Machine Learning

KNN

Dr. Shuang LIANG

Recall: Logistic Regression

- Model

$$f_{w,b}(x) = \sigma\left(\sum_i w_i x_i + b\right)$$

Output: between 0 and 1

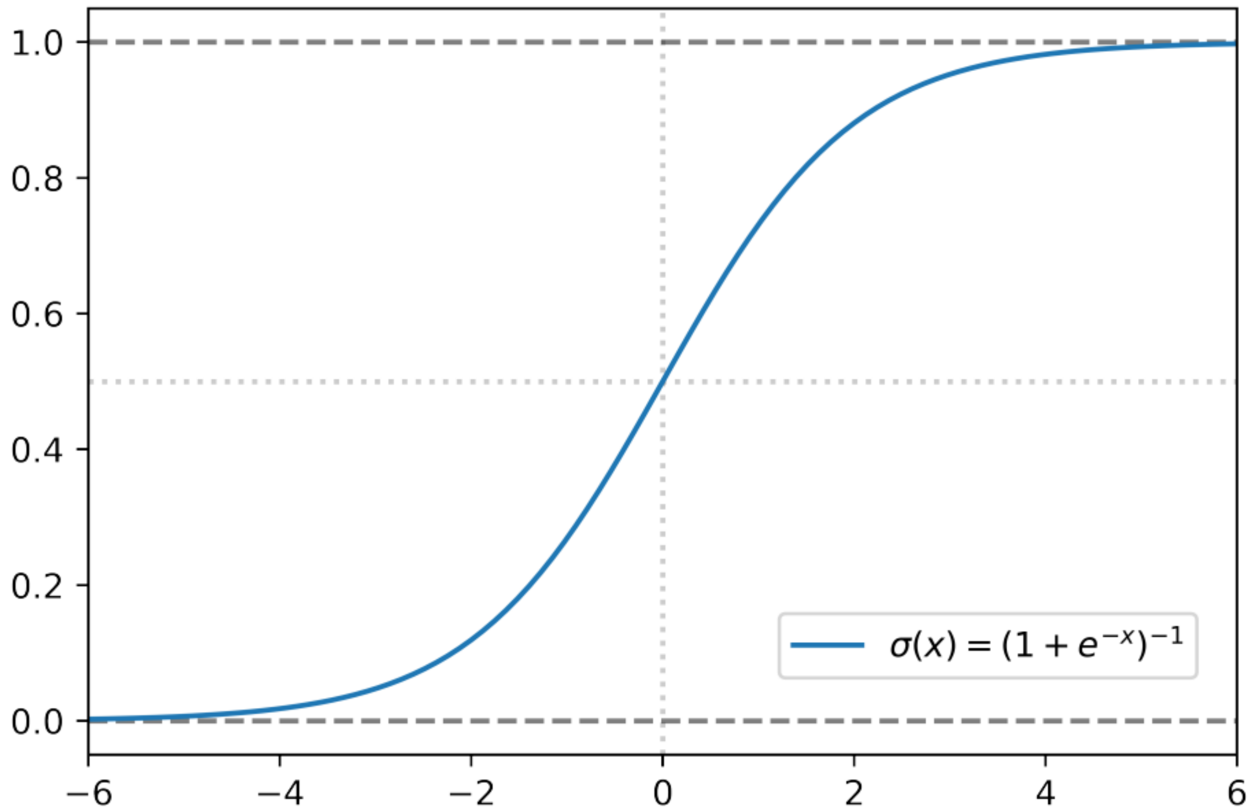
- Loss: Cross Entropy

$$= \sum_n -\left[\hat{y}^n ln f_{w,b}(x^n) + (1 - \hat{y}^n) ln\left(1 - f_{w,b}(x^n)\right)\right]$$

- Optimization: Gradient Descent

$$w_i \leftarrow w_i - \eta \sum_n -\underline{\left(\hat{y}^n - f_{w,b}(x^n)\right)} x_i^n$$

Recall: Sigmoid



Today's Topics

- Type of classifiers
- KNN
- Setting Parameters
- Analysis of KNN

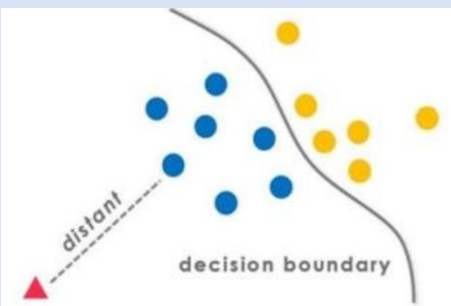
Today's Topics

- ***Type of classifiers***
- KNN
- Setting Parameters
- Analysis of KNN

Types of Classifiers

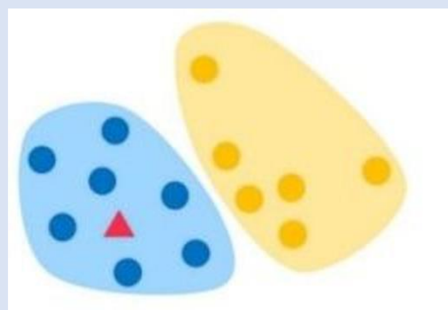
Model-based

Discriminative
directly estimate a decision rule/boundary



Logistic regression
Decision tree
Neural network
……

Generative
build a generative statistical model



Naïve Bayes
Bayesian Networks
HMM
……

No Model

Instance-based
Use observation directly

KNN

Discriminative
- Only care about estimating the conditional probabilities $P(y|x)$
- Very good when underlying distribution of data is really complicated (e.g. texts, images, movies)

