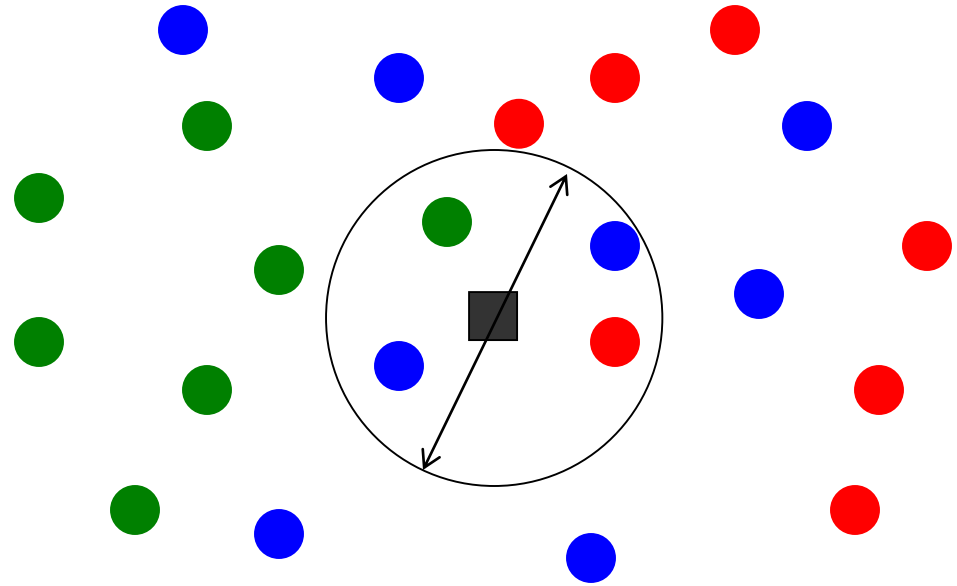
The Nearest-Neighbor Classifier

- Let $D = \{\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_n\}$ be a set of "labeled" *training* samples (or *prototypes*)
- This means that each $\mathbf{x}_i \in \omega_j$ (a class)
- What is the rule?... quite simple:
  - Given an unknown sample **x**
  - Decide the class of sample **x**′ that is **closest** to **x**
  - This is known as the 1-NN rule
- What is the definition of "closeness" ?
- For this, we need a "metric" to measure the "distance" between two samples (later)

The *k*-NN rule

- Start with sample **x** we want to classify
- Grow a region until it encloses *k* samples
  or the *k* nearest neighbors
- Obtaining:
  - k_1 samples $\in \omega_1$
  - k_2 samples $\in \omega_2$
  - :
  - k_c samples $\in \omega_c$
- Decide ω_j
  - where k_j is the max. of
    $k_1, k_2, \ldots, k_c$



