Nearest Neighbors

Adopted from slides by Roger Grosse, Alex Ihler

Supervised learning

- **Would like to do prediction:**

  estimate a function **f(x)** so that ***y = f(x)***

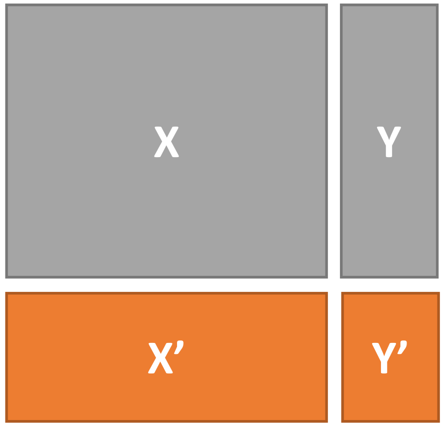
Hope that the same *f(x)* also works on unseen *X', Y'*

- **Where *y* can be:**
  - **Real number**: Regression
  - **Categorical**: Classification

- **Data is labeled:**
  - Have many pairs **{(x, y)}**
    - **x** … vector of binary, categorical, real valued features
    - **y** … class: {+1, -1}, or a real number

Cross Validation

- To evaluate how the model performs on unseen data.
- Process:
  - Randomly partition dataset into k equal-sized subsamples.
  - Retain 1 subsample as test set, use k-1 as training set.
  - Repeat k times, each subsample is test set once.
  - Average results from the k folds to get a single estimation.

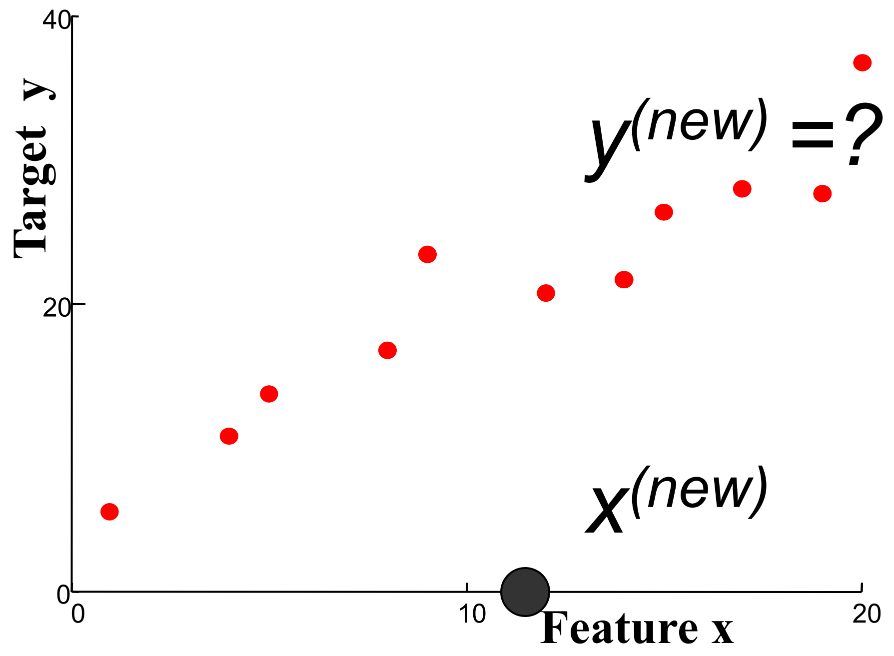


Training and test set

Nearest neighbor

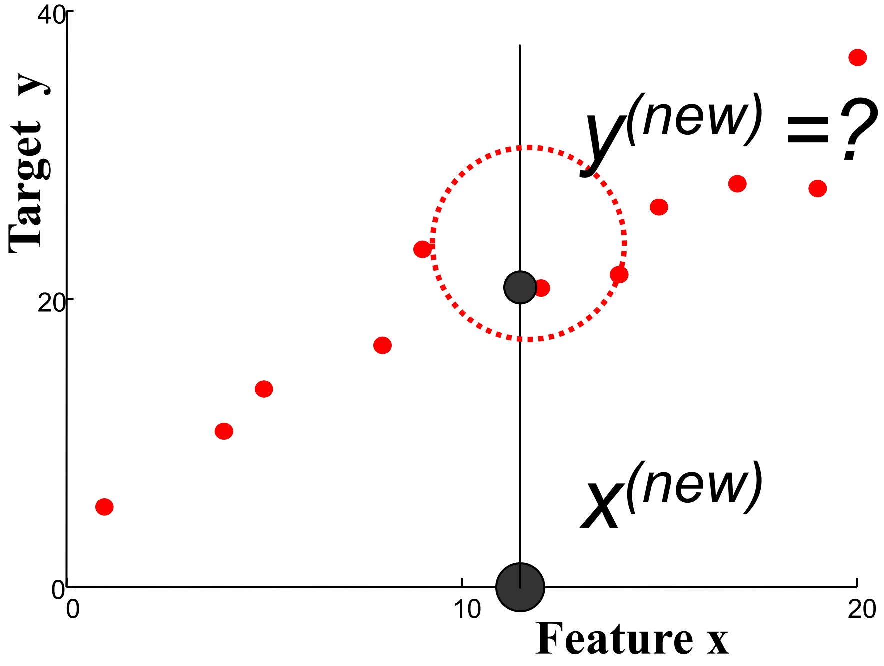
- Keep the whole training dataset: $\{(\boldsymbol{x}, \boldsymbol{y})\}$
- A query example (vector) $\boldsymbol{x}$ comes
- Find closest example(s) $\boldsymbol{x}^*$
- Predict $\widehat{\boldsymbol{y}}$
- **Works for both regression and classification**

