



CS 540 Introduction to Artificial Intelligence

Classification – Nearest Neighbors & Naïve Bayes

University of Wisconsin-Madison

Fall 2026 Sections 1 & 2

Announcements

- **Homework:**
 - HW4 due **Friday October 9th at 11:59 PM**
 - HW5 will also be released on Friday
- **Midterm cheatsheet:** one double-sided 8.5 x 11 page permitted, but we aim to make the midterm doable without one

- **Class roadmap:**

Machine Learning: k-NN & NB	Supervised Learning
Machine Learning: Neural Networks I (Perceptron)	
Machine Learning: Neural Networks II	
Machine Learning: Neural Networks III	

Outline

- Review linear and logistic regression
- Nearest Neighbors
 - parametric vs. **nonparametric** models
- Naïve Bayes
 - discriminative vs. **generative** models

Review: Linear Regression

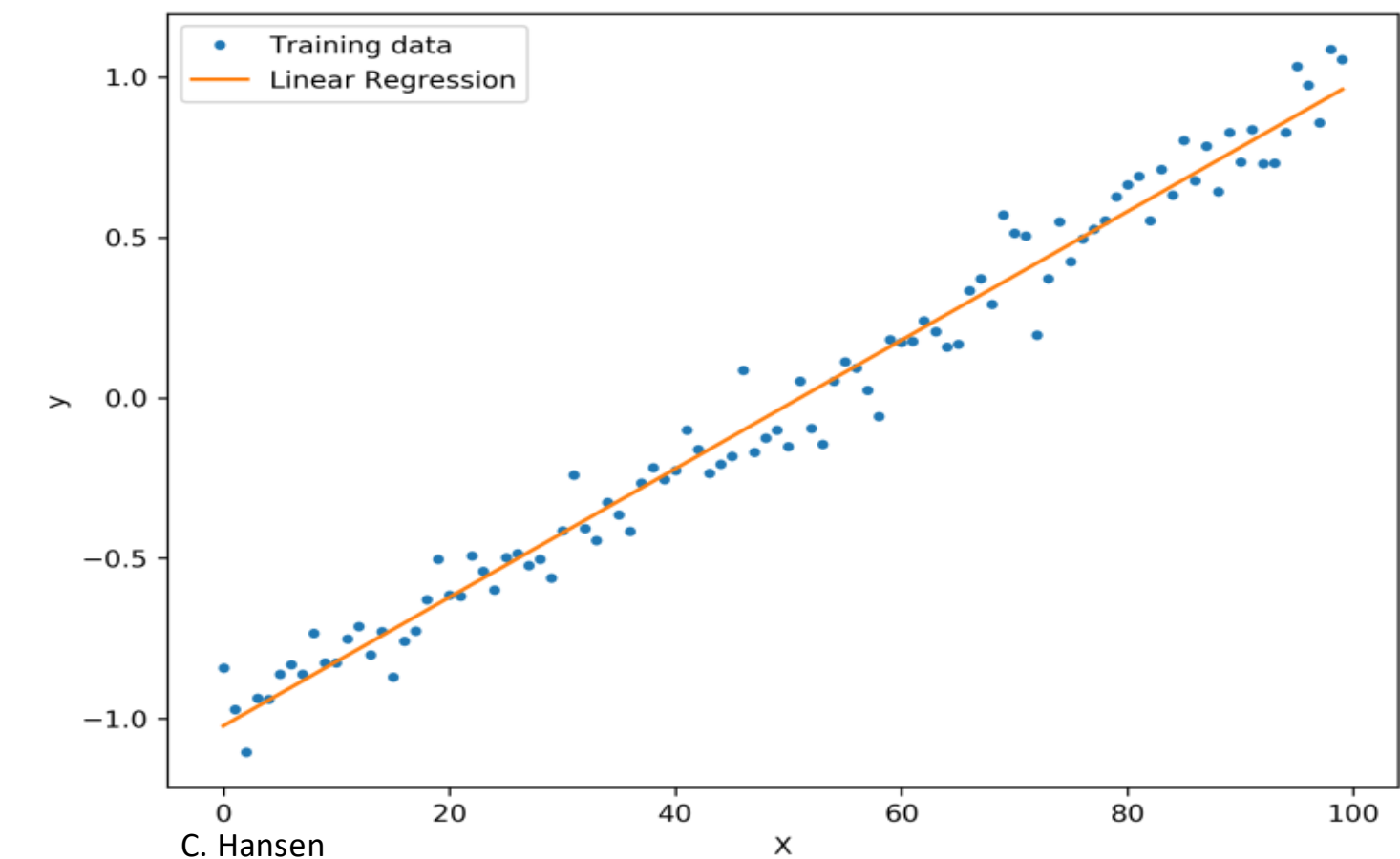
Simplest form of regression: find a line that fits the data

- When x is a scalar: $f_{(a,b)}(x) = ax + b$
- When x is a vector:

$$f(x) = \theta_0 + \theta_1 x_1 + \theta_2 x_2 + \cdots + \theta_d x_d = \theta_0 + x^T \theta$$

Give x a “dummy dimension” to simplify notation

Old $x = \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_d \end{bmatrix}$ New $x = \begin{bmatrix} 1 \\ x_1 \\ x_2 \\ \vdots \\ x_d \end{bmatrix}$ $f(x) = \begin{bmatrix} 1 & x_1 & x_2 & \cdots & x_d \end{bmatrix} \begin{bmatrix} \theta_0 \\ \theta_1 \\ \theta_2 \\ \vdots \\ x_d \end{bmatrix} = \langle x, \theta \rangle = x^T \theta$