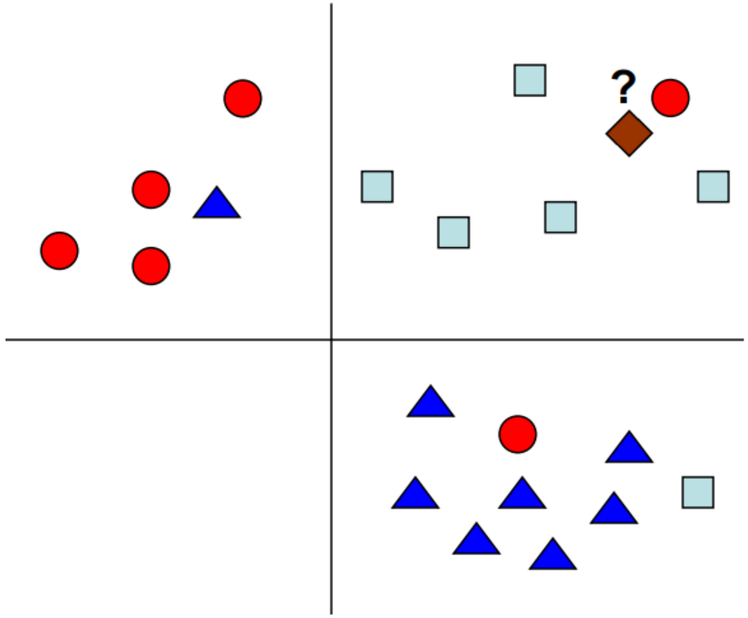
Generative
- Model observations (x, y) first ($P(x, y)$), then infer $P(y|x)$
- Good for missing variables, better diagnostics
- Easy to add prior knowledge about data

Today’s Topics

- Type of classifiers
- ***KNN***
- Setting Parameters
- Analysis of KNN

KNN

- A simple, yet surprisingly efficient algorithm
- Requires the definition of a **distance function** or similarity measures between samples
- Select the class based on the **majority vote** in the k closest points

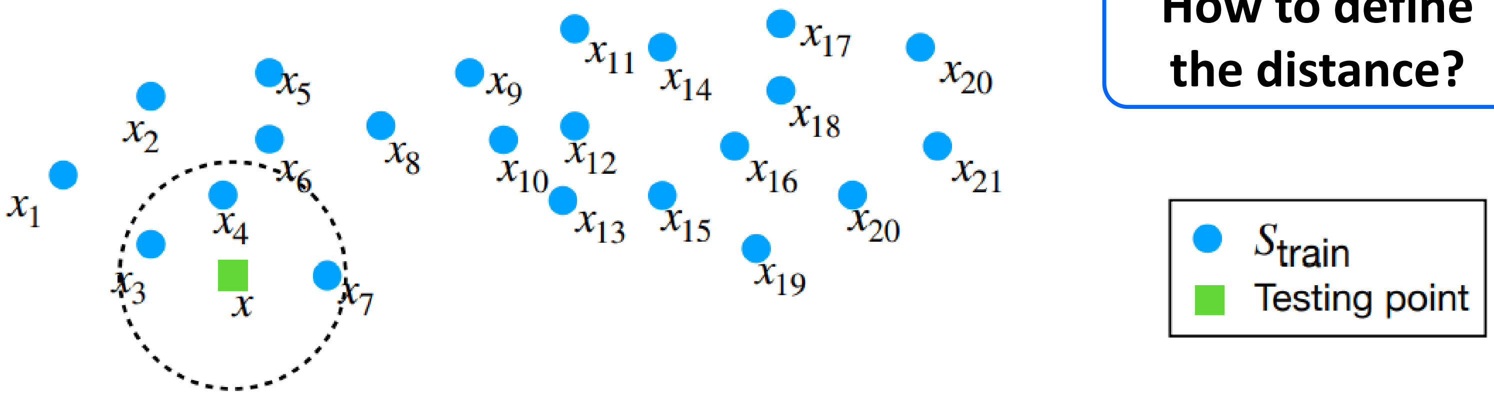


Step1: Find nearest neighbors

$$nbh_{S_{train},k}: \mathcal{X} \to \mathcal{X}^k$$

$$x \longmapsto \{k \text{ elements of } S_{train} \text{ which are } \textcolor{red}{\text{the closest}} \text{ to } x\}$$

How to define the distance?



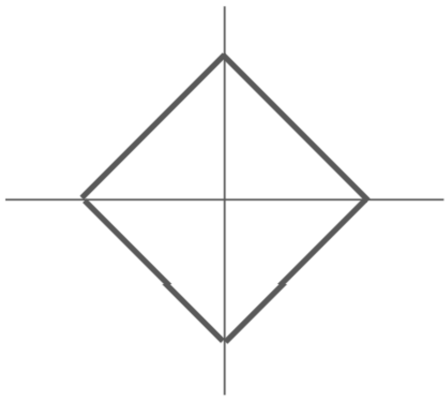
$$nbh_{S_{train},3}(x) = \{x_3, x_4, x_7\}$$

