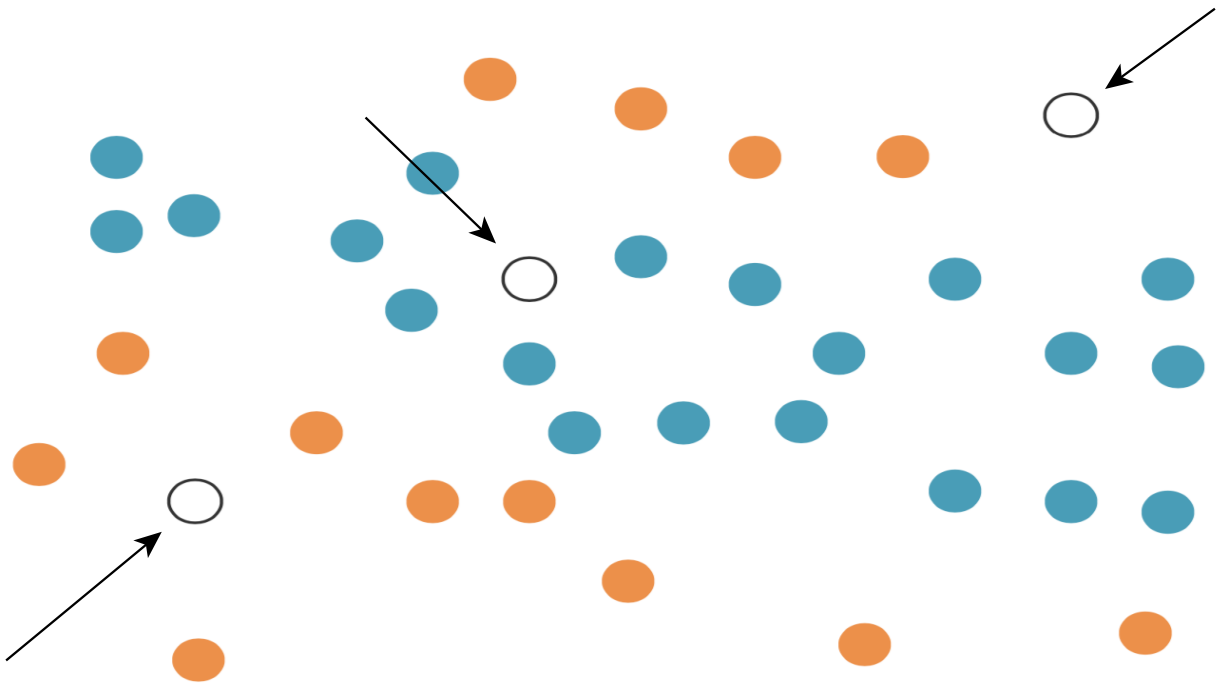
k-Nearest Neighbors (kNN)