Review: Linear Regression

Problem Setup

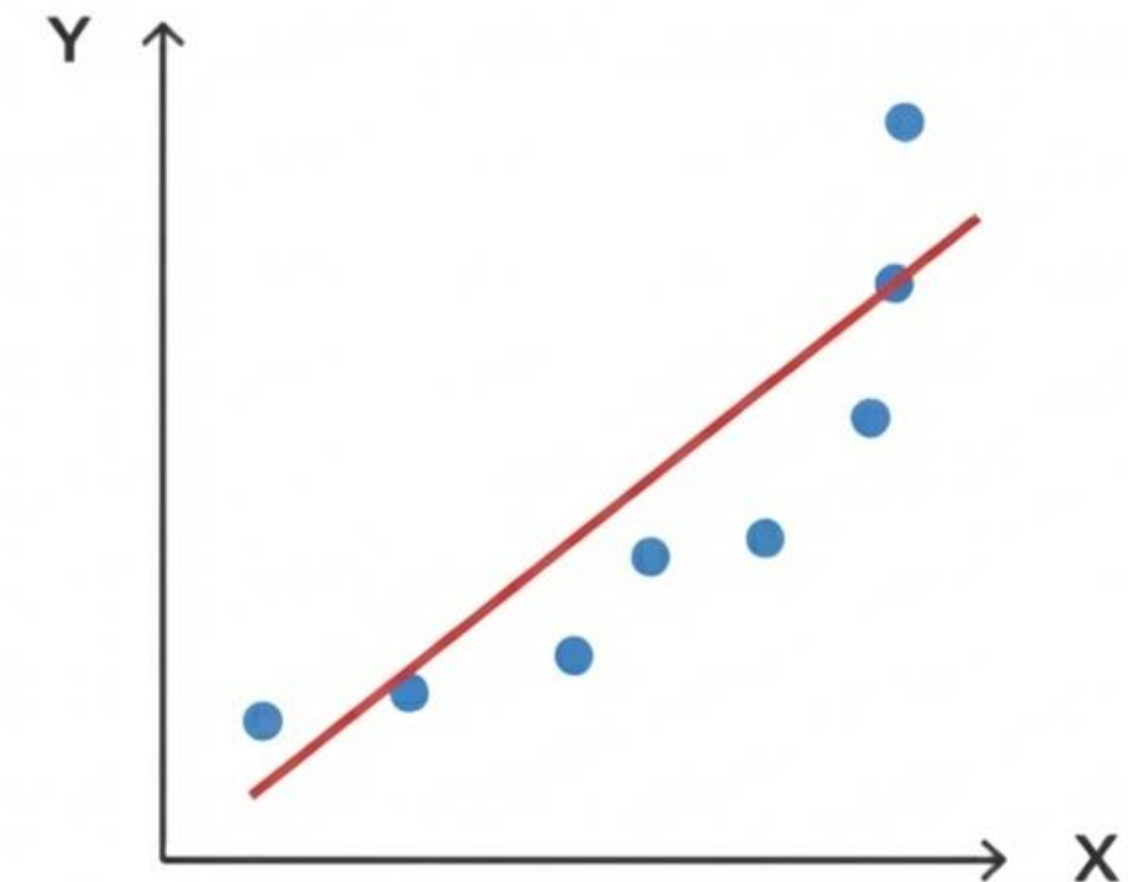
- Train set $(\mathbf{x}_1, y_1), (\mathbf{x}_2, y_2), \dots, (\mathbf{x}_n, y_n)$
- Take train features and make it a $n \times (d+1)$ matrix, and y a vector:

$$X = \begin{bmatrix} x_1^T \\ \dots \\ x_n^T \end{bmatrix} \quad y = \begin{bmatrix} y_1 \\ \dots \\ y_n \end{bmatrix}$$

- Loss: $\frac{1}{n} \|X\theta - y\|^2$
- Closed form solution: $\hat{\theta} = (X^T X)^{-1} X^T y$
- Another way to find the best parameters is using Gradient Descent and its variants

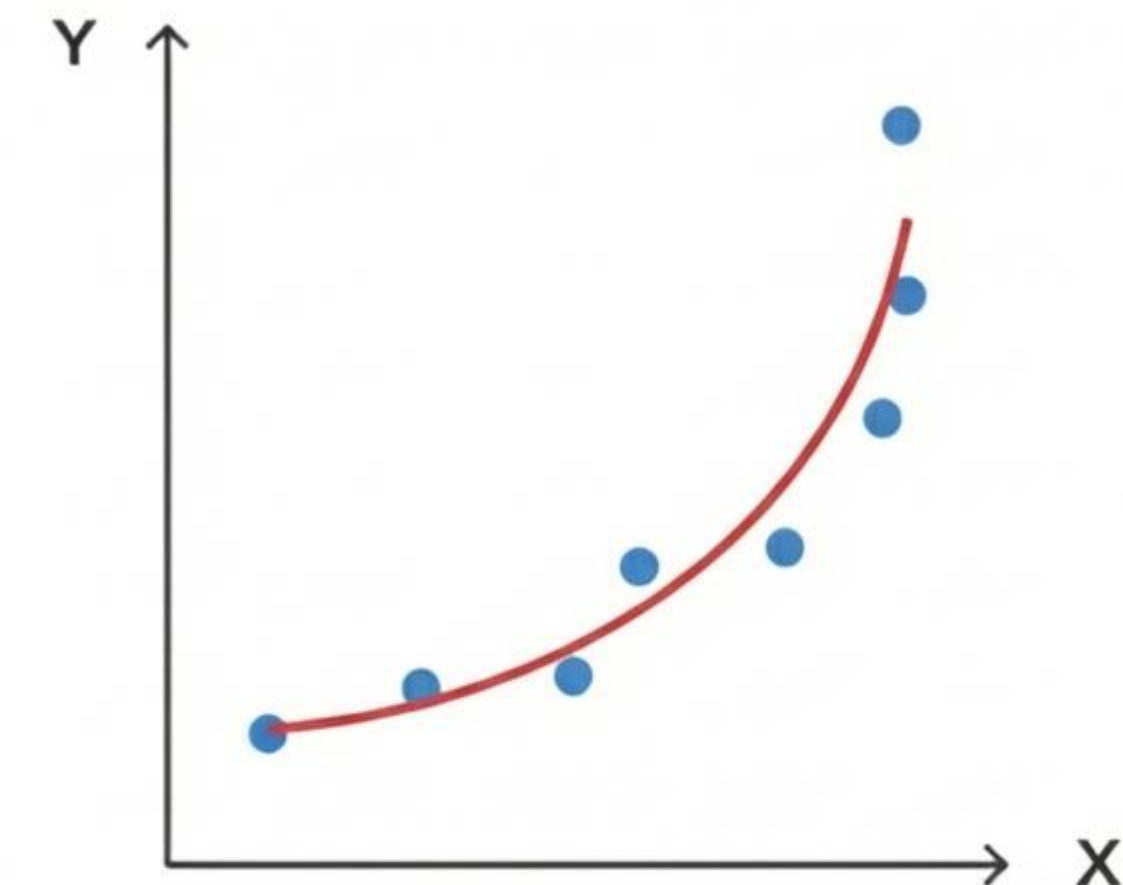
Polynomial Regression

- Linear regression tries to draw a straight line on your data.



- If data naturally curves, we may need to use polynomial regression.

