Bayesian Classification

Bayesian Classification

- A statistical classifier: predicts class membership probabilities
- Foundation: Based on Bayes' Theorem
- Performance: A simple Bayesian classifier, naïve Bayesian classifier, has comparable performance with decision tree and selected neural network classifiers

Bayes' Theorem: Basics

- Let **X** be a data sample ("*evidence*"), its class label is unknown
- Let H be a *hypothesis* that X belongs to class C
- Classification is to determine P(C|**X**), the probability that the hypothesis holds given the observed data sample **X**
- P(H|**X**) or P(C|**X**) is a posterior probability
- P(C) is a *prior probability*
  - the initial probability of C
  - e.g., **X** will buy computer, regardless of age, income, …
- P(**X**): probability that sample data is observed
- P(**X**|C) (likelihood), the probability of observing the sample **X**, given that the hypothesis H holds

Bayes' Theorem

- Given training data X, posteriori probability of a hypothesis H, P(C|X), follows the Bayes theorem

$$P(C|X) = \frac{P(X|C)P(C)}{P(X)}$$

- Predicts X belongs to C_i iff the probability $P(C_i|X)$ is the highest among all the $P(C_k|X)$ for all the k classes

Towards Naïve Bayesian Classifier

- Let D be a training dataset
- Each tuple is represented as vector **X** = $(x_1, x_2, \ldots, x_n)$
- Suppose there are *m* classes $C_1, C_2, \ldots, C_m$.
- Classification is to find the maximum $P(C_i|\mathbf{X})$
- This can be derived from Bayes' theorem (1<= i <= m)

$$P(C_i|\mathbf{X}) = \frac{P(\mathbf{X}|C_i)P(C_i)}{P(\mathbf{X})}$$

- Since P(X) is constant for all classes, only

$$P(C_i|\mathbf{X}) \approx P(\mathbf{X}|C_i)P(C_i)$$

needs to be maximized

Derivation of Naïve Bayes' Classifier

- A simplified assumption: attributes are conditionally independent

$$P(\mathbf{X}|C_i) = \prod_{k=1}^{n} P(x_k|C_i) = P(x_1|C_i) \times P(x_2|C_i) \times \ldots \times P(x_n|C_i)$$

- This greatly reduces the computation cost
- If A_k is categorical, $P(x_k|C_i)$ is the # of tuples in C_i having value x_k for A_k divided by $|C_{i,D}|$ (# of tuples of C_i in D)
- If A_k is continuous-valued, $P(x_k|C_i)$ is usually computed based on Gaussian distribution with a mean μ and standard deviation σ

$$g(x, \mu, \sigma) = \frac{1}{\sqrt{2\pi}\sigma} e^{-\frac{(x-\mu)^2}{2\sigma^2}}$$

and $P(x_k|C_i)$ is

$$P(\mathbf{X}_k|C_i) = g(x_k, \mu_{C_i}, \sigma_{C_i})$$

How to Estimate Probabilities from Data?

Tid	Refund	Marital Status	Taxable Income	Evade
1	Yes	Single	125K	No
2	No	Married	100K	No
3	No	Single	70K	No
4	Yes	Married	120K	No
5	No	Divorced	95K	Yes
6	No	Married	60K	No
7	Yes	Divorced	220K	No
8	No	Single	85K	Yes
9	No	Married	75K	No
10	No	Single	90K	Yes

- Normal distribution:

$$P(A_i|c_j) = \frac{1}{\sqrt{2\pi\sigma_j^2}} e^{-\frac{(A_i-\mu_j)^2}{2\sigma_j^2}}$$

- For (Income, Class=No):
  - If Class=No
    - sample mean = 110K
    - sample variance = 2975K

$$P(\mathbf{X}_k \mid C_i) = g(x_k, m_{C_i}, s_{C_i})$$

$$P(Income = 120|No) = \frac{1}{\sqrt{2\pi}(54.54)} e^{-\frac{(120-110)^2}{2(2975)}} = 0.0072$$

