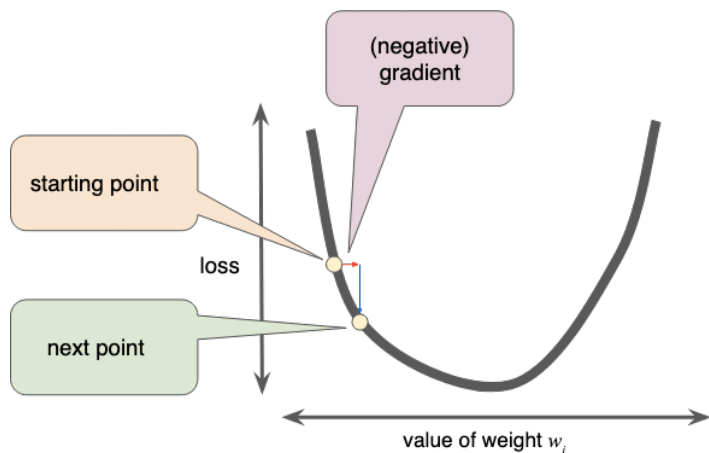# Machine Learning

Prof. Adil Khan

Objectives

1. A quick recap of last week

2. What is Naïve Bayes? How does it work? Why is it called "Naïve"?

3. Nearest Neighbor Methods

   - What is KNN? What is its objective function? How does it work?
   - Pitfalls of KNN

4. Hyperparameter tuning and cross-validation

5. What is regularization? Why is it important? How does it work?

Recap (1)

Classification

- Eye color $\in \{brown, blue, green\}$
- Email $\in \{Spam, Ham\}$
- Expression $\in \{Happy, Sad, Angry\}$
- Action $\in \{Walking, Running, Cycling\}$

Gradient Descent

Repeat {

$$w_j = w_j - \alpha \frac{1}{n} \sum_{i=1}^{n} (\hat{y}_i - y_i) x_i^j$$

}

Logistic Regression

$$p(x) = \frac{1}{1 + e^{-z}}$$

$$z = w_0 + w_1 x_1 + \cdots + w_p x_p$$
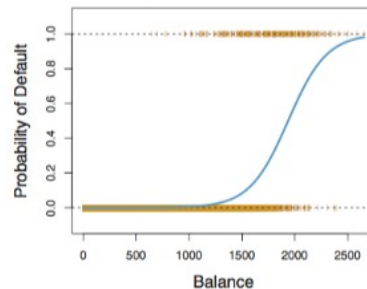
Recap (2)

MSE Loss Function for Logistic Regression

$$y = \frac{1}{1 + e^{-\boldsymbol{w}^T \boldsymbol{x}^i}}$$

$$\mathcal{L}(\boldsymbol{w}) = \frac{1}{n}\sum_{i=1}^{n}\left(y^i - \boldsymbol{w}^T \boldsymbol{x}^i\right)^2$$

- Due to non-linearity of the sigmoid function in hypothesis of Logistic Regression, MSE is **not convex** anymore

Loss Function of Logistic Regression
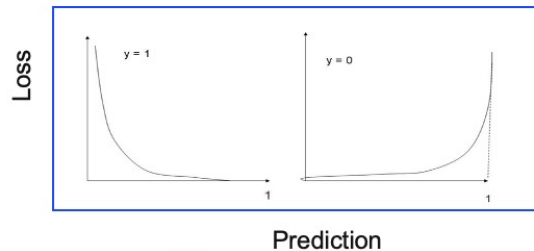
$$\mathcal{L}(\boldsymbol{w}) = -\frac{1}{n}\sum_{i=1}^{n} y^i \log\left(p(\boldsymbol{x}^i)\right) + (1 - y^i)\log(1 - p(\boldsymbol{x}^i))$$

$$\underset{w_0, w_1}{\text{argmin}} \ \mathcal{L}(\boldsymbol{w})$$

We solve it using **Gradient Descent**

$$w_j = w_j - \alpha \frac{1}{n}\sum_{i=1}^{n}\left(p(\boldsymbol{x}^i) - y^i\right) x_j^i$$

Recap (3)

$$Acc = 1 - \frac{\# \text{ of missclassified examples}}{\# \text{ of samples}}$$

$$Precision = \frac{True\ Positives}{True\ Positives + False\ Positives}$$

$$Recall = \frac{True\ Positives}{True\ Positives + False\ Negatives}$$

$$F1 = 2 \times \frac{Precision \times Recall}{Precision + Recall}$$

Class-imbalance

Data For Cat / Dog Classifier

Data For Normal / Fraud Classifier

■ Cat ■ Dog

■ Fraud ■ Normal

Confusion Matrix

Actual Class Label		Predicted Class Label	
		Negative	**Positive**
	Negative	True Negatives	False Positives
	Positive	False Negatives	True Positives

Naïve Bayes Classifier

Naïve Bayes Classifier

- Takes its name from the equation it is based on

- Goal: given a set of training data points from C classes, compute the predictive probabilities for each of the C potential classes

$$p(y_{new} = c | \boldsymbol{x}_{new}, \boldsymbol{X}, \boldsymbol{y})$$

$$0 \leq p(y_{new} = c | \boldsymbol{x}_{new}, \boldsymbol{X}, \boldsymbol{y}) \leq 1$$

$$\sum_{c=1}^{C} p(y_{new} = c | \boldsymbol{x}_{new}, \boldsymbol{X}, \boldsymbol{y}) = 1$$

Bayes Rules (1)

- Bayes classifier is based on Bayes equation

$$p(y_{new} = c | \boldsymbol{x}_{new}, \boldsymbol{X}, \boldsymbol{y}) = \frac{p(\boldsymbol{x}_{new} | y_{new} = c, \boldsymbol{X}, \boldsymbol{y}) p(y_{new} = c | \boldsymbol{X}, \boldsymbol{y})}{p(\boldsymbol{x}_{new} | \boldsymbol{X}, \boldsymbol{y})}$$

Bayes Rules (2)

- Simplified

$$p(y_{new} = c | \boldsymbol{x}_{new}) = \frac{\textcolor{red}{p(\boldsymbol{x}_{new}|c)}\textcolor{blue}{p(c)}}{\textcolor{green}{p(\boldsymbol{x}_{new})}}$$