Distance Metric

- **Distance Metric**

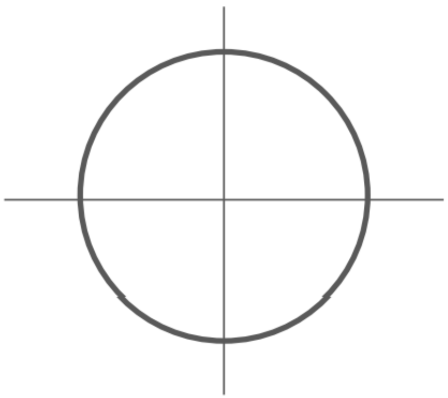
L1 (Manhattan) distance

$$d_1(I_1, I_2) = \sum_p \left| I_1^p - I_2^p \right|$$



L2 (Euclidean) distance

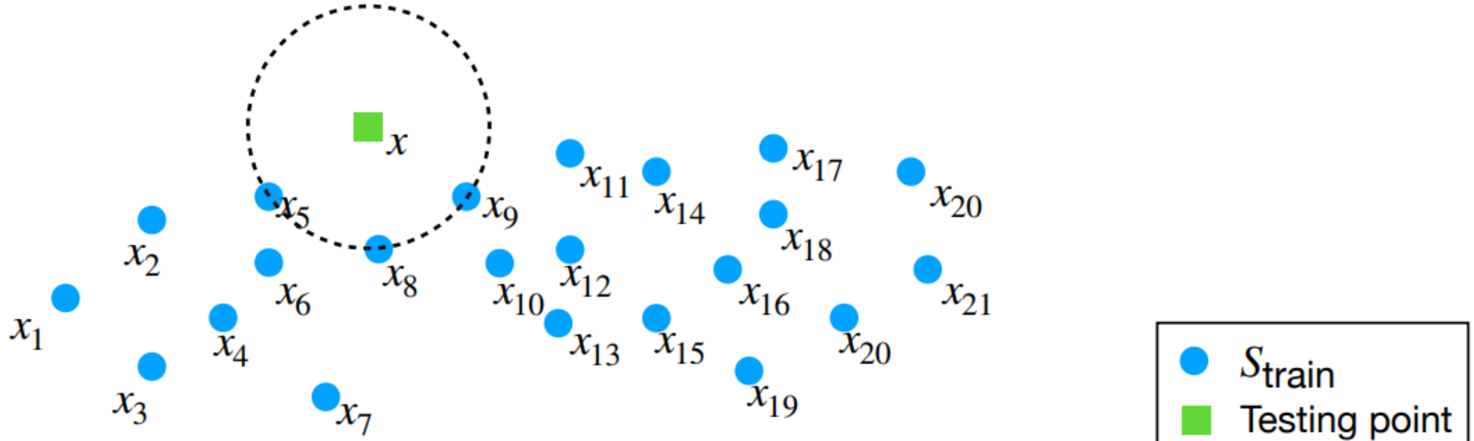
$$d_1(I_1, I_2) = \sqrt{\sum_p \left(I_1^p - I_2^p \right)^2}$$



Step1: Find nearest neighbors

$$nbh_{S_{train},k}: \mathcal{X} \to \mathcal{X}^k$$

$$x \longmapsto \{k \text{ elements of } S_{train} \text{ which are the closest to } x\}$$



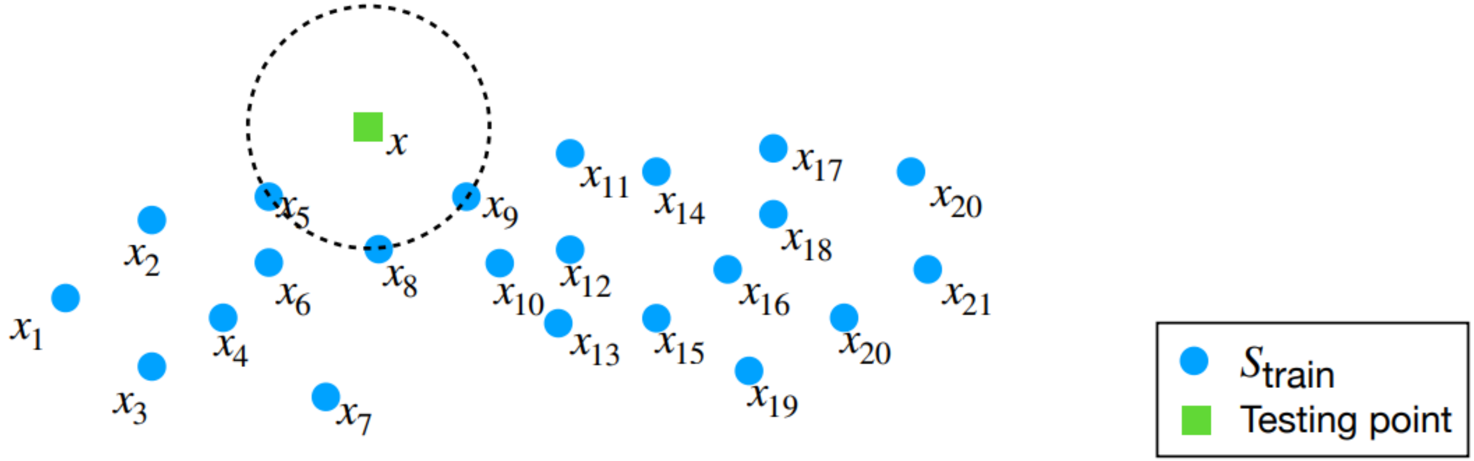
$$nbh_{S_{train},2}(x) = \{x_5, x_8\}$$

It seems that $\{x_5, x_9\}$ and $\{x_8, x_9\}$ work fine as well!

Step1: Find nearest neighbors

$$nbh_{S_{train},k}: \mathcal{X} \rightarrow \mathcal{X}^k$$

$$x \longmapsto \{k \text{ elements of } S_{train} \text{ which are the closest to } x\}$$