Regression; Scatter plots



- Suggests a relationship between x and y
- Regression: given new observed $x^{(new)}$, estimate $y^{(new)}$

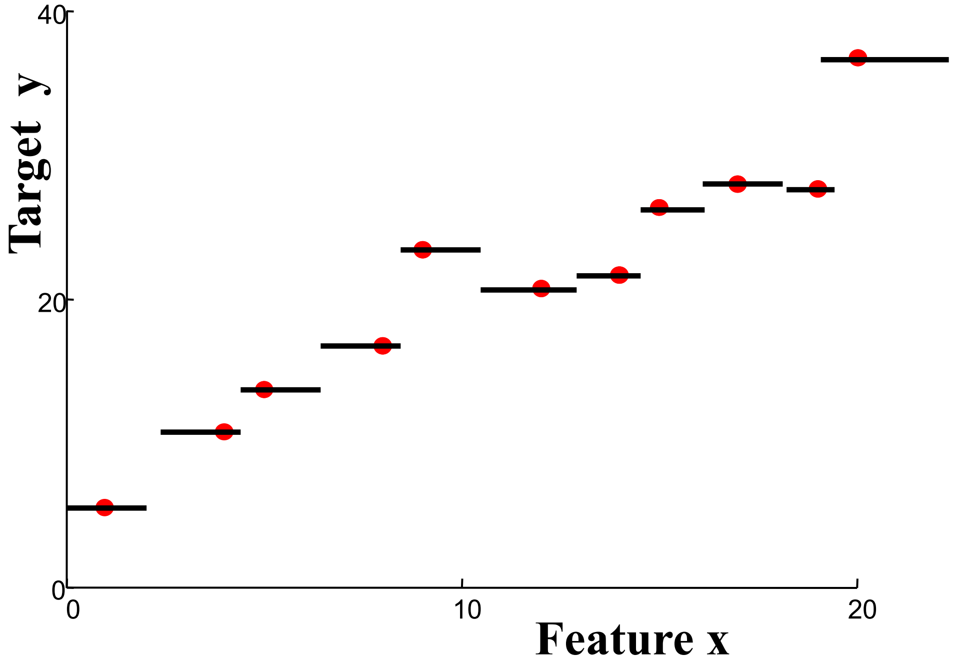
Nearest neighbor regression



"Predictor":
Given new features:
  Find nearest example
  Return its value

- Find training data $x^{(i)}$ closest to $x^{(new)}$; predict $y^{(i)}$

Nearest neighbor regression

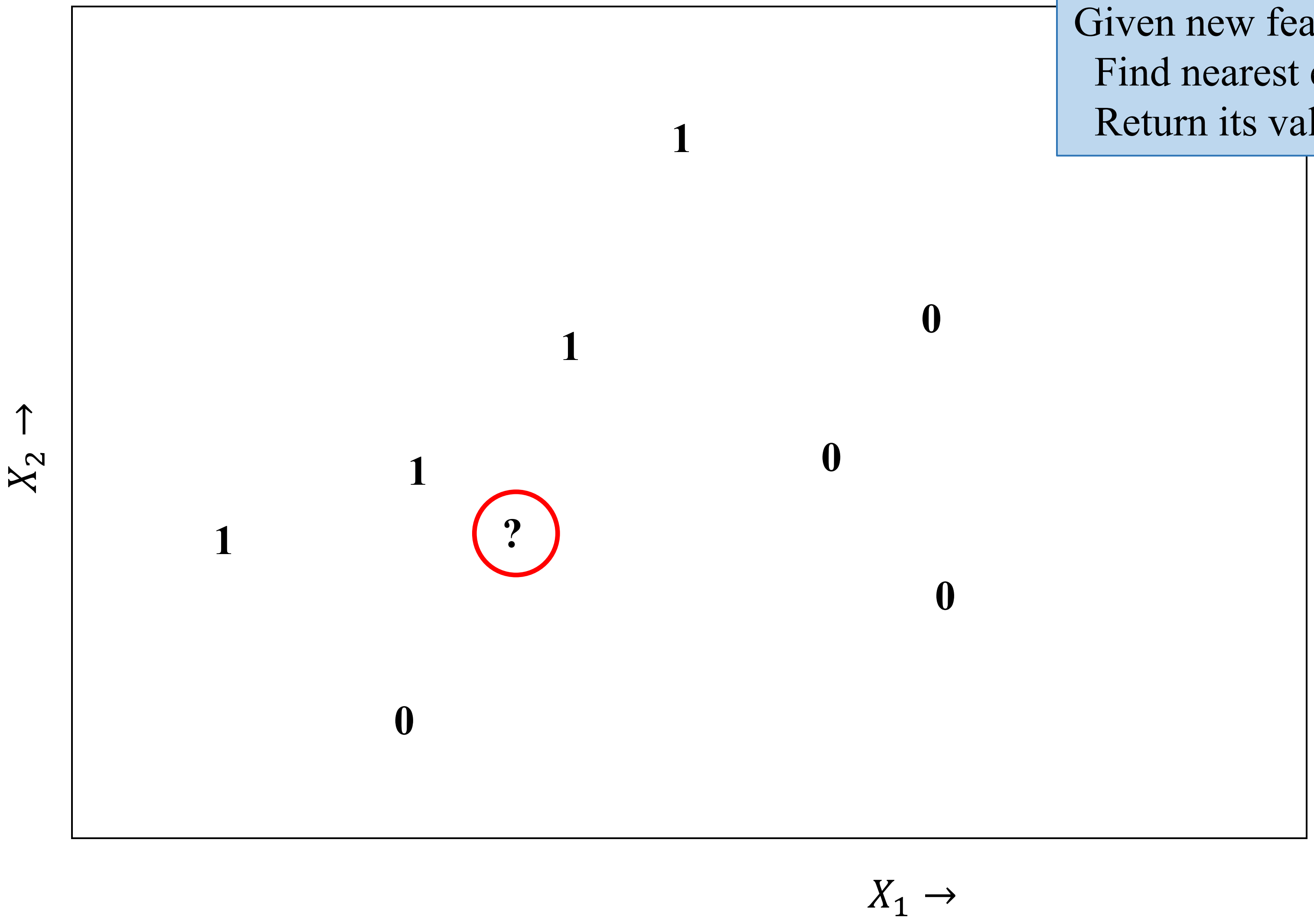