Naïve Bayes' – Example

Class:
C1:buys_computer = 'yes'
C2:buys_computer = 'no'

Data sample
X = (age <=30,
income = medium,
student = yes
credit_rating = Fair)

age	income	student	credit_rating	buys_computer
<=30	high	no	fair	no
<=30	high	no	excellent	no
31…40	high	no	fair	yes
>40	medium	no	fair	yes
>40	low	yes	fair	yes
>40	low	yes	excellent	no
31…40	low	yes	excellent	yes
<=30	medium	no	fair	no
<=30	low	yes	fair	yes
>40	medium	yes	fair	yes
<=30	medium	yes	excellent	yes
31…40	medium	no	excellent	yes
31…40	high	yes	fair	yes
>40	medium	no	excellent	no

$$P(C_i|\mathbf{X}) \approx P(\mathbf{X}|C_i)P(C_i)$$

$$P(\mathbf{X}|C_i) = \prod_{k=1}^{n} P(x_k|C_i) = P(x_1|C_i) \times P(x_2|C_i) \times \ldots \times P(x_n|C_i)$$

AVC Sets for the Training Dataset

Training Examples

age	income	student	credit_rating	buys_computer
<=30	high	no	fair	no
<=30	high	no	excellent	no
31…40	high	no	fair	yes
>40	medium	no	fair	yes
>40	low	yes	fair	yes
>40	low	yes	excellent	no
31…40	low	yes	excellent	yes
<=30	medium	no	fair	no
<=30	low	yes	fair	yes
>40	medium	yes	fair	yes
<=30	medium	yes	excellent	yes
31…40	medium	no	excellent	yes
31…40	high	yes	fair	yes
>40	medium	no	excellent	no

AVC-set on *Age*

Age	Buy_Computer	
	yes	no
<=30	2	3
31..40	4	0
>40	3	2

AVC-set on *income*

income	Buy_Computer	
	yes	no
high	2	2
medium	4	2
low	3	1

AVC-set on *Student*

student	Buy_Computer	
	yes	no
yes	6	1
no	3	4

AVC-set on *credit_rating*

Credit rating	Buy_Computer	
	yes	no
fair	6	2
excellent	3	3

Naïve Bayes' – Example

$$P(C_i|\mathbf{X}) \approx P(\mathbf{X}|C_i)P(C_i)$$

- $P(C_i)$: P(buys_computer = "yes") = 9/14 = 0.643
  P(buys_computer = "no") = 5/14= 0.357

- Compute $P(X|C_i)$ for each class
  P(age = "<=30" | buys_computer = "yes") = 2/9 = 0.222
  P(age = "<= 30" | buys_computer = "no") = 3/5 = 0.6
  P(income = "medium" | buys_computer = "yes") = 4/9 = 0.444
  P(income = "medium" | buys_computer = "no") = 2/5 = 0.4

Age	Buy_Computer	
	yes	no
<=30	2	3
31..40	4	0
>40	3	2

- **X = (age <= 30 , income = medium, student = yes, credit_rating = fair)**

income	Buy_Computer	
	yes	no
high	2	2
medium	4	2
low	3	1

$$P(\mathbf{X}|C_i) = \prod_{k=1}^{n} P(x_k|C_i) = P(x_1|C_i) \times P(x_2|C_i) \times \ldots \times P(x_n|C_i)$$

Naïve Bayes' – Example

$$P(C_i|\mathbf{X}) \approx P(\mathbf{X}|C_i)P(C_i)$$

student	Buy_Computer	
	yes	no
yes	6	1
no	3	4

Credit rating	Buy_Computer	
	yes	no
fair	6	2
excellent	3	3

- $P(C_i)$: P(buys_computer = "yes") = 9/14 = 0.643
  P(buys_computer = "no") = 5/14= 0.357

