K-nearest Neighbor Algorithm

Hongchang Gao

Spring 2024

Non-parametric model

- Parametric methods:
  - Assume some functional form (Gaussian, Bernoulli, Multinomial, logistic, Linear, Quadratic) for P(Y|X)
  - Estimate parameters
  - Pro:
    - need few data points to learn parameters
  - Con:
    - Strong distributional assumptions, not satisfied in practice

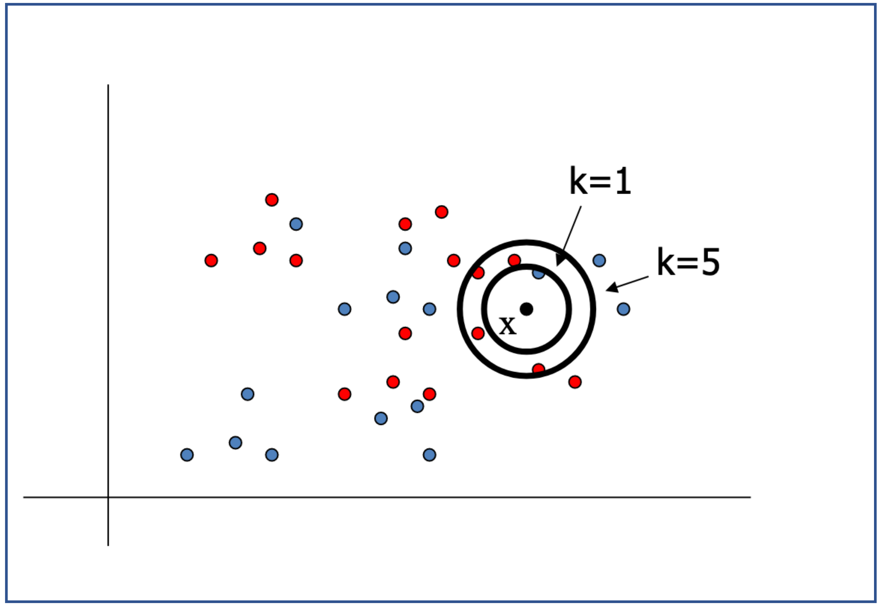
$$p(y = 1|\mathbf{x}) = \frac{\exp(\mathbf{w}_1^T\mathbf{x})}{\exp(\mathbf{w}_0^T\mathbf{x})+\exp(\mathbf{w}_1^T\mathbf{x})}$$

$$p(y = 0|\mathbf{x}) = \frac{\exp(\mathbf{w}_0^T\mathbf{x})}{\exp(\mathbf{w}_0^T\mathbf{x})+\exp(\mathbf{w}_1^T\mathbf{x})}$$

logistic regression: parametric model

Non-parametric model

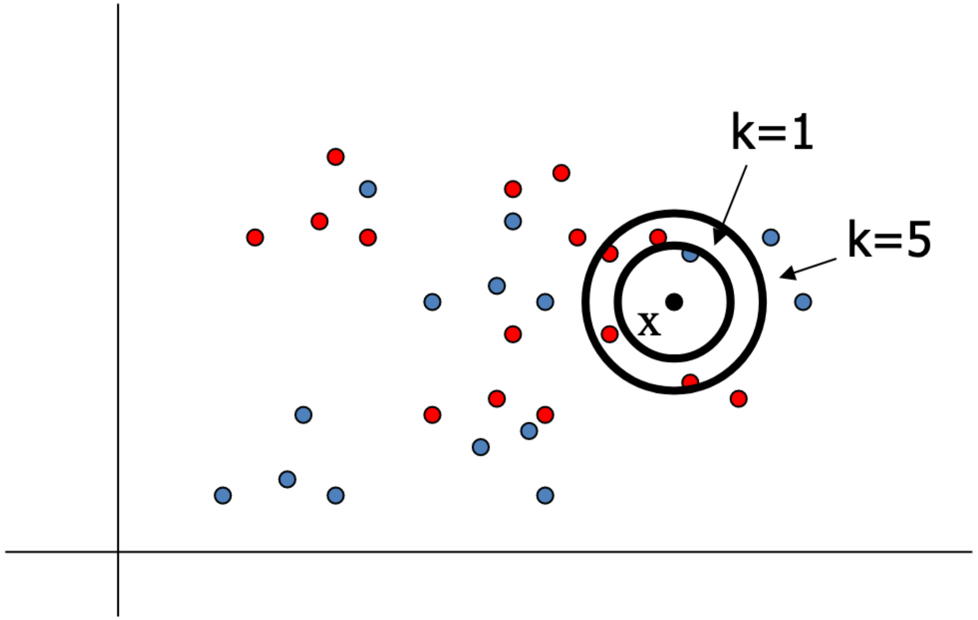
- Typically, don't make any distributional assumptions
- As we have more data, we should be able to learn more complex models



