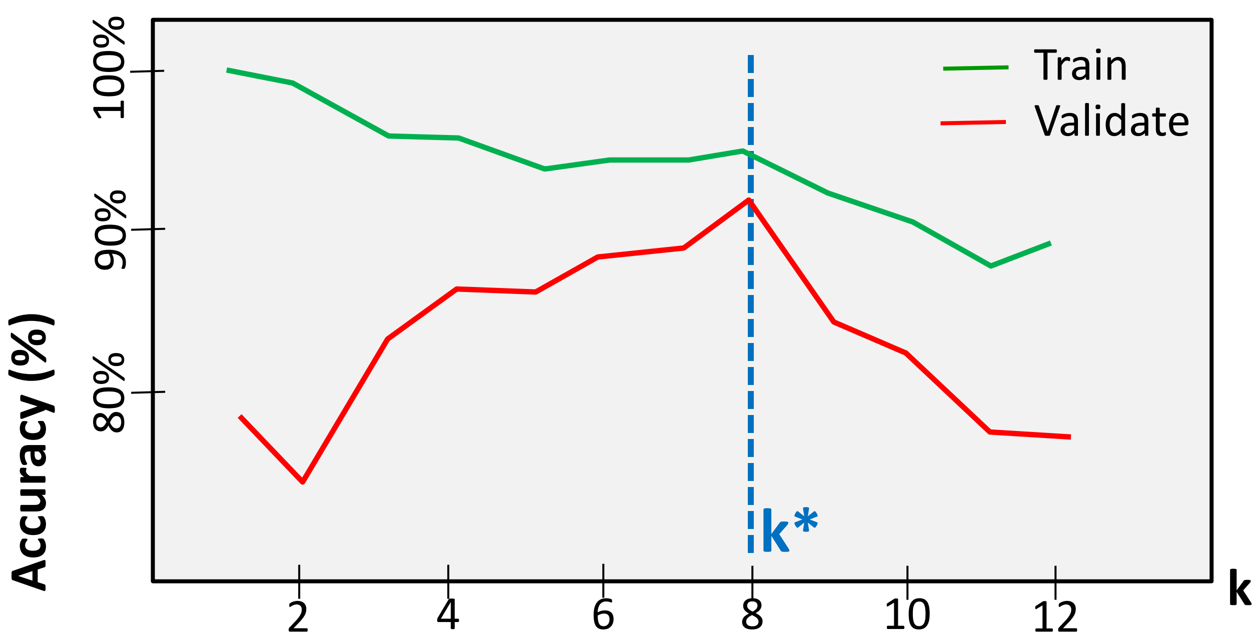
k is an odd value for no ties (if binary class)

As **k** gets larger (larger neighborhood) ➔ New data point gets classified as the most probable class (large bias, inaccurate estimation)

As **k** gets smaller (smaller neighborhood) ➔ Sensitive to noise and highly variable (unreliable estimation as a consequence of overfitting)

A probabilistic estimate could be given based on the classes of the neighbors if you want to assign a score to the prediction (other than a simple Yes or No)

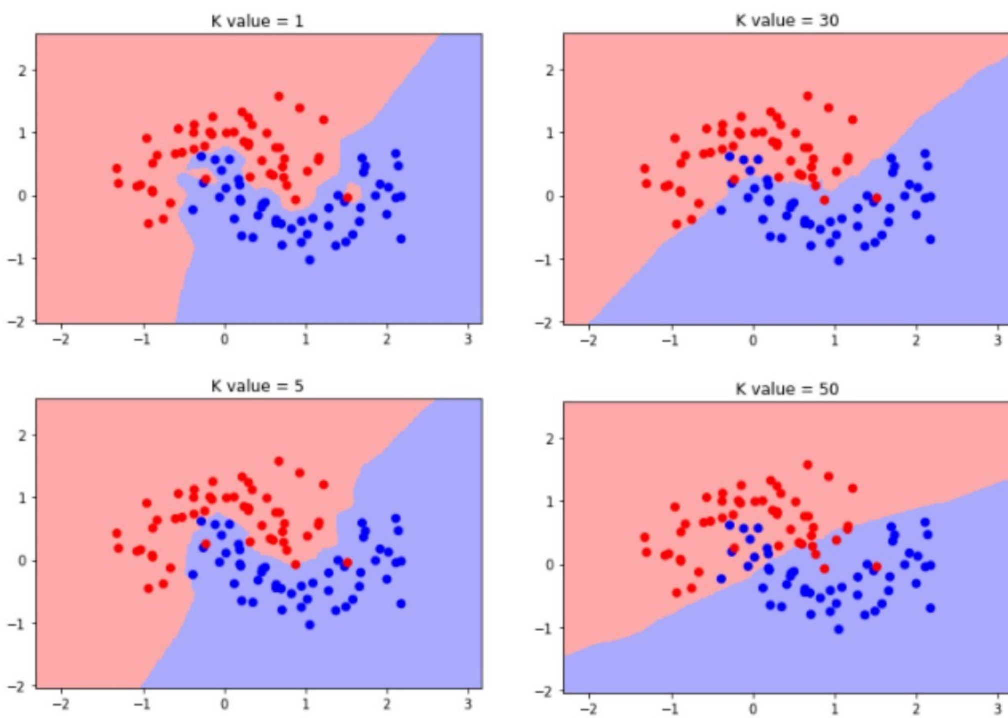
- **Methodology**:
  - Split the whole data into train and validation sets
  - Select a range of "**k**" for the number of neighbors
  - For each "**k**", do a cross validation and compute the error (or accuracy) for both train and validation sets
  - Plot the train-validate error over the range of **k**'s



Pick **k*** which gives the lowest validation error (for the best generalization performance)

How many neighbors? – cont’d

- How does k affect the performance of the model?
  - The bigger the k, the smoother the decision boundary. If the difference in the CV errors is negligible for increasing k, a larger value of k may be chosen if computational expense isn't an issue.
  - If the CV error doesn't start to rise again, it could mean that the features are not informative and a constant output is the best it can do.

