k-Nearest Neighbors (KNN)

CS 530
Chapman

Spring 2021

1

Take 🏠 message for rest of course: *k*-NN

- *k*-NN does not make any assumptions about the distribution and does not require the construction of an actual model on the training set. But it has various free parameters (k, distance measure) that need optimizing.

2

Table of Contents

3

k-Nearest Neighbors

The "k-nearest neighbors" algorithm is a simple—though often powerful—method to make predictions on previously unseen data based on nearby data points from the training set or feature similarity between the training and test sets.

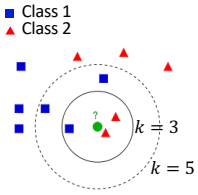


Image Source

4

k-Nearest Neighbors

Say we wish to predict the label of ● (green sample). Does it make more sense that it belongs to Class 1 or to Class 2?

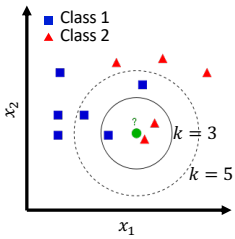


Image Source

5

k-Nearest Neighbors

We can use <u>*k*-nearest neighbors</u> (also known as *k*-NN) to predict the class of ●. We pick a value for k, find the k closest samples to ● in the training set, and take majority vote.

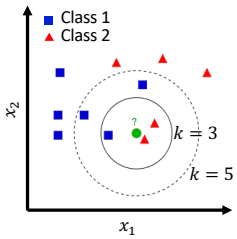


Image Source

6

k-Nearest Neighbors

Note that, in this example, we would assign ● to

- Class 2 for $k = 3$,
- Class 1 for $k = 5$.

Here Class 1/2 might be dog/cat and x_1/x_2 might be *pointy ears* and *size*, for example.

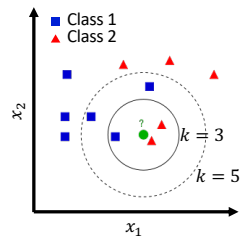


Image Source

7

k-Nearest Neighbors (*k*-NN) algorithm

For training set $\overleftrightarrow{x}^{\text{train}} \in \mathbb{R}^{n^{\text{train}} \times p}$ and associated labels $\vec{y}^{\text{train}} \in \{0,1\}^{n^{\text{train}}}$ and test set $\overleftrightarrow{x}^{\text{test}} \in \mathbb{R}^{n^{\text{test}} \times p}$, we predict the test-set labels, $\hat{\vec{y}}^{\text{test}} \in \{0,1\}^{n^{\text{test}}}$.

8

k-Nearest Neighbors (*k*-NN) algorithm

For training set $\overleftrightarrow{x}^{\text{train}} \in \mathbb{R}^{n^{\text{train}} \times p}$ and associated labels $\vec{y}^{\text{train}} \in \{0,1\}^{n^{\text{train}}}$ and test set $\overleftrightarrow{x}^{\text{test}} \in \mathbb{R}^{n^{\text{test}} \times p}$, we predict the test-set labels, $\hat{\vec{y}}^{\text{test}} \in \{0,1\}^{n^{\text{test}}}$.

1. First, pick a k, typically an odd number.

9

k-Nearest Neighbors (*k*-NN) algorithm

For training set $\overleftrightarrow{x}^{\text{train}} \in \mathbb{R}^{n^{\text{train}} \times p}$ and associated labels $\vec{y}^{\text{train}} \in \{0,1\}^{n^{\text{train}}}$ and test set $\overleftrightarrow{x}^{\text{test}} \in \mathbb{R}^{n^{\text{test}} \times p}$, we predict the test-set labels, $\hat{\vec{y}}^{\text{test}} \in \{0,1\}^{n^{\text{test}}}$.