Bayes Rules (3)

- Expand the denominator over all classes

$$p(y_{new} = c | \boldsymbol{x}_{new}) = \frac{p(\boldsymbol{x}_{new}|c)p(c)}{\sum_{c=1}^{C} p(\boldsymbol{x}_{new}|\boldsymbol{c})\ p(c)}$$

Bayes Rules (4)

Posterior Class Probability

Likelihood

Prior Class Probability

Marginal Likelihood

$$p(y_{new} = c | \boldsymbol{x}_{new}) = \frac{p(\boldsymbol{x}_{new} | c) p(c)}{\sum_{c=1}^{C} p(\boldsymbol{x}_{new} | \boldsymbol{c}) \; p(c)}$$

Likelihood

$$p(\boldsymbol{x}_{new}|c) = p\left(\left(x_1, \cdots, x_p\right)_{new}|c\right)$$

- Now, to simplify the matter, an assumption is made

- **Independence Assumption:** predictors are are <u>conditionally independent</u> given the target

$$p\left(\left(x_1, \cdots, x_p\right)_{new}|c\right) = \prod_{i=1}^{p} p(x_i|c)$$

- That is why it is called the ***naïve** Bayes*

Naïve Bayes Properties

- Estimating $P(x_i \mid c_j)$ instead of $P(x_1, x_2, \ldots, x_n \mid c_j)$ greatly reduces the number of parameters

- The learning step in Naïve Bayes consists of estimating $P(x_i \mid c_j)$ and $P(c_j)$ based on the frequencies in the training data

- An unseen instance is classified by computing the class that maximizes the posterior

$$c_{NB} = \arg\max_{c_j \in C} P(c_j) \prod_i P(x_i \mid c_j)$$

Naïve Bayes Example

Day	Outlook	Temperature	Humidity	Wind	Play Tennis
1	Sunny	Hot	High	Weak	No
2	Sunny	Hot	High	Strong	No
3	Overcast	Hot	High	Weak	Yes
4	Rain	Mild	High	Weak	Yes
5	Rain	Cool	Normal	Weak	Yes
6	Rain	Cool	Normal	Strong	No
7	Overcast	Cool	Normal	Strong	Yes
8	Sunny	Mild	High	Weak	No
9	Sunny	Cool	Normal	Weak	Yes
10	Rain	Mild	Normal	Weak	Yes
11	Sunny	Mild	Normal	Strong	Yes
12	Overcast	Mild	High	Strong	Yes
13	Overcast	Hot	Normal	Weak	Yes
14	Rain	Mild	High	Strong	No

For the day <sunny, cool, high, strong>

What's the play prediction?

Naïve Bayes Example

- Based on the examples in the table, classify the data

x=(Outlook=Sunny, Temp=Cool, Hum=High, Wind=strong)

$$h_{NB} = \arg\max_{h \in [yes,no]} P(h)P(\mathbf{x}\,|\,h) = \arg\max_{h \in [yes,no]} P(h)\prod_t P(a_t\,|\,h)$$

$$= \arg\max_{h \in [yes,no]} P(h)P(Outlook = sunny\,|\,h)P(Temp = cool\,|\,h)P(Humidity = high\,|\,h)P(Wind = strong\,|\,h)$$

Naïve Bayes Example

Day	Outlook	Temperature	Humidity	Wind	Play Tennis
1	Sunny	Hot	High	Weak	No
2	Sunny	Hot	High	Strong	No
3	Overcast	Hot	High	Weak	Yes
4	Rain	Mild	High	Weak	Yes
5	Rain	Cool	Normal	Weak	Yes
6	Rain	Cool	Normal	Strong	No
7	Overcast	Cool	Normal	Strong	Yes
8	Sunny	Mild	High	Weak	No
9	Sunny	Cool	Normal	Weak	Yes
10	Rain	Mild	Normal	Weak	Yes
11	Sunny	Mild	Normal	Strong	Yes
12	Overcast	Mild	High	Strong	Yes
13	Overcast	Hot	Normal	Weak	Yes
14	Rain	Mild	High	Strong	No

$P(PlayTennis = yes) = 9/14 = 0.64$

$P(PlayTennis = no) = 5/14 = 0.36$

$P(Wind = strong \mid PlayTennis = yes) = 3/9 = 0.33$

$P(Wind = strong \mid PlayTennis = no) = 3/5 = 0.60$

......

$P(yes)P(sunny \mid yes)P(cool \mid yes)P(high \mid yes)P(strong \mid yes) = 0.0053$

$P(no)P(sunny \mid no)P(cool \mid no)P(high \mid no)P(strong \mid no) = \mathbf{0.0206}$

$\Rightarrow answer : PlayTennis(x) = no$

Underflow

- Multiplying lots of probabilities, which are between 0 and 1 by definition, can result in **floating-point underflow.**

$$c_{NB} = \arg\max_{c_j \in C} P(c_j) \prod_i P(x_i \mid c_j)$$

$$c_{NB} = \operatorname*{argmax}_{c_j \in C} \log P(c_j) + \sum_{i \in positions} \log P(x_i \mid c_j)$$

- Since *log(xy) = log(x) + log(y),* it is better to perform all computations by summing logs of probabilities rather than multiplying probabilities.

- Class with highest final unnormalized log probability score is still the most probable.

Observations

- **Advantages**
  - It is easy and fast to predict class of test data set. It also perform well in multi class prediction

  - When assumption of independence holds, a Naive Bayes classifier performs better compare to other models like logistic regression and you need less training data.

  - Allows online learning

Observations

- **Disadvantages**
  - If a predictor was not observed in training data set, then model will assign a 0 (zero) probability and will be unable to make a prediction. This is often known as "Zero Frequency".

  - **Solution:** To solve this, we can use the smoothing technique. One of the simplest smoothing techniques is called Laplace estimation.

