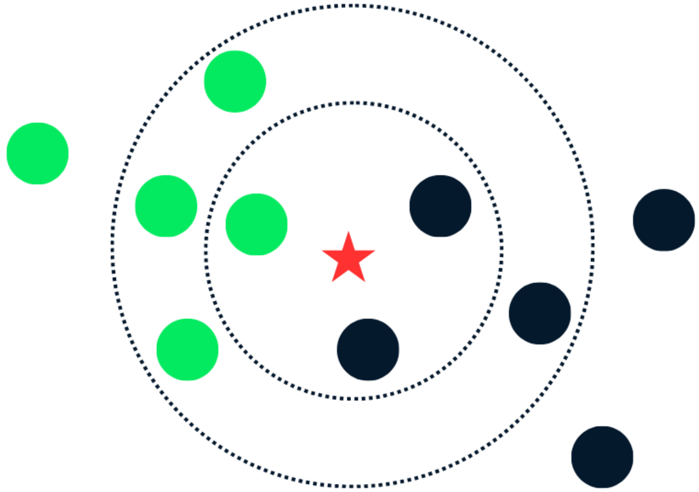
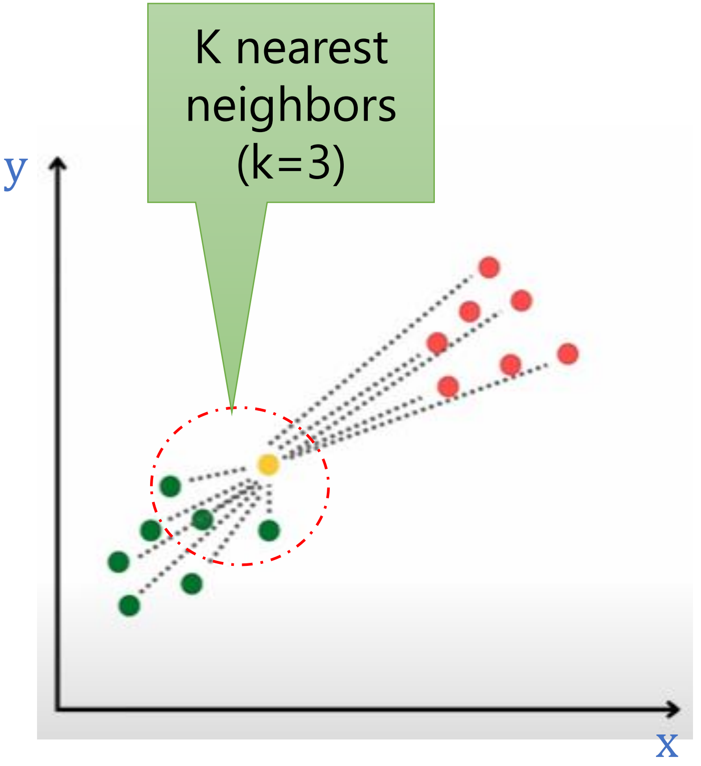
K-Nearest Neighbour (*k*NN) Classifier



kNN Algorithm steps

- K-Nearest Neighbors (kNN) is a classification technique designed to predict outcomes based on **similar** observed instances
- Given a data point that we want to classify:
  - Compute its distance from all data points in the dataset
  - Locate the closest **k** points
  - For classification: Get the label with majority vote
  - For regression: Get the average of their values