1. First, pick a k, typically an odd number.
2. Then, for $i = 1, \ldots, n^{\text{test}}$:
   a) For sample $\overleftrightarrow{x}^{\text{test}}(i,:)$ (i.e., for the i'th row of $\overleftrightarrow{x}^{\text{test}}$), find the k nearest samples in $\overleftrightarrow{x}^{\text{train}}$—rows $i_1, \ldots, i_k$
   b) Assign the majority class of those k samples to $\hat{\vec{y}}^{\text{test}}(i)$
   $\hat{\vec{y}}^{\text{test}}(i) = \text{round}(\text{mean}([y(i_1), \ldots, y(i_k)]))$

10

k-Nearest Neighbors (*k*-NN) algorithm

For training set $\overleftrightarrow{x}^{\text{train}} \in \mathbb{R}^{n^{\text{train}} \times p}$ and associated labels $\vec{y}^{\text{train}} \in \{0,1\}^{n^{\text{train}}}$ and test set $\overleftrightarrow{x}^{\text{test}} \in \mathbb{R}^{n^{\text{test}} \times p}$, we predict the test-set labels, $\hat{\vec{y}}^{\text{test}} \in \{0,1\}^{n^{\text{test}}}$.

1. First, pick a k, typically an odd number.
2. Then, for $i = 1, \ldots, n^{\text{test}}$:
   a) For sample $\overleftrightarrow{x}^{\text{test}}(i,:)$ (i.e., for the i'th row of $\overleftrightarrow{x}^{\text{test}}$), find the k nearest samples in $\overleftrightarrow{x}^{\text{train}}$—rows $i_1, \ldots, i_k$
   b) Assign the majority class of those k samples to $\hat{\vec{y}}^{\text{test}}(i)$
   $\hat{\vec{y}}^{\text{test}}(i) = \text{round}(\text{mean}([y(i_1), \ldots, y(i_k)]))$

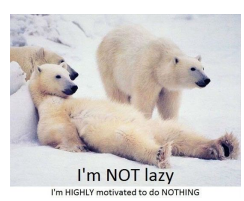
For 3 or more classes, assign the *plurality* of $[y(i_1), \ldots, y(i_k)]$ to $\hat{\vec{y}}^{\text{test}}(i)$ in step (b).

11

About *k*-NN algorithm

k-NN often termed non-parametric, lazy learning algorithm

- Non-parametric: makes no assumptions about any underlying model for data
- Lazy: does not strictly learn and store a model
  - Requires access to entire training set to compute labels for test set

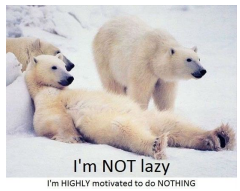


12

About KNN algorithm

Nevertheless, *k*-NN has 1 main assumption about data:

Smoothness—samples close to each other belong to the same class



13

Distance Metrics

How do we determine which of the neighboring points are "nearest"? That depends on the distance metric we use (i.e., on how we define distance). Distance metrics include:

- Minkowski Distances (l_p norm)
  - Euclidean Distance ($p = 2$)
  - Manhattan Distance ($p = 1$)
  - Chebyshev Distance ($p = \infty$)
- Mahalanobis Distance

14

Distance Metrics: Minkowski Distance

A broad class of distance metrics are known as Minkowski distance metrics (a.k.a. Minkowski norms, or l_p norms).

If we have two points $\vec{x}$ and $\vec{y}$ in n-dimensional space, with coordinates

$$\vec{x} = (x_1, \dots, x_n)$$

$$\vec{y} = (y_1, \dots, y_n)$$

the Minkowski distance between $\vec{x}$ and $\vec{y}$ is

$$d(\vec{x}, \vec{y})_p = \|\vec{x} - \vec{y}\|_p = \left(\sum_{i=1}^{n} |x_i - y_i|^p\right)^{\frac{1}{p}}$$

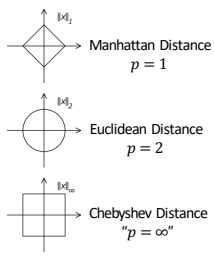


Image Source

15

Distance Metrics: Euclidean Distance ($p = 2$)

The most commonly used distance metric is Euclidean distance:

$$d(\vec{x}, \vec{y})_2 = \sqrt{\sum_{i=1}^{n} (x_i - y_i)^2}$$

Its formula comes from the Pythagorean theorem.