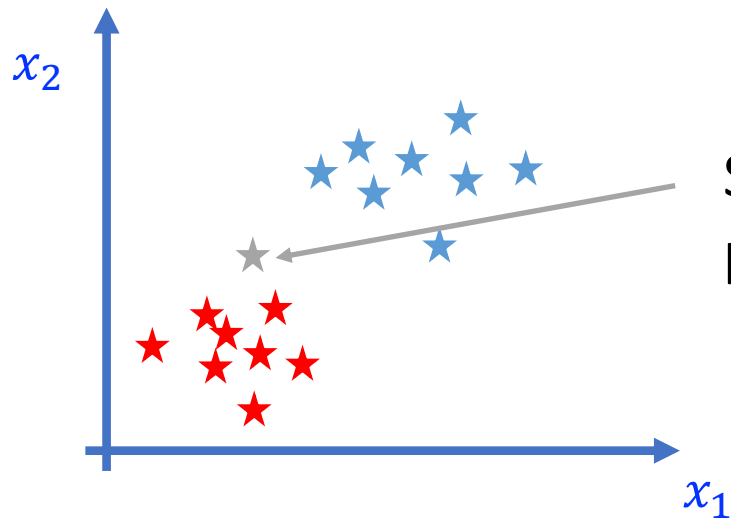# Nearest Neighbor Methods

Nearest Neighbor (1)
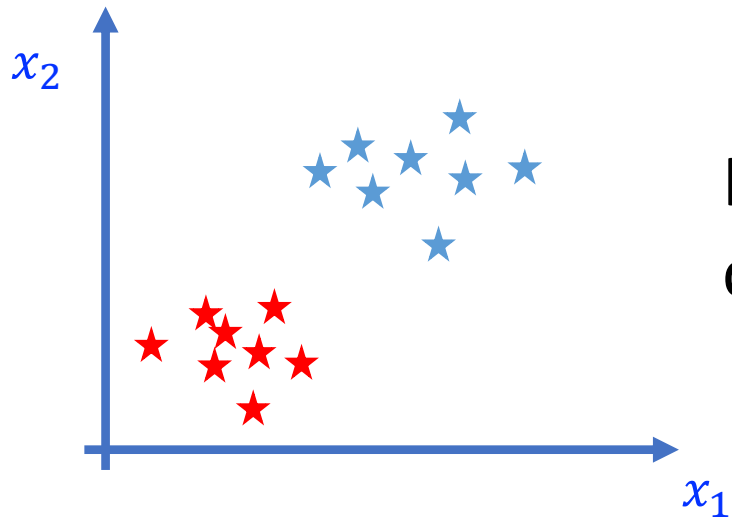
Imagine a binary classification case

Nearest Neighbor (2)
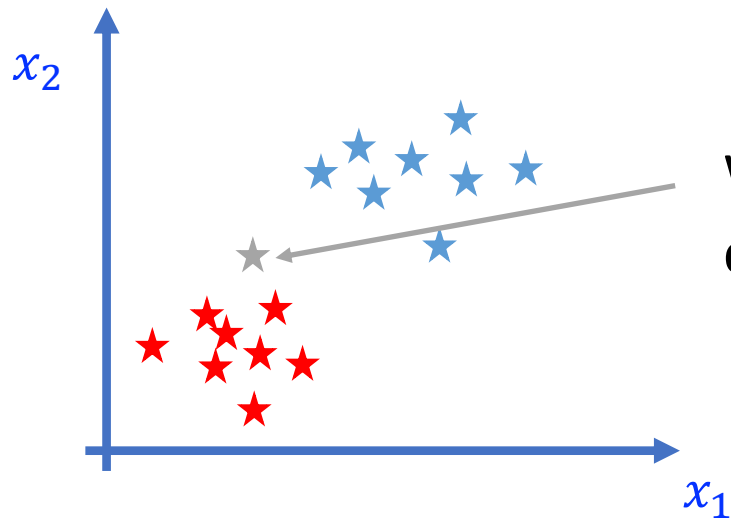
We are given a new data point to classify

Nearest Neighbor (3)

Should we call it a BLUE or a RED point?

Nearest Neighbor (4)

- **The Assumption:** *Similar points have similar labels,*

- Thus find new point's nearest input vector in the training set and copy its label

- How to formalize the concept "nearest"?

- **Solution:** formalize it in the form of a distance

- Euclidean distance

$$\left\| x^a - x^b \right\|_2 = \sqrt{\sum_{i=1}^{d} (x_j^a - x_j^b)^2}$$

Nearest Neighbor (5)

- More formally

1. Find the example $(\boldsymbol{x}^*, \boldsymbol{t}^*)$ from the stored training set which is closest to $\boldsymbol{x}$,

   That is

$$x^* = argmin_{x^i \in train} dis \tan ce \left(x^i, x\right)$$

2. Output: $y = t^*$

Nearest Neighbor (6)

New point classified as a RED point
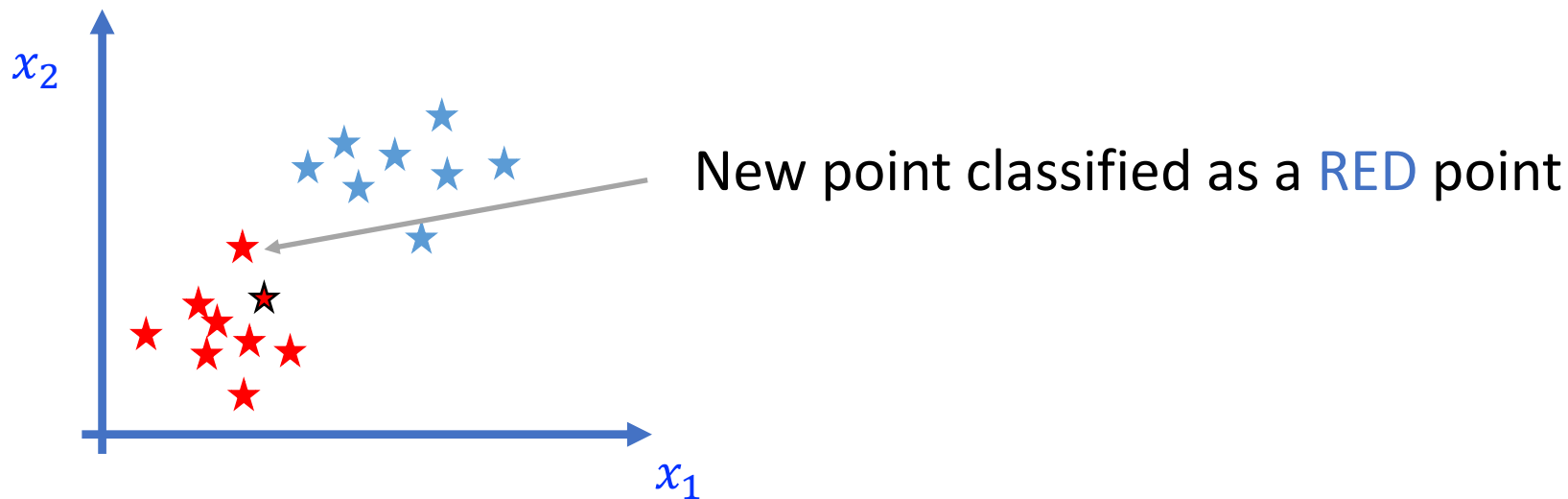
Noisy or Mis-labeled Samples