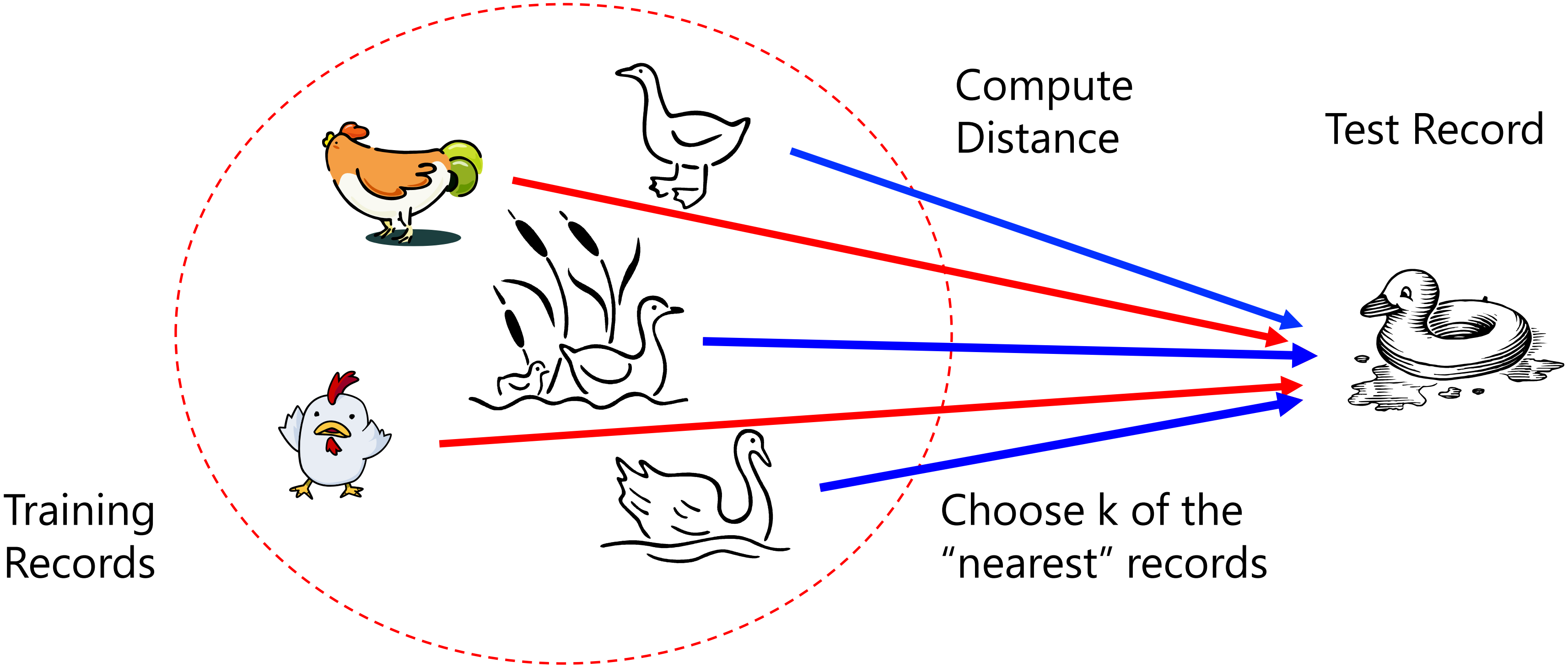


K-Nearest Neighbour Model

- **Lazy Learner**:
  - It does not build models explicitly
  - Unlike eager learners such as decision tree induction
  - Delaying the decision to the time of classification => ideal where data is constantly changing
- **Instance-based** learning, i.e., no explicit model, but rather stores instances of the training data
- **Non-parametric** algorithm, i.e., no assumptions about the underlying distribution of the data
- **Similarity-based** learning:
  - Similar examples have similar label
  - Classify new unseen examples "like" most similar training examples

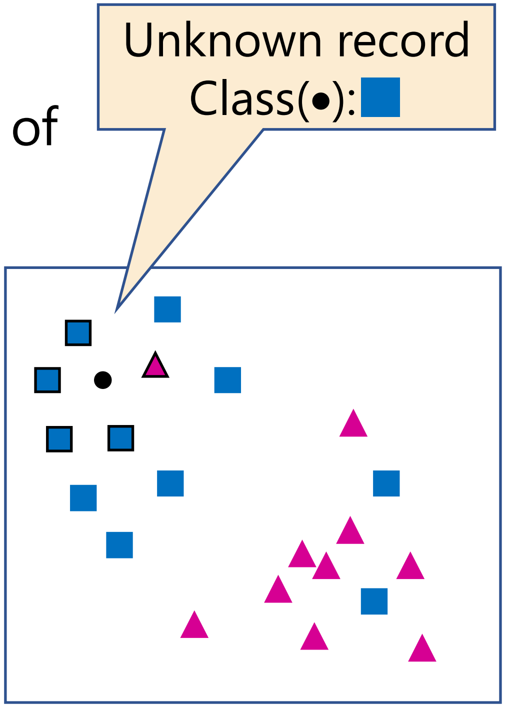
Rationale

- "If it walks like a duck, quacks like a duck, then it's probably a duck"



Requirements & Steps

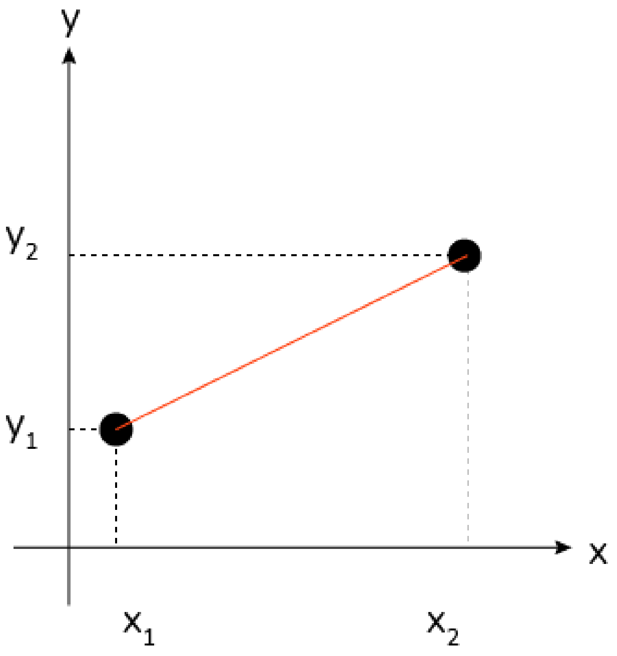
- **Requirements:**
  - The set of training instances
  - Distance Metric to compute distance between instance
  - The value of the hyperparameter *k*, i.e., the number of nearest neighbors to consider
- **To predict an unknown instance:**
  - Compute distance to all training instances
  - Locate ***k*** nearest neighbors
  - Take **majority vote** of class labels of nearest neighbors. In the case of a tie:
    - Use odd *k* (doesn't solve multi-class)
    - Winner is the class with the **shortest distance** to the point to classify
    - Choose the class label randomly
    - Pick the class dominant in the entire dataset
    - decrease *k* by 1 until you break the tie