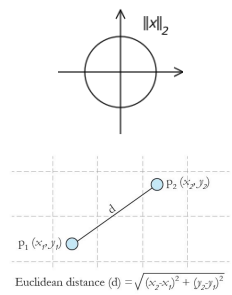


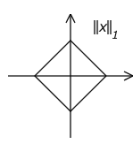
Image Source

16

Distance Metrics: Manhattan Distance ($p = 1$)

Manhattan distance (a.k.a. the taxicab metric, city-block distance) is

$$d(\vec{x}, \vec{y})_1 = \|\vec{x} - \vec{y}\|_1 = \sum_{i=1}^{n} |x_i - y_i|$$



Sum of distances (absolute value of differences) across dimensions. Gets its name from rectangular grid of current-day Manhattan streets. Traveling between two points requires moving along streets, not a direct route through buildings.

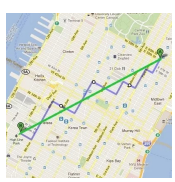


Image Source

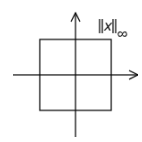
17

Distance Metrics: Chebyshev Distance ($p = \infty$)

Chebyshev distance (a.k.a. maximum metric, chessboard distance) is

$$d_{\max}(\vec{x}, \vec{y}) = \lim_{p \to \infty} \left(\sum_{i=1}^{n} |x_i - y_i|^p\right)^{\frac{1}{p}}$$

$$= \max_i |x_i - y_i|$$



This is also called chessboard distance because it defines how many steps are required for a king to reach another square on the board.

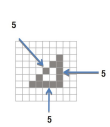


Image Source

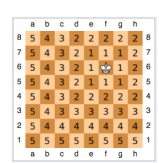
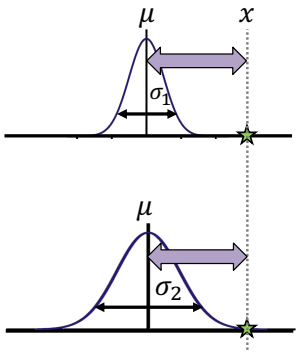


Image Source

18

Distance Metrics: Mahalanobis Distance

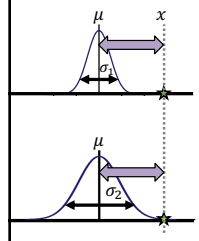


$$d_{\mathrm{M}_1}(\vec{x}, D) = \frac{|x-\mu|}{\sigma_1}$$

$$d_{\mathrm{M}_2}(\vec{x}, D) = \frac{|x-\mu|}{\sigma_2}$$

19

Distance Metrics: Mahalanobis Distance



$$d_{\mathrm{M}_1}(\vec{x}, D) = \frac{|x-\mu|}{\sigma_1}$$

$$d_{\mathrm{M}_2}(\vec{x}, D) = \frac{|x-\mu|}{\sigma_2}$$

$\sigma_1 < \sigma_2 \;\rightarrow\; d_{\mathrm{M}_1} > d_{\mathrm{M}_2}$

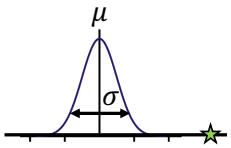
20

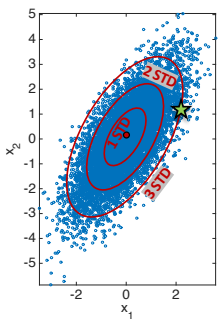
Distance Metrics: Mahalanobis Distance

Mahalanobis distance of sample $\vec{x}$ from a multi-dimensional distribution, D, with mean vector $\vec{\mu}$ and covariance matrix $\vec{S}$ is defined

$$d_{\mathrm{M}}(\vec{x}, D) = \sqrt{(\vec{x}-\vec{\mu})^T \vec{S}^{-1} (\vec{x}-\vec{\mu})}$$

It is thus the multidimensional generalization of measuring how many standard deviations a sample is from a distribution's mean.