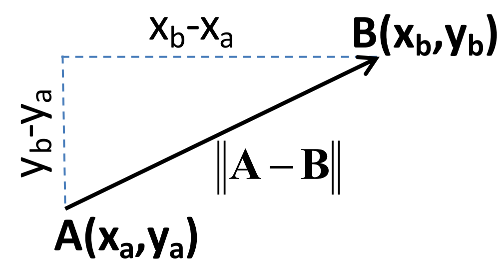


Large k will over-smooth ignoring local structure

Influence of neighbors on prediction

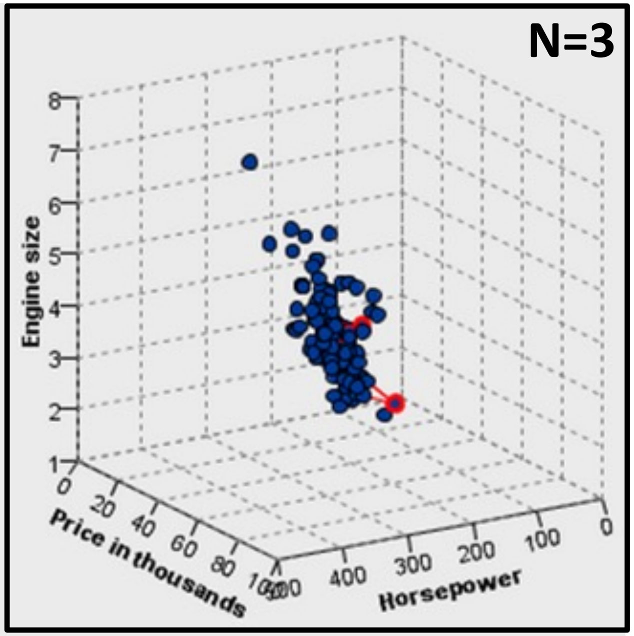
- **Distance between neighbors?**
- A key component of kNN algorithm
  - A metric for measuring how similar instances are
  - Has a strong impact on classification performance
- **k** neighbors that are closest to the new data point "**x**" are determined by a distance measure
- Measures of distances used in kNN:
- **Euclidean distance**
  - Euclidean (L2 norm) distance in 2D (valid for numerical variables)

$$\boxed{d(\mathbf{A},\mathbf{B}) = \sqrt{(x_b - x_a)^2 + (y_b - y_a)^2}}$$



Influence of neighbors on prediction

Euclidean distance in higher dimensions



Brand	Engine size	Horsepower	Price
Ford Torino	302	140	15000
AMC rebel	304	150	16500
Buick skylark	350	165	19000

$$d(\text{ford,amc}) = \sqrt{(302-304)^2 + (140-150)^2 + (15000-16500)^2}$$
$$= 1500$$

$$d(\text{ford,buick}) = \sqrt{(302-350)^2 + (140-165)^2 + (15000-19000)^2}$$
$$= 4000.4$$

Source: http://www.datasciencecentral.com/profiles/blogs/introduction-to-the-k-nearest-neighbor-knn-algorithm

- Euclidean distance in "N" dimensions:

$$d(\mathbf{A},\mathbf{B}) = \sqrt{(d_{1,A}-d_{1,B})^2 + (d_{2,A}-d_{2,A})^2 + \ldots + (d_{N,A}-d_{N,A})^2}$$

$$= \sqrt{\sum_{i=1}^{N}(d_{i,A}-d_{i,A})^2}$$ (valid for continuous variables)

Influence of neighbors on prediction

- Once we find **x**'s nearest neighbors using the distance function, neighbors vote to predict **x**'s class:
- Method 1: **Majority voting**
  - All votes are equal and majority (democracy) wins
- Method 2: **Weighted voting**
  - Closer neighbors get higher votes (shareholder democracy)
    1. Distance-based weight: A neighbor's vote is the inverse of its distance to **x**:

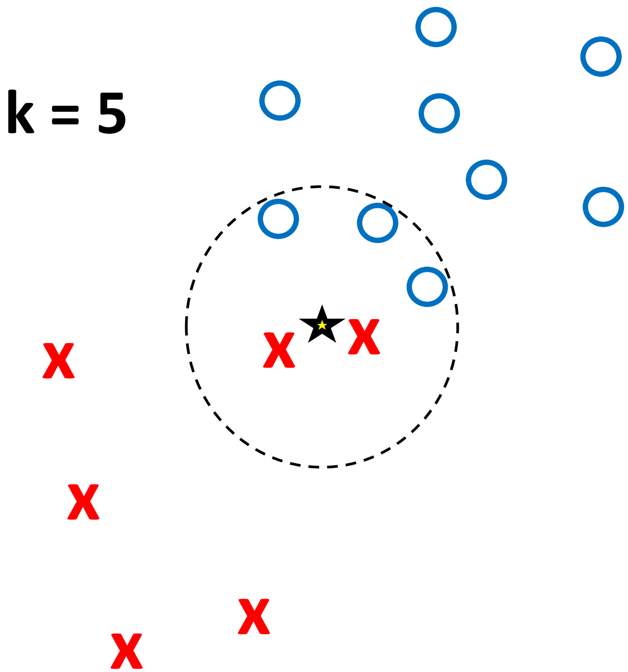
$$\boxed{vote(\mathbf{x}_i) = 1/d(\mathbf{x}_i, \mathbf{x})}$$

    2. Heuristic: Based on domain-specific characteristics of neighbors

Influence of neighbors on prediction

- **Majority Voting** vs **Weighted Voting** mechanisms

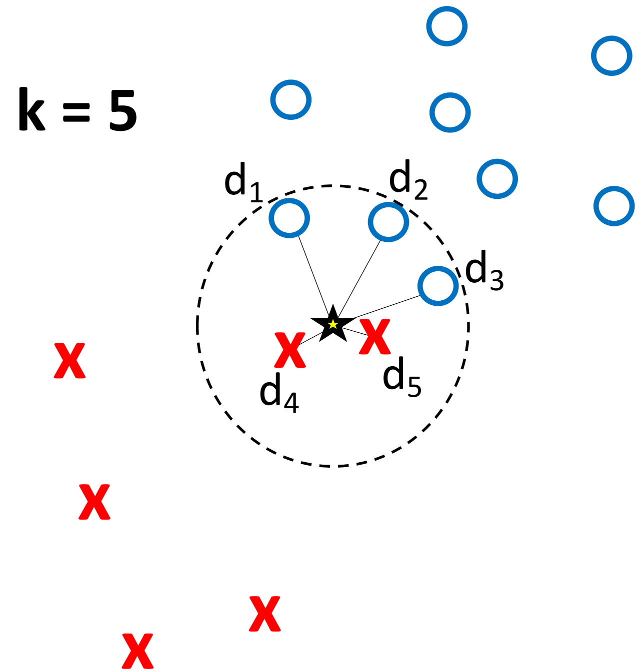
Majority-voting
(**uniform** in sklearn)