Example: 2 classes – *k* must be odd to avoid ties

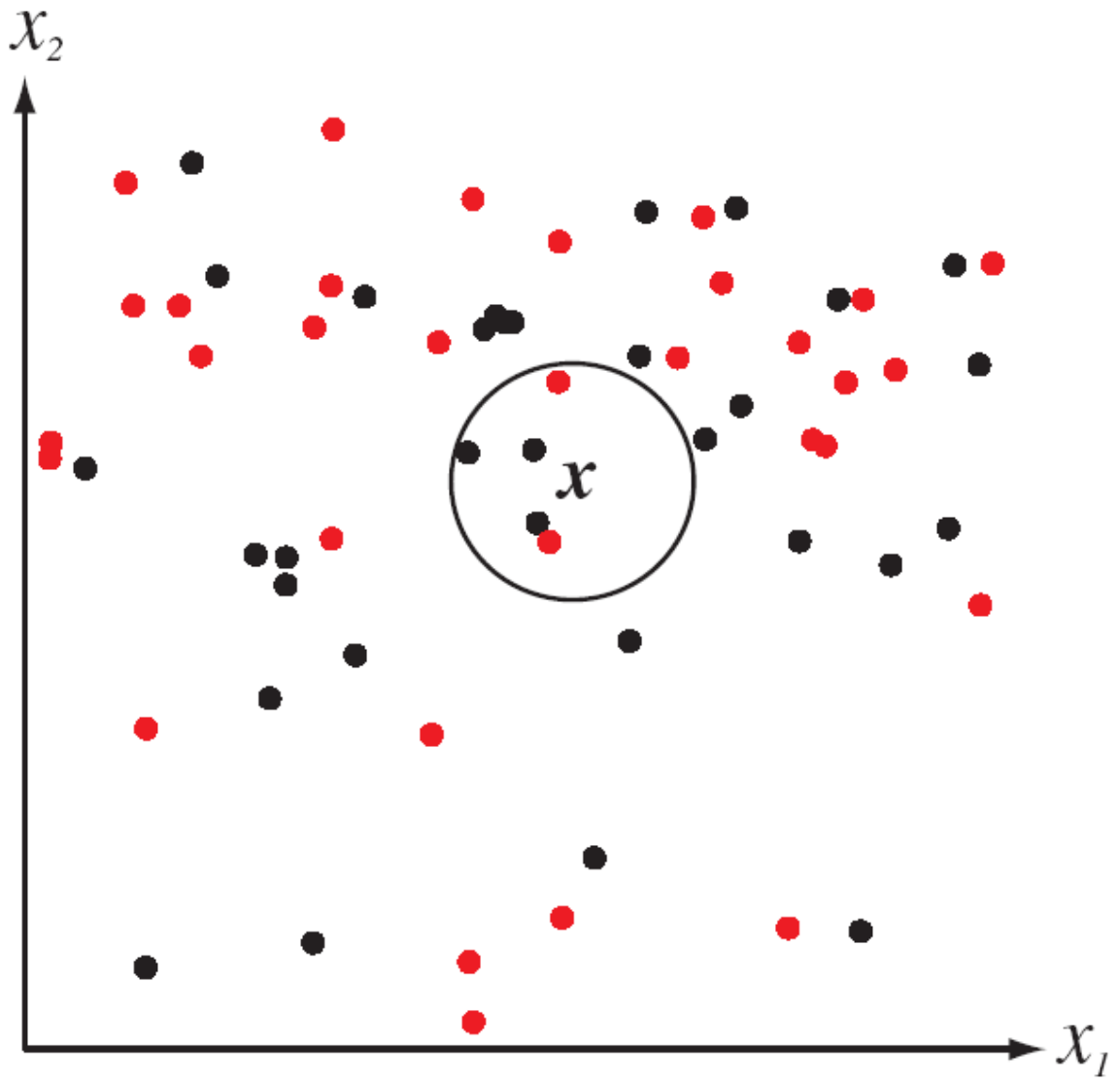
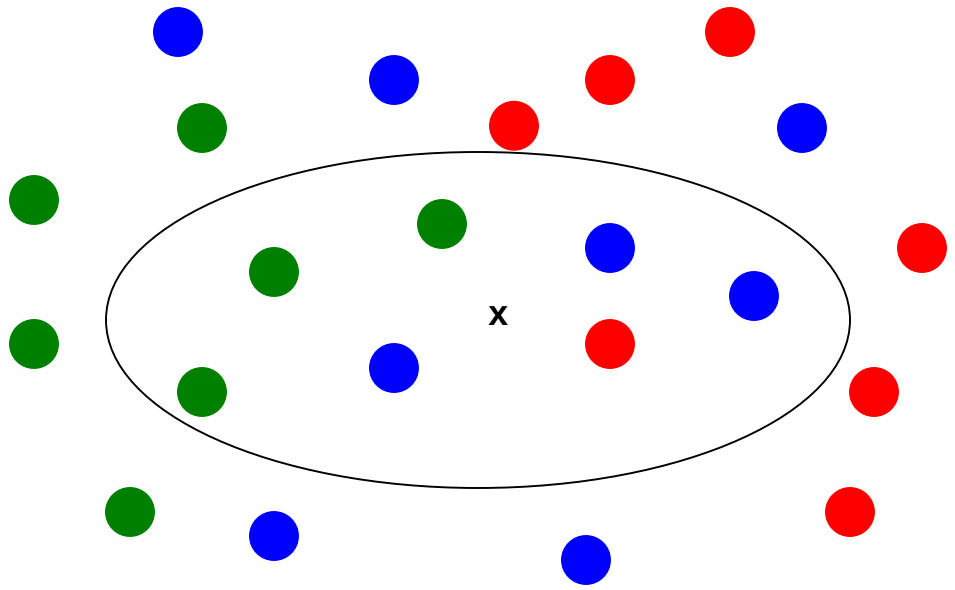


Figure from Duda et al.

Example: Multi-class – ties resolved *arbitrarily*

- Distance: Mahalanobis
- Classify: **x**
- $k = 7$
  - $k_1 = 3$
  - $k_2 = 1$
  - $k_3 = 3$



- … results in a tie between ω_1 and ω_3
- … design a rule that resolves the ties!