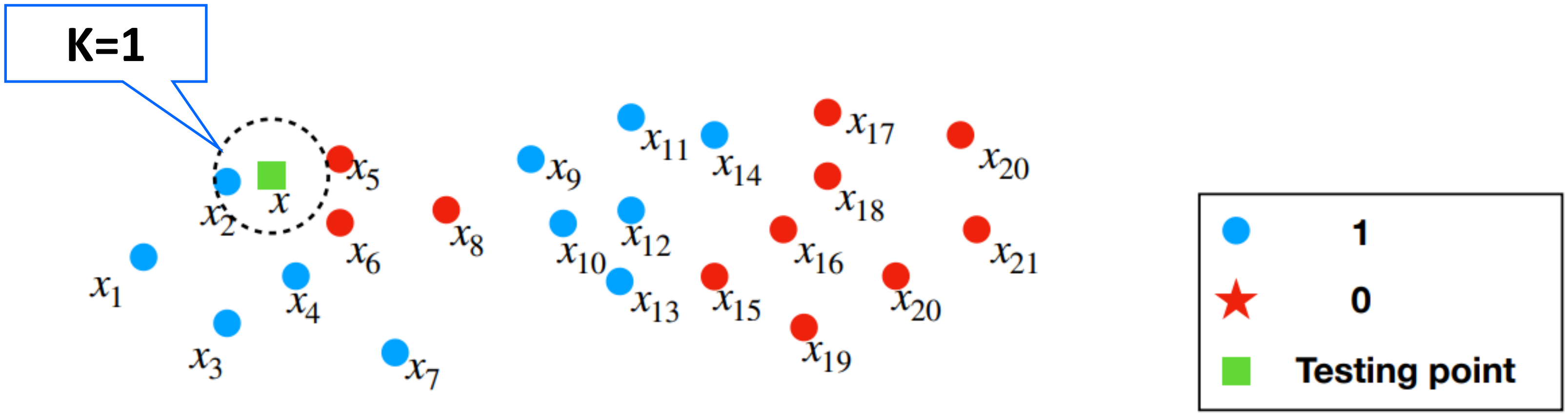


$$nbh_{S_{train},2}(x) = \{x_5, x_8\}$$

Not uniquely defined!
It will depend on the strategy
Often ties are broken randomly

Step2: Select Class

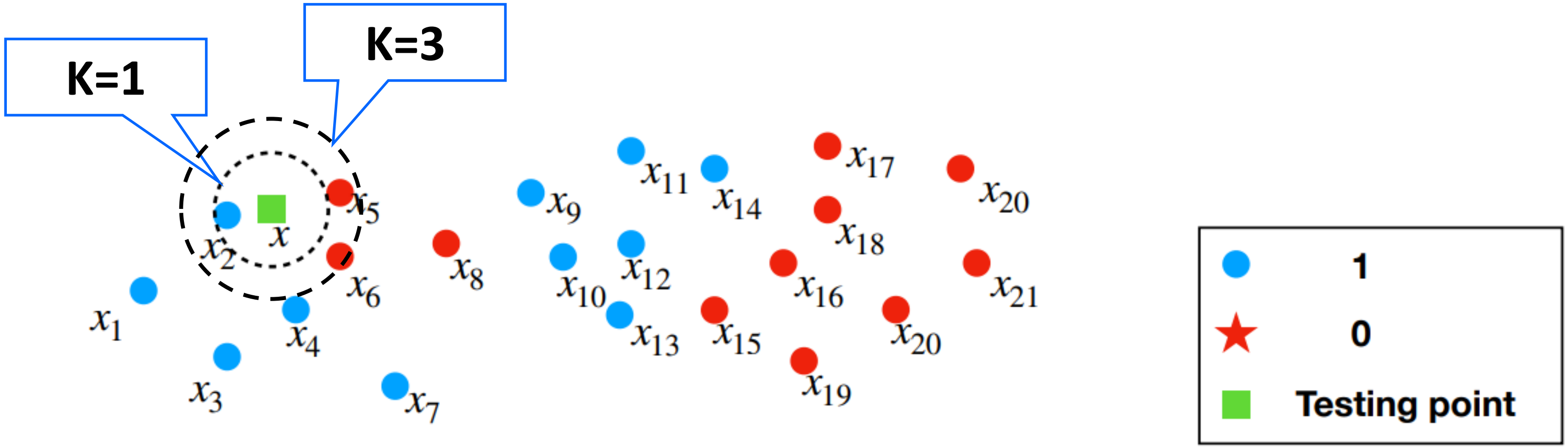
$$f_{S_{train},k}(x) = majority\{y_i : x_i \in nbh_{S_{train},k}(x)\}$$



$$f_{S_{train},1}(x) = 1$$
$$f_{S_{train},3}(x) = ?$$

Step2: Select Class

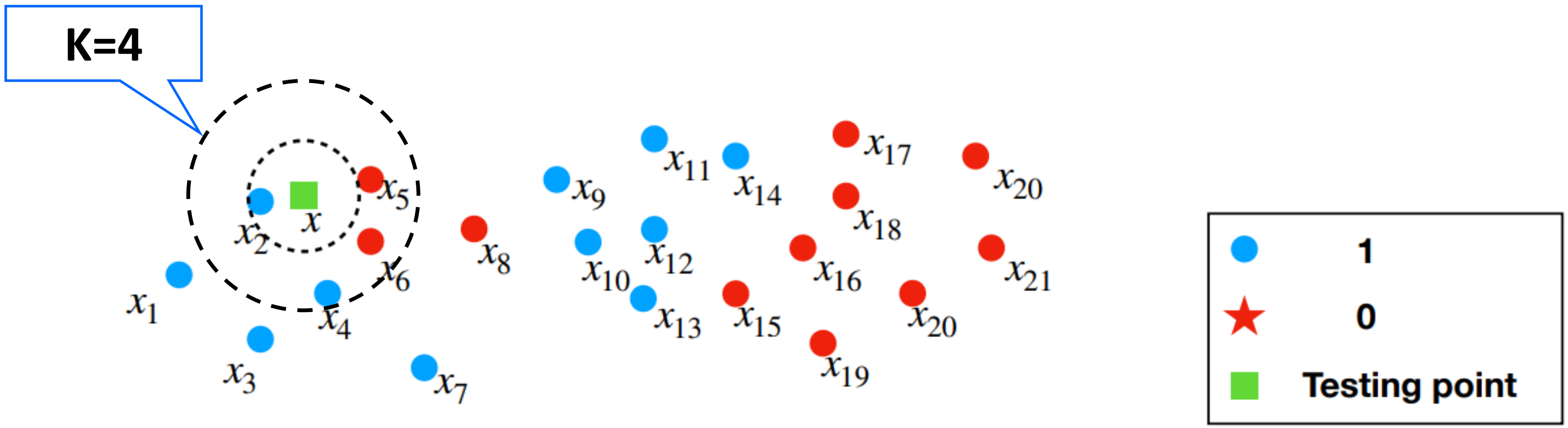
$$f_{S_{train},k}(x) = majority\{y_i : x_i \in nbh_{S_{train},k}(x)\}$$



$$f_{S_{train},1}(x) = 1$$
$$f_{S_{train},3}(x) = 0$$

Step2: Select Class

$$f_{S_{train},k}(x) = majority\{y_i : x_i \in nbh_{S_{train},k}(x)\}$$



$$f_{S_{train},4}(x) = ?$$

Tie!