Noisy sample

Every point in the shaded region will be wrongly classified

$\boldsymbol{k}$-nearest Neighbor

New point

Noisy sample

$k = 3$

Smoothing effect by considering multiple neighbors
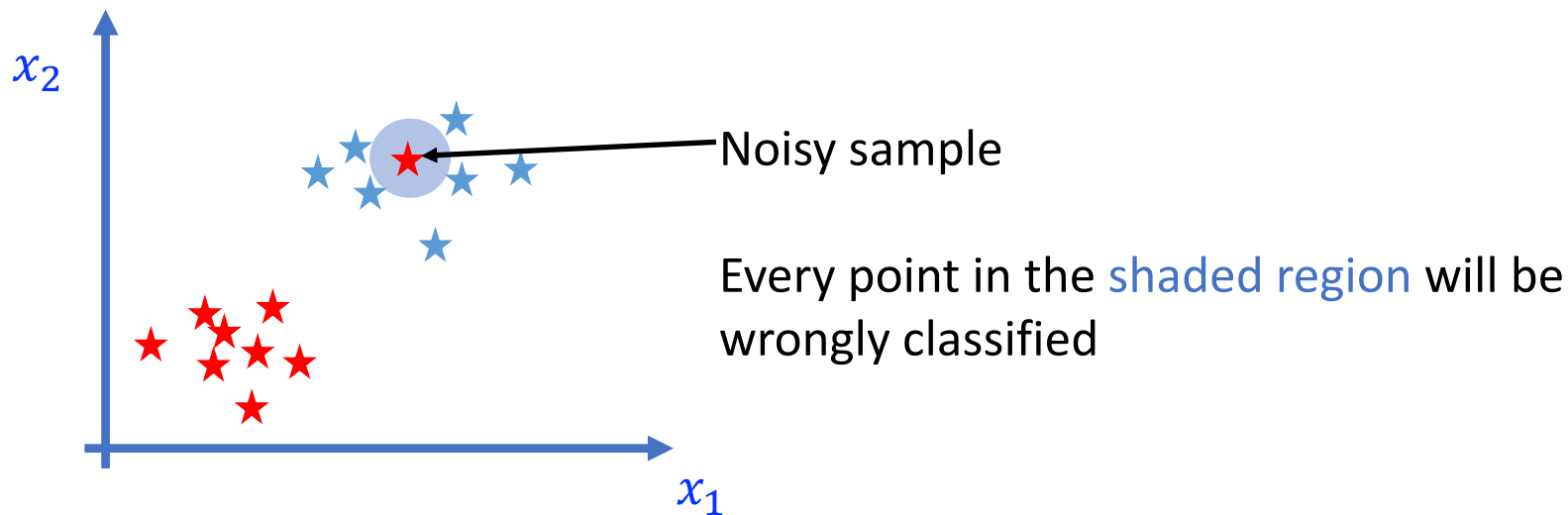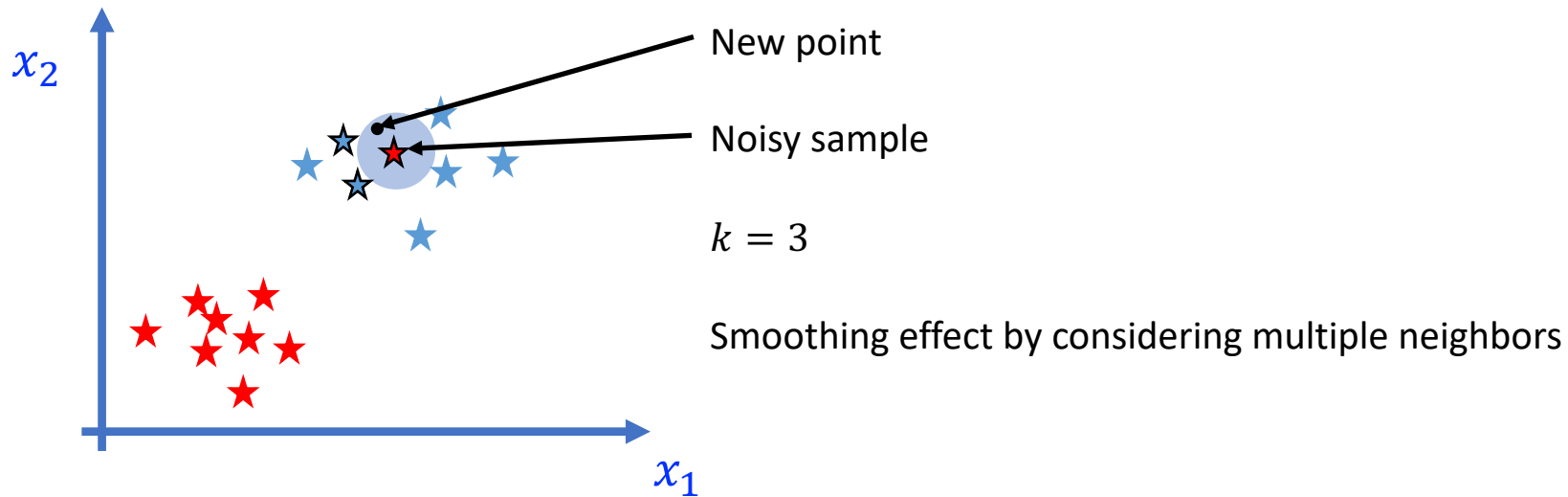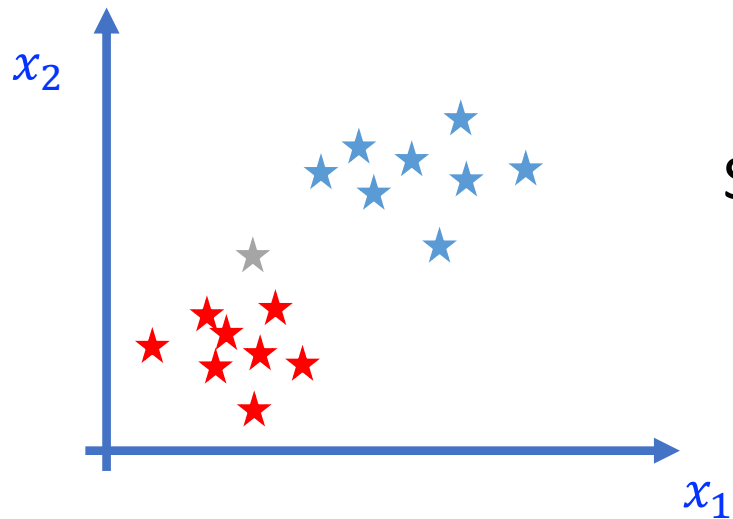
k-nearest Neighbor (KNN)