O : 3
X : 2

Prediction: O

Weighted-voting
(**distance** in sklearn)



$$O : \frac{1}{d_1} + \frac{1}{d_2} + \frac{1}{d_3}$$

$$X : \frac{1}{d_4} + \frac{1}{d_5}$$

Prediction: X

Other measures of distance

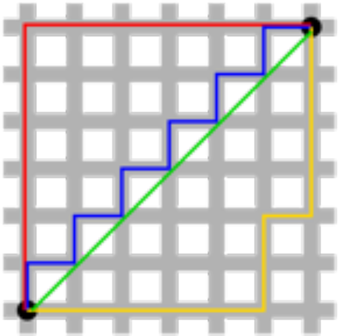
- **Minkowski distance** (m-norm)

$$d(\mathbf{A},\mathbf{B}) = \sqrt[p]{|d_{1,A}-d_{1,B}|^p + |d_{2,A}-d_{2,B}|^p + \ldots + |d_{N,A}-d_{N,B}|^p}$$

$$= \sqrt[p]{\sum_d |d_{i,A}-d_{i,B}|^p} = \left(\sum_d |d_{i,A}-d_{i,B}|^p\right)^{1/p}$$

- If p=0 => **Hamming distance** (later)
- If p=1 => **Manhattan distance** (L1 norm)
  - Suitable for grid-like paths such as maps (usually preferred in the presence of many features – high dimensional data)

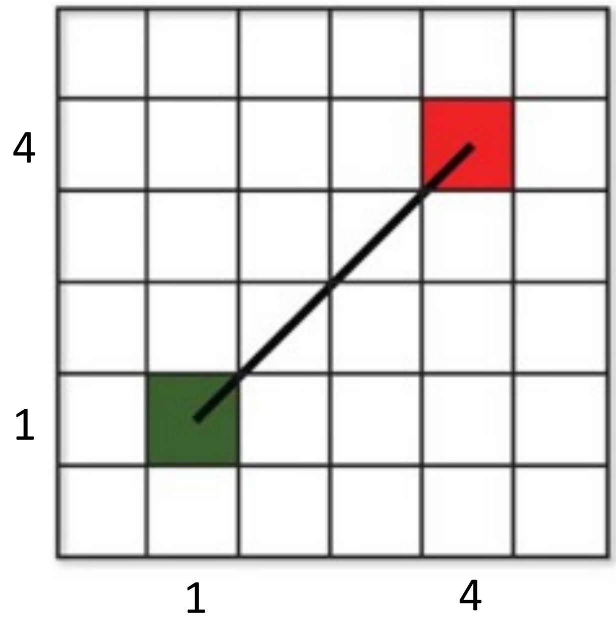
$$d(\mathbf{A},\mathbf{B}) = \sum_d |d_{i,A}-d_{i,B}|$$



Taxicab or city block distance

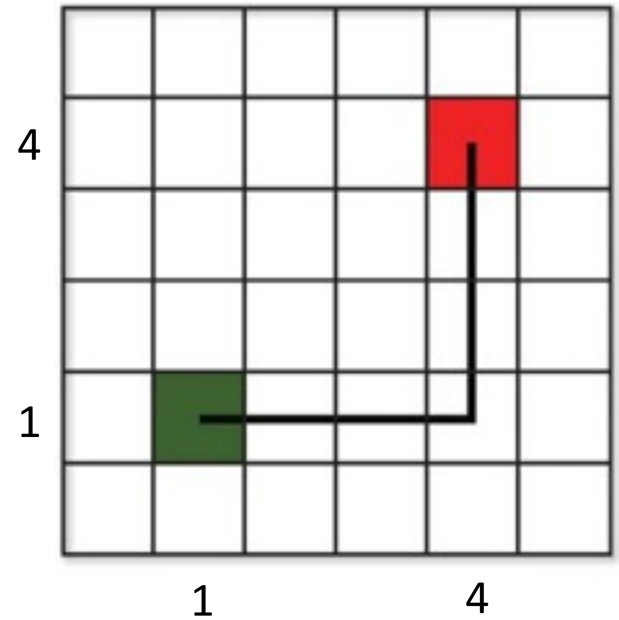
- If p=2 => **Euclidean distance** (L2 norm)

- **Euclidean vs Minkowski**



Euclidean distance between A and B: Straight line distance (most commonly used metric – default one)

$$d(\mathbf{A},\mathbf{B}) = \sqrt{(4-1)^2 + (4-1)^2} = 4.24$$

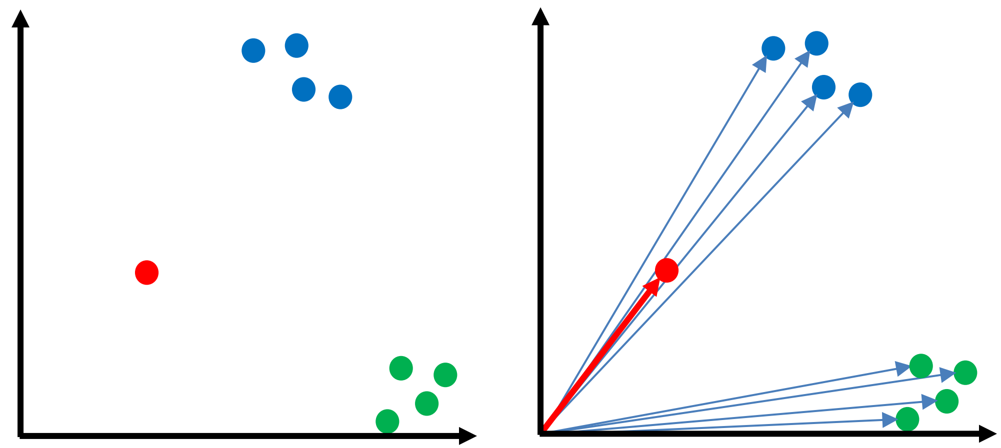


Manhattan distance between A and B: The distance between two points is the sum of the absolute differences of their Cartesian coordinates, i.e.,

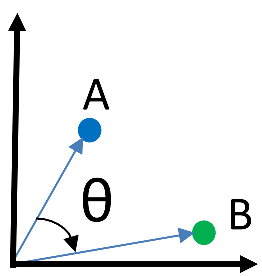
$$d(\mathbf{A},\mathbf{B}) = \sum_{i=1}^{2} |x_i - y_i| = |4-1| + |4-1| = 6$$

Other measures of distance – cont'd

- **Distance based on Cosine Similarity**
  - A class distribution as given below will not lead to a good separation between the classes using Euclidian distance. Instead, we can use a metric based on Cosine similarity (measuring the angle between the class vectors)



  - So a Cosine similarity between two data points is given as:



$$\cos(\theta) = \frac{\mathrm{A.B}}{\| A \| \, \| B \|} = \frac{\sum A_i B_i}{\sqrt{\sum A_i^2}\,\sqrt{\sum B_i^2}}$$

Cos distance = 1 – Cos(θ)

The closer the class vectors are by angle, the higher is the Cosine Similarity (Cos theta), thus smaller the distance.