- Compute $P(X|C_i)$ for each class
  P(age = "<=30" | buys_computer = "yes") = 2/9 = 0.222
  P(age = "<= 30" | buys_computer = "no") = 3/5 = 0.6
  P(income = "medium" | buys_computer = "yes") = 4/9 = 0.444
  P(income = "medium" | buys_computer = "no") = 2/5 = 0.4
  P(student = "yes" | buys_computer = "yes) = 6/9 = 0.667
  P(student = "yes" | buys_computer = "no") = 1/5 = 0.2
  P(credit_rating = "fair" | buys_computer = "yes") = 6/9 = 0.667
  P(credit_rating = "fair" | buys_computer = "no") = 2/5 = 0.4

- **X = (age <= 30 , income = medium, student = yes, credit_rating = fair)**

$$P(\mathbf{X}|C_i) = \prod_{k=1}^{n} P(x_k|C_i) = P(x_1|C_i)\times P(x_2|C_i)\times \ldots \times P(x_n|C_i)$$

Naïve Bayes' – Example

$$P(C_i|\mathbf{X}) \approx P(\mathbf{X}|C_i)P(C_i)$$

- $P(C_i)$: P(buys_computer = "yes") = 9/14 = 0.643
  P(buys_computer = "no") = 5/14= 0.357

- Compute $P(X|C_i)$ for each class
  P(age = "<=30" | buys_computer = "yes") = 2/9 = 0.222
  P(age = "<= 30" | buys_computer = "no") = 3/5 = 0.6
  P(income = "medium" | buys_computer = "yes") = 4/9 = 0.444
  P(income = "medium" | buys_computer = "no") = 2/5 = 0.4
  P(student = "yes" | buys_computer = "yes) = 6/9 = 0.667
  P(student = "yes" | buys_computer = "no") = 1/5 = 0.2
  P(credit_rating = "fair" | buys_computer = "yes") = 6/9 = 0.667
  P(credit_rating = "fair" | buys_computer = "no") = 2/5 = 0.4

- **X = (age <= 30 , income = medium, student = yes, credit_rating = fair)**

$P(X|C_i)$: P(X|buys_computer = "yes") = 0.222 x 0.444 x 0.667 x 0.667 = 0.044
P(X|buys_computer = "no") = 0.6 x 0.4 x 0.2 x 0.4 = 0.019

$P(X|C_i)*P(C_i)$: P(X|buys_computer = "yes") * P(buys_computer = "yes") = 0.028
P(X|buys_computer = "no") * P(buys_computer = "no") = 0.007

Therefore, X belongs to class ("buys_computer = yes")

$$P(\mathbf{X}|C_i) = \prod_{k=1}^{n} P(x_k|C_i) = P(x_1|C_i) \times P(x_2|C_i) \times \ldots \times P(x_n|C_i)$$

Exercise

- Consider the training dataset given in the table below, where A1, A2 and A3 are the input attributes and C is the class label. Use Naïve Bayes' to find the class for the object X = {1, 2, 2}.

Sample	Attribute A1	Attribute A2	Attribute A3	**Class C**
1	1	2	1	**1**
2	0	0	1	**1**
3	0	1	2	**1**
4	1	0	1	**1**
5	2	1	2	**2**
6	1	2	1	**2**
7	2	2	2	**2**

Avoiding the 0-Probability Problem

- Naïve Bayesian requires each conditional prob. be non-zero. Otherwise, the predicted prob. will be zero

$$P(X \mid C_i) = \prod_{k=1}^{n} P(x_k \mid C_i)$$

- Ex. Suppose that a dataset with 1000 tuples. We have income=low (0), income= medium (990), and income = high (10),
- Use Laplacian correction (or Laplacian estimator)
  - Adding 1 to each case

    Prob(income = low) = 1/1003

    Prob(income = medium) = 991/1003

    Prob(income = high) = 11/1003
  - The "corrected" prob. estimates are close to their "uncorrected" counterparts

Naïve Bayes' – Advantages and Disadvantages

- Advantages:
  - Easy to implement
  - Very efficient
  - Good results obtained in many applications
  - Can easily handle missing data by omitting that probability in the calculations
- Disadvantages
  - Assumption: class conditional independence, therefore loss of accuracy when the assumption is seriously violated