a lazy learning algorithm

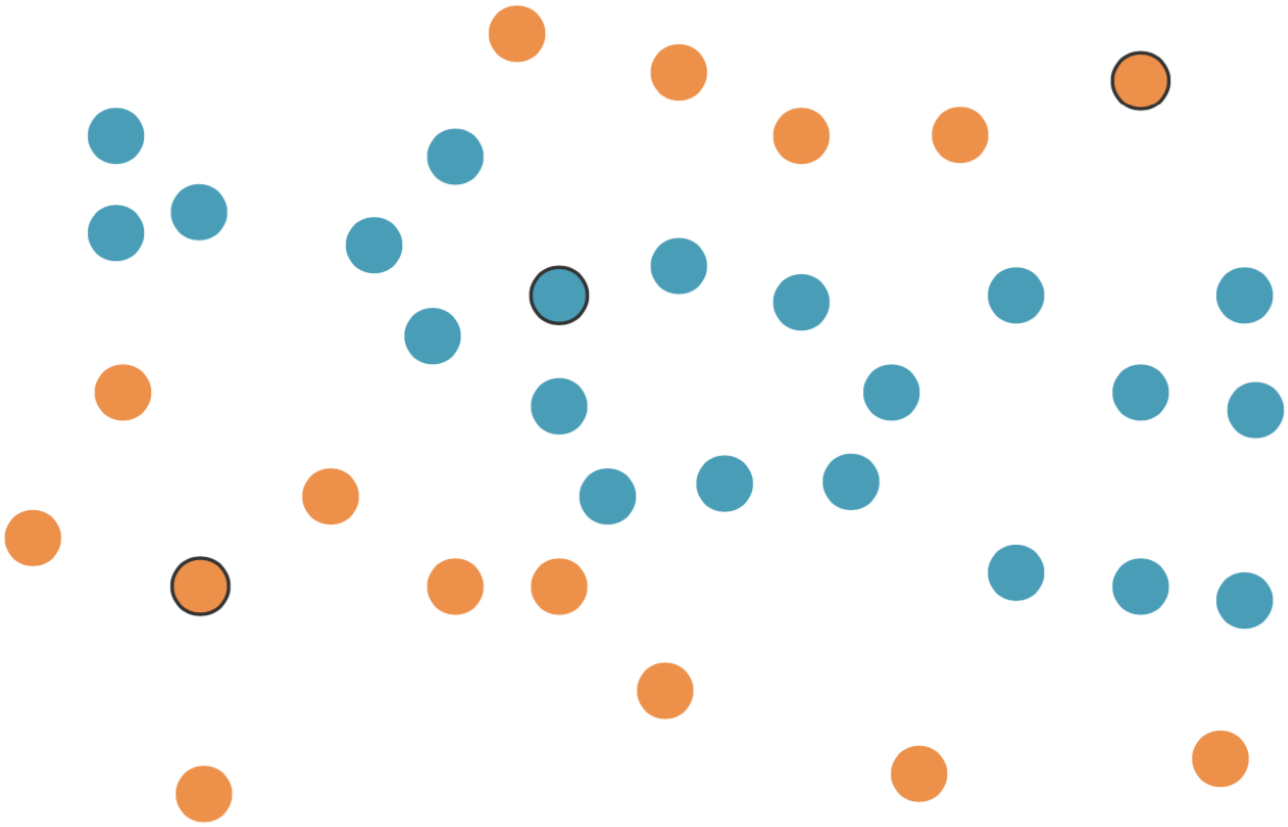
April 15th, 2019

k-Nearest Neighbors

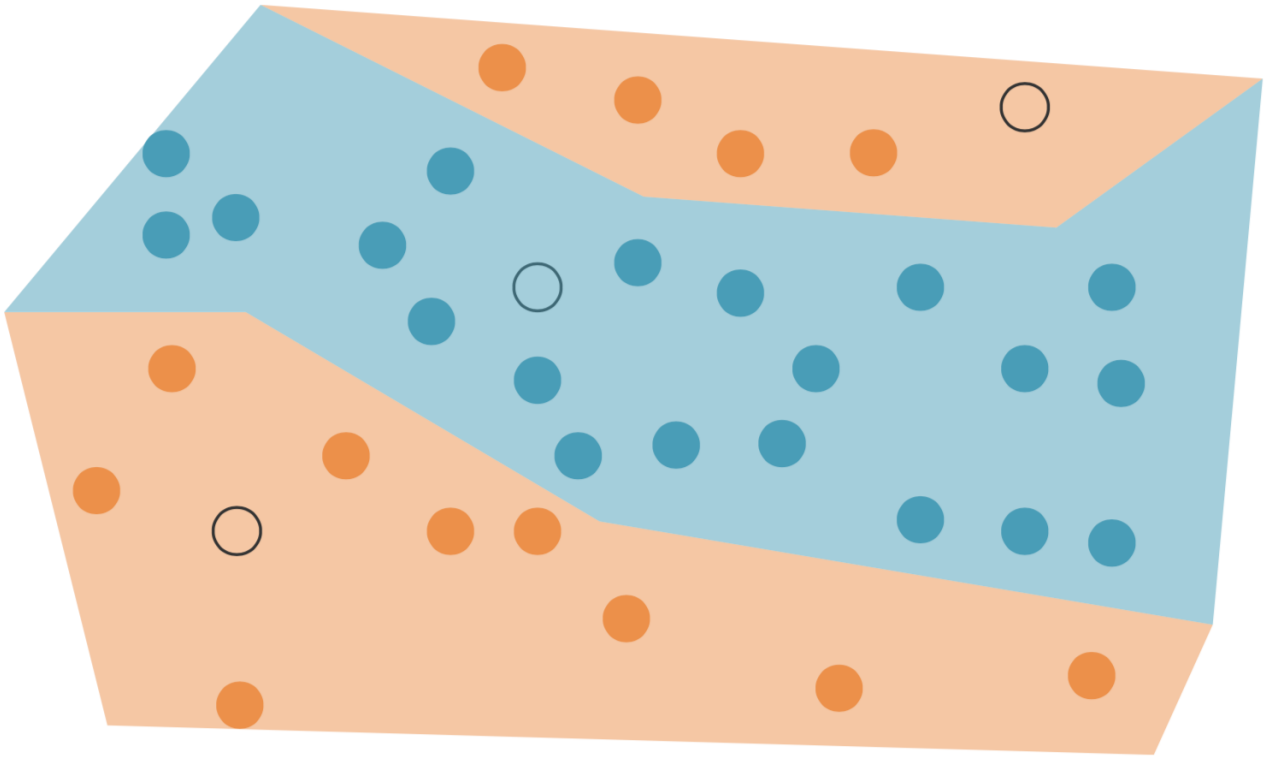
How would you classify the new data points



How would you classify the new data points



Partitioning the whole training space



kNN belongs to a subcategory of non-parametric models

- In ***parametric*** models, we estimate the parameters from the training dataset to learn a function that can classify new data points without requiring the original dataset
- ***Non-parametric*** models cannot be characterized by a fixed set of parameters
  - the complexity of the decision function grows with the number of data points
  - The decision function is expressed directly in terms of the training set

kNN is an instance-based learning

- kNN belongs to a subcategory of non-parametric models know as ***instance-based learning***
  - it learns by memorizing the training dataset
  - the cost of learning is zero
- kNN learning strategy is similar to the ***case-based reasoning***
  - Doctors treats a patient based on how patients with similar symptoms were treated
  - Judges rule court cases based on legal precedent

What are the characteristics of kNN?

- It ***memorizes*** all the ***training set***
- It is a typical example of ***lazy learner***: it doesn't learn a discriminative function from the training data
- When a new data point x is presented, it looks for k training examples that are closest to it and then, labels it accordingly
- In the ***classification*** scenario, it adopts the ***majority vote strategy***, which means that it tries to predict the ***class*** of the most frequent label among the k neighbors
- For regression, the prediction is based on the ***average*** of the labels of the k neighbors

kNN algorithm

1. Choose the number of k and a distance metric
2. Find the k nearest neighbors of the sample that we want to classify
3. Assign the label by majority vote

- **How do we define the value of k?**

Choosing the number of k neighbors

- We choose more than one neighbor to average out the noise of the data
- Therefore, a large value of k increase the computing time (i.e., it is computationally intensive)
- k can be set by ***cross-validation***
- A heuristic comprises in setting $k \approx \sqrt{n}$

What are the advantages and drawbacks of kNN?