Other measures of distance – cont'd

- Cosine similarity is very useful when we are interested in the orientation but not the magnitude of the vectors.
- Frequently used in text analytics where the similarity between documents is investigated.

	Example:	I	think	therefore	am	Can	you	don't	know	who
1	I think, therefore I am	2	1	1	1	0	0	0	0	0
2	Can you think?	0	1	0	0	1	1	0	0	0
3	I don't think, therefore I don't know who I am	3	1	1	1	0	0	2	1	1

$$d_{12} = 1 - \frac{1}{\sqrt{2^2+1+1+1}\sqrt{1+1+1}} = 0.782$$

$$d_{13} = 1 - \frac{6+1+1+1}{\sqrt{2^2+1+1+1}\sqrt{3^2+1+1+1+2^2+1+1}} = 0.198$$

Sentence 1 and 3 are closest.

$$d_{23} = 1 - \frac{1}{\sqrt{1+1+1}\sqrt{3^2+1+1+1+2^2+1+1}} = 0.864$$

Other measures of distance – cont'd

- What if the features are categorical?
- Categorical attributes
  - Univalent: Single level (an object could be black or white but not both)
    - Nominal variables
    - Ordinal variables
  - Multivalent: A specific object may contain multiple levels (a movie genre could be both Action and Comedy at the same time)
- There are several distance functions measuring dissimilarity for categorical attributes.

Distance for categorical attributes

- **Nominal attributes**
  - "**Hamming distance**" for binary variables (p=0):
  - Example: {male, female}, {green, blue, red}

$$d_H(\mathbf{x},\mathbf{y}) = \sum_{k=1}^{N} |x_k - y_k| \begin{cases} x = y \Rightarrow D = 0 \\ x \neq y \Rightarrow D = 1 \end{cases}$$

Person1	Person2	Distance
Male	Female	1
Female	Female	0

  - **Method I (Simple Matching)**: (A more general formula including multi-class for number of categories > 2):

$$\boxed{d(i,j) = \frac{p-m}{p} = \frac{\#\ \text{of mismatches}}{\text{total}\ \#\ \text{of variables}}}$$

m: Number of matches

d(male, female)=?

(1,0) <=> (0,1) => zero matches : m = 0, p = 2 => d(male, female) = 1

  - **Method II**: If many levels in an attribute, create a new binary attribute for each of the M nominal states

Distance for categorical attributes – cont'd

- **Ordinal attributes**
  - An ordinal variable can be discrete or continuous
  - Order (rank) is important: {small, medium, large}
  - Can be treated like interval-scaled
    - Replace an ordinal variable by its rank $r_{ik} \in \{1, 2, \ldots M_k\}$
    - Map the range of each variable onto [0,1] by replacing the ith object in kth variable by:

$$\boxed{z_{ik} = \frac{r_{ik} - 1}{M_k - 1}}$$

Data set = {cold , cool , warm , hot }

cold = (1-1)/(4-1) = 0 cool = (2-1)/(4-1) = 1/3

warm = (3-1)/3 = 2/3 hot = (4-1)/3 = 1

Example: {cold: 0, cool: 1/3, warm: 2/3, hot: 1}

- Distance between warm and hot: 1 - 2/3 = **1/3**
- Distance between cold and warm: 2/3 – 0 = **2/3**

Distance for multivalent categorical attributes

- **Jaccard Similarity**
- Instead of calculating distances between vectors, we will work with sets (unordered collection of objects).
- Jaccard similarity is not a distance, 1-JS is a distance. Can be used for multivalent categorical variables.
- Let's say we have two sets of objects: A and B.
  - Elements common to A and B (intersection): $A \cap B$
  - All elements in A and B (union): $A \cup B$.
- So the Jaccard Similarity (JS) is given by: **$JS = |A \cap B| / |A \cup B|$**
- Example:

Movie1 (A)	Movie2 (B)	$A \cap B$	$A \cup B$	distance 1-JS
Comedy	Comedy, Action	1	2	½
Comedy, Action	Comedy, Action	2	2	0
Comedy, Action	Comedy, Drama	1	3	2/3
Comedy, Action	Drama, Non-fiction	0	4	1

- Euclidean distance between animals




Target

Animal	Egg-laying	Scales	Poisonous	Slender	Legs	Reptile?
Rattlesnake	True	True	True	True	0	Yes
Boa constrictor	False	True	False	True	0	Yes
Dart Frog	True	False	True	False	4	No
Alligator	True	True	False	True	4	Yes

Feature vectors for data instances (animals):

```
Rattlesnake     = [1, 1, 1, 1, 0]
Boa constrictor = [0, 1, 0, 1, 0]
Dart Frog       = [1, 0 ,1, 0, 4]
Alligator       = [1, 1, 0, 1, 4]
```

Ref: Example taken from MIT Opencourseware – 6.0002 Introduction to ML

- Euclidean distance between animals

```
data = [ [1,1,1,1,0], [0,1,0,1,0], [1,0,1,0,4], [1,1,0,1,4] ]
cols = ['Rattlesnake', 'Boa Constrictor', 'DartFrog', 'Alligator']
df = pd.DataFrame(euclidean_distances(data, data), columns=cols, index=cols)
```

		Rattlesnake	Boa Constrictor	DartFrog	Alligator
[1, 1, 1, 1, 0]	**Rattlesnake**	0.000000	1.414214	4.242641	4.123106
[0, 1, 0, 1, 0]	**Boa Constrictor**	1.414214	0.000000	4.472136	4.123106
[1, 0 ,1, 0, 4]	**DartFrog**	4.242641	4.472136	0.000000	1.732051
[1, 1, 0, 1, 4]	**Alligator**	4.123106	4.123106	1.732051	0.000000

- Alligator closer to Dart Frog than to snakes. Why?
- Alligator differs from Frog in 3 features, from Boa in only 2
- But scale on "Legs" is from 0 to 4, on other features is 0 to 1
- So "Legs" dominate as its dimension is disproportionately large

Example I – cont'd

- What if "Legs" converted to "4-Legs?" and used as a binary feature?