Naïve Bayes in Scikit-Learn

- There are 3 classes that implement Naïve Bayes in the submodule sklearn.naive_bayes
  - GaussianNB
  - MultinomialNB
  - BernoulliNB
- Each assumes a different likelihood distribution of the attributes' values
- Which one to use depends on features' types: continuous, categorical, binary

Preparing the Dataset

```
# Load data
iris = datasets.load_iris()
X = iris.data
y = iris.target

# Split the dataset:
from sklearn.model_selection import
                                train_test_split



X_train, X_test, y_train, y_test =
         train_test_split(X, y, test_size=0.25,
                          random_state=1)
```

Create a GaussianNB Model and Train It

- **`from sklearn.naive_bayes import GaussianNB`**

```
# Create Gaussian Naive Bayes object
classifer = GaussianNB()

# Train model
model = classifer.fit(X_train, y_train)
```

Test the Model

```
# Test the model

y_pred = model.predict(X_test)

from sklearn.metrics import accuracy_score

accuracy_score(y_test, y_pred)
#0.97

new_sample = [[ 5, 4, 3, 2]] # Create new sample

model.predict(new_sample) # Classify it
#array([1])
```

Using NB to Work with Categorical Data

- MultinomialNB is commonly used when working with categorical attributes
  - Ex: movie ratings ranging from 1 to 5
  - Ex: working with text data (e.g., documents of text)
    - Common approach: create a bag of words with a vector for each document containing counts of the appearance of the words in the vocabulary

Prepare the Dataset

```
# Load libraries
import numpy as np
from sklearn.naive_bayes import MultinomialNB
from sklearn.feature_extraction.text import CountVectorizer

# Create text
text_data = np.array(['I love Brazil. Brazil!', 'Brazil is best’,
                      'Germany beats both’])

# Create bag of words
count = CountVectorizer()
bag_of_words = count.fit_transform(text_data)
```

CountVectorizor and BOW

```
# Show feature matrix
bag_of_words
Output: <3x7 sparse matrix of type '<class 'numpy.int64'>'
  with 8 stored elements in Compressed Sparse Row format>

bag_of_words.toarray()
      array([[0, 0, 0, 2, 0, 0, 1,],
             [0, 1, 0, 1, 0, 1, 0],
             [1, 0, 1, 0, 1, 0, 0]], dtype=int64)

# Show feature names
count.get_feature_names()
    ['beats', 'best', 'both', 'brazil', 'germany', 'is', 'love']
```

CountVectorizor and BOW (cont.)

- The feature matrix with the words as column names and each row is one observation

beats	best	both	brazil	germany	is	love
0	0	0	2	0	0	1
0	1	0	1	0	1	0
1	0	1	0	1	0	0

- The tuples: (['I love Brazil. Brazil! ', 'Brazil is best', 'Germany beats both’])
- count.get_feature_names()

  ['beats', 'best', 'both', 'brazil', 'germany', 'is', 'love']

Preparing the Dataset and Creating the Model

```
# Create text
text_data = np.array(['I love Brazil. Brazil!', 'Brazil is best', 'Germany beats both'])
# Create bag of words
count = CountVectorizer()
bag_of_words = count.fit_transform(text_data)

# Create feature matrix
features = bag_of_words.toarray()

# Create target vector
target = np.array([0,0,1])

# Create multinomial naive Bayes object with prior probabilities of each class
classifier = MultinomialNB(class_prior=[0.25, 0.5])

# Train model
model = classifier.fit(features, target)
```

Using the Model

```
# Create new observation
new_observation = [[0, 0, 0, 1, 0, 1, 0]]

# Predict new observation's class
model.predict(new_observation)
# array([0])
```

BernoulliNB

- Works similar to MultinomialNB but use with binary data
- See example in the notebook