- Step 1: choose a value for k

- Step 2: Take the k neighbors of the new data point according to Euclidean distance

- Step 3: Among these k neighbor data points, count the number of points in each category

- Step 4: Assign the new data points to the category where you counted the most neighbors
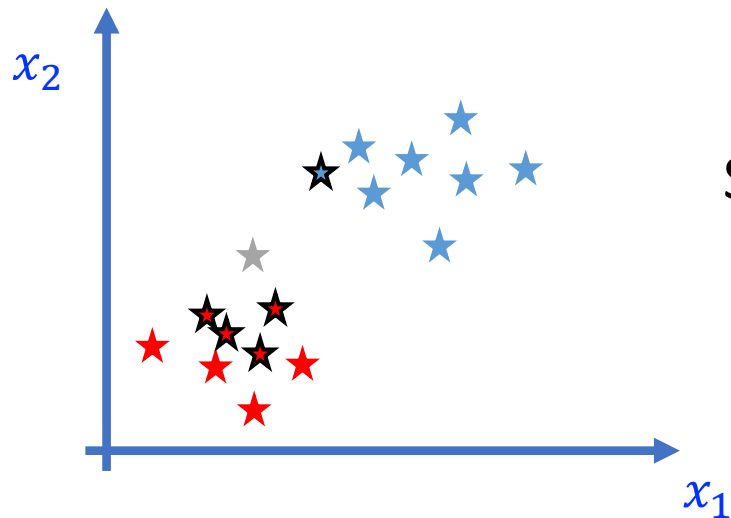
KNN on Our Example (1)

A binary classification case
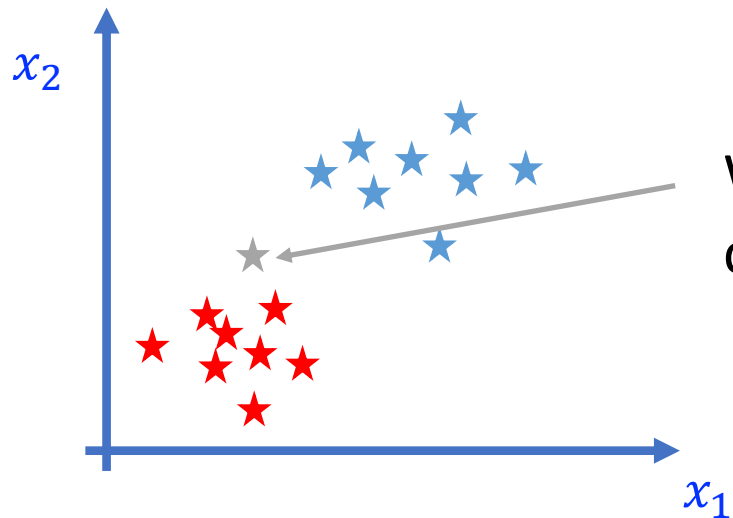
KNN on Our Example (2)

We are given a new data point to classify

KNN on Our Example (2)
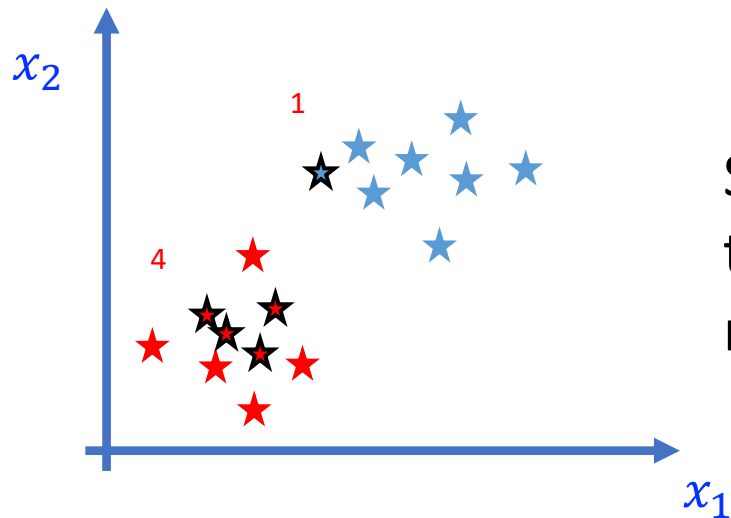

Step 1: set $k = 5$

KNN on Our Example (3)

Step 2: Find $k = 5$ nearest neighbors

KNN on Our Example (4)