	Egg-laying	Scales	Poisonous	Slender	Legs	4-Legs?
Rattlesnake	1	1	1	1	0	No
Boa Constrictor	0	1	0	1	0	No
DartFrog	1	0	1	0	1	Yes
Alligator	1	1	0	1	1	Yes

	Rattlesnake	Boa Constrictor	DartFrog	Alligator
Rattlesnake	0.000000	1.414214	1.732051	1.414214
Boa Constrictor	1.414214	0.000000	2.236068	1.414214
DartFrog	1.732051	2.236068	0.000000	1.732051
Alligator	1.414214	1.414214	1.732051	0.000000

- Now Alligator is closer to snakes than it is to Dart Frog, which makes more sense.
- This shows why Feature Engineering is an important concept.

Example II

Attribute	Person1	Person2	...	PersonN	
Gender	Male	Female		Male	➔ Binary
Residential status	rent	owner	...	other	➔ Multi-class (3)

- Distance between Person1-Person2, Person1-PersonN, etc?
- Assign a binary dummy variable to each value of "Residential status" and compute the average distances:

Person1 = [1,(1,0,0)]

Person2 = [0,(0,1,0)]

PersonN = [1,(0,0,1)]

simple average: a weight of 1 for each feature

Distance: Person[1-2] = [1 , 2/3] = (1+2/3)/2 = 5/6

Person[1-N] = [0 , 2/3] = (0+2/3)/2 = 1/3

Person[2-N] = [1 , 2/3] = (1+2/3)/2 = 5/6

distance between two genders

$d_{i,j} = (p - m) / p$

m: # of matches

p: # of variables

- **Heterogeneous attributes and differences in scale**

Attribute	Person1	Person2	. . .	PersonN
Gender	Male	Female		Male
Age	29	47		38
Residential status	rent	owner		other
Salary	36,000	140,000		90,000

2

1

- Issues:

  1 Numerical variables with very different scales
    - Need to apply scaling (normalization)
    - Without scaling, our distance function will treat 10 TL difference in salary as significant as 10 yrs difference in age

  2 Distance issue with attributes of mixed nature
    - For distance computations, "gender" and "age" variables must be combined numerically.

      So how do we do this?

Issues with kNN: Scaling

- **Data Scaling**
- As the distance between data points is essential in kNN algorithm, attribute scaling is critical.
- Min-Max scaling [0-1 range]: $x_s = \dfrac{x - x_{\min}}{x_{\max} - x_{\min}}$

ID	Gender	Age	Salary
1	F	27	19,000
2	M	51	64,000
3	M	52	100,000
4	F	33	55,000
5	M	45	45,000

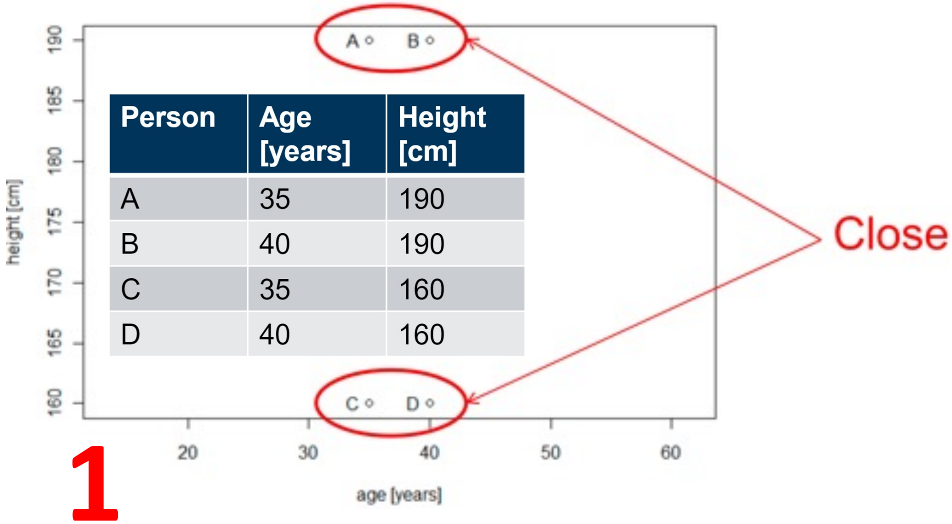
ID	Gender	Age	Salary
1	1	0.00	0.00
2	0	0.96	0.56
3	0	1.00	1.00
4	1	0.24	0.44
5	0	0.72	0.32

- Normalizing (standardization) data:

$$x_s = \frac{x - \bar{x}}{\sigma}$$ (0 mean, unit variance)