21

Distance Metrics

Which distance metric should we use?

Euclidean distance is most commonly used, but sometimes other metrics provide better performance. Finding the best metric can sometimes involve trial and error (cross validation?).

22

Curse of Dimensionality

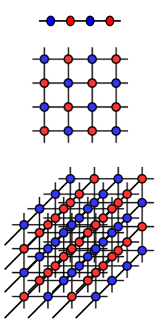
When fitting models like k-NN, we must be vigilant about the curse of dimensionality.

With more features (variables), the feature space is higher-dimensional. More dimensions lead to sparser tiling of the space. And this can interfere with the performance of k-NN, so we should be mindful about feature selection.

1D:

2D:

3D:



23

Which value of k for k-NN?

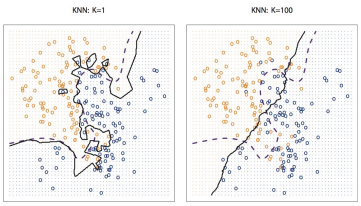


Image Source

Which value of k should we use for k-NN? Generally,

- Small values of k tend to overfit—high variance—so resulting model very sensitive to specific training set, noise, etc.
- Large values of k tend to underfit—high bias—miss some classification features

24

$k=1$ and Voronoi Diagrams

When applying k-NN for k=1, the space gets divided into regions closest to each point. This is known as a Voronoi diagram (or Voronoi tessellation). This is helpful for visualizing how 1-NN classifies.

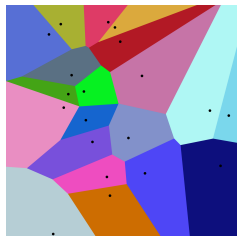


Image Source

25

Choosing k for k-NN

We can use cross-validation to pick the best value for k. We can create k-NN models for each value of k, apply the model to both the training and test sets, and find a model that minimizes test error.

The graph suggests that—here specifically—a value of $1/k \approx 0.11$ —or $k \approx 9$—yields the lowest test error. This finds a balance between underfitting, towards the left side of the graph (with k too large) and overfitting, towards the right side of the graph (with k too small).

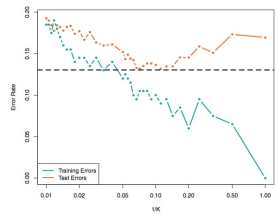


Image Source

26

Scaling Variables

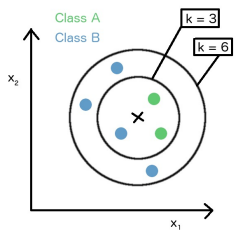
Good model fitting sometimes requires scaling variables (a.k.a. feature weighting). Say we use k-NN to classify the level of education (college degree yes/no) compared to income and age:

$x_1 \leftarrow$ parents' income

$x_2 \leftarrow$ age

Class A $\leftarrow$ College degree

Class B $\leftarrow$ No college



27

Scaling Variables

Suppose we are trying to use k-NN, and we want to determine which is closer to Test Subject—Subject 1 or Subject 2.

Intuitively, Subject 1 is closer to Test Subject: no difference in age & slight difference in income

However, if we used the same scales for age and income, the distance between Test Subject and Subject 1 is 5,000, while the distance between Test Subject and Subject 2 is 45.

A common scaling method is z-scoring.