Unknown record
Class(•):🟦

Euclidean Distance

- **Euclidean Distance** is a measure of the **straight-line distance** between two points $(x_1, y_1)(x_2, y_2)$ in a two-dimensional space



It's derived from the **Pythagorean** theorem

$$\text{Distance} = \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2}$$

- Numerical measure of **Dissimilarity/Distance**
  - How **different** are two data objects
  - Lower when objects are more alike

Euclidean Distance for data objects with multiple features

- When dealing with instances (data objects) with multiple $\mathbf{n}$ features, the Euclidean distance between two instances $\mathbf{p}$ and $\mathbf{q}$ with features ($p_1, p_2, \ldots p_n$) and ($q_1, q_2, \ldots q_n$) respectively, is given by:
  - the square root of the sum of the squared differences between corresponding feature values

$$\text{Distance} = \sqrt{\sum_{i=1}^{n} (q_i - p_i)^2}$$

- p_i and q_i represent the values of the i^{th} feature for data objects $\mathbf{p}$ and $\mathbf{q}$ respectively
- $\mathbf{n}$ is the number of features

slides\knn.xlsx

Distance Metrics

- Euclidean distance: $d_{Eucl}(a,b) = \sqrt{\sum_{k=1}^{m}(a_k - b_k)^2}$

- Manhattan distance: $d_{Manh}(a,b) = \sum_{k=1}^{m}|a_k - b_k|$

- Minkowski distance: $d_{Mink}(a,b) = \sqrt[p]{\sum_{k=1}^{m}|a_k - b_k|^p}$

  - When *p* = 1, the Minkowski distance is the Manhanttan distance
  - When *p* = 2, the Minkowski distance is the Euclidean distance
  - Although there are infinite number of Minkowski-based distance metrics to choose from, Euclidean distance and Manhattan distance are the most used ones

Similarity Between Binary Vectors using Simple Matching Coefficient (SMC)

- Simple Matching Coefficient (SMC) is a similarity measure used to quantify the similarity between two **binary** vectors

$$\mathrm{SMC} = \frac{\mathrm{M}_{11} + \mathrm{M}_{00}}{\mathrm{M}_{11} + \mathrm{M}_{00} + \mathrm{M}_{01} + \mathrm{M}_{10}}$$

In the case of multiple features, the formula is applied to each feature individually, and then averaged across all features:

$$\mathrm{SMC} = \frac{1}{n}\sum_{i=1}^{n} \frac{M_{11}{}^{(i)} + M_{00}{}^{(i)}}{M_{11}{}^{(i)} + M_{00}{}^{(i)} + M_{01}{}^{(i)} + M_{10}{}^{(i)}}$$

- Where:
  - ***M*₁₁**
  - $\boldsymbol{M_{11}}$ = number of attribute pairs where both vectors have the same value (1 in one vector, 1 in the other)
  - $\boldsymbol{M_{00}}$ = number of attribute pairs where both vectors have the same value (0 in one vector, 0 in the other)
  - $\boldsymbol{M_{01}}$ = number of attribute pairs where the first vector has a value of 0 and the second vector has a value of 1
  - $\boldsymbol{M_{10}}$ = number of attribute pairs where the first vector has a value of 1 and the second vector has a value of 0
  - **n** is the number of features

Similarity Between Binary Vectors using Jaccard Coefficient

- Jaccard Coefficients is used to quantify the similarity between two **binary** vectors

$$J = \frac{number\ of\ 11\ matches}{number\ of\ not{-}both{-}zero\ attribute\ values}$$

$$J = \frac{M_{11}}{M_{01} + M_{10} + M_{11}}$$

- SMC / Jaccard ranges from 0 to 1, where:
  - 0 indicates no similarity between the instances
  - 1 indicates perfect similarity between the instances
  - Higher SMC values suggest greater similarity between the instances, while lower values suggest greater dissimilarity

SMC versus Jaccard - Example

$p =$ 1, 0, 1, 0, 1

$q =$ 1, 1, 0, 0, 1

$M_{11} = 2$ (the number of attributes where p was 1 and q was 1)

$M_{00} = 1$ (the number of attributes where p was 0 and q was 0)

$M_{01} = 1$ (the number of attributes where p was 0 and q was 1)