- ***Training phase is fast***: just store the training examples
- ***Keeps the training data***: useful there is something using to do with it later on
- ***Almost robust to noisy data***: averaging the k votes
- ***Can learn complex function***
- ***Memory and storage requirements*** is a big issue when dealing with large amounts of data
  - this requires efficient data structures such as KD-trees[1]
- ***Prediction can be slow***: the complexity of labeling a new data point is $\mathcal{O}(pn + n \log k)$
- It can be fooled by ***irrelevant features***
- kNN is susceptible to overfitting due to the ***curse of dimensionality***

[1] Jerome H. Friedman, Jon Louis Bentley, and Raphael Ari Finkel. "An Algorithm for Finding Best Matches in Logarithmic Expected Time". In: *ACM Transactions on Mathematical Software* 3.3 (1977), pp. 209–226.

Computing distances between examples

- ***Euclidean distance***

$$d\left(\boldsymbol{x}^1, \boldsymbol{x}^2\right) = \left\|\boldsymbol{x}^1 - \boldsymbol{x}^2\right\|_2 = \sqrt{\sum_{j=1}^{p}\left(x_j^1 - x_j^2\right)^2}$$

- ***Manhatan distance***

$$d\left(\boldsymbol{x}^1, \boldsymbol{x}^2\right) = \left\|\boldsymbol{x}^1 - \boldsymbol{x}^2\right\|_1 = \sum_{j=1}^{p}\left|x_j^1 - x_j^2\right|$$

- ***Minkowski distance***

$$d\left(\boldsymbol{x}^1, \boldsymbol{x}^2\right) = \left\|\boldsymbol{x}^1 - \boldsymbol{x}^2\right\|_q = \left(\sum_{j=1}^{p}\left|x_j^1 - x_j^2\right|^q\right)^{\frac{1}{q}}$$

Computing similarities between examples

- ***Pearson's correlation***

$$\rho(\boldsymbol{x}, \boldsymbol{z}) = \frac{\sum_{j=1}^{p} (x_j - \overline{x})(z_j - \overline{z})}{\sqrt{\sum_{j=1}^{p} (x_j - \overline{x})^2} \sqrt{\sum_{j=1}^{p} (z_j - \overline{z})^2}}$$

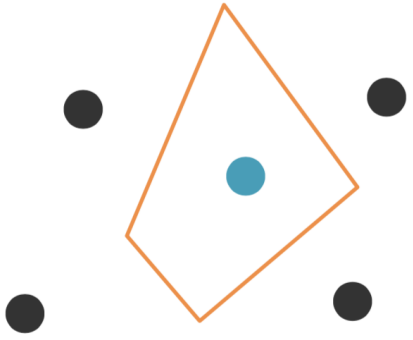
$$\overline{x} = \frac{1}{p} \sum_{j=1}^{p} x_j$$