For the binary case it is good to pick *k* to be odd so that there is a clear winner.

KNN

- **Summary**
- **Step1: Find nearest neighbors**

L1 (Manhattan) distance

$$d_1(I_1, I_2) = \sum_p |I_1^p - I_2^p|$$

L2 (Euclidean) distance

$$d_1(I_1, I_2) = \sqrt{\sum_p (I_1^p - I_2^p)^2}$$

- **Step2: Select Class (majority vote)**

Today's Topics

- Type of classifiers
- KNN
- ***Setting Parameters***
- Analysis of KNN

Setting Parameters

- What do we need to set for KNN?

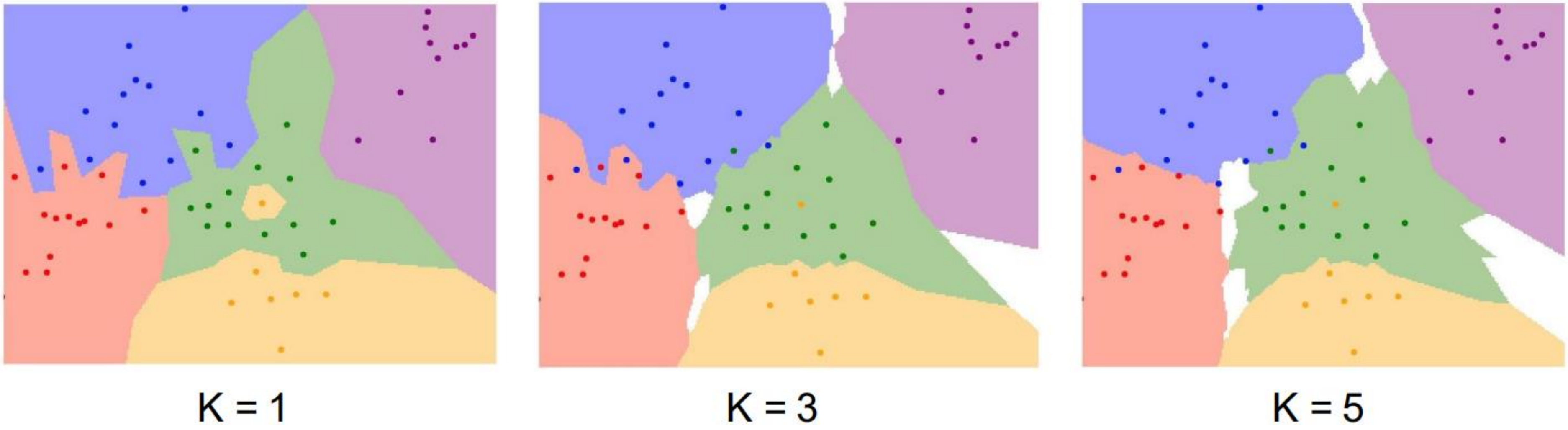
- What is the best **value of k** to use?
- What is the best **distance** to use?

hyperparameters: choices about the algorithm that we set rather than learn.

Very problem-dependent.
Must try them all out and see what works best.

Setting Hyperparameters

- Results in different **value of k**

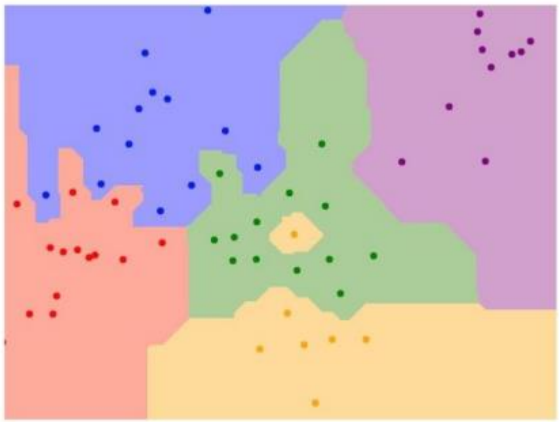


Setting Hyperparameters

- Results in different **distance metrics**

L1 (Manhattan) distance

$$d_1(I_1, I_2) = \sum_p |I_1^p - I_2^p|$$



K = 1

L2 (Euclidean) distance

$$d_1(I_1, I_2) = \sqrt{\sum_p (I_1^p - I_2^p)^2}$$

K = 1

Setting Hyperparameters

Idea #1: Choose hyperparameters that work best on the data

Your Dataset

Setting Hyperparameters

Idea #1: Choose hyperparameters that work best on the data

BAD: K = 1 always works perfectly on training data

Your Dataset

Setting Hyperparameters

Idea #1: Choose hyperparameters that work best on the data

BAD: K = 1 always works perfectly on training data

Your Dataset

Idea #2: Split data into **train** and **test**, choose hyperparameters that work best on test data

train | test

Setting Hyperparameters

Idea #1: Choose hyperparameters that work best on the data

BAD: K = 1 always works perfectly on training data

Your Dataset

Idea #2: Split data into **train** and **test**, choose hyperparameters that work best on test data

BAD: No idea how algorithm will perform on new data

train	test

Setting Hyperparameters

Idea #1: Choose hyperparameters that work best on the data

BAD: K = 1 always works perfectly on training data

Your Dataset

Idea #2: Split data into **train** and **test**, choose hyperparameters that work best on test data

BAD: No idea how algorithm will perform on new data

train | test

Idea #3: Split data into **train**, **val**, and **test**; choose hyperparameters on val and evaluate on test

Better!

train | validation | test

Any better solutions?