"Predictor":
Given new features:
Find nearest example
Return its value

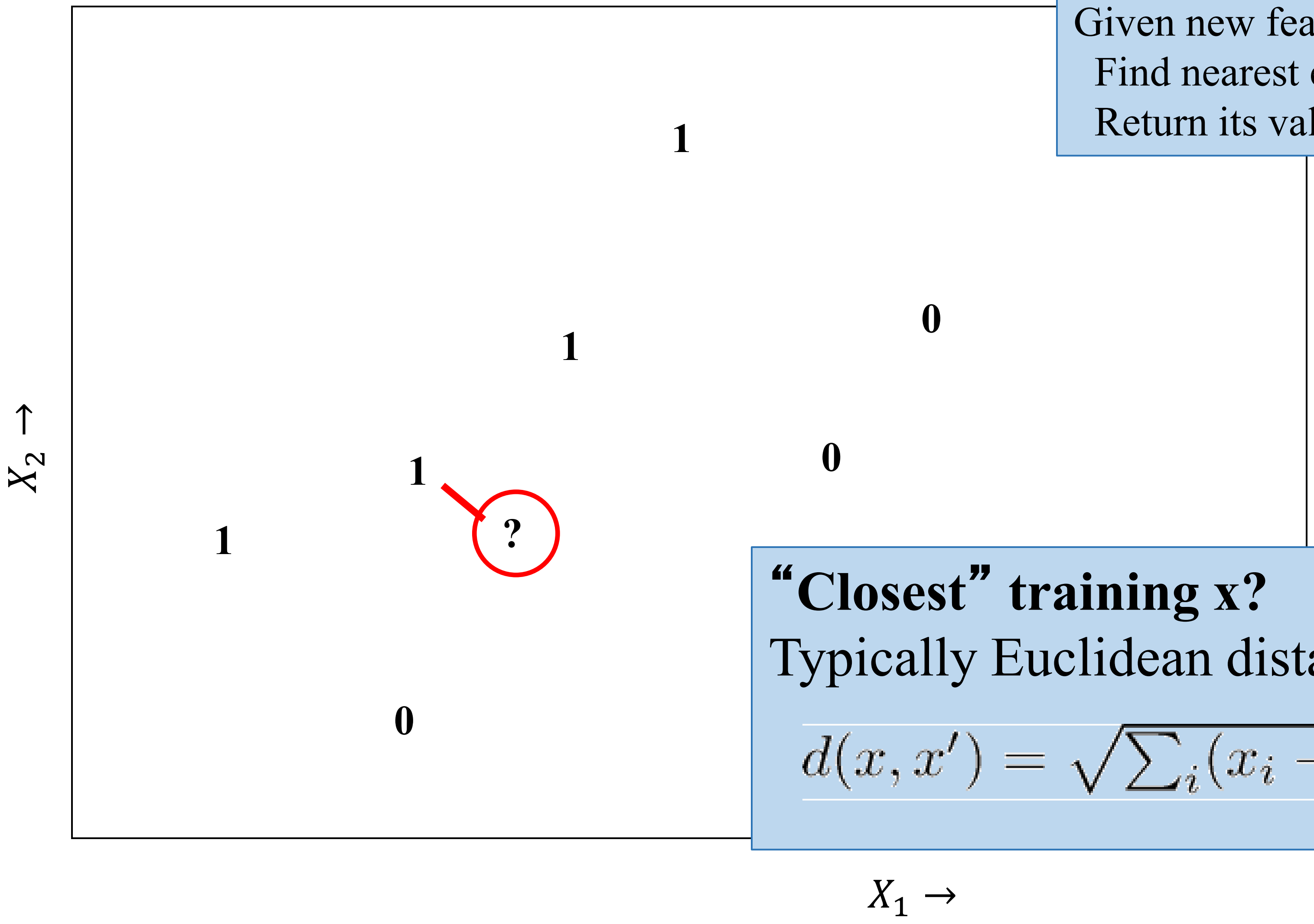
- Find training data $x^{(i)}$ closest to $x^{(new)}$; predict $y^{(i)}$
- Defines an (implict) function f(x)
- "Form" is piecewise constant

Nearest neighbor classifier

"Predictor":
Given new features:
  Find nearest example
  Return its value



Nearest neighbor classifier



"Predictor":

Given new features:

  Find nearest example

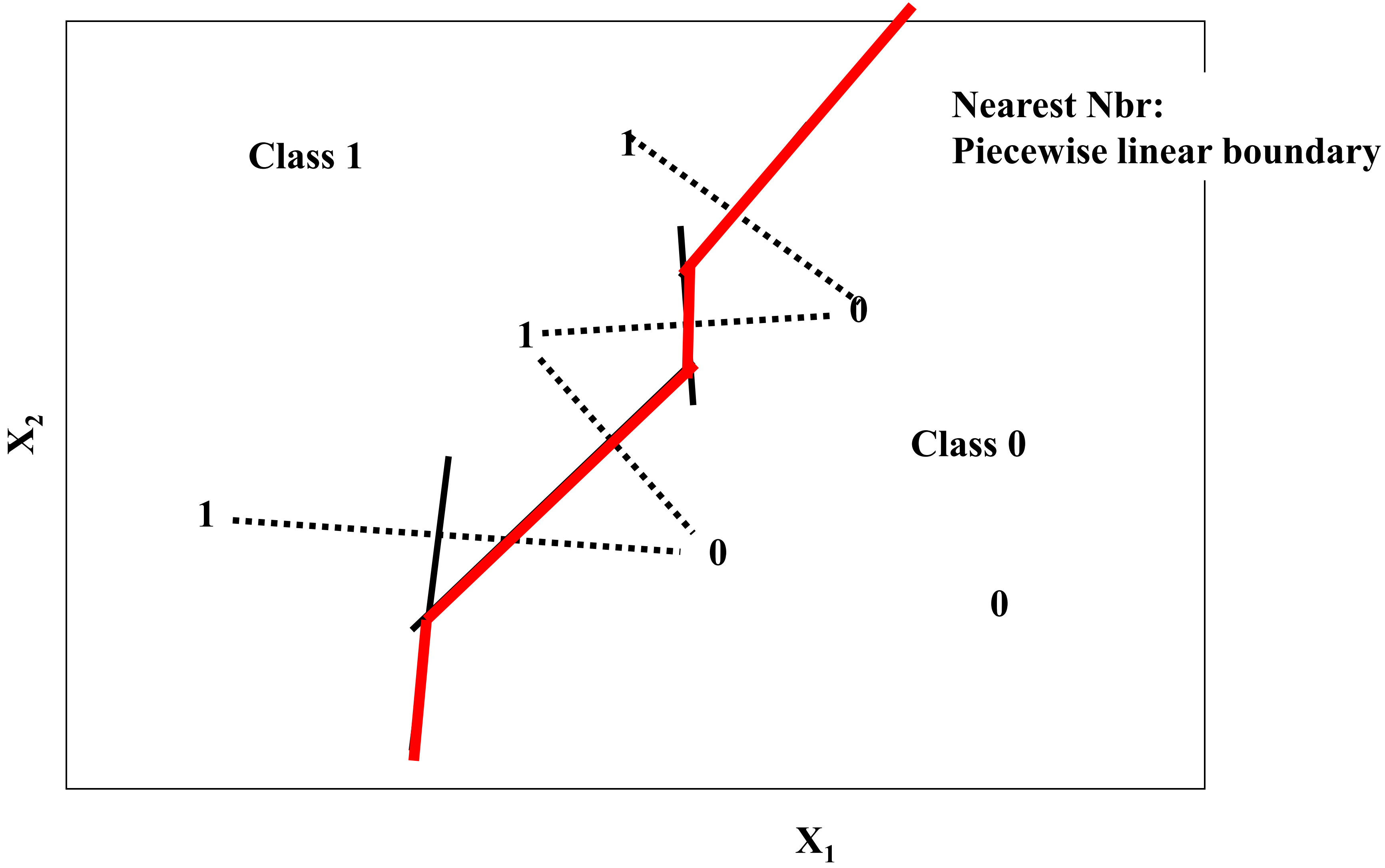
  Return its value

"Closest" training x?

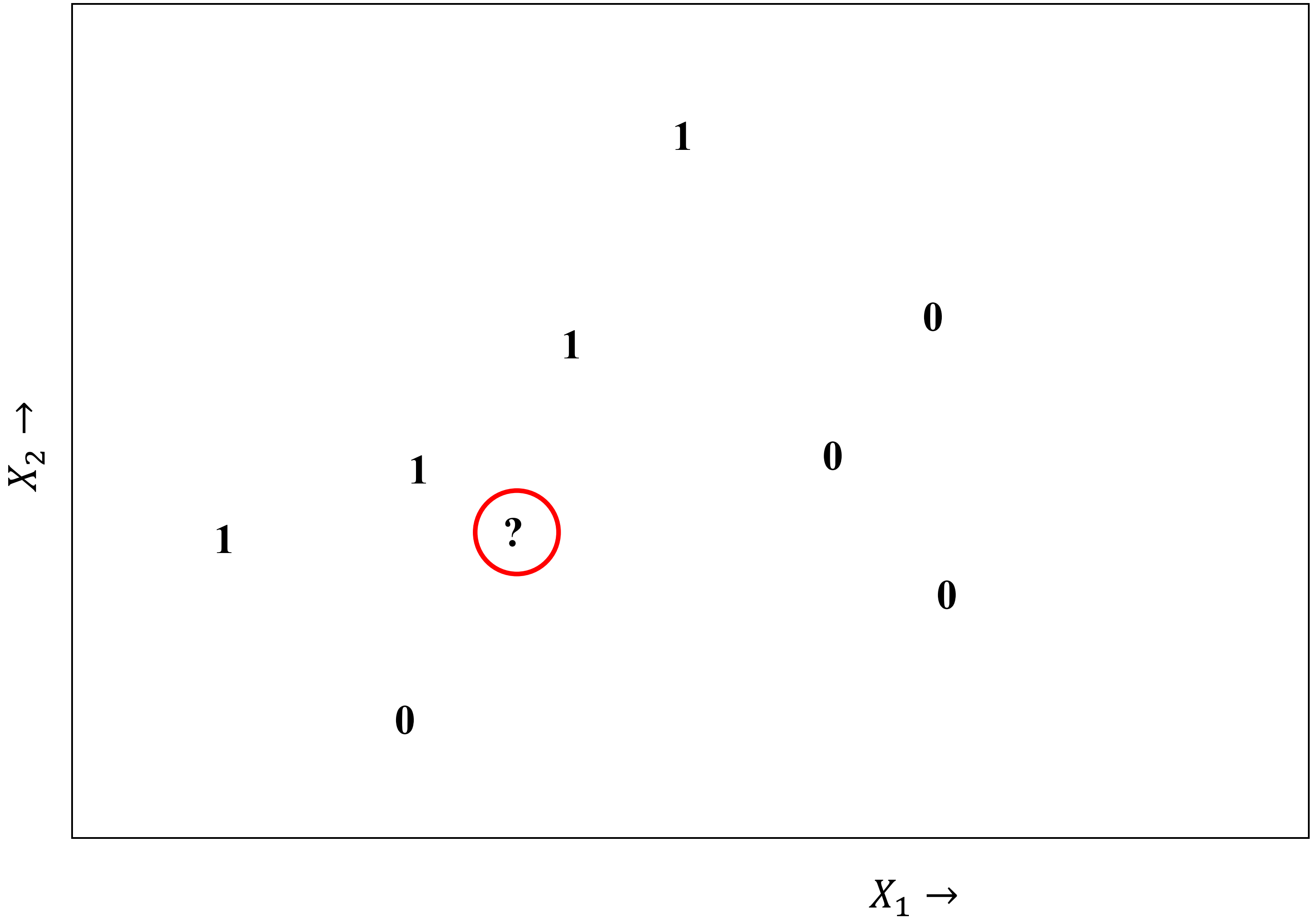
Typically Euclidean distance:

$$d(x, x') = \sqrt{\sum_i (x_i - x'_i)^2}$$

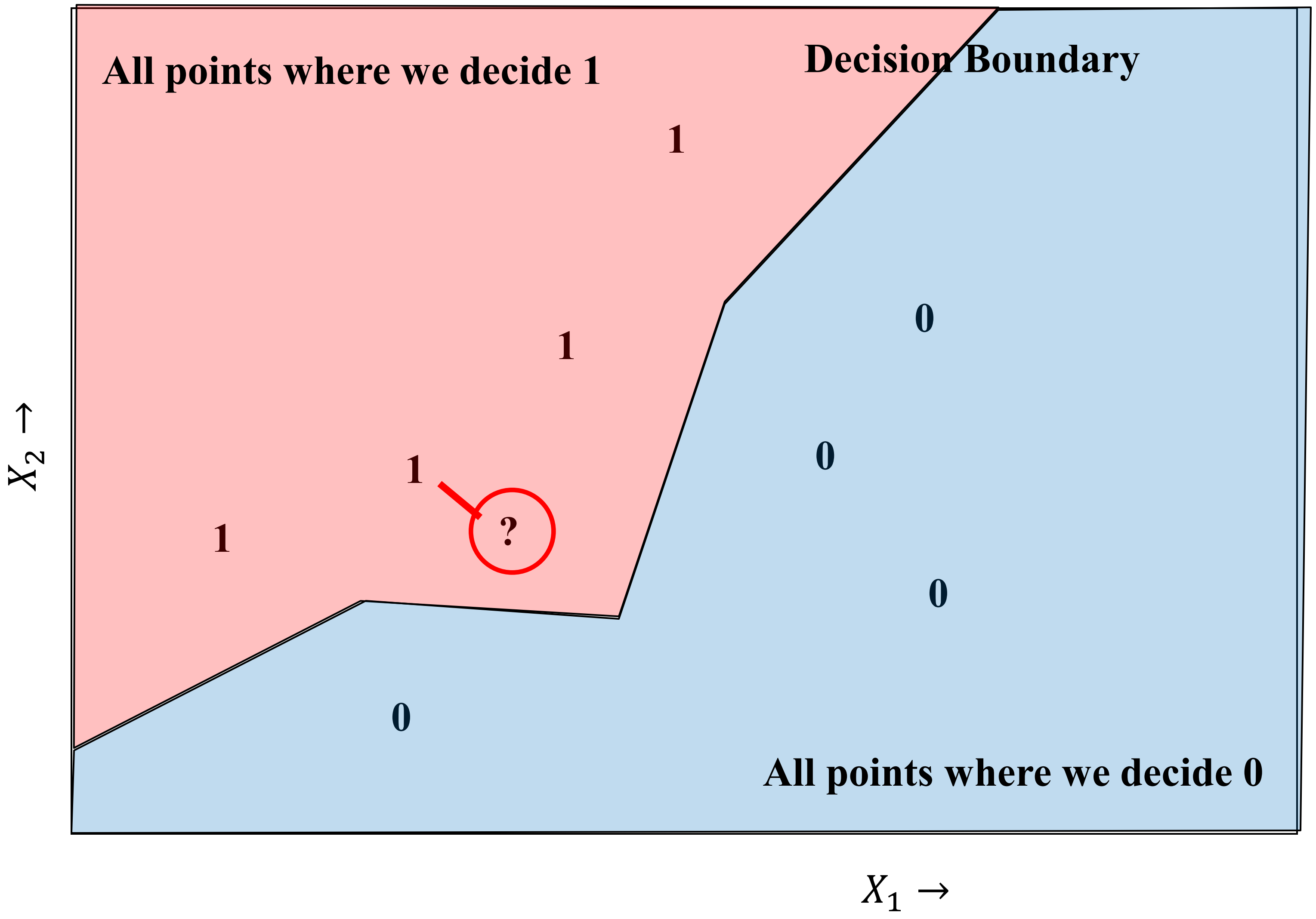
Nearest neighbor classification



Nearest neighbor classifier

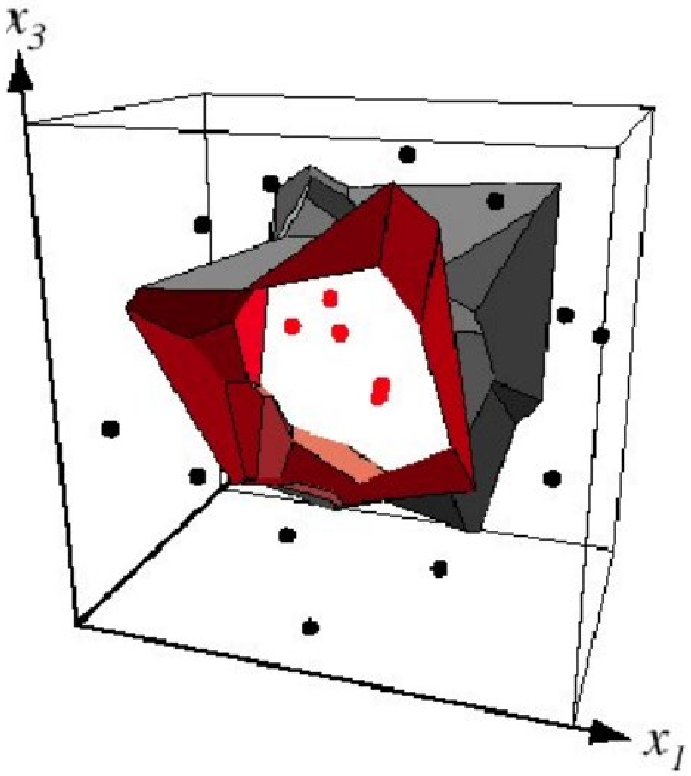


Nearest neighbor classifier