Proximity Measures - Metrics

- Proximity can be seen as a generalization of:
  - Dissimilarity (distance)
  - Similarity (closeness)
- Similarity and dissimilarity functions can satisfy certain properties.
  - These are called “metrics” if the following properties are satisfied…

- **Properties of metrics (distance):**
  - Let **x**, **y**, **z** be vectors in $\Re^d$
  - Let $d(\mathbf{x},\mathbf{y})$ be the distance between **x** and **y**
- **Reflexivity:**

$$d(\mathbf{x},\mathbf{y}) = 0 \quad \text{iff} \quad \mathbf{x} = \mathbf{y}$$

- **Symmetry:**

$$d(\mathbf{x},\mathbf{y}) = d(\mathbf{y},\mathbf{x})$$

- **Positivity (nonnegativity):**

$$d(\mathbf{x},\mathbf{y}) \geq 0$$

- **Triangle inequality:**

$$d(\mathbf{x},\mathbf{z}) \leq d(\mathbf{x},\mathbf{y}) + d(\mathbf{y},\mathbf{z})$$

Distance functions (not all are metrics)

- **Euclidean distance:**

$$d(\mathbf{x},\mathbf{y}) = \left(\sum_{i=1}^{d} (x_i - y_i)^2 \right)^{\frac{1}{2}} = \sqrt{(\mathbf{x}-\mathbf{y})^t (\mathbf{x}-\mathbf{y})}$$

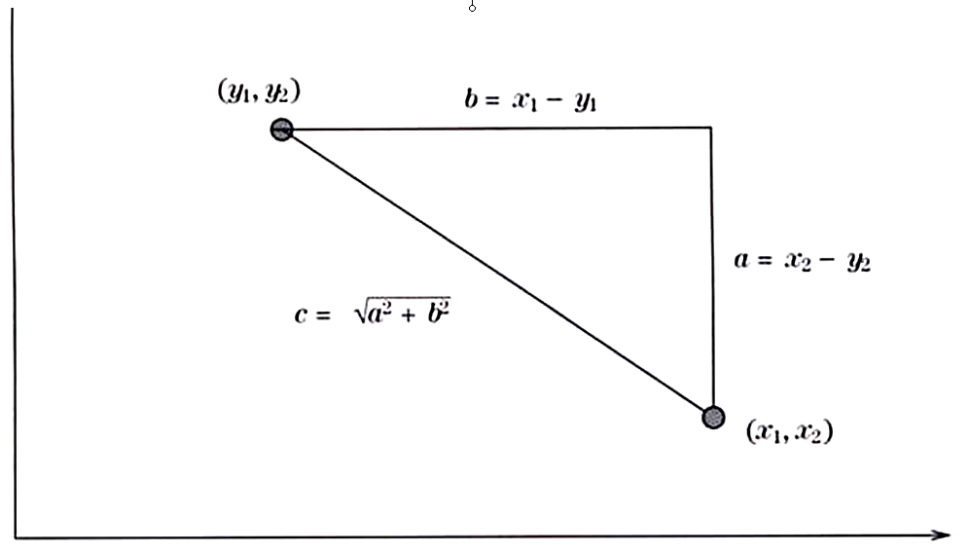
- **Minkowski distance:**

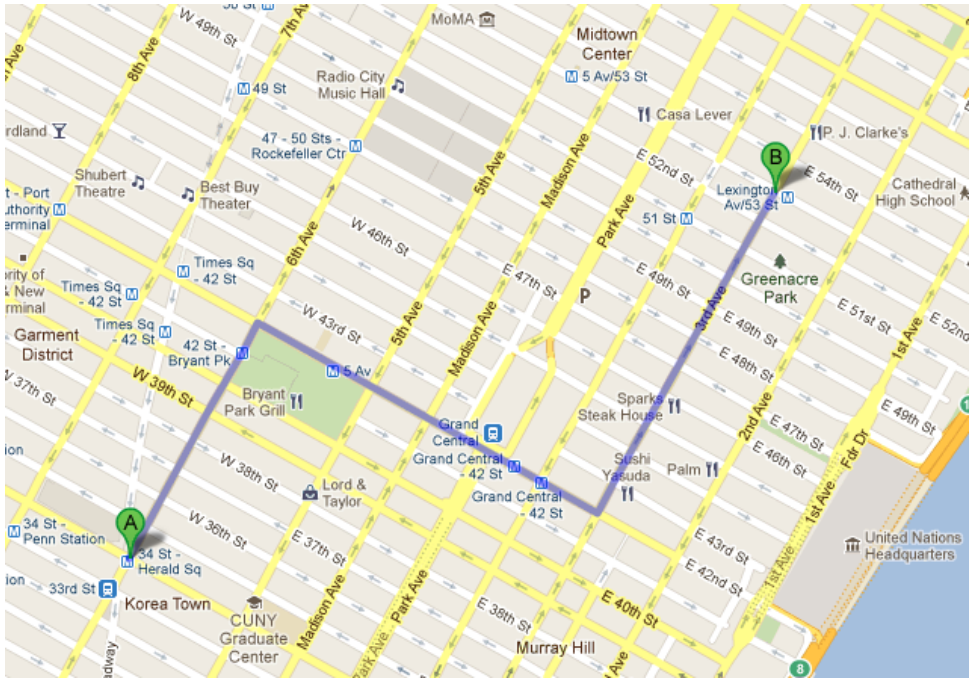
$$d(\mathbf{x},\mathbf{y}) = \left(\sum_{i=1}^{d} | x_i - y_i |^p \right)^{\frac{1}{p}}$$