Setting Hyperparameters

Your Dataset

Idea #4: **Cross-Validation**: Split data into **folds**,
try each fold as validation and average the results

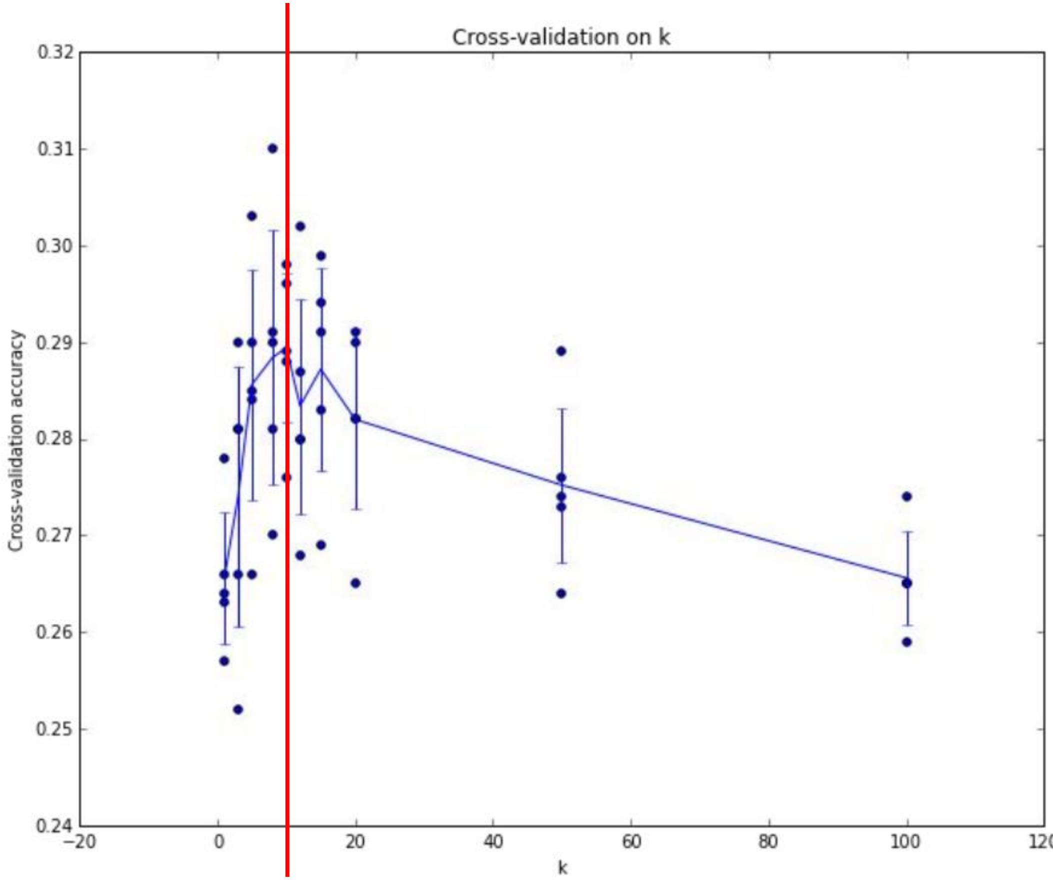
fold 1	fold 2	fold 3	fold 4	fold 5	test
fold 1	fold 2	fold 3	fold 4	fold 5	test
fold 1	fold 2	fold 3	fold 4	fold 5	test

Useful for small datasets, but not used too frequently in deep learning

Why?

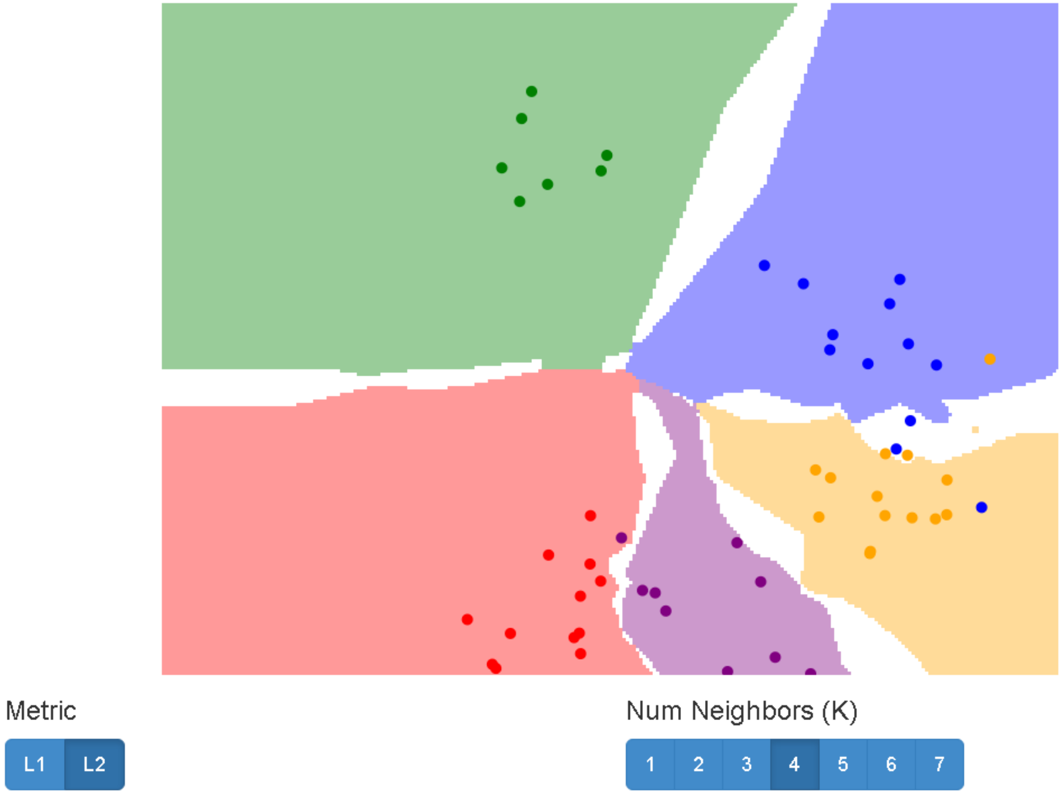
Setting Hyperparameters

- Example of **5-fold cross-validation** for the value of k.
- Each point: single outcome.
- The line goes through the mean, bars indicated standard deviation
- **Seems that k ~= 7 works best for this data**