Distance-Based Classification Algorithms

Distance-Based Classification Algorithms

- Tuples in the same class are more similar to each other
- A similarity or distance measure is used to determine how similar tuples are
- The problem is how to define similarity measures

Distance Measures

- $x_i = (x_{i1}, x_{i2}, \ldots, x_{ip})$ and $x_j = (x_{j1}, x_{j2}, \ldots, x_{jp})$ are two p-dimensional data objects
- ***Minkowski distance***:

$$d(x_i, x_j) = \sqrt[q]{(|x_{i1} - x_{j1}|^q + |x_{i2} - x_{j2}|^q + \ldots + |x_{ip} - x_{jp}|^q)}$$

  where q is a positive integer
- If $q = 2$, d is ***Euclidean distance***

$$d(x_i, xj) = \sqrt{\sum_{k=1}^{p} (x_{ik} - x_{jk})^2}$$

- If $q = 1$, d is ***Manhattan distance***

$$d(x_i, x_j) = |x_{i1} - x_{j1}| + |x_{i2} - x_{j2}| + \ldots + |x_{ip} - x_{jp}|$$

Distance-Based Classification Algorithms

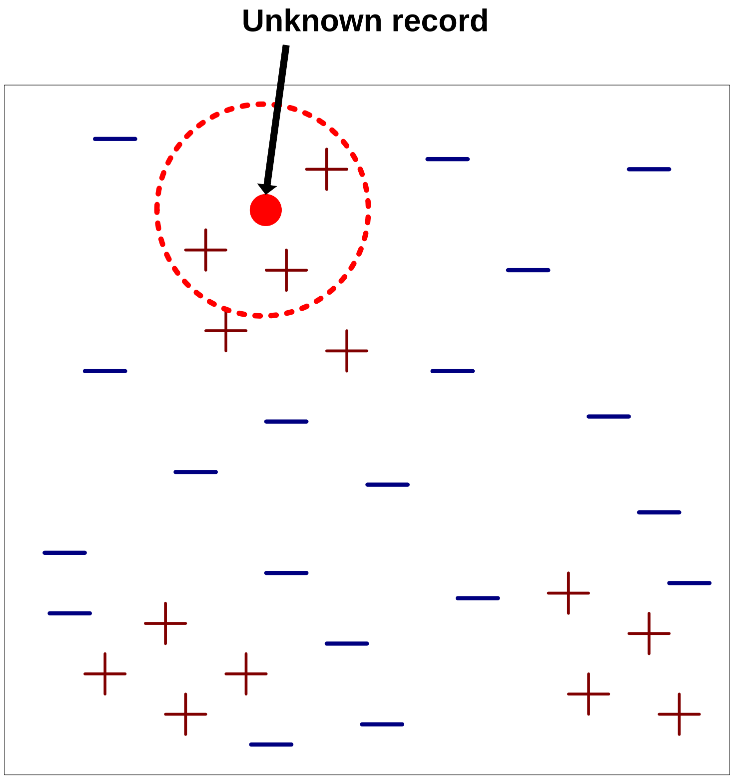
- Place items in class to which they are "closest"
- Must determine distance between an item and a class
- Classes represented by
  - ***Centroid:*** central value
  - ***Medoid:*** actual point near the centroid
  - ***Individual points***
- Example: C = {1, 4, 9}
  - Centroid is _________
  - Medoid is __________