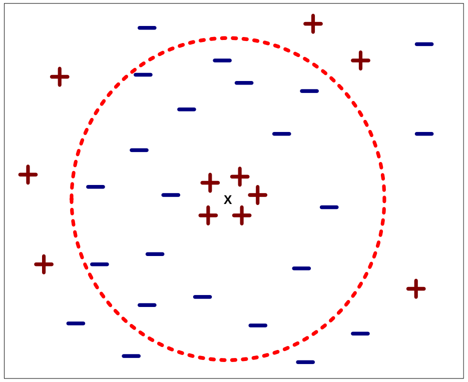
$M_{10} = 1$ (the number of attributes where p was 1 and q was 0)

$$\text{SMC} = \frac{M_{11} + M_{00}}{M_{11} + M_{00} + M_{01} + M_{10}} = \frac{2 + 1}{2 + 1 + 1 + 1} = \mathbf{0.6}$$

$$J = \frac{M_{11}}{M_{01} + M_{10} + M_{11}} = \frac{2}{2 + 1 + 1} = \mathbf{0.5}$$

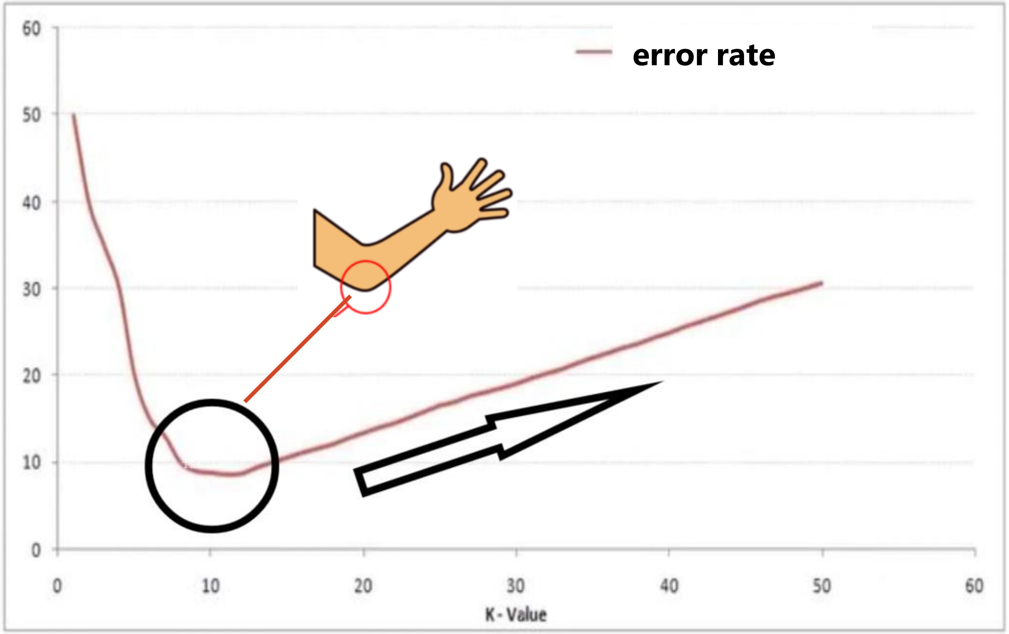
Choosing the value of k

- If k is too small, sensitive to noise points (results in high variance, overfitting) => performing poorly on unseen data
- If k is too large, neighborhood may include points from other classes => leading to underfitting and poor performance on both training and test data



The Elbow Method

- Choose a range of values for k
- Fit the KNN model with each value of k
- Compute the error rate for each k
- Plot the error rate against the corresponding k values



- Identify the "elbow" point on the plot, where the rate of decrease in error rate slows down significantly.
- Choose the value of k at the elbow point as the optimal number of neighbors for the KNN model