- **Manhattan distance (city block distance):**

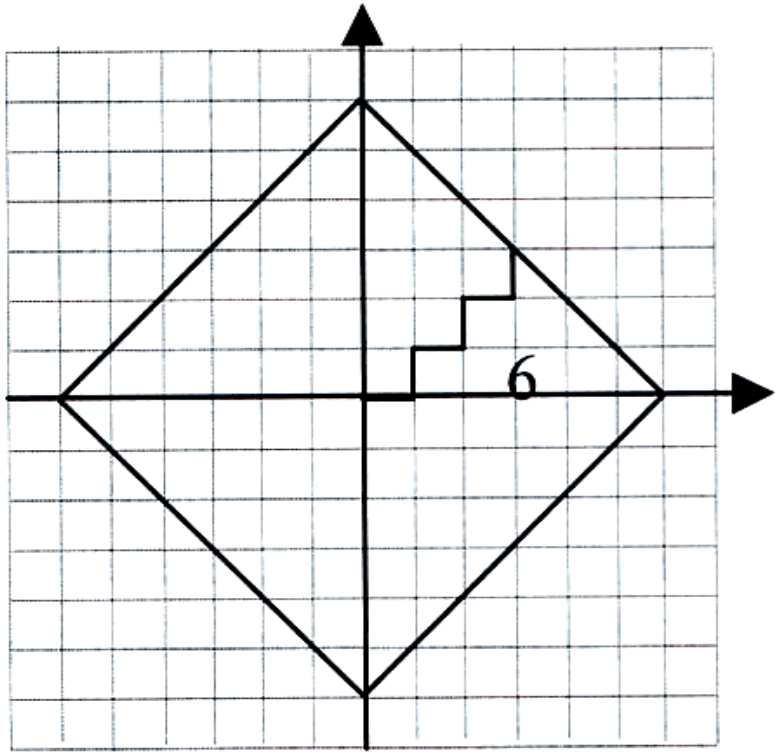
$$d(\mathbf{x},\mathbf{y}) = \sum_{i=1}^{d} | x_i - y_i |$$

- A particular case of the Minkowski distance, $p = 1$
- Measured in “blocks”