KNN: non-parametric model

K-nearest neighbor method

- Basic Idea
  - Find the K nearest neighbors of sample x
  - Find the majority category label within these neighbors
  - Assign the majority label to sample x



K-nearest neighbor method

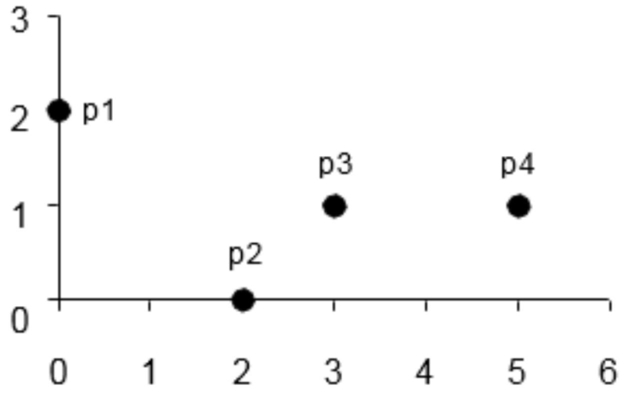
- KNN:
  - Training phase:
    - Store all training samples
  - Testing phase
    - Find the *k* training examples $(x_1,y_1),\ldots(x_k,y_k)$ that are closest to the test example *x*
    - Predict the most frequent class among those y_i's.
- Logistic Regression
  - Training phase:
    - Learn model parameters from training samples
  - Testing phase:
    - Apply the learned model to testing samples

Step 1: find neighbors

- How to measure the distance?
  - Euclidean distance

Euclidean:

$$D(\boldsymbol{x},\boldsymbol{y}) = \sqrt{\sum_{i=1}^{m}(x_i - y_i)^2}$$



point	x	y
p1	0	2
p2	2	0
p3	3	1
p4	5	1

	p1	p2	p3	p4
p1	0	2.828	3.162	5.099
p2	2.828	0	1.414	3.162
p3	3.162	1.414	0	2
p4	5.099	3.162	2	0

Step 1: find neighbors

- How to measure the distance?
  - Manhattan distance

Manhattan / city-block:

$$D(\boldsymbol{x},\boldsymbol{y}) = \sum_{i=1}^{m} |x_i - y_i|$$

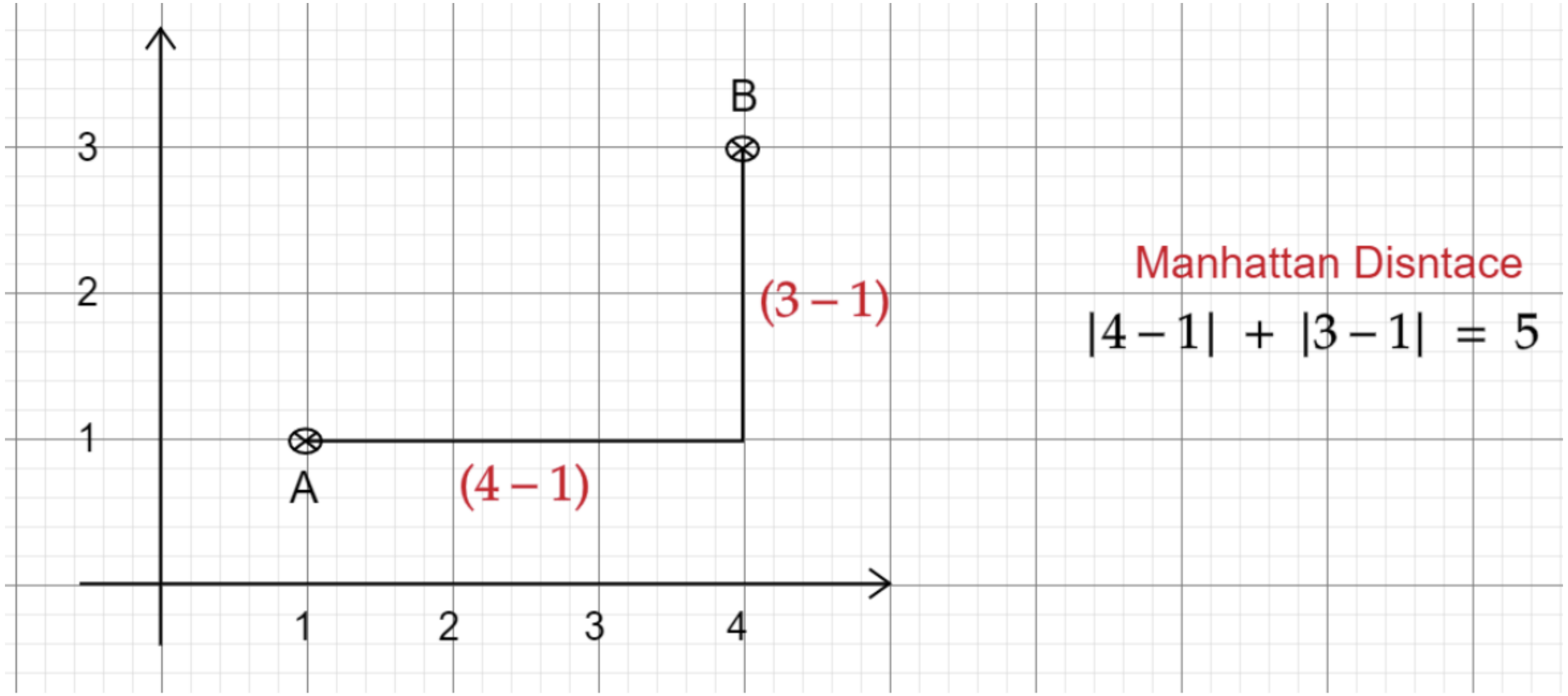


Fig: https://datascienceparichay.com/article/manhattan-distance-python/

Step 1: find neighbors

- How to measure the distance?
    - Correlation

Correlation:

$$D(\boldsymbol{x}, \boldsymbol{y}) = \frac{\sum_{i=1}^{m} (x_i - \overline{x_i})(y_i - \overline{y_i})}{\sqrt{\sum_{i=1}^{m} (x_i - \overline{x_i})^2 \sum_{i=1}^{m} (y_i - \overline{y_i})^2}}$$

