AI6102: Machine Learning Methodologies & Applications

L8a: K-NN Classifiers

Sinno Jialin Pan

Nanyang Technological University, Singapore

Homepage: http://www.ntu.edu.sg/home/sinnopan

Typical Learning Procedure

Inductive Learning

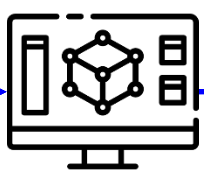
Labeled training data $\{\boldsymbol{x}_i, y_i\}, i = 1, \ldots, N$

ID	Gender	Profession	Income	Saving	Repay
1	F	Engineer	60k	200k	Yes
2	M	Student	10k	20k	Yes
…	…	…	…	…	…
10	M	Student	8k	5k	No

Training phase

Test data $\boldsymbol{x}^*$

ID	Gender	Profession	Income	Saving
11	F	Lawyer	70k	100k



Test phase

Repay
Yes or No

Model

$f: \boldsymbol{x} \rightarrow y$

Lazy Learning Procedure

Lazy Learning

Labeled training data $\{\boldsymbol{x}_i, y_i\}, i = 1, \ldots, N$

ID	Gender	Profession	Income	Saving	Repay
1	F	Engineer	60k	200k	Yes
2	M	Student	10k	20k	Yes
…	…	…	…	…	…
10	M	Student	[illegible]	[illegible]	No

Training phase

- A model is not learned during "training phase"
- Instead, hashing table or indexing can be built

Test phase

- Retrieve similar training instances

Test data $\boldsymbol{x}^*$

ID	Gender	Profession	Income	Saving
11	F	Lawyer	70k	100k

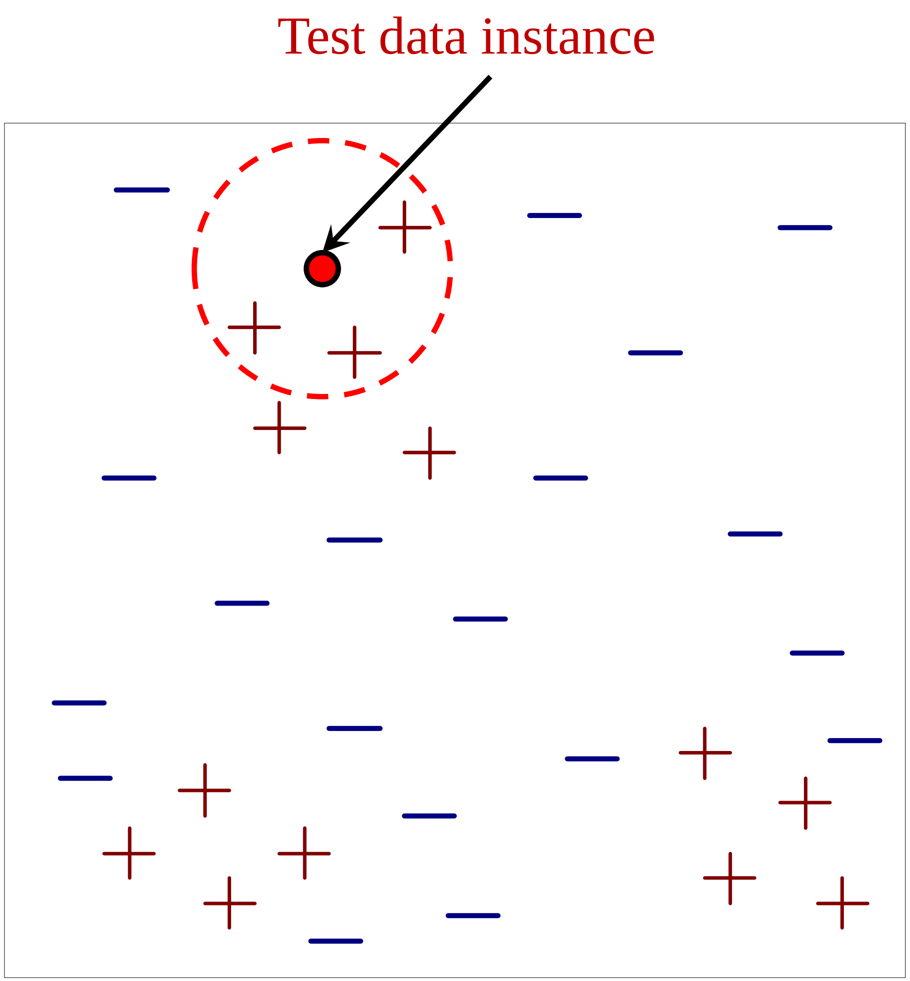
Repay
Yes or No

Based on the labels of the retrieved data instances

K-Nearest Neighbors Classifiers

- Algorithm:
  - For each test instance $\boldsymbol{x}^*$, retrieve K training data instances that are the most similar to $\boldsymbol{x}^*$ (K nearest neighbors) from the training set
  - Based on the class labels of the K nearest neighbors to make a prediction on $\boldsymbol{x}^*$