Setting Hyperparameters

- **Run the demo with different hyperparameters**

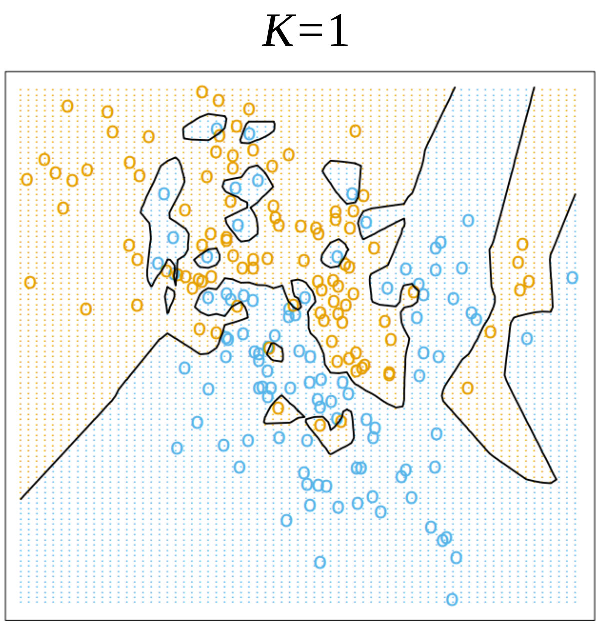


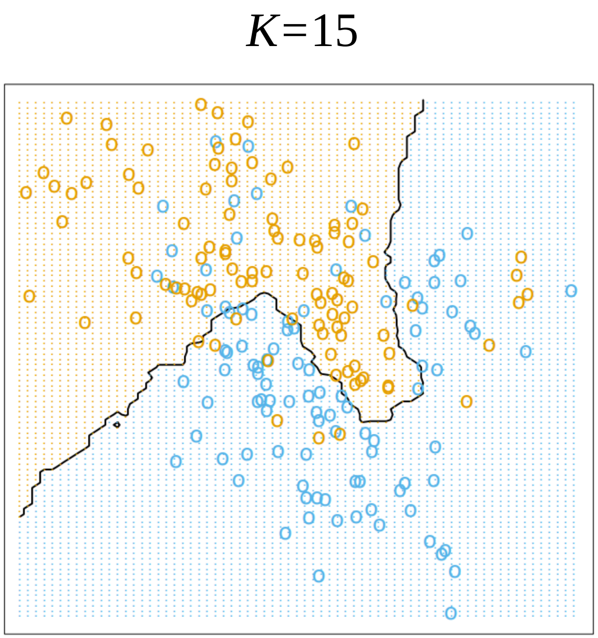
http://vision.stanford.edu/teaching/cs231n-demos/knn/

Today's Topics

- Type of classifiers
- KNN
- Setting Parameters
- ***Analysis of KNN***

Bias-Variance for KNN





Small k

Small bias
Very complex decision boundary
Large variance
Overfitting

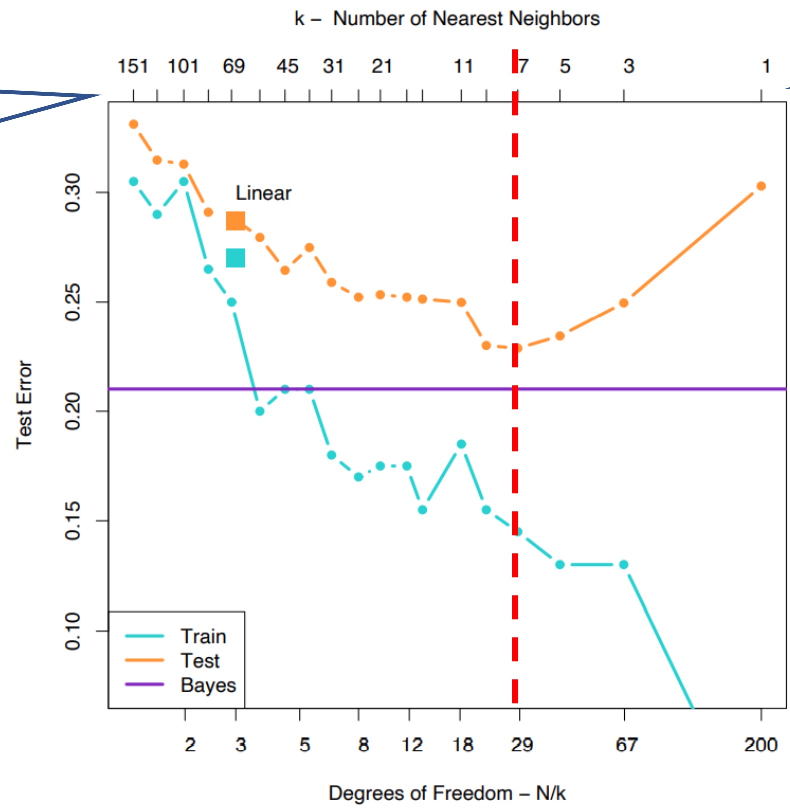
Large k

Large bias
extreme case: k=n, constant prediction
Small variance

Bias-Variance for KNN

Complexity increases when k decreases

Large k
Large bias
Small variance



Small k
Small bias
Large variance

Good k
Small bias
complex enough decision boundary
Small variance
no overfitting

Complexity of KNN

Q: With N examples, how fast are **training** and **prediction**?