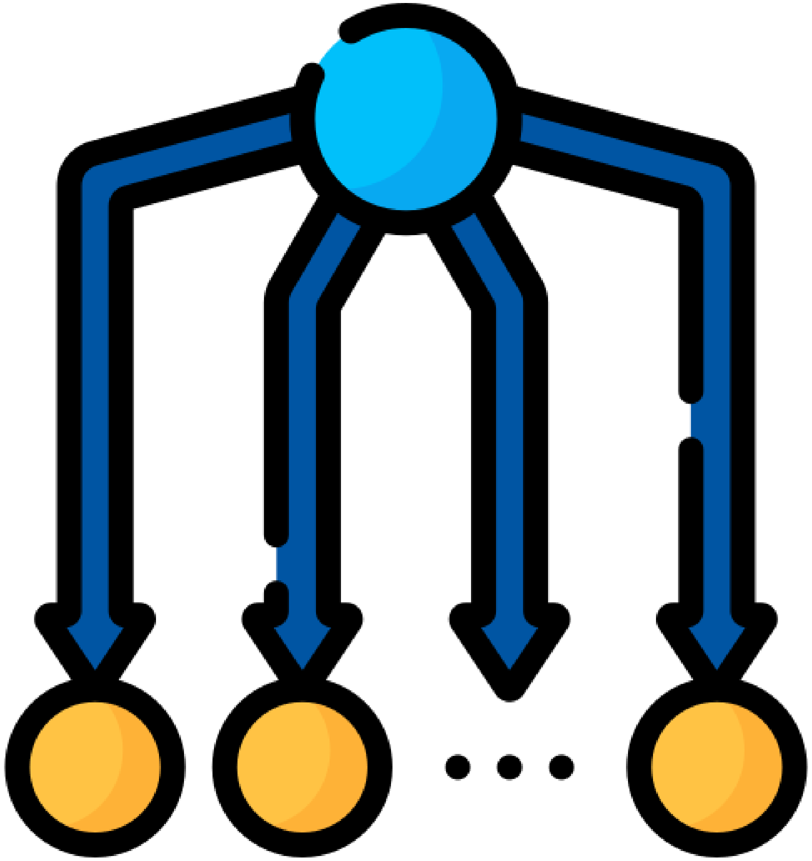
K-NNs Properties

- k-NNs are lazy (or instance-based) learners
  - It does not build models explicitly
  - Unlike eager learners such as decision tree induction
- Simple to implement algorithm
- Feature scaling is necessary, if scales differ, to prevent distance measures from being dominated by attributes having higher magnitude, e.g.,
  - height of a person 1.5m to 1.8m, weight 90lb to 300lb and monthly income QR10K to QR500K
- Requires little tuning
- Often performs quite well! => Try it first on a new learning problem
- Predicting unknown records are relatively expensive
  - Must make a pass through the entire dataset for each classification. This can be prohibitive for large data sets

KNN Limitations

- Slow when the training data set is large, as it requires calculating distances to every training data point
- It doesn't work well with high-dimensional data, as the prediction accuracy can degrade as the number of attributes grows
- Sensitive to the choice of distance metric, and different distance metrics may lead to different results
- It doesn't handle imbalanced data well, as it may assign the majority class label to new data points
- Missing values must be processed with little effect on the distances. Replace with average is reasonable but may have side effects

Naïve Bayes Classifier



Naïve Bayes Classifier

- Naïve Bayes uses a probabilistic approach to build a classifier. It uses **conditional probabilities** to predict multi-class or binary outcomes
  - It determines the most probable class label for an unseen instance given its features. Example:
    - Determine the class an object given its attributes {shape, color, weight}
    - A given object that is {shape = 'spherical', color = 'yellow', weight < 60 grams}, may be classified as a tennis ball

- "naive" classifier as it assumes that the object features are independent of each other, which is often not the case in real-world datasets
  - Nevertheless, it is still widely used and can achieve high accuracy in many applications

Naïve Bayes Classifier - Applications

- **Text classification**: for spam filtering, sentiment analysis, and topic classification
- **Image classification**: for face recognition, object detection, and image segmentation
- **Medical diagnosis**: for disease prediction and risk assessment
- **Recommendation systems**: for personalized product recommendations and content recommendations

Example of Bayes Theorem

- Given:
  - A doctor knows that meningitis causes stiff neck 50% of the time
  - Prior probability of any patient having meningitis is 1/50,000 (*prior knowledge*)
  - Prior probability of any patient having stiff neck is 1/20
- If a patient has stiff neck, what' s the probability he/she has meningitis? **(Conditional Probability)**

$$\boldsymbol{P(M|S) = \frac{P(S|M)P(M)}{P(S)} = \frac{0.5 \times 1/50000}{1/20} = 0.0002}$$

Naïve Bayes Classifier - Example

- Heart disease dataset. Class label ***Y*** means has heart disease, and the class label ***N*** means that the disease is absent
- **Prior** Probabilities for the Class labels: {Y, N}
  - P(Class = Y) = 6/10 = 0.6
  - P(Class = N) = 4/10 = 0.4

- **Conditional Probabilities given Class = Y**
  - P(BloodPressure=high | Class=Y) = 4/6
  - P(BloodPressure=normal | Class=Y) = 2/6
  - P(BloodSugarLevel=1 | Class=Y) = 1/6
  - P(BloodSugarLevel=2 | Class=Y) = 2/6
  - P(BloodSugarLevel=3 | Class=Y) = 3/6
  - P(Habit=smoker | Class=Y) = 5/6
  - P(Habit=nonsmoker | Class=Y) = 1/6

- **Conditional Probabilities given Class = N**
  - P(BloodPressure=high | Class=N) = 2/4
  - P(BloodPressure=normal | Class=N) = 2/4
  - P(BloodSugarLevel=1 | Class=N)= 1/4
  - P(BloodSugarLevel=2 | Class=N) = 2/4
  - P(BloodSugarLevel=3 | Class=N) = 1/4
  - P(Habit=smoker | Class=N) = 1/4
  - P(Habit=nonsmoker | Class=N) = 3/4

Blood Pressure	Blood Sugar Level	Habit	Class
High	3	Smoker	Y
High	3	Nonsmoker	Y
High	2	Smoker	Y
High	1	Smoker	Y
Normal	3	Smoker	Y
Normal	2	Smoker	Y
High	2	Nonsmoker	N
High	2	Nonsmoker	N
Normal	3	Smoker	N
Normal	1	Nonsmoker	N

Classify patient *A* (1/2)

- Classify the unseen patient ***A*** using the Naïve Bayes classifier

Blood Pressure	Blood Sugar Level	Habit	Class
High	2	Nonsmoker	?