Step 2: make classification

- Assigned to the most common class amongst its *K*- nearest neighbors

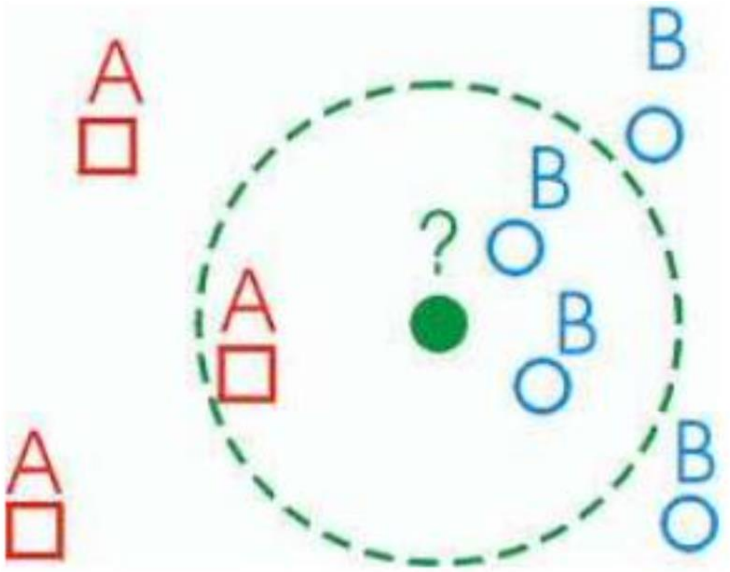


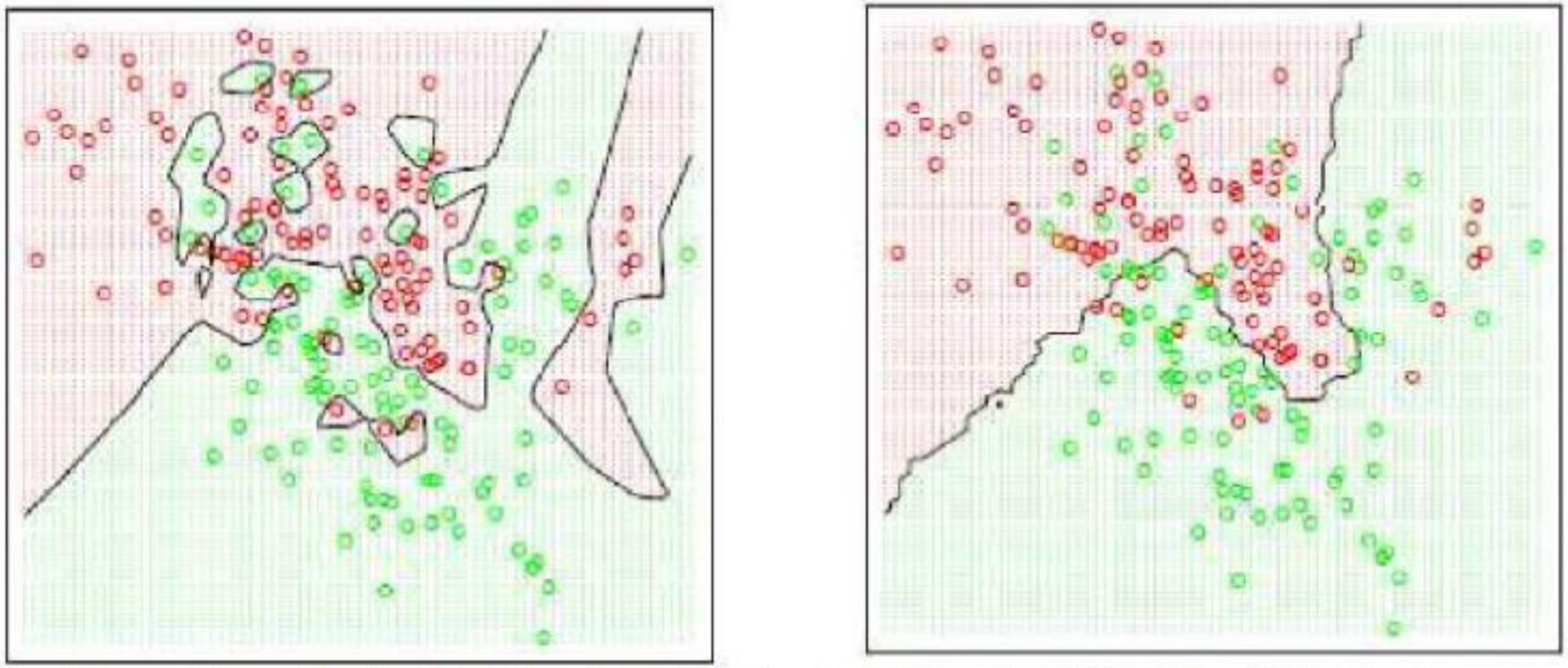
Fig: https://www.slideshare.net/tilanigunawardena/k-nearest-neighbors