Comparison: Euclidean vs. Manhattan



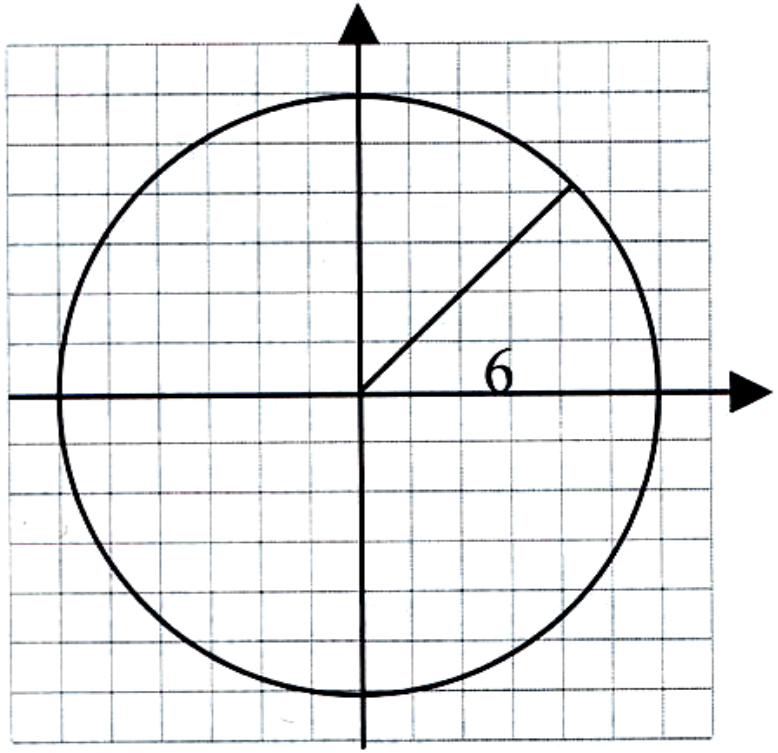
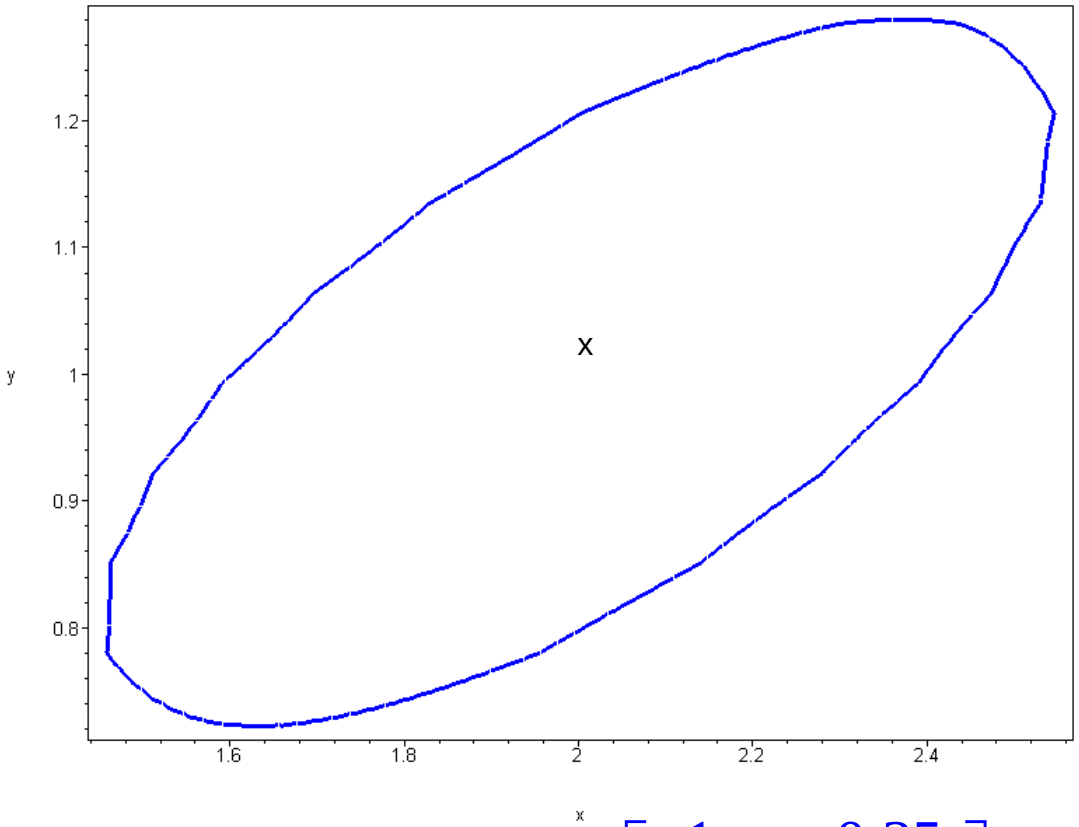


Image from Draghici et al.