Test Subject:
Age: 25
Income: $85,000

Subject 1:
Age: 25
Income: $90,000

Subject 2:
Age: 70
Income: $85,000

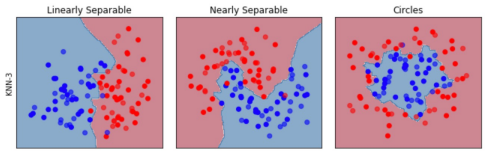
28

Python code

```python
# K-NN
from sklearn.neighbors import KNeighborsClassifier

clf = KNeighborsClassifier(n_neighbors=3)  # number of neighbors
f, axarr = visualize_clf(datasets, clf)
axarr[0].set_ylabel("KNN-3")
```

```
Text(77,0.5,'KNN-3')
```



29

Choosing K in KNN

```python
# split X and y into training and testing sets
from sklearn.cross_validation import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.4, random_state=4)

from sklearn import metrics
# try K=1 through K=25 and record testing accuracy
k_range = range(1, 26)

scores = []
for k in k_range:
    knn = KNeighborsClassifier(n_neighbors=k)
    knn.fit(X_train, y_train)
    y_pred = knn.predict(X_test)
    scores.append(metrics.accuracy_score(y_test, y_pred))

print(scores)
```

```
[0.92500000000000004, 0.94999999999999996, 0.97499999999999998, 0.94999999999999996, 0.97499999999999998, 0.97499999999999998, 0.97499999999999998, 0.97499999999999998, 0.94999999999999996, 0.97499999999999998, 1.0, 0.97499999999999998, 0.97499999999999998, 0.97499999999999998, 0.94999999999999996, 0.94999999999999996, 0.94999999999999996, 0.94999999999999996, 0.94999999999999996, 0.94999999999999996, 0.94999999999999996, 0.94999999999999996, 0.94999999999999996, 0.94999999999999996, 0.94999999999999996, 0.94999999999999996, 0.94999999999999996, 0.94999999999999996, 0.94999999999999996, 0.94999999999999996]
```

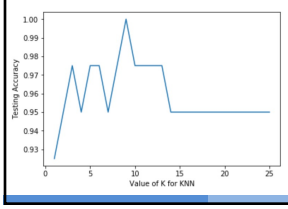
30

```
%matplotlib inline
import matplotlib.pyplot as plt

# plot the relationship between K and testing accuracy
# plt.plot(x_axis, y_axis)
plt.plot(k_range, scores)
plt.xlabel('Value of K for KNN')
plt.ylabel('Testing Accuracy')


#Training accuracy rises as model complexity increases
#Testing accuracy penalizes models that are too complex or not complex enough
#For KNN models, complexity is determined by the value of K (lower value = more complex)
```

```
Text(0,0.5,'Testing Accuracy')
```



31

Advantages of *k*-NN

Advantages of *k*-nearest neighbors:

- Instance-based and lazy learning algorithm—all computation is put off until we need to classify. There's no model to train. And new training data samples can be easily incorporated
- The model is intuitive—easy to understand and to implement
- It is simple and powerful. No complex tuning parameters are required to build the model
- Provides good classification when the number of samples is large with respect to the number of features (good tiling)
  - It sometimes outperforms more complex models

32

Disadvantages of *k*-NN

Disadvantages of k-nearest neighbors:

- Can be computationally expensive and slow. Computing time is $O(np)$, where n is number of samples, p number of dimensions
- Choosing k not trivial and can have large influence on classification outcome
- Requires exponentially large training set with respect to p (number of dimensions or features) to predict well. In other words, requires good tiling of the training space

33

Summary

- *k*-Nearest Neighbors
  - Classify test-set samples based on majority (or plurality) vote between *k* nearest neighbors of each
  - Distance Metrics: Euclidean, Manhattan, Chebyshev, Mahalanobis
  - Choosing *k* optimal for *k*-NN difficult, cross-validation can be used
  - Scaling Variables often useful for *k*-NN
  - Suffers from Curse of Dimensionality: number of samples required to well define higher-dimensional space rises exponentially with space's dimension

34

Take 🏠 message for rest of course: Naïve Bayes & *k*-NN

- <u>*k*-NN</u> does not make any assumptions about data distributions and does not require training a model on the training set. But it has various free parameters (k, distance measure) that need optimizing. And it is especially prone to the curse of dimensionality.

35