$$f(x) = \theta_0 + \theta_1 x + \theta_2 x^2 + \dots + \theta_n x^n$$



Review: Linear Regression

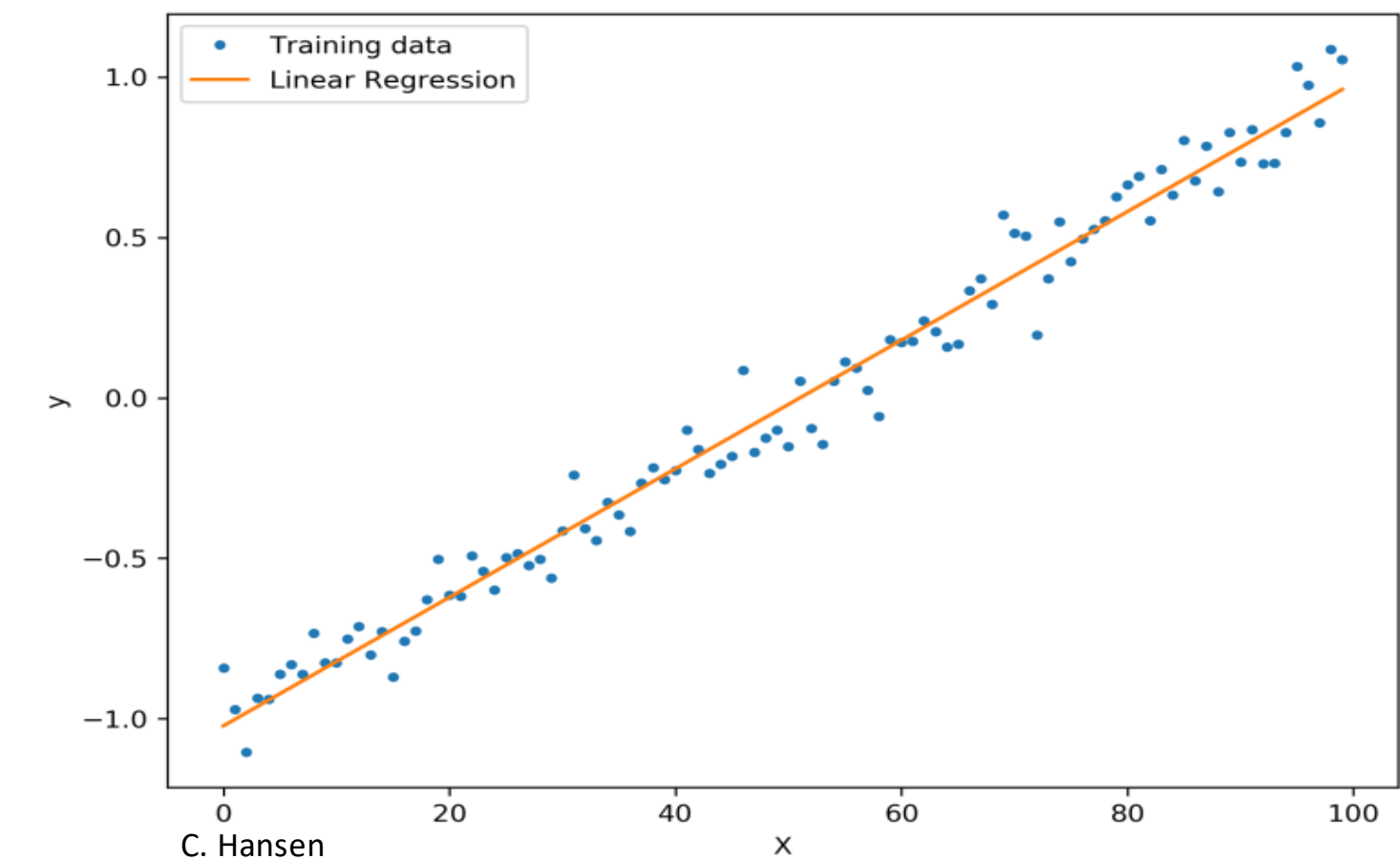
Simplest form of regression: find a line that fits the data

- When x is a scalar: $f_{(a,b)}(x) = ax + b$
- When x is a vector:

$$f(x) = \theta_0 + \theta_1 x_1 + \theta_2 x_2 + \cdots + \theta_d x_d = \theta_0 + x^T \theta$$

Give x a “dummy dimension” to simplify notation

Old $x = \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_d \end{bmatrix}$ New $x = \begin{bmatrix} 1 \\ x_1 \\ x_2 \\ \vdots \\ x_d \end{bmatrix}$ $f(x) = \begin{bmatrix} 1 & x_1 & x_2 & \cdots & x_d \end{bmatrix} \begin{bmatrix} \theta_0 \\ \theta_1 \\ \theta_2 \\ \vdots \\ x_d \end{bmatrix} = \langle x, \theta \rangle = x^T \theta$



Review: Logistic regression

- **Logistic regression = sigmoid conditional distribution + MLE**

- More precisely:

- Give training data iid from some distribution D ,

- **Train:**
$$\max_{\theta} \ell(\theta; X, Y) = \min_{\theta} - \sum_{i=1}^n \log p_{\theta}(y_i | x_i)$$

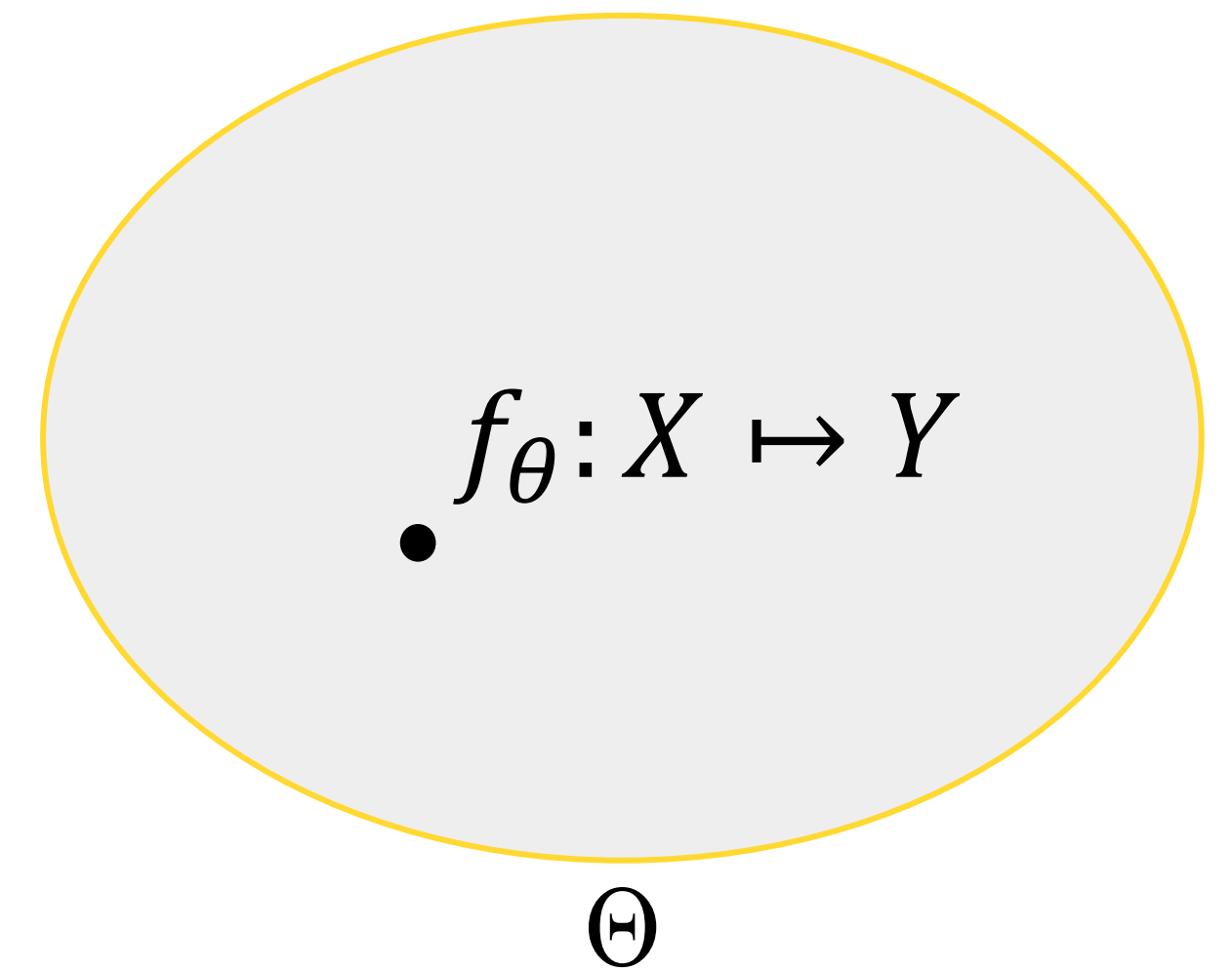
- **Test:** output label probabilities

$$P_{\theta}(y = 1 | x) = \sigma(\theta^T x) = \frac{1}{1 + \exp(-\theta^T x)}$$

Big picture: Nonparametric learning

So far, all learning approaches we have studied have been **parametric**:

- each possible model is defined via a setting of a **fixed-dimensional parameter vector**
- learning proceeds by **finding a parameter** that yields a good model according to some loss
- Examples: linear models, neural networks



Nonparametric methods:

- models are defined by an unbounded number of parameters, typically determined by the data itself
- Examples: **nearest neighbors**, decision trees, kernel density estimation

Example 1: Predict if a user likes a song or not



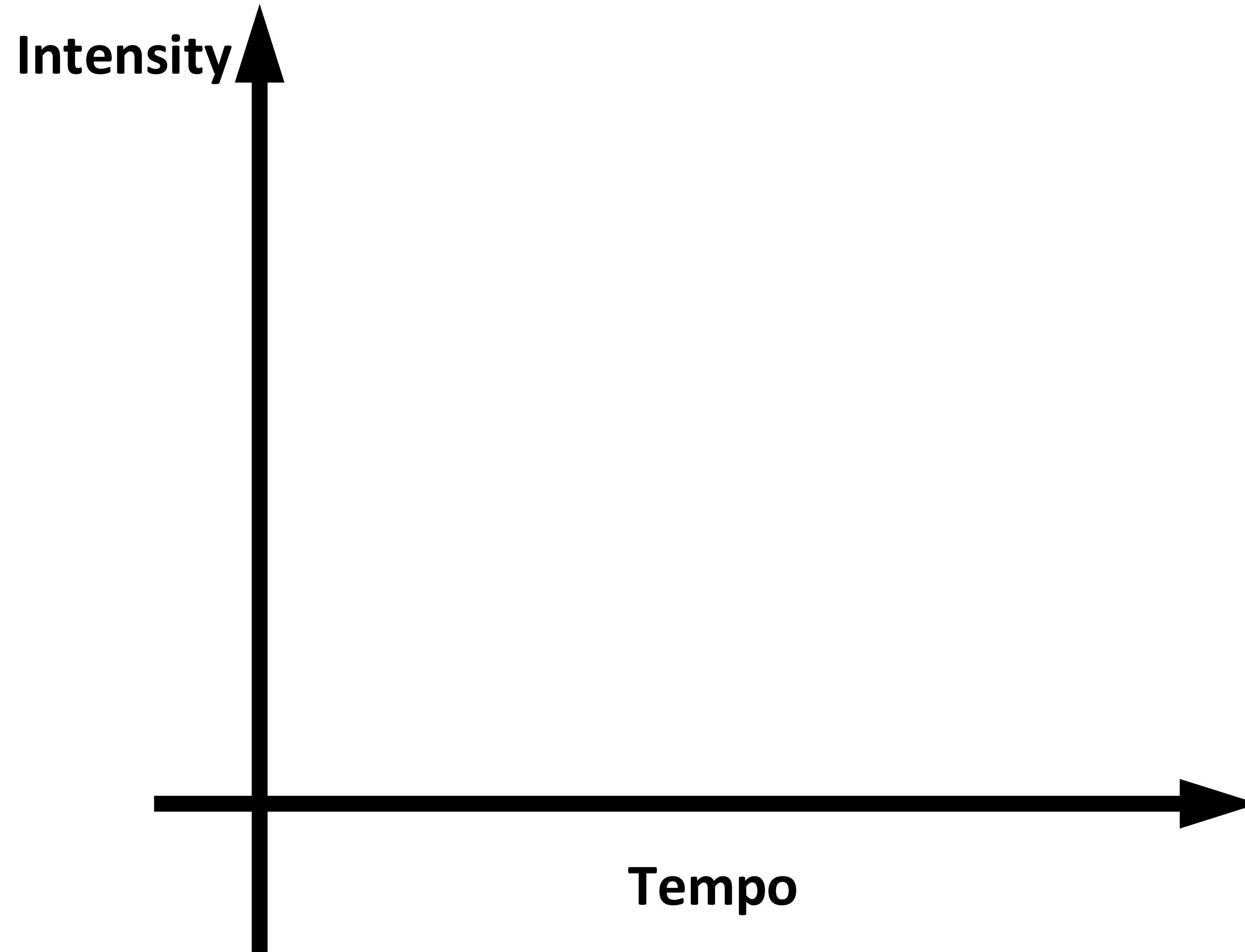
model



Example 1: Predict if a user likes a song or not



User Sharon



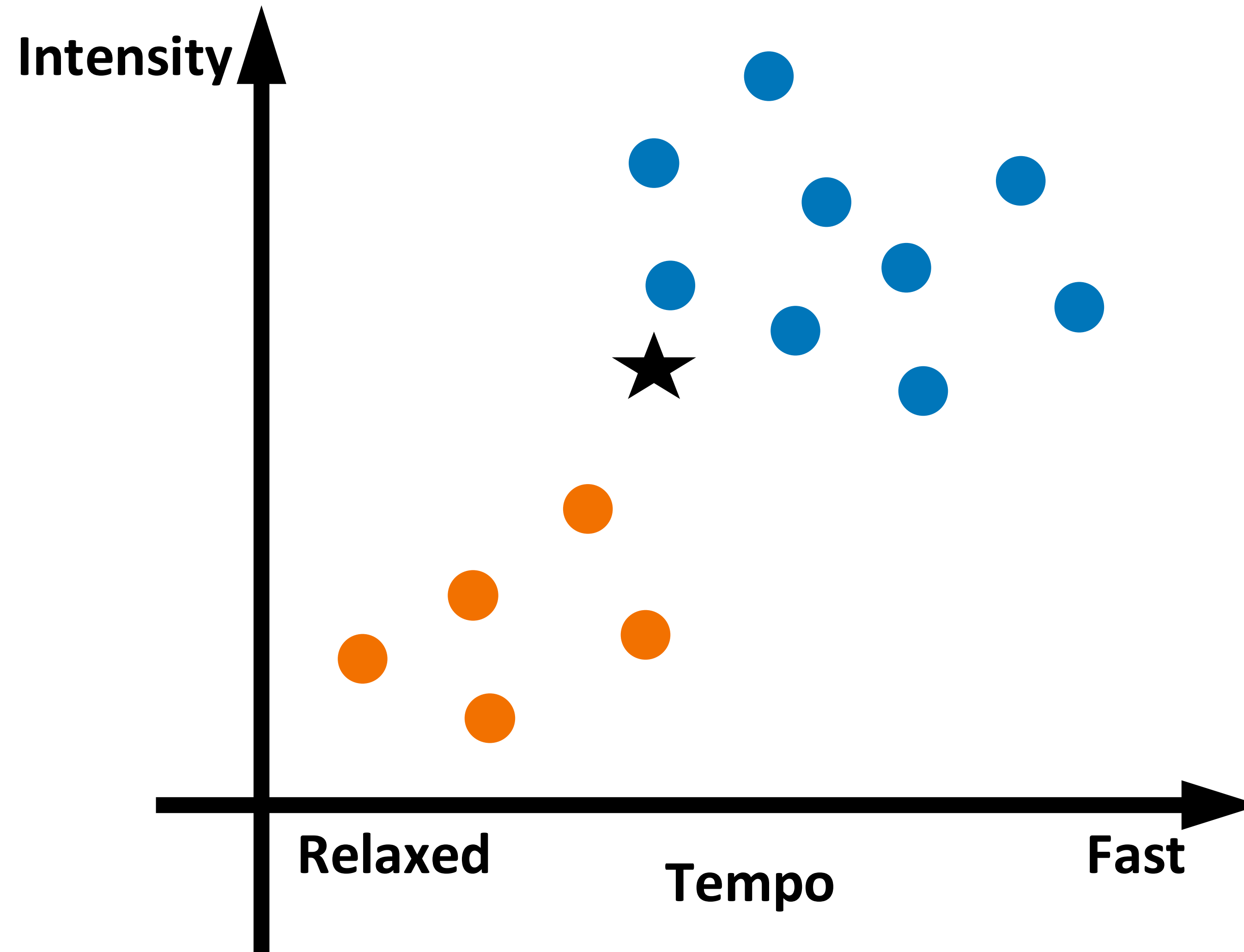
Example 1: Predict if a user likes a song or not