Mahalanobis distance

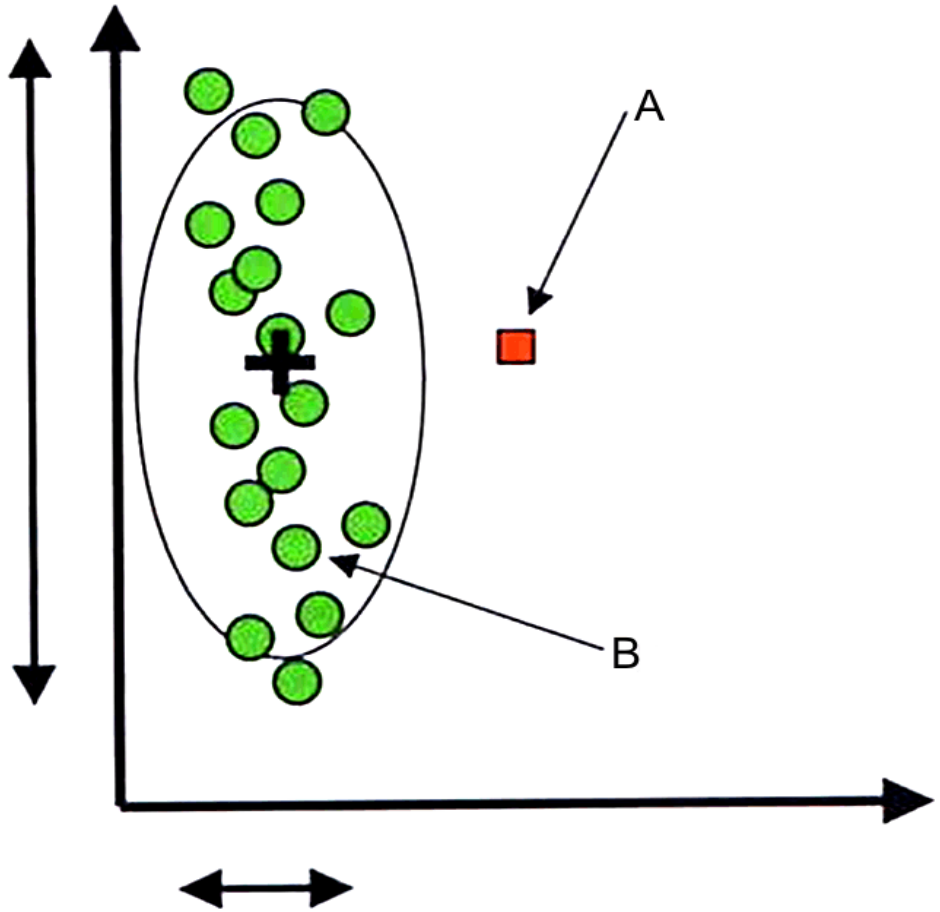
- A positive, definite, symmetric matrix: Σ

$$d(\mathbf{x},\mathbf{y}) = \left[(\mathbf{x}-\mathbf{y})^t \Sigma^{-1} (\mathbf{x}-\mathbf{y})\right]^{\frac{1}{2}}$$



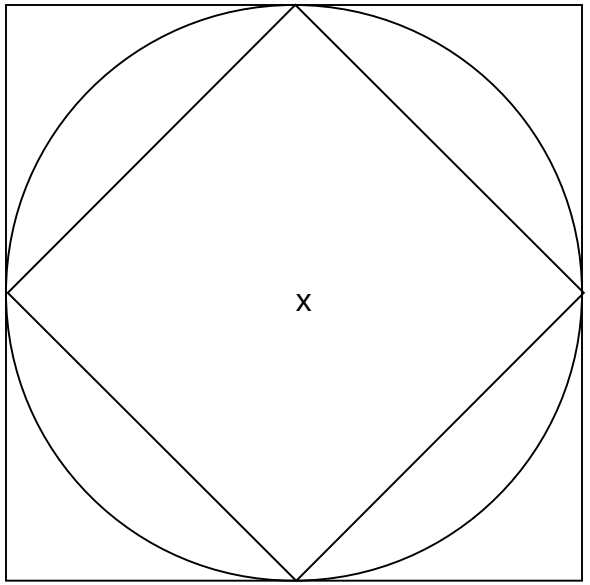
$$\Sigma := \begin{bmatrix} 1 & 0.35 \\ 0.35 & 0.2625 \end{bmatrix}$$

Uncorrelated random variables:



L_p-norm

- It's the Minkowski distance
- When:
  - $p = 2$: L_2-norm or Euclidean distance
  - $p = 1$: L_1-norm or Manhatan distance
  - $p = \infty$: L_∞-norm or sup distance



- **Chebychev distance:**
  - Largest difference between any two components

$$d(\mathbf{x},\mathbf{y}) = \max_i \left| x_i - y_i \right|$$

- **Angle distance:**
  - Angle between two vectors

$$d(\mathbf{x},\mathbf{y}) = \frac{\mathbf{x}^t\mathbf{y}}{\|\mathbf{x}\|\|\mathbf{y}\|}$$

Correlation distances

Pearson correlation distance between **x** and **y**

Linear correlation

$$d(\mathbf{x},\mathbf{y}) = 1 - \frac{\sum_{i=1}^{d}(x_i - \bar{x})(y_i - \bar{y})}{\left[\sum_{i=1}^{d}(x_i - \bar{x})^2\right]^{1/2}\left[\sum_{i=1}^{d}(y_i - \bar{y})^2\right]^{1/2}}$$

where $\bar{x} = \sum_{i=1}^{d} x_i$ and $\bar{y} = \sum_{i=1}^{d} y_i$

Spearman correlation distance between **x** and **y**

Nonlinear correlation

- Create two vectors **a** = [a_1,…, a_d] and **b =** [b_1,…, b_d],
- where a_i and b_i are the ranks of x_i and y_i respectively. Compute:

$$d(\mathbf{x},\mathbf{y}) = 1 - \frac{6\sum_{i=1}^{d}(a_i - b_i)^2}{d(d^2 - 1)}$$