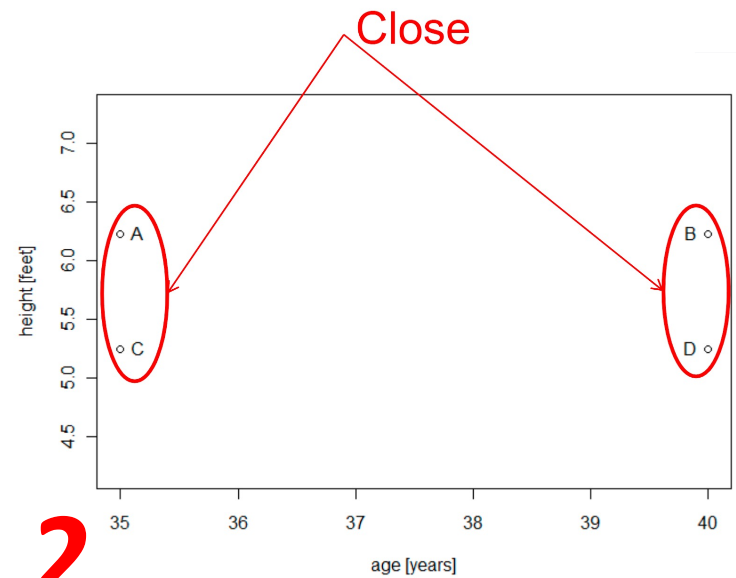
Issues with kNN: Scaling

- **Why the need for scaling?**



Person	Age [years]	Height [cm]
A	35	190
B	40	190
C	35	160
D	40	160

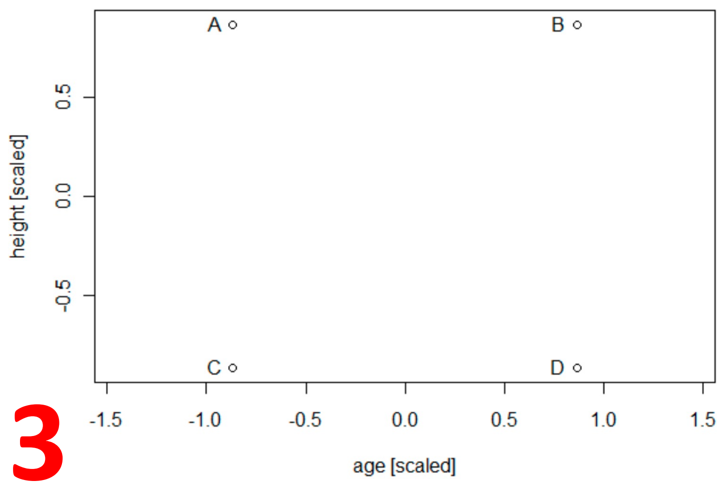
1



2

Person	Age [years]	Height [feet]
A	35	6.232
B	40	6.232
C	35	5.248
D	40	5.248

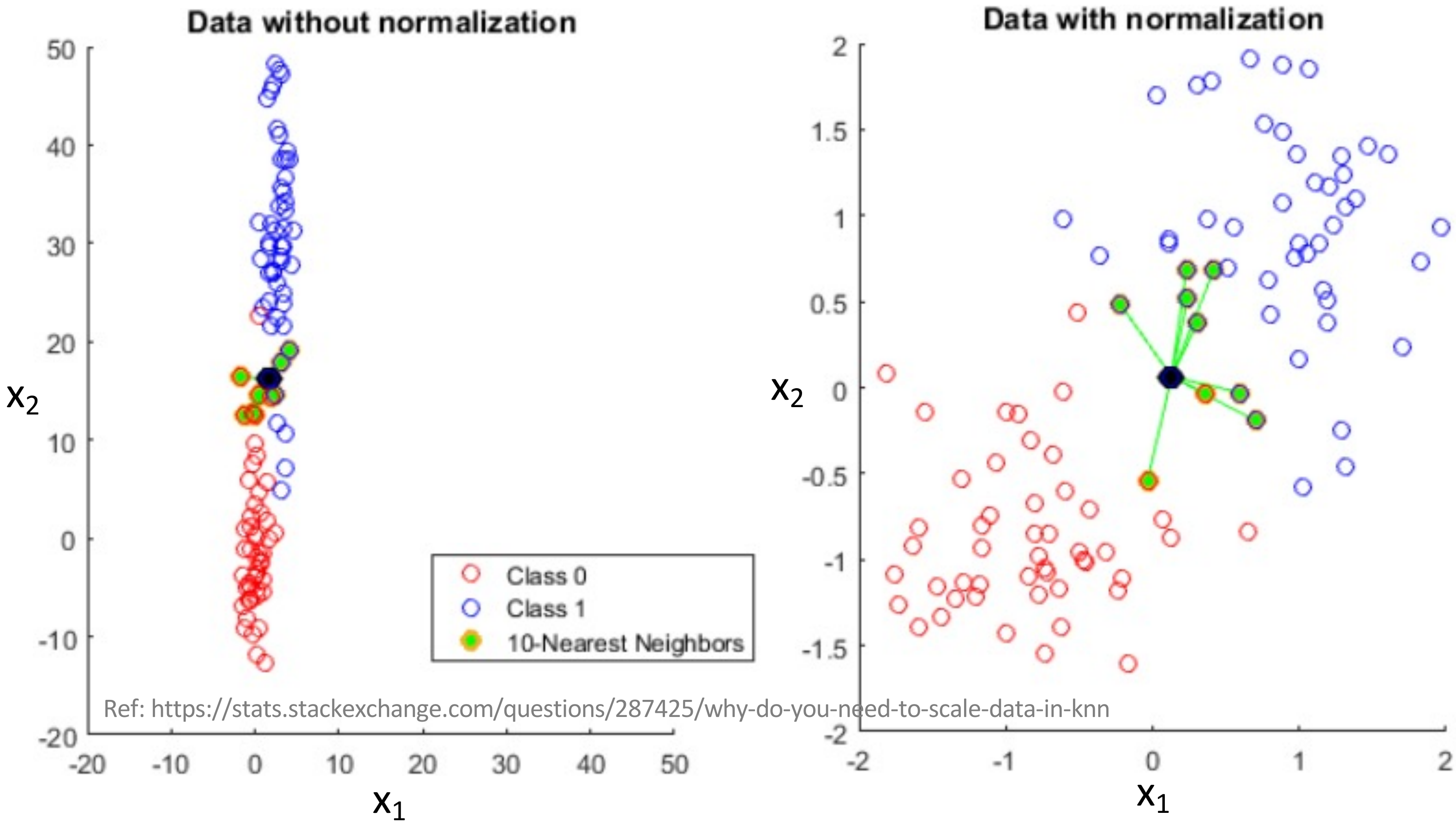
Person	Age [scaled]	Height [scaled]
A	-0.87	0.87
B	0.87	0.87
C	-0.87	-0.87
D	0.87	-0.87



3

Ref: https://stat.ethz.ch/education/semesters/ss2012/ams/slides/v4.2.pdf

- **Why the need for scaling?**



Ref: https://stats.stackexchange.com/questions/287425/why-do-you-need-to-scale-data-in-knn

Consider a simple binary classification problem, where a Class 1 sample is chosen (black) along with its 10-nearest neighbors (filled green). Without normalization, all the nearest neighbors are aligned in the direction of the axis with the smaller range, i.e., x_1 leading to incorrect classification. Normalization solves this problem.

Issues with kNN: Scaling

- **Why the need for scaling?**

Unscaled Data

Age	Loan	Default	Distance	
25	$40,000	N	102000	
35	$60,000	N	82000	
45	$80,000	N	62000	
20	$20,000	N	122000	
35	$120,000	N	22000	2
52	$18,000	N	124000	
23	$95,000	Y	47000	
40	$62,000	Y	80000	
60	$100,000	Y	42000	3
48	$220,000	Y	78000	
33	$150,000	Y	8000	1
48	**$142,000**	**?** Y		

Euclidean Distance

closest ~ most similar

$$D = \sqrt{(\mathrm{Age}_1\text{-}\mathrm{Age}_2)^2 + (\mathrm{Loan}_1\text{-}\mathrm{Loan}_2)^2}$$

Scaled Data

Age	Loan	Default	Distance
0.125	0.11	N	0.7652
0.375	0.21	N	0.5200
0.625	0.31	N	0.3160
0	0.01	N	0.9245
0.375	0.50	N	0.3428
0.8	0.00	N	0.6220
0.075	0.38	Y	0.6669
0.5	0.22	Y	0.4437
1	0.41	Y	0.3650
0.7	1.00	Y	0.3861
0.325	0.65	Y	0.3771
0.7	**0.61**	**?** N	