Simple Distance Based Classification Algorithm

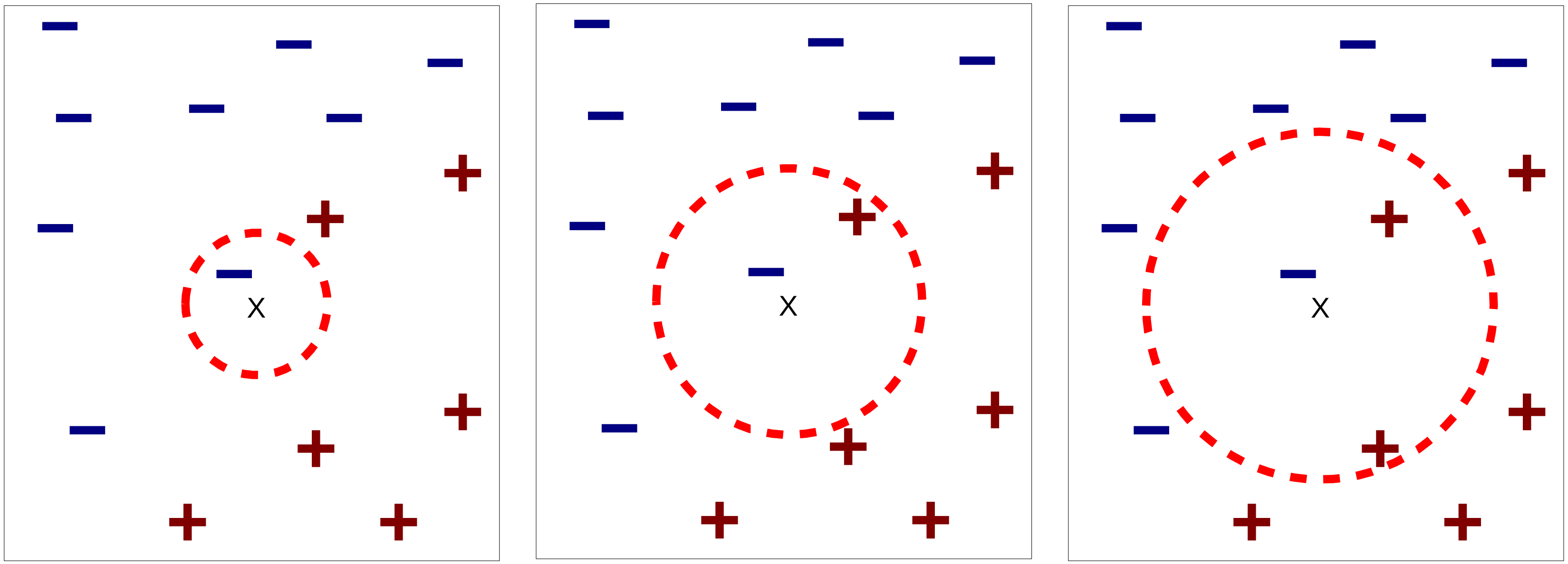
- Decide on the representative for each class
- Decide on the similarity “or distance” measure to use
- To classify a new sample X, it will be compared to the representative of each class
  - Place X in the same class with the representative that is most similar “or closest” to it

K Nearest Neighbors Algorithm

- Requires three things
  - Set of stored objects
  - Distance measure to compute distance between objects
  - The value of k, the number of nearest neighbors to retrieve
- To classify an unknown object:
  - Compute distance to the training objects
  - Identify k nearest neighbors
  - Use class labels of nearest neighbors to determine the class label of unknown object (e.g., by taking majority vote)



Definition of Nearest Neighbors



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

K-nearest neighbors of an object x are data points that have the k smallest distance to x