Example

$$\mathbf{x} = \begin{bmatrix} 0.1 & 0.7 & 1.2 & 0.5 \end{bmatrix} \qquad \mathbf{y} = \begin{bmatrix} 3.5 & 4.2 & 5.7 & 6.2 \end{bmatrix}$$

$$\mathbf{a} = \begin{bmatrix} 1 & 3 & 4 & 2 \end{bmatrix} \qquad \mathbf{b} = \begin{bmatrix} 1 & 2 & 3 & 4 \end{bmatrix}$$

$$\sum_{i=1}^{d} (\mathrm{a_i} - \mathrm{b_i})^2 = (\mathbf{a}-\mathbf{b})^{\mathrm{t}}(\mathbf{a}-\mathbf{b}) = 0^2 + 1^2 + 1^2 + 2^2 = 6$$

$$d(\mathbf{x},\mathbf{y}) = 1 - \frac{6 \cdot 6}{4(16-1)} = 1 - \frac{36}{60} = 1 - 0.6 = 0.4$$

Jaccard index/distance

Let S_1 and S_2 be two sets, the Jaccard index:

$$J(S_1, S_2) = \frac{|S_1 \bigcap S_2|}{|S_1 \bigcup S_2|}$$

where $0 \leq J(S_1, S_2) \leq 1$

If S_1 and S_1 are both empty: $J(S_1, S_2) = 1$

Jaccard distance is defined as:

$$d_J = 1 - J(S_1, S_2)$$

Jaccard distance is used for nominal or textual features

For example, phone brand name (Apple, Samsung, Huawei, LG, Nokia, Blackberry), city, province, car make, etc.

Comparison of Distance Functions

- **Euclidean distance:** Most common distance function used by human beings – it's like an optimal way to "reach something"
- **Angle between vectors:** Considers *only* angle, and *not* magnitude of the vectors

  e.g. **x** = [1,1], **y** = [100,100] ($\cos \pi/2 = 0$)
- **Orthogonal vectors:** Two vectors are orthogonal iff angle distance is cos 0 = 1

  e.g. **x** = [1,1], **y** =[-100,100]
- **Correlation distance:** Looks for similar *variations*, and not for actual numerical values

  e.g. **x** = [1,2,3,4,5], **y** = [10,20,30,40,50], **z** = [5,4,3,2,1]
  - d(x,y) will be small, while d(x,z) will be large
  - Euclidean distance is the opposite

- **Mahalanobis distance:** Takes correlation of r.v. into consideration
  If r.v. are uncorrelated, takes covariances into account
- **Manhattan:** More appropriate for discrete r.v.
- **Minkowski:** A generalization of the Euclidean distance.
- **Chebychev:** Focuses on the *most important* difference.

  e.g. **x** = [1,2,3,4], **y** = [2,3,4,5]

  x = [1,2,3,4], **y** = [1,2,3,6]

Edit distance

- When samples are not real-valued vectors, but strings…
- Can use *edit distance*
- In this case, there is no *obvious* measure for the distance
- For example,
- It is not clear whether *abbccc*

  is closer to *aabbcc*

  or to *abbcccb*

- **Example:**

  X = *excused* is transformed into

  Y = *exhausted*

- Steps:
  - First, substitute *h* for *c*, yielding
    X = *exhused*
  - Second, insert *a*, obtaining
    X = *exhaused*
  - Third, insert *t*, obtaining
    X = *exhausted*
  - Since the cost of each operation is 1, the distance between these two strings is 3

Edit distance – cont'd

- Given two strings, **x** and **y**.
- The edit distance is based on three operations:
- **Substitution:** A character in **x** is replaced by the corresponding character in **y**.
- **Insertion:** A character in **y** is inserted into **x**.
- **Deletion:** A character in **x** is deleted.
- Another operation:
  - **Interchange:** Exchange two neighbor characters in **x**
    e.g., transform **x** = *abc* into **y** = *cba*
- A dynamic programming algorithm for the edit distance follows…
- It uses a matrix of costs, or "distances", and
- δ(**x**[*i*],**y**[*j*]), which is 1 if **x**[*i*] = **y**[*j*], and 0 otherwise