- Assuming the data are centered

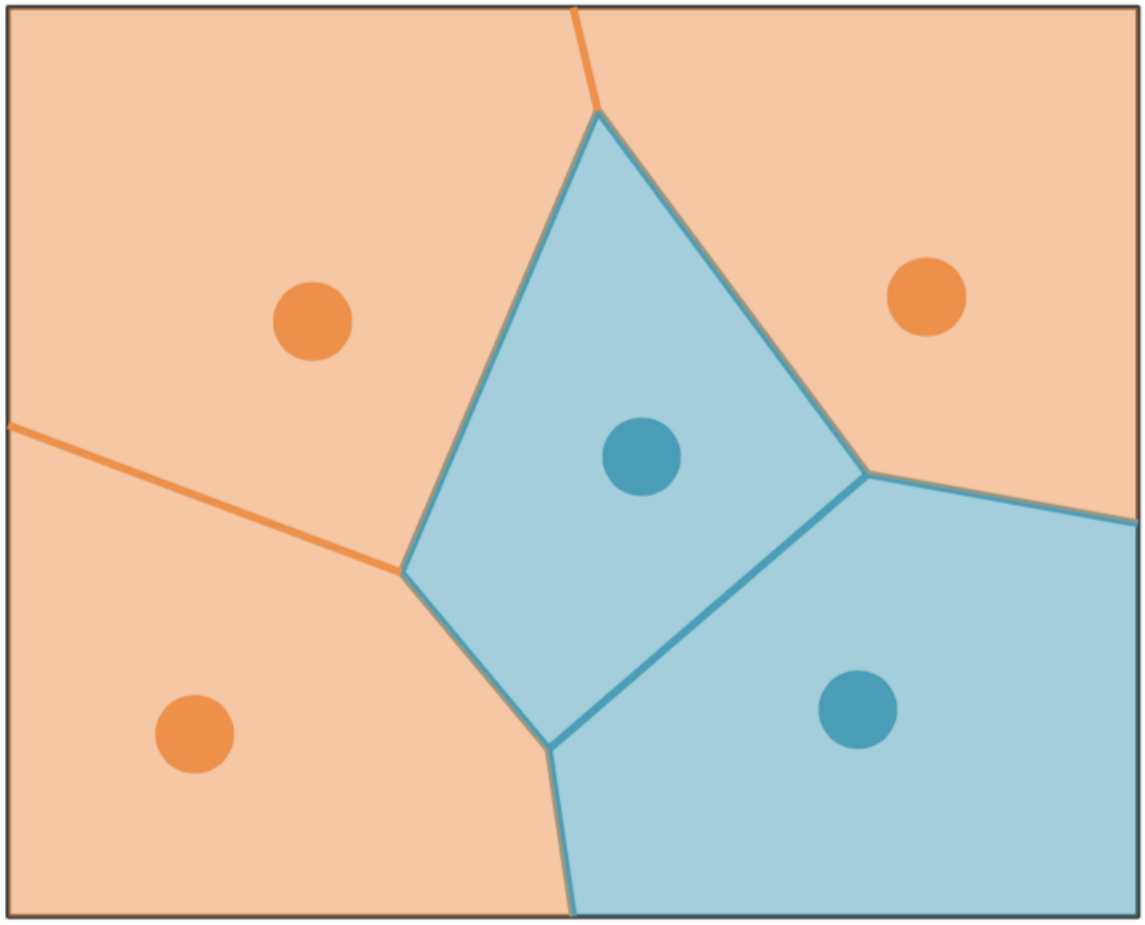
$$\rho(\boldsymbol{x}, \boldsymbol{z}) = \frac{\sum_{j=1}^{p} x_j z_j}{\sqrt{\sum_{j=1}^{p} x_j^2} \sqrt{\sum_{j=1}^{p} z_j^2}}$$

What is kNN's decision boundary?

- ***Decision boundary***: a line separating the positive from negative regions
- ***Voronoi cell*** of x
  - set of all points of the space closer to x than any other points of the training dataset
  - its region comprise a polyhedron
- ***Voronoid tesselation of the space***: union of all Voronoi cells
- Example



Voronoi tesselation defines the decision boundary of the 1-NN



kNN variants

- ε-ball neighbors:
  - Instead of using the k-nearest neighbors, use all points within a distance ε to the new point
- Weighted kNN:
  - Weigh the vote of each neighbor according to the distance to the new data point

$$w_l = \exp\left(\frac{1}{2} d\left(\boldsymbol{x}, \boldsymbol{x}^l\right)\right)$$

Summary

- The cost of the training phase is zero
- Prediction can be ***computationally expensive***: requires efficient data structures (e.g., KD-trees)
- Depends on good distance/similarity function between examples
- Decision boundary is the ***Voronoi tesselation***
- Susceptible to overfit due to the ***curse of dimensionality***

References

- Hal Daume III. *A Course in Machine Learning*. 2nd. Self-published, 2017. URL: http://ciml.info/dl/v0_99/ciml-v0_99-all.pdf

  kNN: sessions 3.2 and 3.2

- Andrew Moore. *An introductory tutorial on kd-trees*. Tech. rep. Technical Report No. 209, Computer Laboratory, University of Cambridge. Pittsburgh, PA: Carnegie Mellon University, 1991. URL: https://bit.ly/2GeVatO

- Voronoi tessellation (*bit.ly/2GlGqdV*)