Step 3: Count the data points in each category
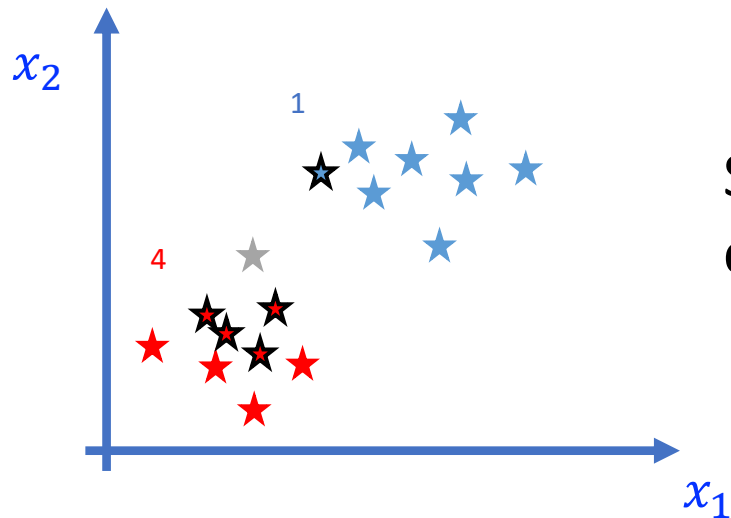
KNN on Our Example (5)

Step 4: Assign the new data point to the class where you counted the most neighbors

Computational Cost of KNN

1. Number of computations performed at training time: 0

2. For a naïve implementation, the number of computations performend at test time, per new point
   - Total points: n
   - Number of dimensions: d
   - Computing d-dimensional distance for n points: $O(nd)$
   - Sorting the distances: $O(nlogn)$
   - Thus an expensive test time algorithm

3. Also, all the data have to be stored in memory

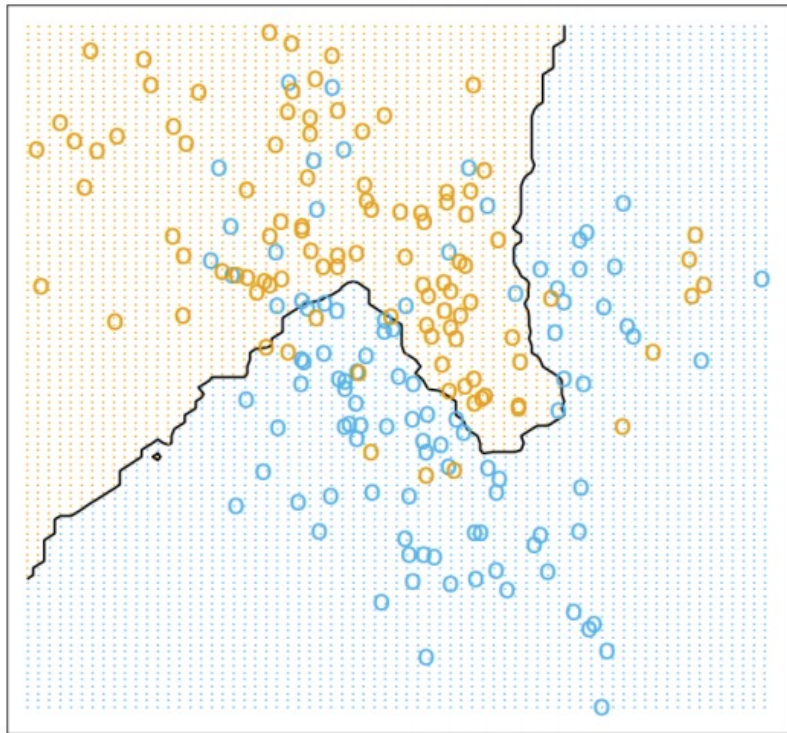4. Many efficient implentations have been implemented, but are out of scope for us

Impact of k

$k = 1$
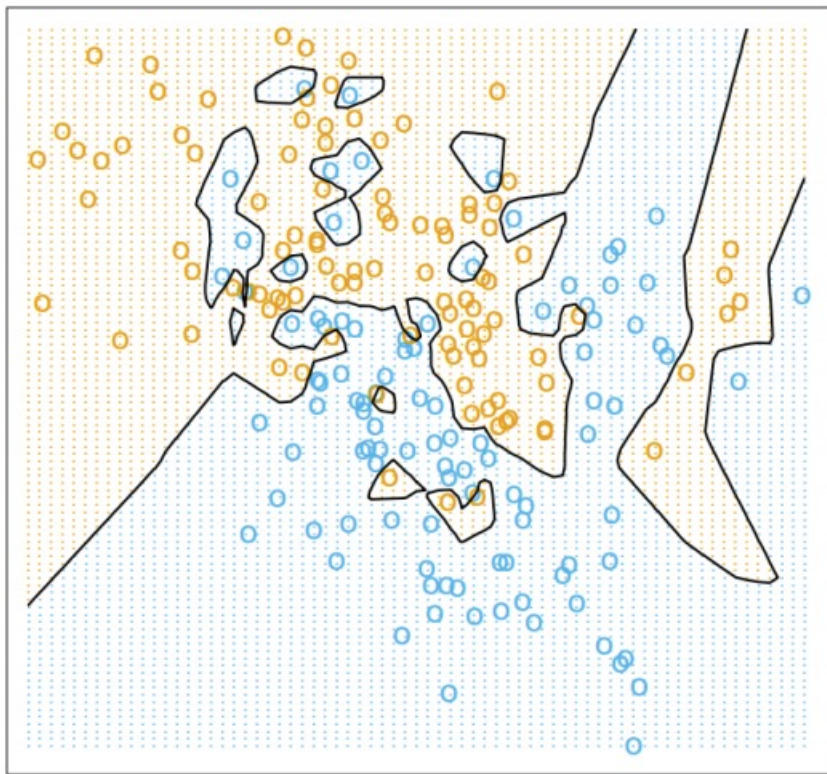
[Image credit: "The Elements of Statistical Learning"]

$k = 15$

[Image credit: "The Elements of Statistical Learning"]

Impact of k

- Small k
  - Good at capturing fine-grained paterns
  - May overfit

- Large k
  - Makes stable prediction by averaging over a large number of neighbors
  - May underfit

- Balancing k
  - Optimal choice depend on the number of data points n
  - Rule of thumb: $k < \sqrt{n}$
  - Choose the optimal value using a validation set

Things to Remember about KNN (1)

- Do we learn a model?

  - NO!!