- Calculate **P(A | Class = Y)** and **P(A | Class = N)**

$$\mathbf{P(A \mid Class = Y)} = P(BloodPressure = High \mid Class = Y)$$

$$\times\ P(BloodSugarLevel = 2 \mid Class = Y)$$

$$\times\ P(Habit = nonsmoker \mid Class = Y)$$

$$= 4/6 \ \ x\ 2/6 \ \ x\ 1/6 = \frac{\mathbf{8}}{\mathbf{6x6x6}}$$

$$\mathbf{P(A \mid Class = N)} = P(BloodPressure = High \mid Class = N)$$

$$\times\ P(BloodSugarLevel = 2 \mid Class = N)$$

$$\times\ P(Habit = nonsmoker \mid Class = N)$$

$$= 2/4 \ \ x\ 2/4 \ \ x\ 3/4 = \frac{\mathbf{12}}{\mathbf{64}}$$

Classify patient A (2/2)

- Calculate **P(Class=Y | A)** and **P(Class=N | A)**

$$\mathbf{P(Class{=}Y \mid A)} \approx P(A \mid Class{=}Y) \times P(Class{=}Y)$$

$$= 8 /(6x6x6) \times (6/10) = \mathbf{32/360}$$

$$\mathbf{P(Class{=}N \mid A)} \approx P(A \mid Class{=}N) \times P(Class{=}N)$$

$$= (12/64) \times (4/10) = 12/160$$

Since **P(Class=N | A) > P(Class=Y | A)**

Therefore, assign the patient A class **N** (not having a heart disease)

Naïve Bayes Classifier

Thomas Bayes

Bayes theorem for a
Single predictor variable A

Probability of C given A is observed
(i.e., conditional probability)

$$P(C|A) = \frac{P(A|C)P(C)}{P(A)}$$

Bayes' Theorem provides a way to calculate the probability of a class label given the instance's features

Now we have object **A** having **m** *attributes:* $a_1, a_2, a_3, \ldots, a_m$

C is the class label:
$C \in \{C_1, C_2, \ldots C_n\}$

$$P(C_i|A) = \frac{P(A|C_i)P(C_i)}{P(A)} = \frac{P(a_1, a_2, \ldots, a_m|C_i)P(C_i)}{P(a_1, a_2, \ldots, a_m)}$$

$$i = 1, 2, \ldots, n$$

Apply the Naïve Assumption and Remove a Constant

- For observed attributes A = (a_1, a_2, … a_m), we want to compute

$$P(C_i|A) = \frac{P(a_1, a_2, \ldots, a_m|C_i)P(C_i)}{P(a_1, a_2, \ldots, a_m)} \qquad i = 1, 2, \ldots, n$$

  Then assign to the object A the class **C_i** having the having the highest probability **P(C_i|A)**

- Two simplifications to the calculations
  - Apply naïve assumption - each **a_j** is conditionally independent of each other, then

$$P(a_1, a_2, \ldots, a_m|C_i) = P(a_1|C_i)P(a_2|C_i) \cdots P(a_m|C_i) = \prod_{j=1}^{m} P(a_j|C_i)$$

  - Denominator **P(a_1,a_2,…a_m)** is a constant and can be ignored

Building a Naïve Bayes Classifier

- Naïve Bayes Classifier after applying the two simplifications

$$\boldsymbol{P(C_i|A) = P(a_1, a_2, \ldots, a_m|C_i)P(C_i) = \left(\prod_{j=1}^{m} P(a_j|C_i)\right)P(C_i) \quad i = 1, 2, \ldots, n}$$

conditional probability of feature j given class i

- To build a Naïve Bayes Classifier, collect the following statistics from the training data:
  - **P(C$_i$)** for every class labels
  - **P(a$_j$| C$_i$)** for every distinct value of attribute **a$_j$** given class **C$_i$**
    - Fraction of training examples of class **C$_i$** that take the attribute value **a$_j$**
  - Assign to the object A the class C$_i$ having the highest probability P(C$_i$|A)

Note: If attribute **a$_j$** is continuous then discretize it first

Naïve Bayes Classifier for the Credit Example

- Class labels: {good, bad}
  - P(Class = good) = 19/27 = 0.7
  - P(Class = bad) = 8/27 = 0.3
- Conditional Probabilities
  - P(Housing = own | bad) = 4/8 = 0.5
  - P(Housing =own | good) = 13/19 = 0.68
  - P(Housing =rent | bad) = 4/8 = 0.5
  - P(Housing =rent | good) = 6/19 = 0.316
  - … and so on