Nearest neighbor classification

- 3D decision boundary



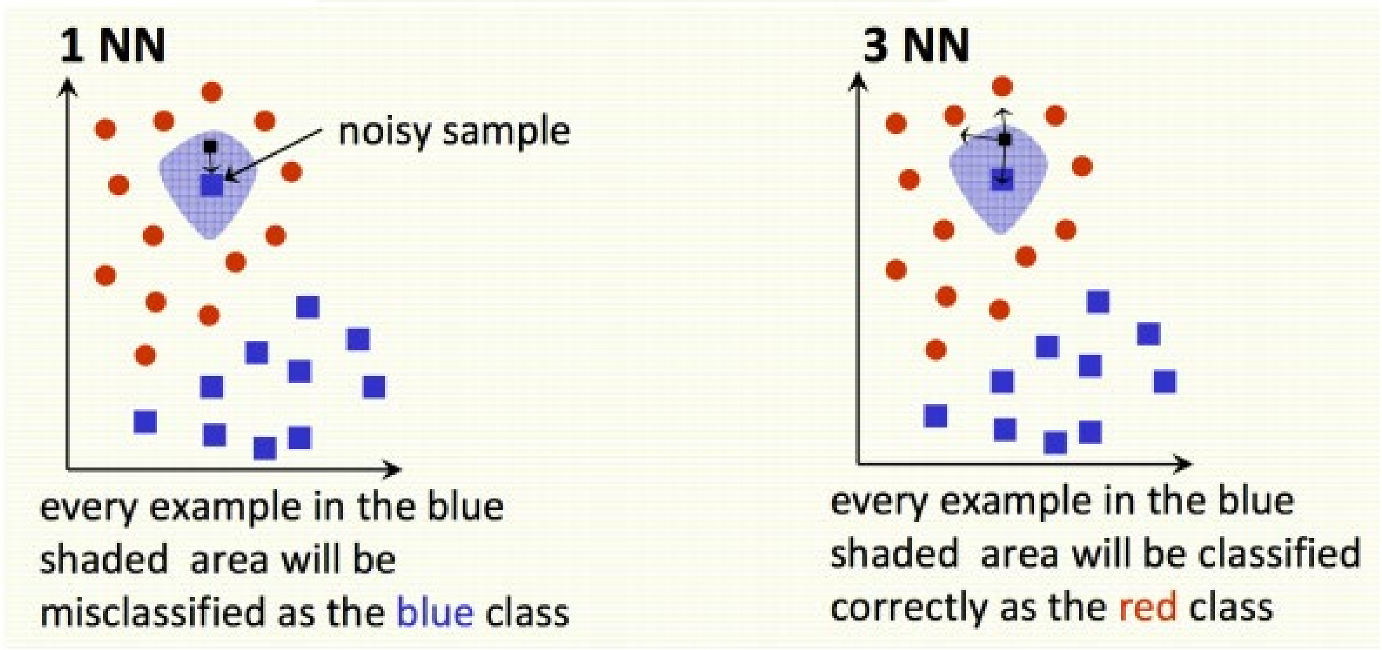
- In general: Nearest-neighbor classifier produces piecewise linear decision boundaries

k-nearest neighbors (kNN)