User Sharon

● Dislike

● Like



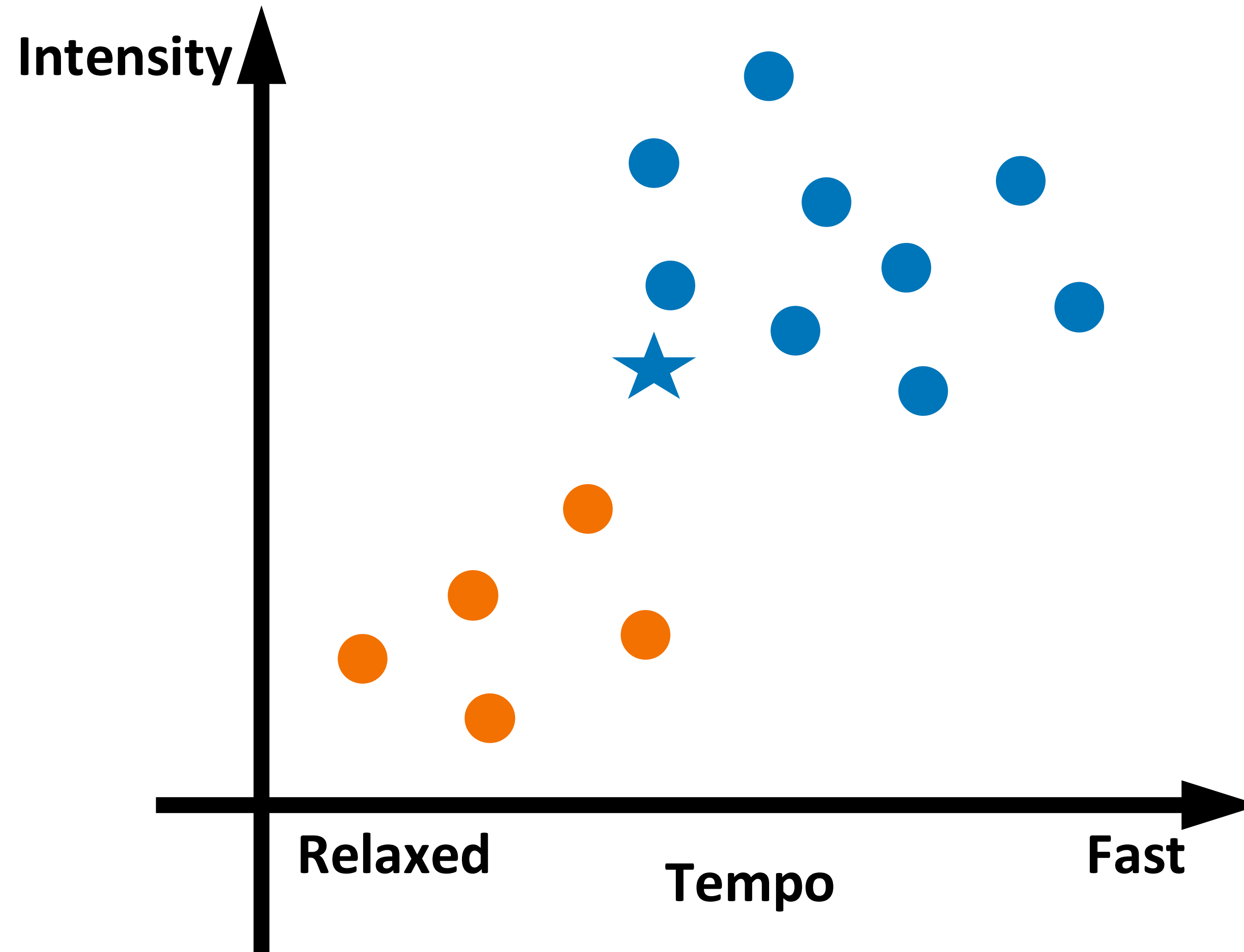
Example 1: Predict if a user likes a song or not



User Sharon

● Dislike

● Like



K-nearest neighbors for classification

Input:

- **Training data** $(\mathbf{x}_1, y_1), (\mathbf{x}_2, y_2), \dots, (\mathbf{x}_n, y_n)$
- **Distance function** $d(\mathbf{x}_i, \mathbf{x}_j)$;
- **number of neighbors** k ;
- **test data** $\mathbf{x}^*$