Effect of K

- Larger k produces smoother boundary effect

K=1

K=15



Figures from Hastie, Tibshirani and Friedman (Elements of Statistical Learning)

Example

```python
import numpy as np
from sklearn.feature_extraction.text import TfidfVectorizer
import matplotlib.pyplot as plt
from sklearn.datasets import fetch_20newsgroups

data_train = fetch_20newsgroups(subset='train', remove=('headers', 'footers', 'quotes'))
data_test = fetch_20newsgroups(subset='test', remove=('headers', 'footers', 'quotes'))

print("Train data target labels: {}".format(data_train.target))
print("Train data target names: {}".format(data_train.target_names))

print('#training samples: {}'.format(len(data_train.data)))
print('#testing samples: {}'.format(len(data_test.data)))
```

```
Train data target labels: [7 4 4 ... 3 1 8]
Train data target names: ['alt.atheism', 'comp.graphics', 'comp.os.ms-windows.misc', 'comp.sys.ibm.pc.hardware', 'com
p.sys.mac.hardware', 'comp.windows.x', 'misc.forsale', 'rec.autos', 'rec.motorcycles', 'rec.sport.baseball', 'rec.spo
rt.hockey', 'sci.crypt', 'sci.electronics', 'sci.med', 'sci.space', 'soc.religion.christian', 'talk.politics.guns', '
talk.politics.mideast', 'talk.politics.misc', 'talk.religion.misc']
#training samples: 11314
#testing samples: 7532
```

Example

```
#TF-IDF representation for each document
vectorizer = TfidfVectorizer()
data_train_vectors = vectorizer.fit_transform(data_train.data)
data_test_vectors = vectorizer.transform(data_test.data)

print(data_train_vectors.shape, data_test_vectors.shape)
```

```
(11314, 101631) (7532, 101631)
```

```python
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import accuracy_score, f1_score

Xtr = data_train_vectors
Ytr = data_train.target

Xte = data_test_vectors
Yte = data_test.target

# train knn
clf_knn =  KNeighborsClassifier(n_neighbors=1)

clf_knn.fit(Xtr, Ytr)

# evaluate knn
y_pred = clf_knn.predict(Xte)

acc = accuracy_score(Yte, y_pred)
macro_f1 = f1_score(Yte, y_pred, average='macro')
micro_f1 = f1_score(Yte, y_pred, average='micro')

print(acc, macro_f1, micro_f1)
```

```
0.1133828996282579 0.1189175441347045 7 0.1133828996282579
```

Model Selection: Cross-validation

- Training data:
  - Randomly partition it into K folds
  - K-1 folds for training set
  - 1 fold for validation set
- How to select the model?
  - For each hyperparameter
    - Train the model for K times
    - Evaluate the model for K times
  - Use the mean of K evaluation to select model