  - Our model is the entire training data (Storage issues)

Things to Remember about KNN (2)

- Normalize data

- Handle missing values

Things to Remember about KNN (3)

- **Other names for KNN**

    - Instance-Based Learning

   - Lazy Learning

    - Nonparametric Learning

Hyperparameters

Hyperparameters

- Hyperparameters
  - In ML, a hyperparameter is a parameter whose value is used to control the learning process
  - They are set (by the us) not learned like mode parameters

- Examples:
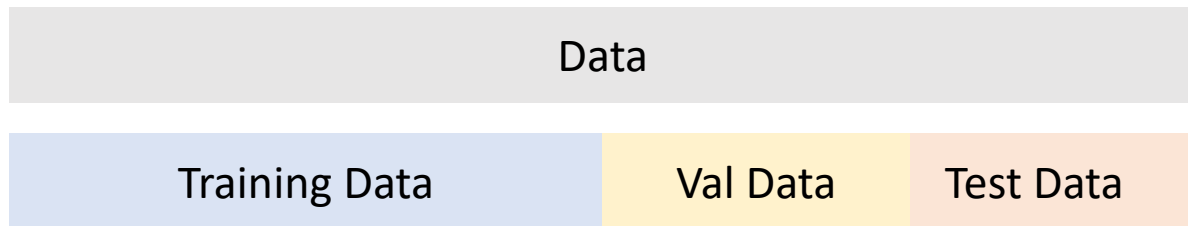 - k in KNN
  - α in GD

Validation and Test Sets

- We can tune hyperparameters using the validation set

- But what is a validation set? Is it the same as test set?

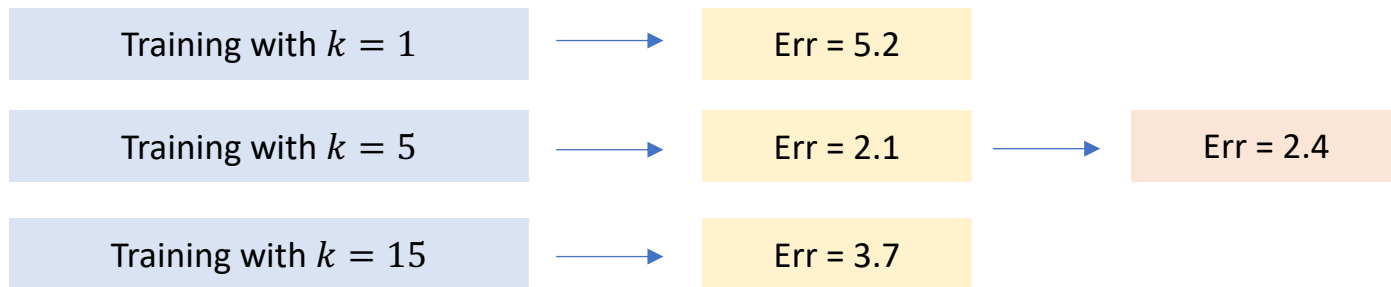- To answer this, let's see how were we splitting our data so far

Data		

Training Data		Test Data
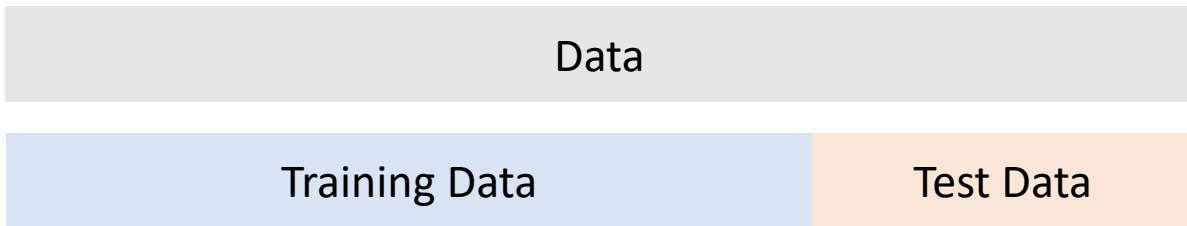
Validation and Test Sets (2)

- But when we have hyperparameters to tune, we split as follows

Data		

Training Data	Val Data	Test Data

- As we train model as follows

Training with $k = 1$	→	Err = 5.2		
Training with $k = 5$	→	Err = 2.1	→	Err = 2.4
Training with $k = 15$	→	Err = 3.7		

Validation and Test Sets (2)

- Solves the "Model Selection" problem
  - Selecting the optimal hyperparameters for a given machine learning model

- But there is another problem
  - Once we have chosen a model, how can we estimate its true error?
  - Yes, we are using an independent "test" set to estimate the true error, BUT
  - The true error is a model's error when tested on the ENTIRE POPULATION

K-fold Cross-Validation

$$E = \frac{1}{K}\sum_{i=1}^{K} E_i$$

- Create K-fold partition of the dataset

K = 3

Data		
Fold 1	Fold 2	Fold 3

|	Fold 1	Fold 2	Fold 3
Exp: 1	Val	Train	Train
Exp: 2	Train	Val	Train
Exp: 3	Train	Tain	Val

# Regularization

Recall

Goal of Machine Learning

$$f: x \rightarrow y$$

- So that we can use the learned function to make predictions on **UNSEEN** data

- For example, we can use it to predict the risk-scores of contracting COVID of **NEW** patients based on their clinical symptoms

Training Error

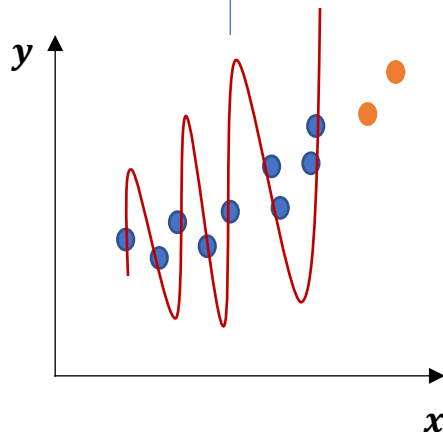- And, naturally, we would like to make **accurate** predictions
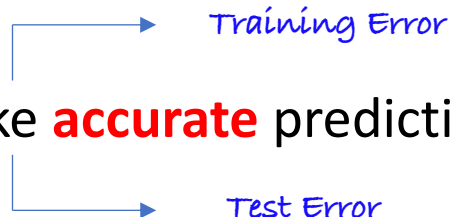
Test Error

What can go wrong?