Job	Housing	Savings	Credit_class
self_employed	own	low	good
skilled	own	low	good
skilled	own	high	good
skilled	own	high	good
skilled	own	high	good
skilled	rent	low	good
skilled	own	low	good
skilled	rent	low	good
skilled	own	high	good
skilled	own	high	good
skilled	rent	low	good
skilled	rent	low	good
skilled	own	high	good
skilled	rent	low	good
skilled	own	high	good
skilled	own	high	good
skilled	rent	low	good
skilled	own	high	good
skilled	own	high	good
skilled	own	low	bad
self_employed	rent	low	bad
self_employed	own	low	bad
self_employed	own	high	bad
self_employed	rent	low	bad
self_employed	rent	low	bad
self_employed	rent	low	bad
self_employed	own	high	bad

Naïve Bayes Classifier for a Particular Applicant

- Given applicant attributes of

A= {Housing = own, Job = self-employed, savings =low}

- We need to compare P(bad|A) against (good|A), and we also need to calculate P(A|bad) and P(A|good)

P(good|A) ≈ P(A|good) × P(good)

P(bad|A) ≈ P(A|bad) × P(bad)

P(good|A) ≈ P(A|good) × 0.7

P(bad|A) ≈ P(A|bad) × 0.3

a_j	C_i	$P(a_j\| C_i)$
Housing = own	good	13/19 = 0.68
Housing = own	bad	4/8 = 0.5
Job = self emp	good	1/19 = 0.05
Job = self emp	bad	6/8 = 0.75
Savings = low	good	9/19 = 0.73
Savings = low	bad	7/8 = 0.875

P(A | good) = P(Housing =own | good) × P(Job=self-employed | good) × P(Savings = low | good)
= 0.68 × 0.05 × 0.73 = 0.2482

P(A | bad) = P(Housing =own | bad) x P(Job=self-employed | bad) × P(Savings = low| bad)
= 0.5 × 0.75 × 0.875 = 0.3281

P(good|A) ≈ 0.2482 × 0.7 = 0.0174
P(bad|A) ≈ 0.3281 × 0.3 = 0.0984

Since P(bad|A) > (good|A), assign the applicant the label "**bad**" credit

Handling zero probabilities in a Naïve Bayes

- Zero probabilities due to unobserved attribute/class pairs
  - The presence of zero probabilities for certain features given a class can cause issues during classification
- Can be handled using **Laplace smoothing** (correction)
  - Adds **1** to all feature counts and adds the **number of attributes** to the total number of records for each class

Attributes			
a1	**a2**	**a3**	**Class**
A	C	E	Yes
B	D	F	Yes
A	C	F	Yes
B	C	E	Yes
B	D	F	No
B	D	F	No
B	D	F	No
B	D	F	No

	Before Correction	Laplace Correction
P(A | Yes)	2/4	(2+1)/(4+3) = 3/7
P(D | Yes)	1/4	(1+1)/(4+3) = 2/7
P(F | Yes)	2/4	(2+1)/(4+3) = 3/7
P(A | No)	**0/4**	**(0+1)/(4+3) = 1/7**
P(D | No)	4/4	(4+1)/(4+3) = 5/7
P(F | No)	3/4	(3+1)/(4+3) = 4/7

Test Record = (a1=A, a2=D, a3 =F) => Class?

Before Correction:

P((a1=A, a2=D, a3 =F) | Yes) = 2/4 × 1/4 × 2/4 = 4/64

P((a1=A, a2=D, a3 =F) | No) = 0/4 × 4/4 × 3/4 = 0

After Correction:

P((a1=A, a2=D, a3 =F) | Yes) = 3/7 × 2/7 × 3/7 = 0.052
P((a1=A, a2=D, a3 =F) | No) = 1/7 × 5/7 × 4/7 = 0.058

Class before Correction:

P(record|Yes) × P(Yes) > P(record|No) × P(No)
((4/64) × 0.5) > (0 × 0.5)

The record should be classified with class label **Yes**

Class after Correction:

P(record|Yes) × P(Yes) < P(record|No) × P(No)
(0.052 × 0.5) < (0.058 × 0.5)

The record should be classified with class label **No**

Smoothing of Conditional Probabilities

- *Avoid* conditional probabilities having 0 or very small non-zeros values, such as 0.00001

$$\boldsymbol{Original}: \quad P(x|y) = \frac{n_{xy}}{n_y}$$

$$\boldsymbol{Laplace}: \quad P(x|y) = \frac{n_{xy} + 1}{n_y + k}$$

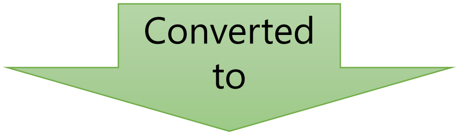
Laplace Smoothing is a technique used to **handle the problem of zero probabilities**, when calculating conditional probabilities from a dataset, for cases where certain combinations of variables have not been observed