K Nearest Neighbors Algorithm

- Compute distance between two points:
  - Euclidean distance

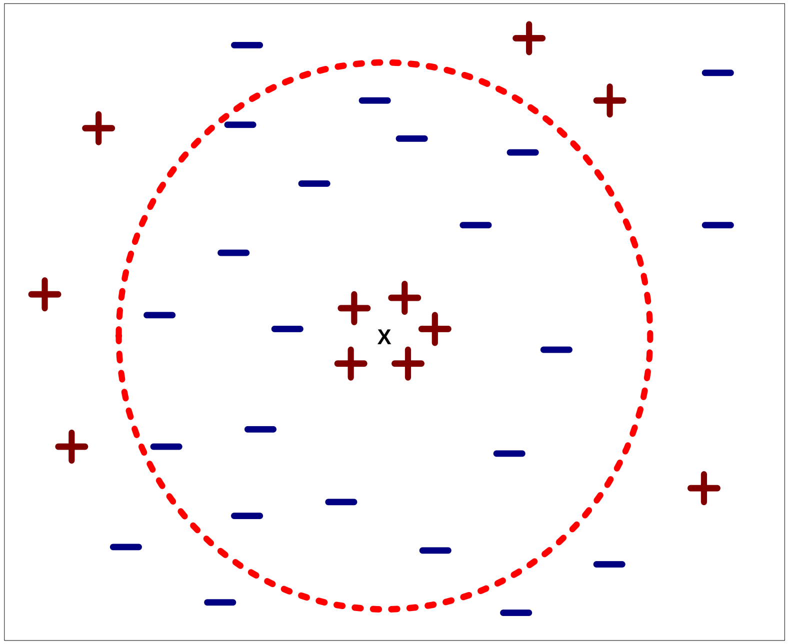
$$d(p,q) = \sqrt{\sum_i (p_i - q_i)^2}$$

- Determine the class from nearest neighbors list
  - take the majority vote of class labels among the k-nearest neighbors (simple voting)
  - Weigh the vote according to distance (weighted voting)
    - weight factor, w = $1/d^2$

Choosing the Value of *K*

- If *k* is too small, sensitive to noise points
- If *k* is too large, neighborhood may include points from other classes

- *k* is usually chosen empirically by trying a range of values



Scaling Values

- Attributes may have to be scaled to prevent distance measures from being dominated by one of the attributes
- Example:
  - height of a person may vary from 1.5m to 1.8m
  - weight of a person may vary from 90lb to 300lb
  - income of a person may vary from $10K to $1M

Eager vs Lazy Learners

- KNN is a lazy learner
  - It does not build a model from the training data
  - Unlike eager learners such as decision tree induction and Naïve Bayes
  - Classifying unknown objects is relatively expensive

KNN Algorithm

Input:

D //Training data

k //Number of neighbors

t //Input tuple to classify

Output:

c //Class to which t is assigned

KNN Algorithm: //Algorithm to classify tuple using KNN

$N = \emptyset$;

//Find set of neighbors, N, for t

foreach d in D do

if $|N| < k$ then

$N = N \cup d$;

else if $\exists$ u in N such that distance(t, d) $<$ distance(t, u)

replace d by the neighbor in N with the largest distance to t

c = class to which most u in N are classified

Source: Dunham, Data Mining – Introductory and Advanced Topics, Algorithm 4.2 page 92

Remarks

- If K = 1, nearest neighbor algorithm
- Choice of K is important for KNN
- [Dunham] As a rule of thumb, $k \le \sqrt{\text{number of training samples}}$; commercial algorithms often use a default value of 10.
- Researchers have shown that the classification accuracy of KNN can be as accurate as more elaborated methods
- KNN is slow at the classification time
- KNN does not produce an understandable model

KNN in Scikit-Learn

```
# Load libraries
from sklearn.neighbors import KNeighborsClassifier
from sklearn.preprocessing import StandardScaler
from sklearn import datasets

# Load data
iris = datasets.load_iris()
X = iris.data
y = iris.target
```

KNN in Scikit-Learn (cont)

```
# Create standardizer
standardizer = StandardScaler()

# Standardize features
X_std = standardizer.fit_transform(X)

# Train a KNN classifier with 5 neighbors
knn = KNeighborsClassifier(n_neighbors=5, n_jobs=-1).fit(X_std, y)

# Create two observations
new_observations = [[ 0.75, 0.75, 0.75, 0.75], [ 1, 1, 1, 1]]

# Predict the class of two observations
knn.predict(new_observations)
array([1, 2])
```

KNeighborsClassifier Parameters

- Parameters we can set include:
  - n_neighbors: value of K (default is 5)
  - metric: 'minkowski' (the default)| 'euclidean'| 'manhattan'
  - p: integer, (default = 2) Power parameter for the Minkowski metric
  - weights: 'uniform' (the default)| 'distance'

- Notebook name: KNN.ipynb