Standardized Variable

$$X_s = \frac{X - Min}{Max - Min}$$

closest ~ most similar

Ref: https://www.saedsayad.com/k_nearest_neighbors.htm

Issues with kNN: Scaling

- If variables are not scaled:
  - variable with largest range has most weight
  - distance depends on scale
- Scaling gives every variable equal weight
- Similar alternative is re-weighing:

$$d(\mathrm{i,j}) = \sqrt{w_1(x_{i1} - x_{j1})^2 + w_2(x_{i2} - x_{j2})^2 + \ldots + w_p(d_{ip} - x_{jp})^2}$$

- Scale if,
  - variables measure different units (kg, meter, sec,...)
  - you explicitly want to have equal weight for each variable
- **Don't scale if units are the same for all variables**
- Most often: Better to scale (doesn't hurt)

Issues with kNN: Attributes of mixed type

- Ideally the distance between two points should be:

$$d(\mathrm{i,j}) = \sqrt{w_1(x_{i1} - x_{j1})^2 + w_2(x_{i2} - x_{j2})^2 + \ldots + w_p(d_{ip} - x_{jp})^2}$$

- It is, however, hard to decide weights on the features. Scaling takes care of this for numerical attributes, but what about in the presence of mixed attributes?
- Example: Suppose we need to predict if a person is obese or not given the height, weight and gender (Male/Female). The gender is probably very important for this problem.
- Example: For predicting the income of a person given age, education and the gender, the gender should have no bearing on the target with 0 or a very small weight.
- It's crucial to give appropriate weights (domain expertise?) to numerical and nominal attributes. See the next page for how.

Ref: https://www.quora.com/How-do-I-perform-a-KNN-algorithm-with-a-mix-of-Categorical-and-Continuous-Variables

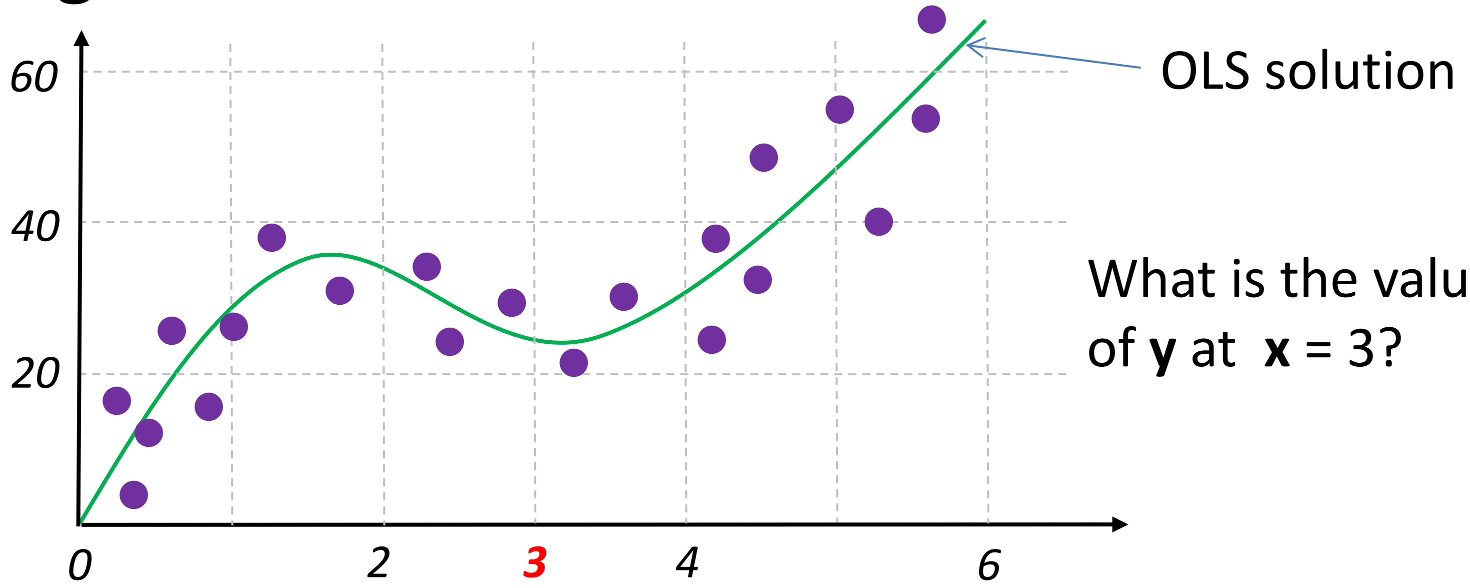
Issues with kNN: Attributes of mixed type

- Nominal (binary or otherwise), ordinal and numeric
- One popular hybrid distance function is Gower's distance (a weighted formula for combined effects):

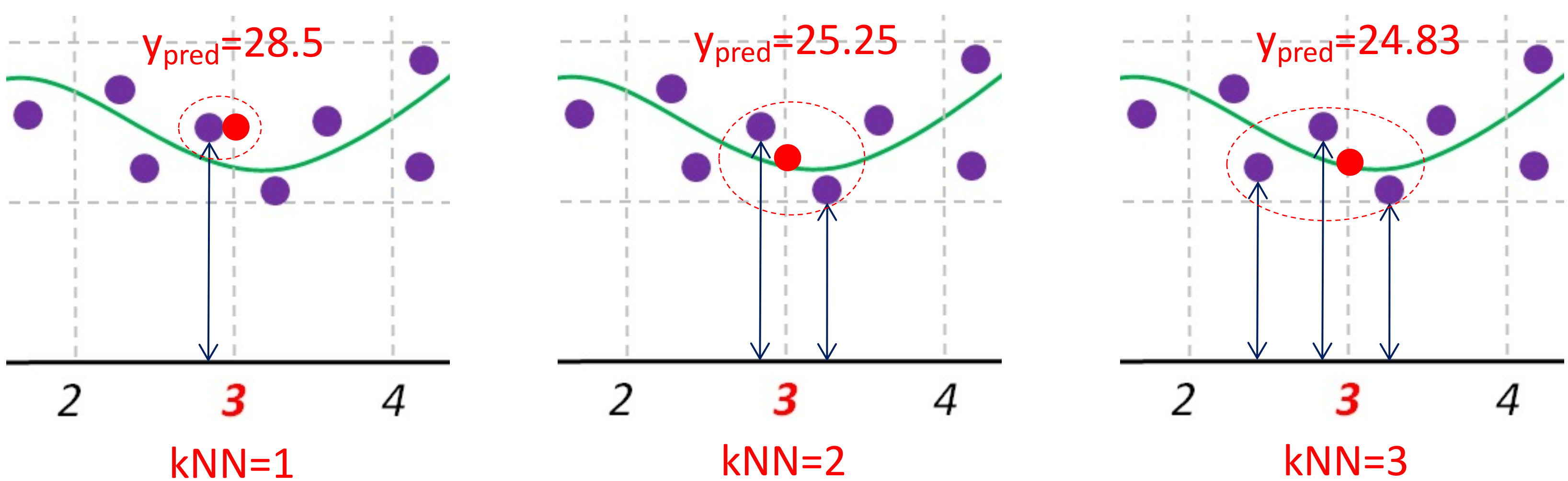
$$d_{ij} = \frac{\sum_{k=1}^{p} w_{ijk} d_{ijk}}{\sum_{k=1}^{p} w_{ijk}}$$