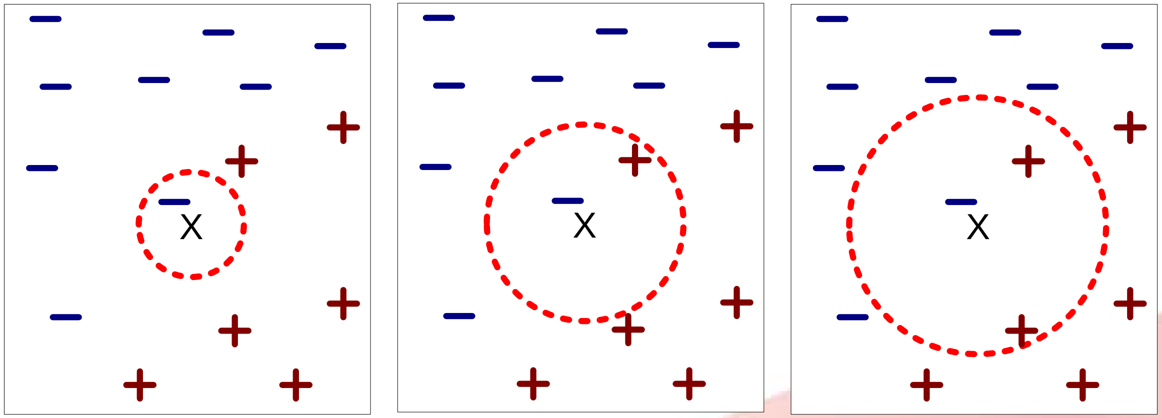
Illustration



- Requirements:
  - A set of stored labeled training instances
  - Distance measure to compute distance between instances
  - The value of *K*, the number of nearest neighbors to retrieve
- To classify a test instance:
  - Compute distance to all the training instances
  - Identify *K* nearest neighbors
  - Use class labels of nearest neighbors to determine the class label of the test instance (e.g., using the majority class)

Distance & Nearest Neighbors

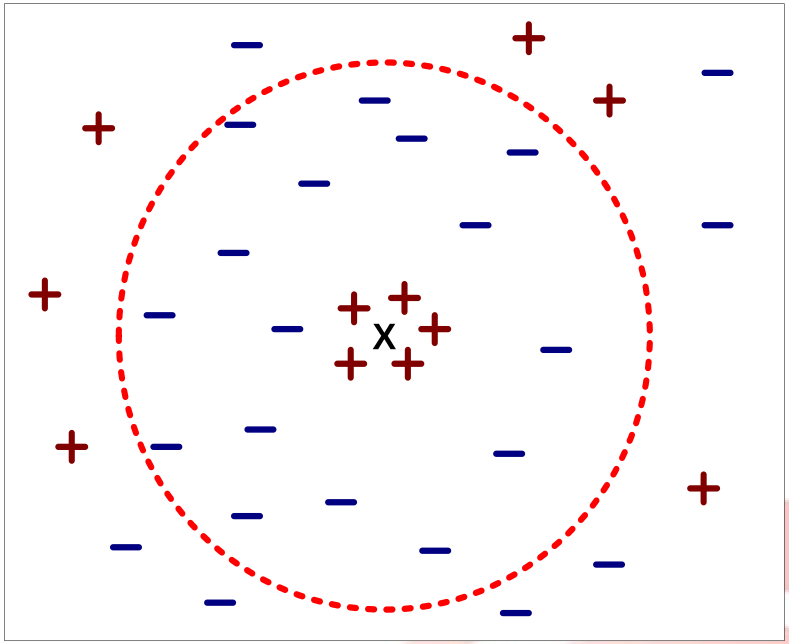
- Euclidean distance $d(\boldsymbol{x}_i, \boldsymbol{x}_j) = \sqrt{\sum_{k=1}^{m}(x_{ik} - x_{jk})^2}$
  - The smaller is the value, the more similar are two data instances
  - K-nearest neighbors of an instance $\boldsymbol{x}$ are data instances that have the K smallest distance to $\boldsymbol{x}$



(a) 1-nearest neighbor (b) 2-nearest neighbor (c) 3-nearest neighbor

Value of K

- K is a hyper-parameter:
  - If K is too small, sensitive to noise points
  - If K is too large, neighborhood may include points from other classes



Determine Class Label

- Determine the class from nearest neighbor list
  - Take the majority vote of class labels among the K-nearest neighbors
- For majority voting:

Indicator function that returns 1 if its input is true, otherwise 0

$$y^* = \arg\max_c \sum_{\boxed{(\boldsymbol{x}_i, y_i) \in \mathcal{N}_{\boldsymbol{x}^*}}} \boxed{I(c = y_i)}$$

Nearest neighbors of the test instance $\boldsymbol{x}^*$

  - Every neighbor has the same impact on the classification
  - This makes the algorithm more sensitive to the choice of K