	Training	**Prediction**
Complexity	**O(1)**	**O(N)**

?

Complexity of KNN

Q: With N examples, how fast are **training** and **prediction**?

	Training	Prediction
Complexity	**O(1)**	**O(N)**
Action	**Simply remembers all the training data** **No explicit training process** ***"Lazy Learning"***	**For each test sample:** **Find closest training sample** **Predict label of nearest sample**

Complexity of KNN

Q: With N examples, how fast are **training** and **prediction**?

	Training	Prediction
Complexity	**O(1)**	**O(N)**
Action	**Simply remembers all the training data** **No explicit training process** ***"Lazy Learning"***	**For each test sample:** **Find closest training sample** **Predict label of nearest sample**

☹ **This is bad.** ?

Complexity of KNN

Q: With N examples, how fast are **training** and **prediction**?

	Training	**Prediction**
Complexity	**O(1)**	**O(N)**
Action	**Simply remembers all the training data** **No explicit training process** ***"Lazy Learning"***	**For each test sample:** **Find closest training sample** **Predict label of nearest sample**

This is bad.

- We want classifiers that are **fast at prediction**; slow for training is ok.
- **Test time performance** is usually much more important in practice.

Can we use KNN on images?

- Very slow at test time
- Distance metrics on pixels are not informative

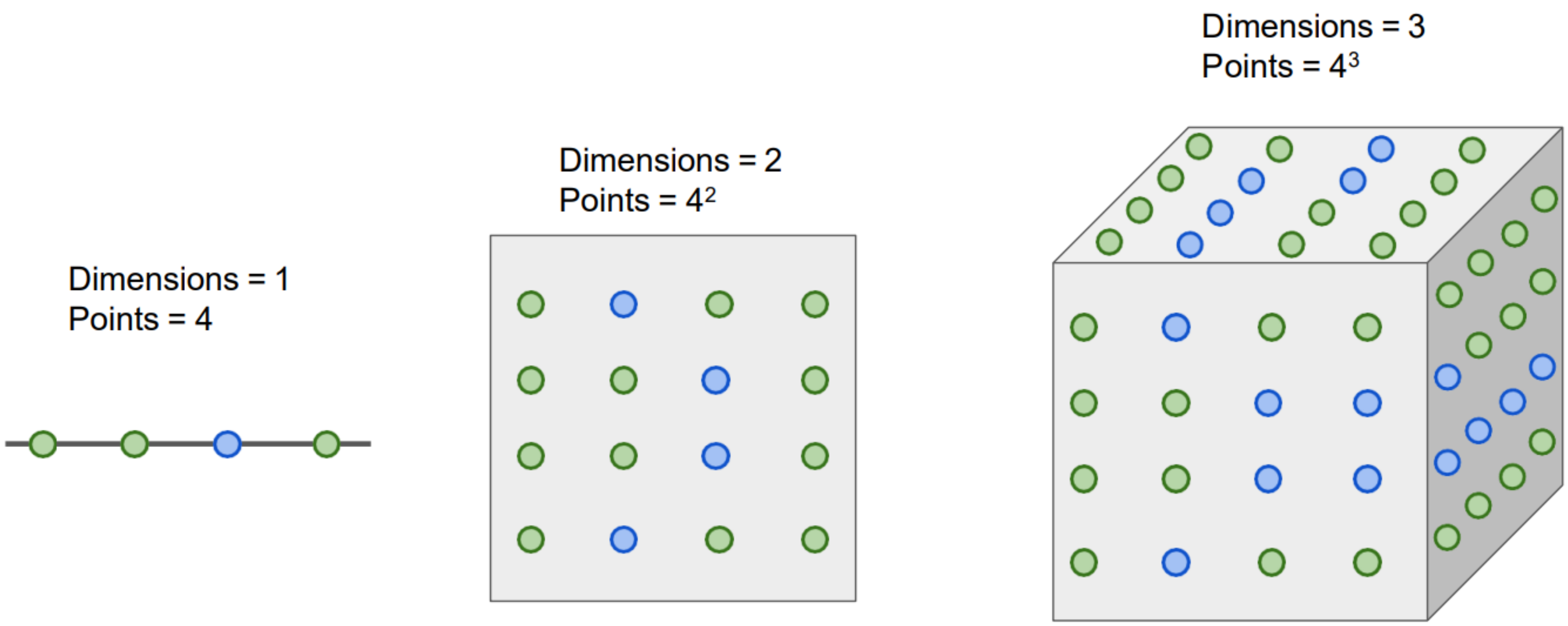
Never

(all 3 images **have same L2 distance** to the one on the left)

Can we use KNN on images?

- *Curse of dimensionality*
- In high-dimensional situations, the data samples are sparse and the distance calculation is difficult

Never



Summary

- **KNN Algorithm**
  - Step1: Find nearest neighbors
  - Step2: Select Class (majority vote)
- **Setting Hyperparameters**
  - value of k
  - distance metric
- **Analysis of KNN**
  - bias and variance
  - complexity(train/predict)

Summary

- **Strength/Weakness of KNN**
  - ✓ Simple to implement and intuitive to understand
  - ✓ Can learn non-linear decision
  - ✓ No Training Time
  - ✕ High prediction complexity for large datasets
  - ✕ Higher prediction complexity with higher dimension
  - ✕ KNN Assumes equal importance to all features
  - ✕ Sensitive to outliers
- **When should we use KNN?**
  - spatial correlation

  - e.g. Recommender system: similarity between users can be viewed as distance)
  - low dimension

  - e.g. Text mining

Practice

- When k=1/3/5, which class will the KNN algorithm discriminate the test sample into?