- n_{xy} count of **x** given **y**
- n_y count of training examples that have the class **y**
- $\mathbf{k}$ number of features in the dataset

Naïve Bayes Implementation Considerations – Handling Numerical underflow

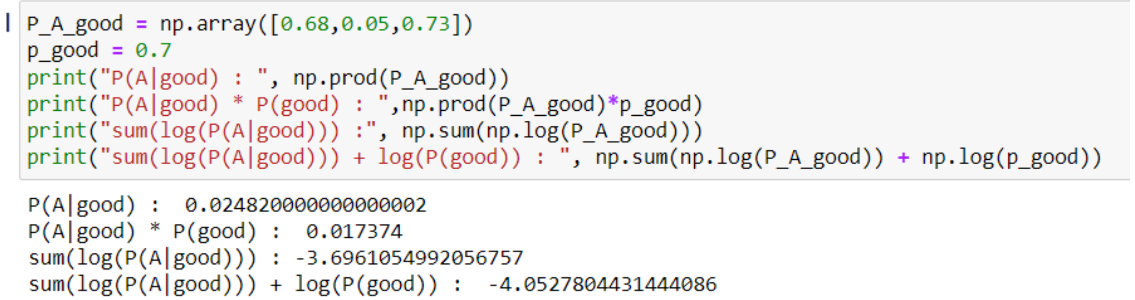
- Numerical underflow resulting from multiplying several probabilities near zero
  - Multiplying many small probabilities can make the result too small to represent accurately with floating-point arithmetic
  - Preventable by converting probabilities to logarithmic scale
  - Instead of multiplying probabilities, sum their logarithms

$$P(c_i|A) = \left(\prod_{j=1}^{m} P\left(a_j|c_i\right)\right) * P(c_i)$$


Converted to

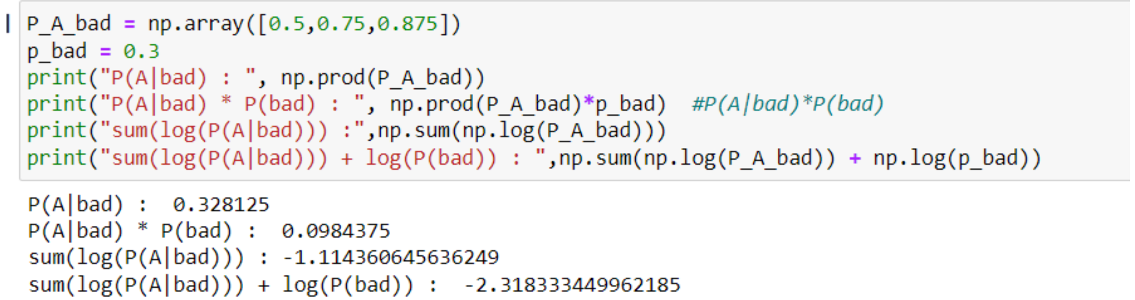
$$\left(\sum_{j=1}^{m} \log P(a_j|C_i)\right) + \log P\left(C_i\right)$$

where i = 1,2,…,n and P denotes the probabilities



```
P_A_good = np.array([0.68,0.05,0.73])
p_good = 0.7
print("P(A|good) : ", np.prod(P_A_good))
print("P(A|good) * P(good) : ",np.prod(P_A_good)*p_good)
print("sum(log(P(A|good))) :", np.sum(np.log(P_A_good)))
print("sum(log(P(A|good))) + log(P(good)) : ", np.sum(np.log(P_A_good)) + np.log(p_good))
```

```
P(A|good) :  0.024820000000000002
P(A|good) * P(good) :  0.017374
sum(log(P(A|good))) : -3.6961054992056757
sum(log(P(A|good))) + log(P(good)) :  -4.0527804431444086
```



```
P_A_bad = np.array([0.5,0.75,0.875])
p_bad = 0.3
print("P(A|bad) : ", np.prod(P_A_bad))
print("P(A|bad) * P(bad) : ", np.prod(P_A_bad)*p_bad)  #P(A|bad)*P(bad)
print("sum(log(P(A|bad))) :",np.sum(np.log(P_A_bad)))
print("sum(log(P(A|bad))) + log(P(bad)) : ",np.sum(np.log(P_A_bad)) + np.log(p_bad))
```

```
P(A|bad) :  0.328125
P(A|bad) * P(bad) :  0.0984375
sum(log(P(A|bad))) : -1.114360645636249
sum(log(P(A|bad))) + log(P(bad)) :  -2.318333449962185
```

Naïve Bayes Classifier

Reasons to Choose (+)	Cautions (-)
• Easy to implement • Computationally efficient o Handles high number of features o Handles categorical variables • Resistant to overfitting • Robust to irrelevant variables • Handles missing values well: o Simply ignores them when estimating the probabilities	• Naïve assumption may not hold in real-world datasets, which can affect the accuracy • Numeric features must be discretized • May not work well with imbalanced datasets, where one class has significantly more samples than the other classes