Weighted Voting

- Alternative scheme: distance-weight voting
  - Weight the influence of each nearest neighbor $\boldsymbol{x}_i$ according to its distance to the test data

$$w_i = \frac{1}{d(\boldsymbol{x}^*, \boldsymbol{x}_i)^2}$$

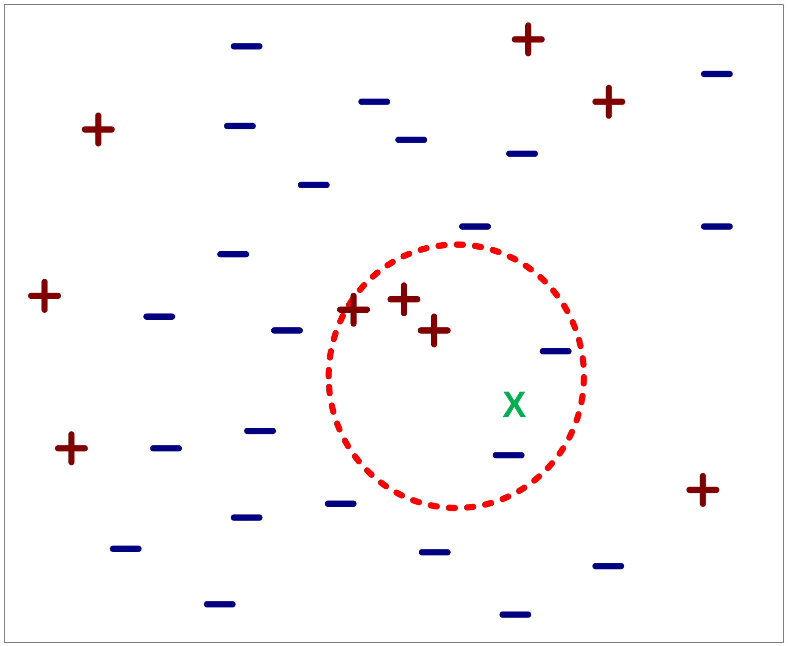
  - That is

$$y^* = \arg\max_c \sum_{(\boldsymbol{x}_i, y_i) \in \mathcal{N}_{\boldsymbol{x}^*}} w_i I(c = y_i)$$

The larger is the distance to the test data, the smaller influence of the corresponding nearest neighbor to the vote

An Example

Consider a binary classification problem, and a 5-NN classifier



Instance ID	Class	Squared distance to test data
1	+	9
2	+	12.25
3	+	16
4	−	2.25
5	−	4

- Majority voting:

$$+ : 3 > - : 2$$

- Distance-weight voting:

$$\text{Distance}-\text{Weight votes for } +: \quad \frac{1}{9}+\frac{1}{12.25}+\frac{1}{16}=0.2552$$

$$<$$

$$\text{Distance}-\text{Weight votes for } -: \quad \frac{1}{4}+\frac{1}{2.25}=0.6944$$

Potential Issues

- Features may have very different scales, e.g.,
  - height of a person may vary from 1.5m to 1.8m
  - weight of a person may vary from 3kg to 200kg
  - income of a person may vary from $10K to $1M
- Features need to be rescaled to prevent distance from being dominated by some features
- Solution: normalization on features of different scales to the same scale
  - Min-Max normalization
  - Standardization (z-score normalization)

Summary

- The *K*NN classifiers are a lazy learner
  - A classification model is not built explicitly
  - "Training" is very efficient
  - Classifying test instances is relatively expensive

Implementation using scikit-learn

- API: sklearn.neighbors.KNeighborsClassifier
  https://scikit-learn.org/stable/modules/generated/sklearn.neighbors.KNeighborsClassifier.html

sklearn.neighbors.KNeighborsClassifier

class `sklearn.neighbors.KNeighborsClassifier`(*n_neighbors=5, *, weights='uniform', algorithm='auto', leaf_size=30, p=2, metric='minkowski', metric_params=None, n_jobs=None, **kwargs*) [source]

Classifier implementing the k-nearest neighbors vote.

Read more in the User Guide.

Parameters:

n_neighbors : *int, default=5*
Number of neighbors to use by default for `kneighbors` queries.

weights : *{'uniform', 'distance'} or callable, default='uniform'*
weight function used in prediction. Possible values:

- 'uniform' : uniform weights. All points in each neighborhood are weighted equally.
- 'distance' : weight points by the inverse of their distance. in this case, closer neighbors of a query point will have a greater influence than neighbors which are further away.
- [callable] : a user-defined function which accepts an array of distances, and returns an array of the same shape containing the weights.

algorithm : *{'auto', 'ball_tree', 'kd_tree', 'brute'}, default='auto'*
Algorithm used to compute the nearest neighbors:

- 'ball_tree' will use `BallTree`
- 'kd_tree' will use `KDTree`
- 'brute' will use a brute-force search.
- 'auto' will attempt to decide the most appropriate algorithm based on the values passed to `fit` method.

Example

```
>>> from sklearn.neighbors import KNeighborsClassifier
>>> import numpy as np

>>> n_samples, n_features = 10, 5
>>> rng = np.random.RandomState(0)
>>> y = rng.integers(2, n_samples)
>>> X = rng.randn(n_samples, n_features)

>>> knnC = KNeighborsClassifier(n_neighbors=3 )
>>> knnC.fit(X, y)
>>> pred= knnC.predict(X)
```

set number of neighbors

Build indices s.t. it is more efficient
when making predictions on test data

Thank you!