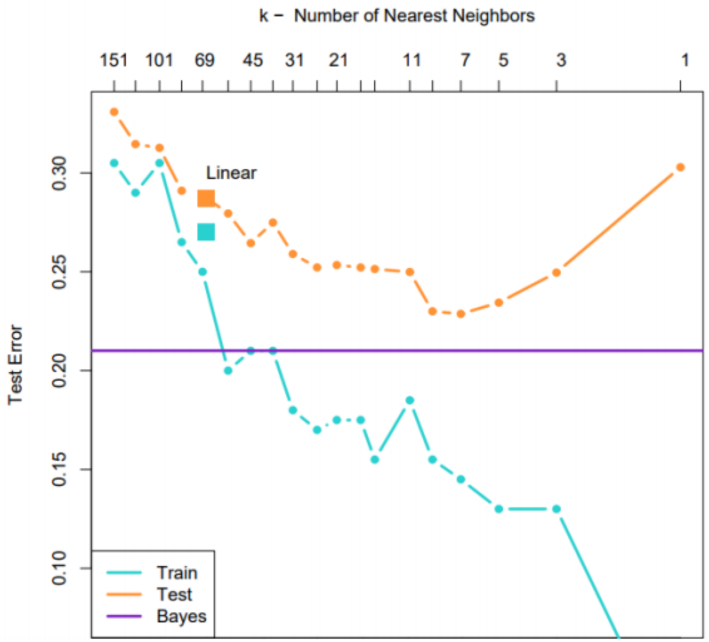
- Find the k-nearest neighbors to $x^{(new)}$ in the data
  - i.e., rank the feature vectors according to Euclidean distance
  - select the k vectors which have smallest distance to $x^{(new)}$
- Regression
  - Usually just average the y-values of the k closest training examples
- Classification
  - ranking yields k feature vectors and a set of k class labels
  - pick the class label which is most common in this set ("vote")
  - Note: for two-class problems, if k is odd ($k = 1, 3, 5, \ldots$) there will never be any "ties"

k-nearest neighbors (k-NN)

- 1-nearest neighbor is sensitive to noise or miss-labeled data
- Solution: smooth by having k nearest neighbors



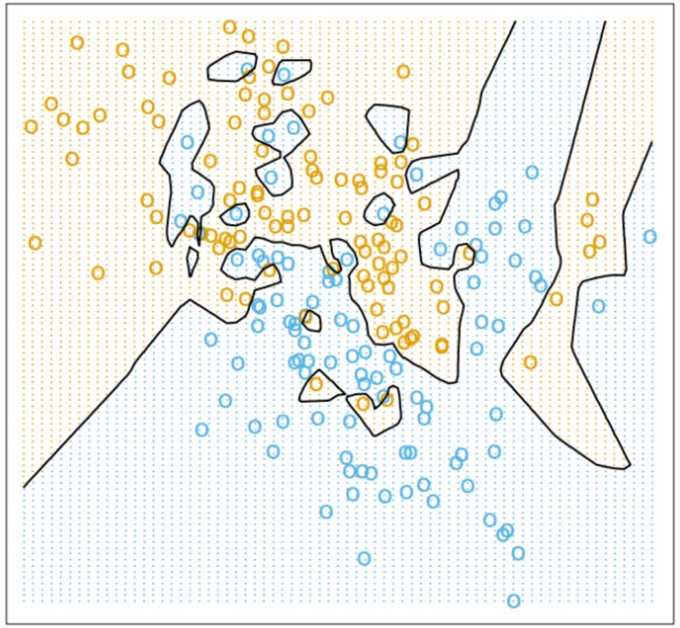
k-nearest neighbors



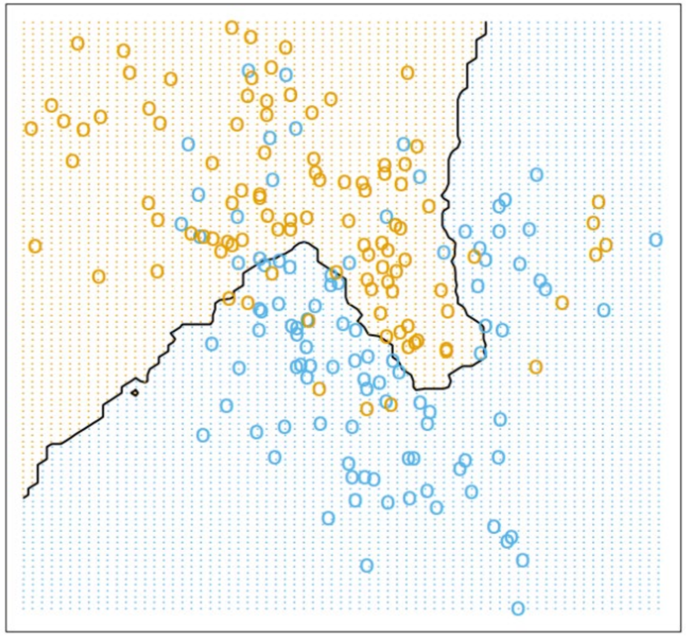
- Tradeoffs in choosing k
  - Small k
    - Good at capturing fine-grained patterns
    - May overfit, i.e. be sensitive to random idiosyncrasies in the training data
  - Large k
    - Makes stable predictions by averaging over lots of examples
    - May underfit, i.e. fail to capture important regularities
  - Rule of thumb: $k < \sqrt{n}$, where n is the number of training examples