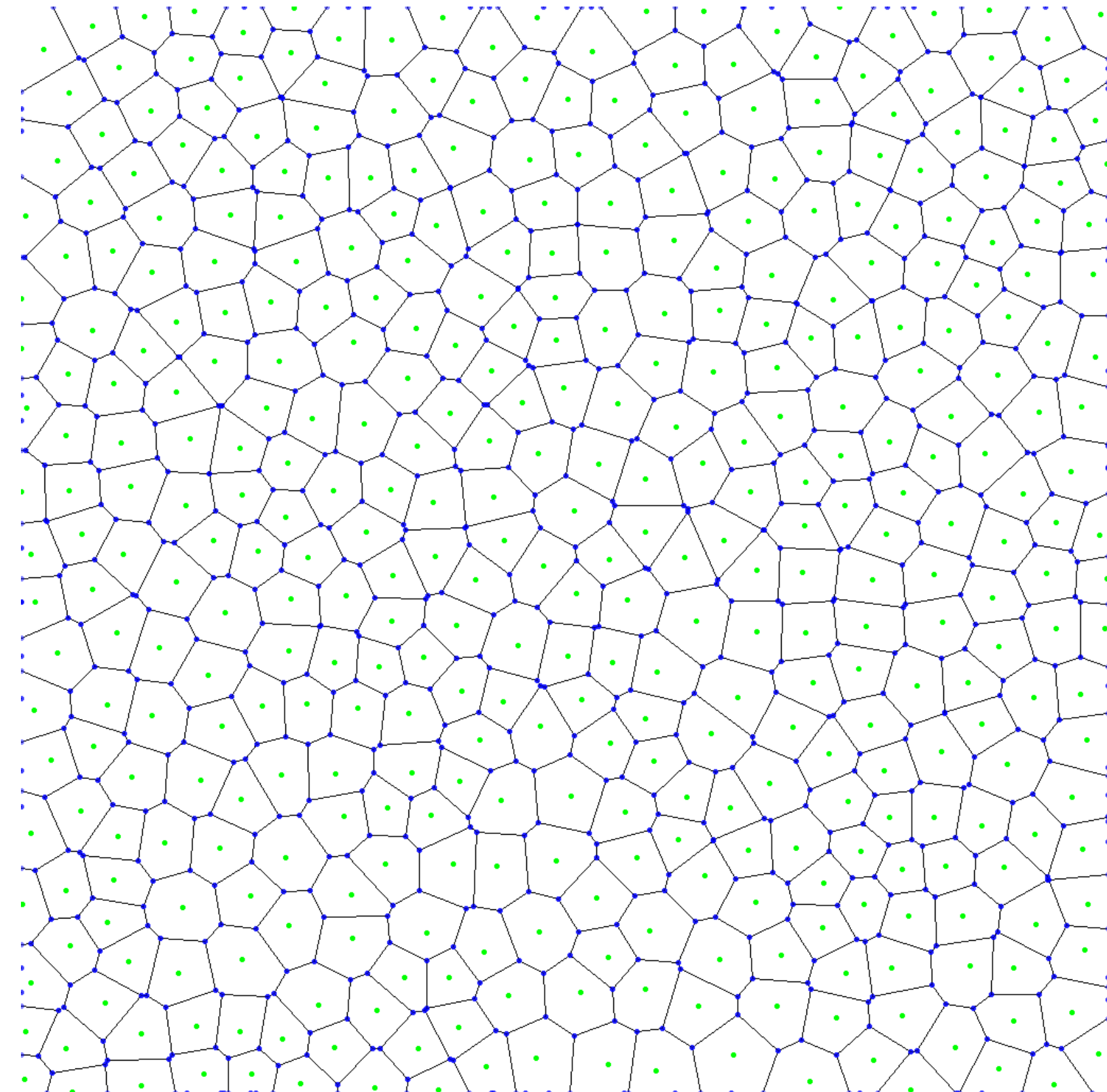
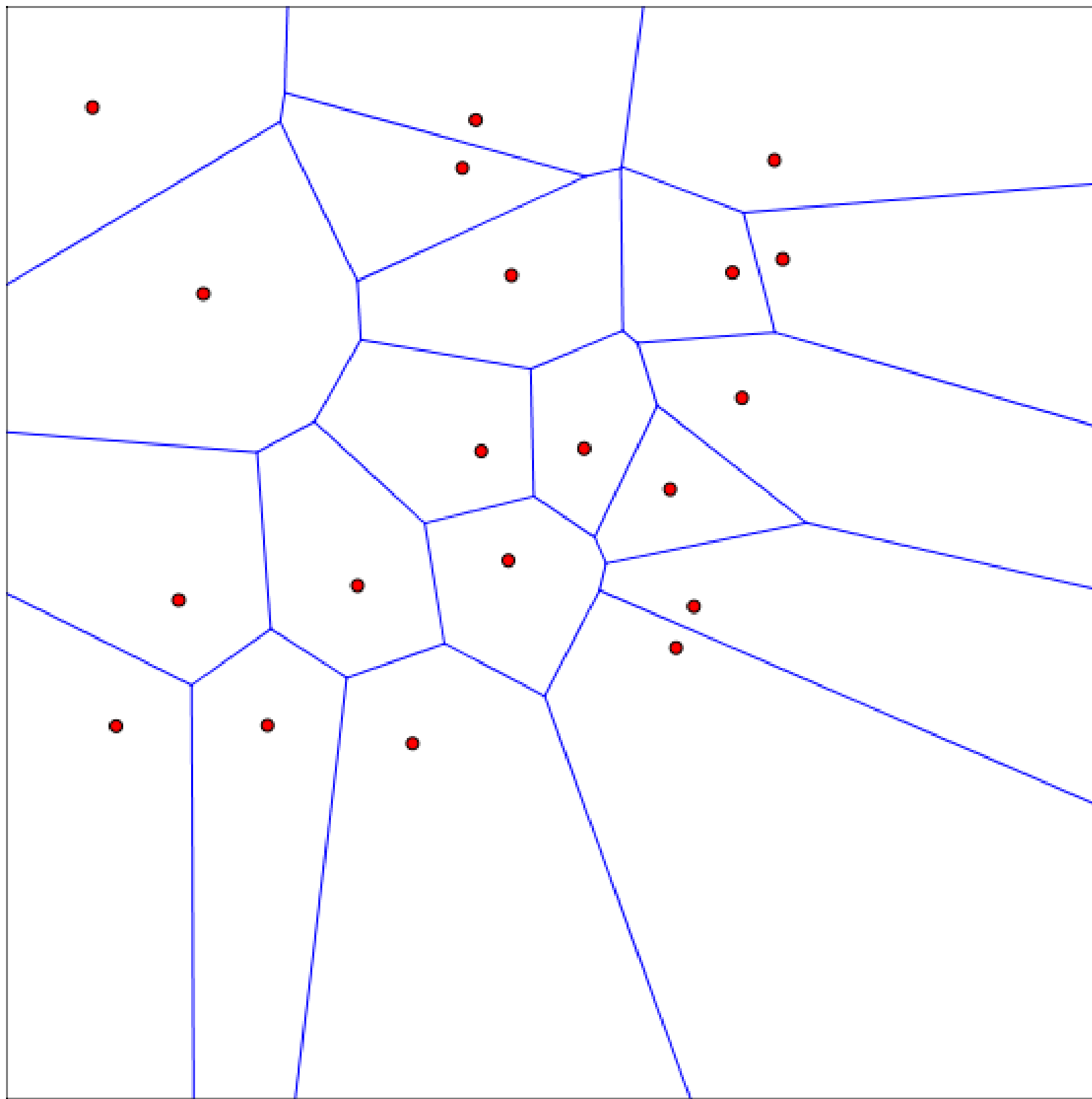
Prediction procedure:

1. Find the k training instances $\mathbf{x}_{i_1}, \dots, \mathbf{x}_{i_k}$ closest to $\mathbf{x}^*$ under $d(\mathbf{x}_i, \mathbf{x}_j)$
2. Output y^* , the majority class of $y_{i_1}, \dots, y_{i_k}$. Break ties randomly.