Training Data

Unseen Test Data
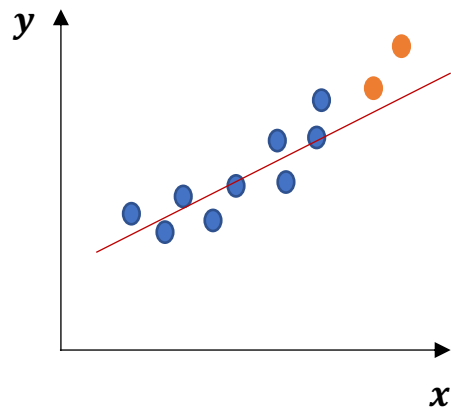
Smallest Training Error

Most Complex

Largest Test Error

$y = \underline{w_o + w_1 x}$
f

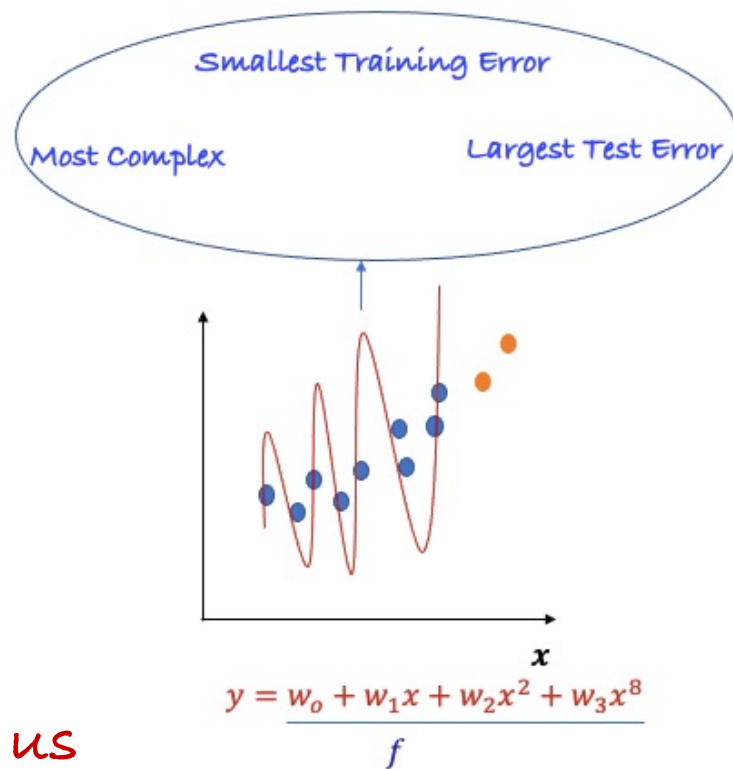$y = \underline{w_o + w_1 x + w_2 x^2}$
f
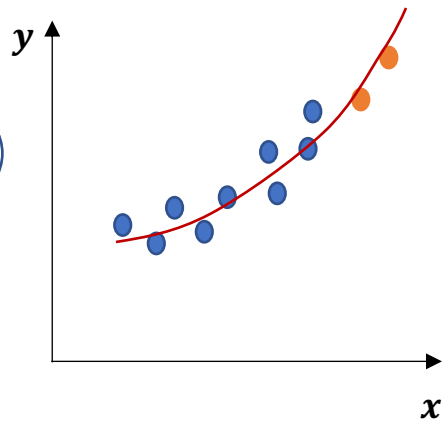
$y = \underline{w_o + w_1 x + w_2 x^2 + w_3 x^8}$
f

Why does it happen?

- It happens because when we are learning the function, we measure the success as

$$Success = Goodness\ of\ Fit$$

This is where Regularization helps us

Smallest Training Error

Most Complex   Largest Test Error

$$y = w_o + w_1 x + w_2 x^2 + w_3 x^8 \over f$$

How does Regularization help?

- Improves the "test error" by compromising on the "training error"

- To achieve this, the most common approach is to change the way we measure success of learning as

$$Success = Goodness\ of\ Fit + Simplicity\ of\ the\ Model$$

Which is usually implemented as

$$Success = Goodness\ of\ Fit + \lambda\ Simplicity\ of\ the\ Model$$

$$where\ \lambda \in [0, \infty)$$

Mathematically

$Success = Goodness\ of\ the\ Fit + \lambda\ Simplicity\ of\ the\ model$

$$\frac{1}{n}\sum_{i=1}^{n}(y_i - \boldsymbol{w}^T.\boldsymbol{x}_i)^2$$

$$\sum_{j=1}^{p} w_j^2$$

L_2 Regularization

$$\mathcal{L} = \frac{1}{n} \sum_{i=1}^{n} (y_i - \boldsymbol{w}^T . \boldsymbol{x}_i)^2 + \lambda \sum_{j=1}^{p} w_j^2$$

$\lambda \geq 0 \ is \ a \ tuning \ parameter$

Ridge Regression

L_2 Regularization

1. If λ is at a "good" value, regularization helps to avoid overfitting

2. Choosing λ may be hard: cross-validation is often used

3. If there are irrelevant features in the input (i.e. features that do not affect the output), L_2 will give them small, but non-zero weights

4. Ideally, irrelevant input should have weights exactly equal to 0

L_1 Regularization

$$\mathcal{L} = \frac{1}{n}\sum_{i=1}^{n}(y_i - \boldsymbol{w}^T.\boldsymbol{x_i})^2 + \lambda\sum_{j=1}^{p}|w_j|$$

$$\lambda \geq 0 \; is \; a \; tuning \; parameter$$

Lasso Regression

L_1 Regularization

- Similar to L_2, L_1 shrinks the coefficient estimates towards zero

- However, L_1 penalty has the effect of forcing some of the coefficient estimates to be exactly equal to zero when the tuning parameter λ is sufficiently large

- Hence, much like best subset selection, L_1 performs variable selection

- As a result, models are generally much easier to interpret

- We say that the L_1 yields sparse models

Recommended Research

1. How does the curse of dimensionality affect KNN?

2. What are the techniques that we can use to tune multiple hyperparameters in k-fold cross-validation manner?

3. Why does L1 regularization give sparse models but L2 does not?

Summary

- Naïve Bayes

- KNN

- Cross-validation

- Regularization