k-nearest neighbors

$k = 1$

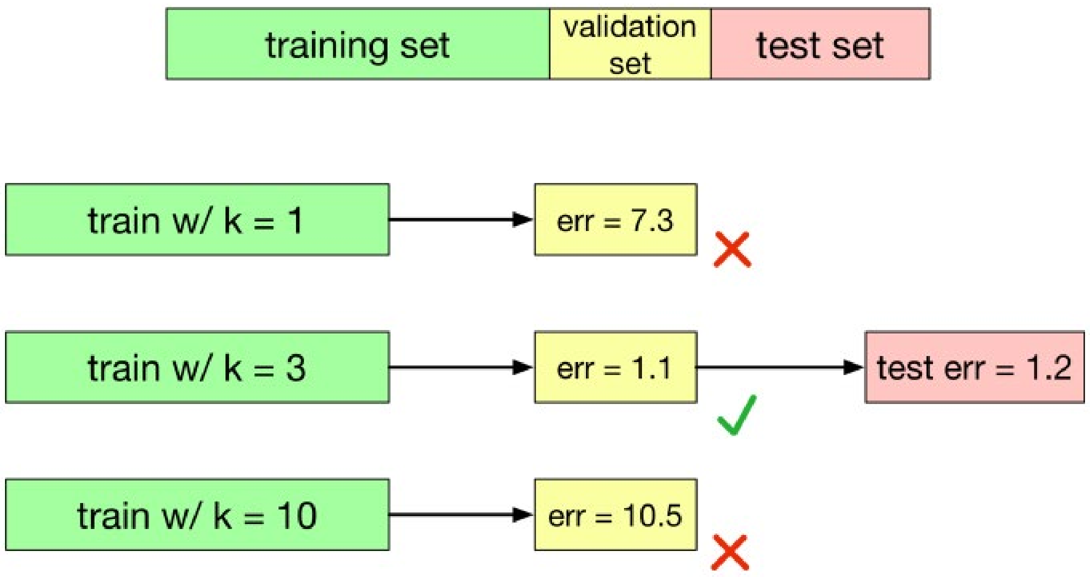


$k = 15$



k-nearest neighbors

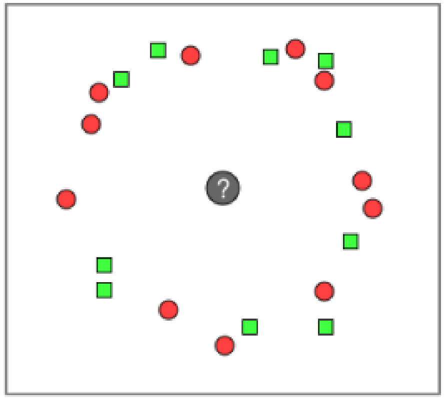
- k is an example of a hyperparameter, something we can't fit as part of the learning algorithm itself
- We can tune hyperparameters using a validation set:

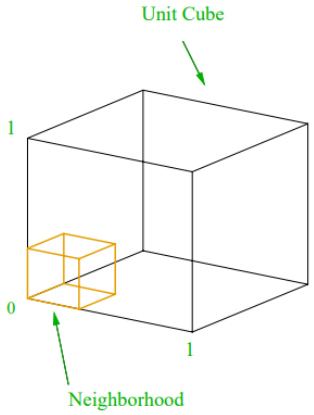


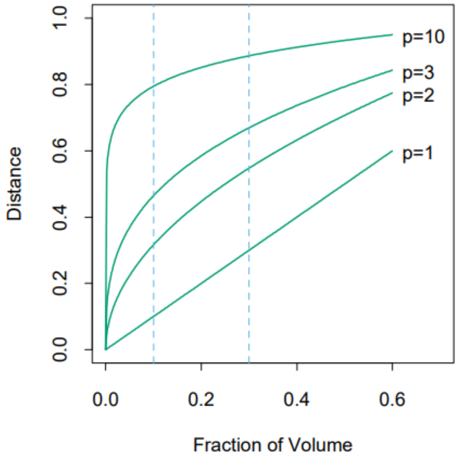
- The test set is used only at the very end, to measure the generalization performance of the final configuration.

Pitfalls: the curse of dimensionality

- Low-dimensional visualizations are misleading! In high dimensions, "most" points are far apart.
- In high dimensions, "most" points are approximately the same distance.







- As the dimensions grow, the amount of data we need to generalize accurately grows exponentially.

Pitfalls: computational cost

- Number of computations at training time: 0 (lazy learning)
- Number of computations at test time, per query (naïve algorithm)
  - Calculate m-dimensional Euclidean distances with n data points: $O(mn)$
  - Sort the distances: $O(n \log n)$
- This must be done for each query, which is very expensive by the standards of a learning algorithm!
- Need to store the entire dataset in memory (non-parametric)!
- Tons of work has gone into algorithms and data structures for efficient nearest neighbors with high dimensions and/or large datasets.

Distance-weighted k-NN

- Might want to weight nearer neighbors more heavily

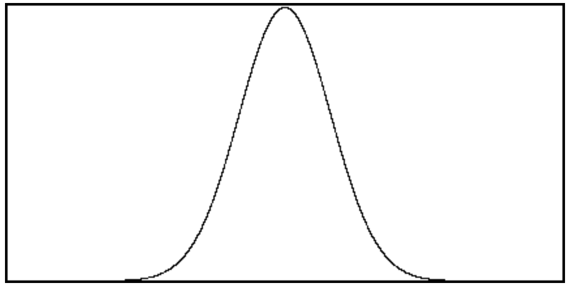
$$f\left(x^{(new)}\right) = \frac{\sum_{i=1}^{n} \omega^{(i)} y^{(i)}}{\sum_{i=1}^{n} \omega^{(i)}}$$

where

$$\omega^{(i)} = \frac{1}{d(x^{(new)}, x^{(i)})^2}$$

and $d\left(x^{(new)}, x^{(i)}\right)^2$ is the distance between $x^{(new)}$ and $x^{(i)}$

- Note now it makes sense to use all training examples instead of k



$$\boldsymbol{w^{(i)}} = \mathbf{exp}\left(-\frac{\boldsymbol{d\left(x^{(i)}, x^{(new)}\right)^2}}{\boldsymbol{K_w}}\right)$$

Summary

- K-nearest neighbor models
  - Classification (vote)
  - Regression (average or weighted average)
- Piecewise linear decision boundary
  - How to calculate
- Simple algorithm that does all its work at test time — in a sense, no learning!
- Test data and overfitting
  - Model “complexity” for knn
  - Use validation data to estimate test error rates & select k