1NN: Decision Regions

Defined by “**Voronoi Diagram**”

- Each cell contains points closer to a particular training point



k-Nearest Neighbors: Distances

Discrete features: Hamming distance

$$d_H(x^{(i)}, x^{(j)}) = \sum_{a=1}^d 1\{x_a^{(i)} \neq x_a^{(j)}\}$$

Continuous features:

• Euclidean distance: $d(x^{(i)}, x^{(j)}) = \left(\sum_{a=1}^d (x_a^{(i)} - x_a^{(j)})^2 \right)^{\frac{1}{2}}$

• L1 (Manhattan) distance: $d(x^{(i)}, x^{(j)}) = \sum_{a=1}^d |x_a^{(i)} - x_a^{(j)}|$

k-Nearest Neighbors: Distances (cont'd)

Mixed distances: Might have features of both, discrete and continuous, types:

- Sum two types of distances component

Often, we will need **standardization**, (e.g. normalize individual distances to maximum value of 1), to ensure that all distances are in the same scale.

k-Nearest Neighbors: Standardization

Typical in data science applications. Recipe:

- Compute empirical mean/stddev for a feature (in train set)

$$\mu_a = \sum_{i=1}^n x_{i,a} \qquad \sigma_a = \left(\frac{1}{n} \sum_{i=1}^n (x_{i,a} - \mu_a)^2 \right)^{\frac{1}{2}}$$

- Standardize features before processing: $\tilde{x}_{i,a} = \frac{x_{i,a} - \mu_a}{\sigma_a}$
 - Only use training points to compute mean/stddev
 - Do the same for test points! (via mean/stddev computed using training points)
 - Very useful for parametric models as well

How to pick k , the number of neighbors

- Split data into training and **tuning (i.e validation) sets**
- Classify tuning set with different k
- Pick k that produces least tuning-set error
- Report error on the test set

(Shuffle whole dataset first)



k-Nearest Neighbors: Regression

Training/learning: given

$$\{(x^{(1)}, y^{(1)}), (x^{(2)}, y^{(2)}), \dots, (x^{(m)}, y^{(m)})\}$$

Prediction: for x , find k most similar training points and return

$$\hat{y} = \frac{1}{k} \sum_{i=1}^k y^{(i)}$$

- i.e. output the mean label of the closest k points

Instance-Based Learning: Strengths & Weaknesses

Strengths

- Easy to explain predictions
- Simple to implement and conceptualize.
- No training!
- Often good in practice

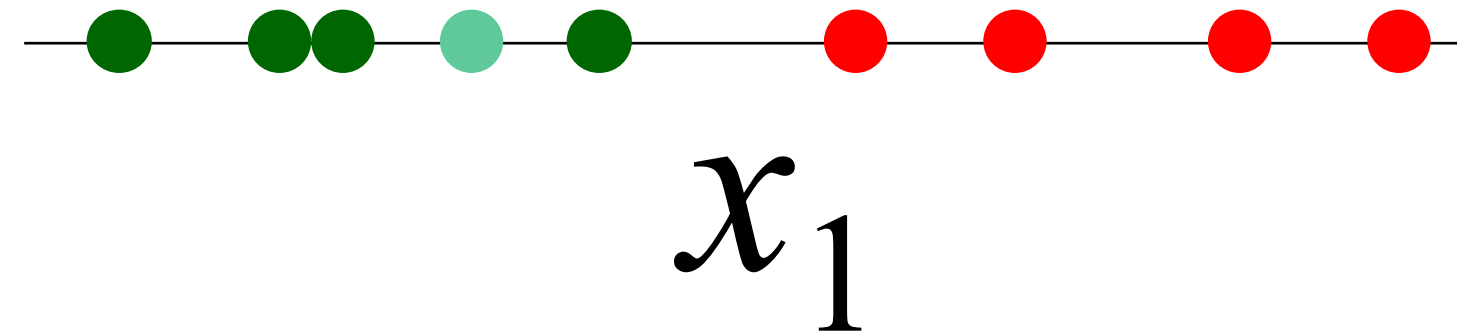
Weaknesses

- Sensitive to irrelevant + correlated features (next slide)
- Prediction stage can be expensive
- No “model” to interpret

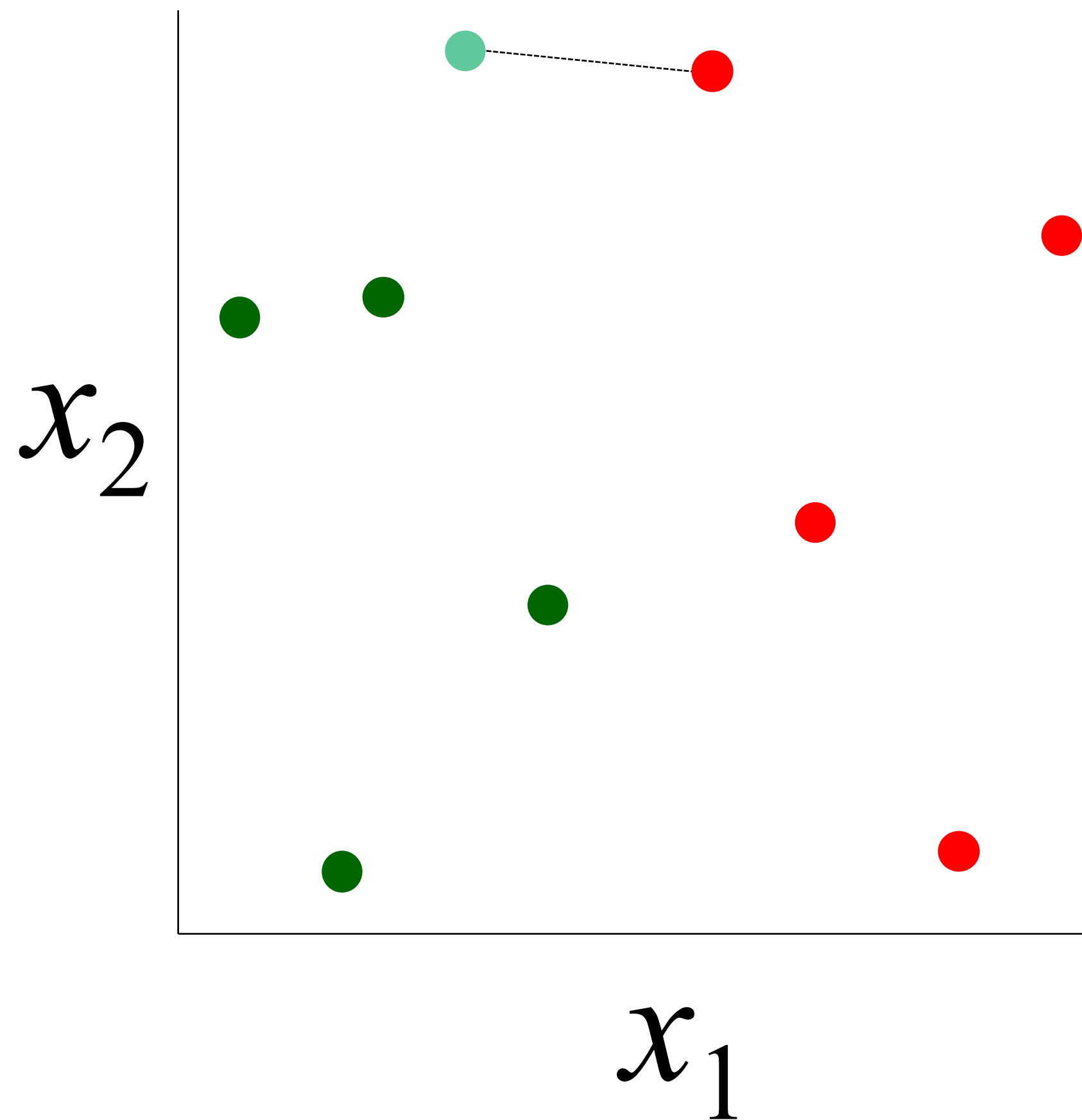
Irrelevant Features

One relevant feature x_1 :

1-NN rule classifies each instance correctly



**Effect of an irrelevant feature x_2
on distances and nearest
neighbors**



Big picture: discriminative vs. generative

So far, all supervised learning approaches we have studied have been **discriminative**: the model $f_\theta: X \mapsto Y$ directly predicts the label or its probability given the features.