- where $\mathbf{w_{ijk}}$ is a **user-specified weight for attributes** *i,j=1,2,…,k*. In the absence of specific information, the weights are often set to 1 (features with equal weight).
- There is not a generally applicable solution to choosing weights effectively (one setting that works well for one scenario may not work well for others)
  - If variable **k** is numeric: Use the normalized distance
  - If variable **k** is nominal/ordinal: Use Hamming, Simple Matching, etc.

- **Regression via kNN**



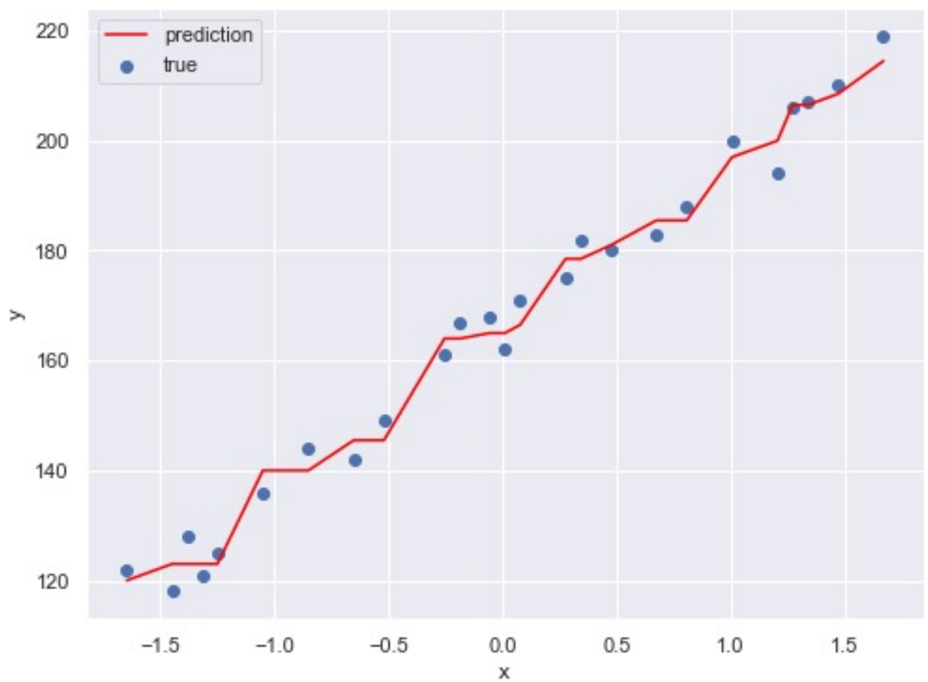
What is the value of **y** at **x** = 3?



- **Regression via kNN**

```
y = [122,118,128,121,125,136,144,142,149,161,167,168,162,
     171,175,182,180,183,188,200,194,206,207,210,219],
x = [50,53,54,55,56,59,62,65,67,71,72,74,75,76,79,80,82,85,87,90,93,94,95,97,100]

kfold = RepeatedKFold(n_splits=5, n_repeats=3, random_state=42)
y = StandardScaler().fit_transforx)
knn_reg = KNeighborsRegressor()
params = {'n_neighbors': [1,2,3,4,5], 'p': [1, 2]}
grid = GridSearchCV(estimator=knn_reg, param_grid=params, cv=kfold)
grid.fit(x,y)
y_pred = grid.predict(x)
plt.scatter(x,y)
plt.plot(x, y_pred);
```



- **Missing value imputation via kNN**
- We have a data set **D** of size **N**
- Assume data in instance '**i**' has a missing value for attribute $\mathbf{C_1}$

#	C_1	C_2	C_3	C_4	Class
1	21	6.5	52	41	Yes
2	17	11	87	15	No
…	…	…	…	…	…
i	?	7	54	39	Yes
…	…	…	…	…	…
N	40	14	99	82	No

Pseudocode for the imputation

- for j = 1 , N (where j ≠ i) :
  - Compute distance $\mathbf{d_j}$ between the data instances '**i**' and '**j**'

$$d(i,j) = d_j = \sqrt{\sum_{m \in D^*} (x_i^{\,m} - x_j^{\,m})^2}$$

D*is the set of attributes with non-missing values

where sum is over all columns with no missing values

  - Save the distance $\mathbf{d_j}$ in a similarity array **S**
- Sort the array **S** in descending order
- Pick the top **k** data instances from **S**
- Impute the missing value of '**i**' by the mean (or mode in case of categorical attribute) of the top **k** known values of attribute $\mathbf{C_1}$

Pros and Cons of kNN

- **Pros**
  - Can be used for both classification and regression
  - Makes no assumptions about the data
  - Simple to understand, highly intuitive, and easy to implement
  - Flexible to feature/distance choices
  - Naturally handles (and particularly powerful for) multiple classes
  - Can do well in practice with enough representative data
  - Good performance on data with few dimensions
  - Easy to update in online settings (could be very slow)
  - Learns nonlinear boundaries
- **Cons**
  - May overfit (low bias/high variance model, choice of k is crucial)
  - Missing data need to be handled
  - Do not work well with high dimensional datasets
  - Slow and cost of classification can be high (large search problem to find nn)
  - Sensitive to outliers and irrelevant attributes, vulnerable to noisy inputs
  - Need to store all samples and may require lots of memory
  - No variable selection of any kind
  - Using all attributes to compute distances is computationally expensive
  - When data is imbalanced, predicted values are biased towards majority class
  - Not a good option for mixed data (categorical and numerical) sets

```python
# using scikit-learn library
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import accuracy_score

# kcv is the optimum k value found via CV
Clf = KNeighborsClassifier(n_neighbors=kcv)
Clf.fit(X_train, y_train)
accuracy_score(y_test, Clf.predict(X_test))

# Hyperparameters tuned:
# n_neighbors: number of neighbors (def=5)
# metric: metric to use for computing distance
```