Algorithm Edit Distance

```
C[0,0] ← 0
for i ← 0 to m                    // length of x
    C[i,0] ← i;
endfor
for i ← 0 to n                    // length of y
    C[0,j] ← j;
endfor
for i ← 0 to m
    for i ← 0 to n
        C[i,j] ← min{ C[i-1,j]+1, C[i, j-1]+1,
                      C[i-1,j-1]+1- δ(x[i],y[j]) }
    endfor
endfor
return C[m,n]
```

Insertion

Deletion

No change / exchange

$$\delta(\mathbf{x}[i],\mathbf{y}[j]) = \begin{cases} 1 \text{ if } \mathbf{x}[i] = \mathbf{y}[j] \\ 0 \text{ otherwise} \end{cases}$$

Complexity: O(*mn*)

Example

C		e	x	h	a	u	s	t	e	d
		1	2	3	4	5	6	7	8	9
e	1	0	1	2	3	4	5	6	7	8
x	*2*	1	0	1	2	3	4	5	6	7
c	*3*	2	1	1	2	3	4	5	6	7
u	*4*	3	2	2	2	2	3	4	5	6
s	*5*	4	3	3	3	3	2	3	4	5
e	*6*	5	4	4	4	4	3	3	3	4
d	*7*	6	5	5	5	5	4	4	4	3

- Algorithm EditDistance runs in O(*nm*)
- Function δ can be generalized to:
  - Other than 0-1 cost function
  - Can consider different costs for different symbols
- Generalization includes protein/DNA sequence alignment

Algorithm *EditDistance*(*X,Y*)
Input: Strings *X* and *Y* of length *n* and *m* respectively
Output: Array C containing prefix edit distances
C[0,0] ← 0
for *i* ← 0 **to** *m* // length of X
  C[*i*,0] ← *i*
for *i* = 0 to *n* // length of X
  C[0,*j*] ← *j*
for *i* = 0 **to** *m*
  for *i* = 0 **to** *n*
    C[*i*,*j*] = min{*C*[*i*-1,*j*]+1, *C*[*i*, *j*-1]+1,
      C[*i*-1,*j*-1]+1- δ(*Xi*,*Yj*) }
return *C*[*m*,*n*]

Bio-sequence alignment

- General case: global alignment
- Bio-sequences can represent:
  - DNA, RNA, proteins
- Uses a scoring system that assigns:
  - score values for **each** pair of symbols
  - e.g. nucleic acid or amino acid pairs
  - penalty values to single gaps
- Score = sum of pair scores – sum of gap penalties

Example: X = **TGATAGCCAG** Y = **AGAGCA**

```
T – G A T A G C C A G
– A G A G – – C – A –
```

Score = **–4–4+6+6+2–4–4+6–4+6–4 = 2**

Score for optimal alignment of X_i and Y_j given by:

$$c_{ij} = \max \{ c_{i-1,j-1} + s(x_i,y_j), \max k \geq 1 (c_{i-k,j} - w_k), \max t \geq 1 (c_{i,j-t} - w_t) \} \quad (1)$$

Score for optimal alignment of X and $Y = c_{mn}$

Algorithm *SeqAlign(X, Y)*

$c_{i0} \leftarrow -w_i$, $1 \leq i \leq m$

$c_{0j} \leftarrow -w_j$, $1 \leq j \leq n$

for i = 1 **to** m

  for j = 1 **to** n

    Compute c_{ij} as in (1)

    Update "traceback" matrix D (with ←↑↖)

  return c_{mn}

Worst-case running time: O(nm)

	A	C	G	T
A	6	2	2	2
C	2	6	2	2
G	2	2	6	2
T	2	2	2	6

DNA sequence alignment – example

	λ	T	G	A	T	A	G	C	C	A	G
λ	0	-4	-8	-12	-16	-20	-24	-28	-32	-36	-40
A	-4	2	-2	-2	-6	-10	-14	-18	-22	-26	-30
G	-8	-2	8	4	0	-4	-4	-8	-12	-16	-20
A	-12	-6	4	14	10	6	2	-2	-6	-6	-10
G	-16	-10	0	10	16	12	12	8	4	0	0
C	-20	-14	-4	2	12	18	14	18	14	10	6
A	-24	-18	-8	2	8	18	20	16	20	20	16

	A	C	G	T
A	6	2	2	2
C	2	6	2	2
G	2	2	6	2
T	2	2	2	6

Optimal alignment (←):

```
A G A - - G C - A -
T G A T A G C C A G
```

c_{mn} = 2+6+6−4−4+6+6−4+6−4 = 16

- Dynamic programming can obtain more than one solution
- e.g., following the pink path
- Algorithm can be extended to protein sequence alignment
  - Considers 20 amino acids and other score matrices