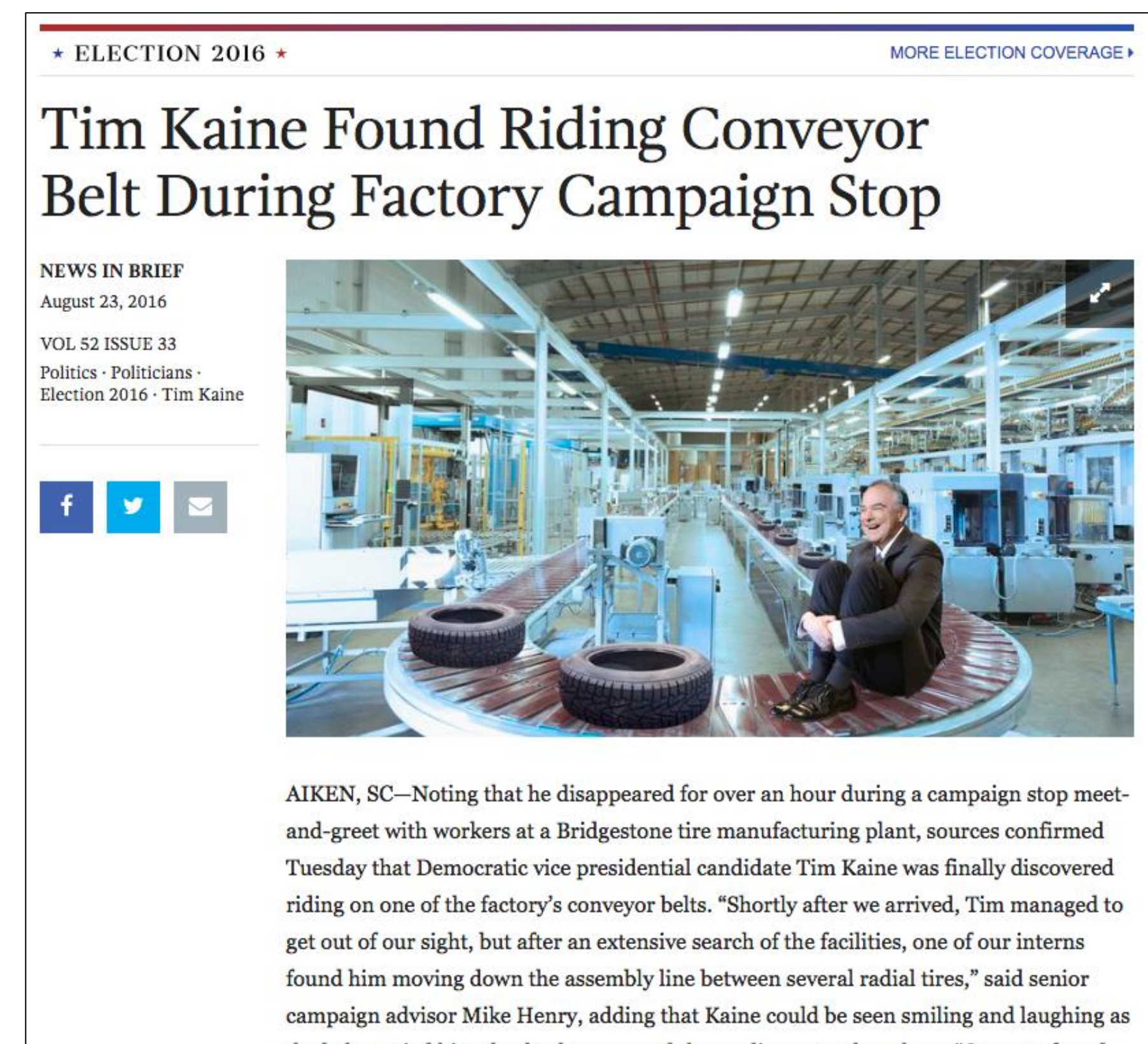
- examples: linear models, nearest neighbors
- strengths: (usually) easier-to-train, targets exact performance measure

The **generative** approach tells a story of how the label was generated: the parameter θ models $p(y|x)$ indirectly via a model of $p(x|y)$, $p(y)$, and Bayes' Rule

- examples: Naïve Bayes, Gaussian Mixture Models
- strengths: interpretability and handling of missing data

Application: Parody Detection

Goal: given a text document of featurized as a **binary** bag-of-words vector of vocabulary size K , model the probability that it is a parody document:



A Generative Model

Generative story:

1. Flip a weighted coin Y
2. If heads, sample a document X from the parody distribution
3. If tails, sample a document X from the regular distribution

$$P(X, Y) = P(X|Y)P(Y)$$

A Generative Model (cont'd)

Generative story (in more detail):

1. Flip a weighted coin (Y)
2. If heads, roll a die with 2^K sides to generate a document vector $\mathbf{X}$ from the parody distribution
3. If tails, roll a **different die** with 2^K sides to generate a document vector $\mathbf{X}$ from the regular distribution

$$P(X_1, \dots, X_K, Y) = P(X_1, \dots, X_K | Y) P(Y)$$

However, this requires modeling $2 * 2^K$ probabilities!

Naïve Bayes: Core Assumption

How do we fix this problem?

- Conditional **independence** of features:

$$\begin{aligned} P(X_1, \dots, X_K, Y) &= P(X_1, \dots, X_K | Y) P(Y) \\ &= \left(\prod_{k=1}^K P(X_k | Y) \right) P(Y) \end{aligned}$$

- What do we gain? With binary features, get one entry per feature per label
- So, number of probabilities goes from $2 * 2^K \rightarrow 2 * K$

Naïve Bayes: Training

Maximum likelihood says to just use counts to obtain probabilities:

$$P(X_k = i|Y) = \frac{\text{\#times } X_k = i \text{ in class } Y}{\text{\#examples of class } Y} = \frac{c_i}{N}$$

But what if $c_i = 0$?

- Then $P(X_k = i|Y) = 0$.
- This makes the probability $P(Y) \prod_k P(X_k = i|Y)$ of the document zero
- Solution: smooth!

$$\hat{P}(X_k|Y) = \frac{c_i + \alpha}{N + \alpha K} \quad \longleftarrow \begin{array}{l} \text{smoothing} \\ \text{parameter} \end{array}$$

Naïve Bayes: Predicting

- With conditional probabilities, how to predict?

$$\hat{y} = \operatorname{argmax}_y p(y|\mathbf{x}) \quad (\text{posterior})$$

$$= \operatorname{argmax}_y \frac{p(\mathbf{x}|y)p(y)}{p(x)} \quad (\text{by Bayes' rule})$$

$$= \operatorname{argmax}_y p(\mathbf{x}|y)p(y)$$

Suggested Readings

- Textbook: Artificial Intelligence: A Modern Approach (4th edition). Stuart Russell and Peter Norvig. Pearson, 2020.
 - Sections 19.7.1, 20.1, 20.2.1-20.2.4



Thanks!