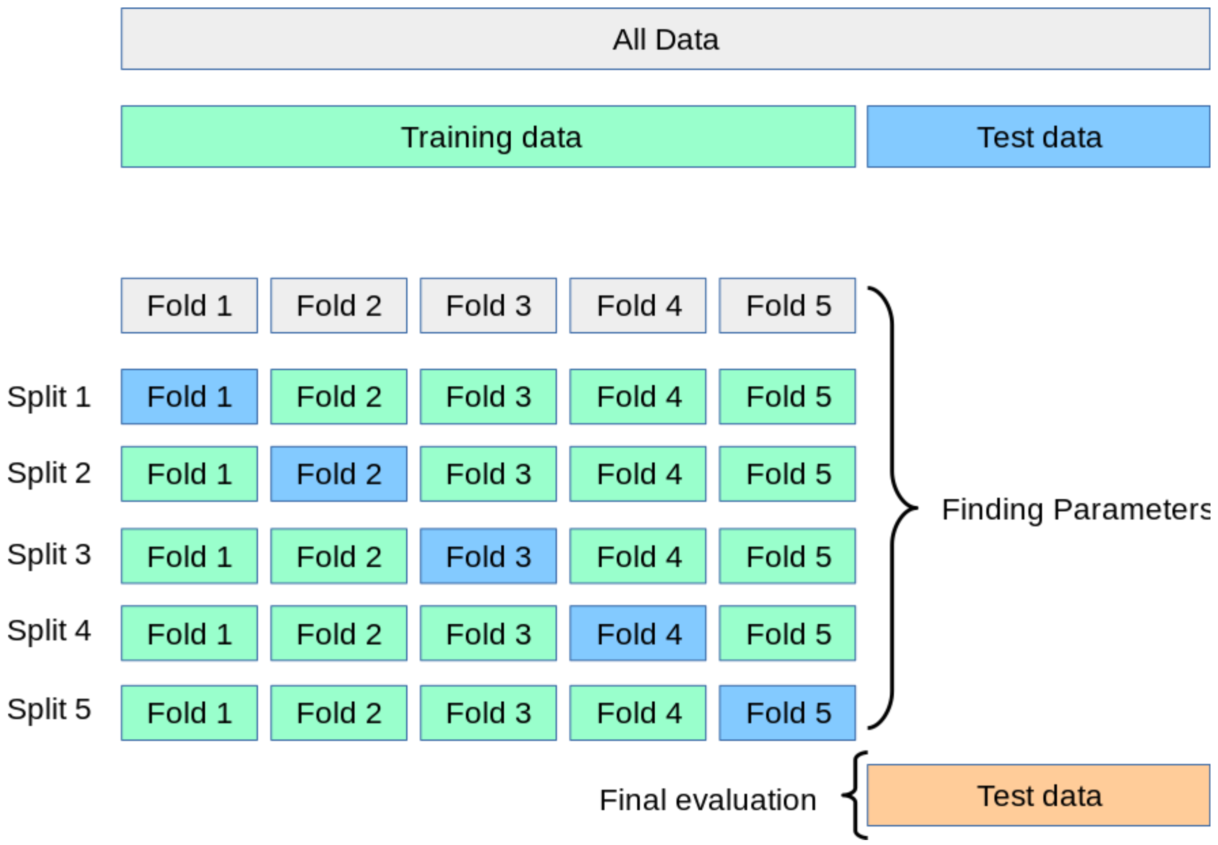


Fig ref: https://scikit-learn.org/stable/modules/cross_validation.html

Example

- Cross Validation to select the best K

```
from sklearn.model_selection import GridSearchCV

k_range = range(1, 5)
param_grid = dict(n_neighbors=k_range)

grid = GridSearchCV(clf_knn, param_grid, cv=5, scoring='accuracy')
grid.fit(Xtr, Ytr)

print(grid.best_score_)
print(grid.best_params_)
```

```
0.16846385009722467
{'n_neighbors': 1}
```

Example

- Comparison with LR

```
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import GridSearchCV
import numpy as np

#=====training with cross validation======
coeff = range(1, 10)
param_grid = dict(C=coeff)

clf_lr = LogisticRegression(penalty='l2')

grid = GridSearchCV(clf_lr, param_grid, cv=5, scoring='accuracy')
grid.fit(Xtr, Ytr)

print(grid.best_params_)
```

```
{'C': 8}
```

Example

- Comparison with LR

```
#=====testing======
clf_lr = LogisticRegression(penalty='l2', C=grid.best_params_['C'])
clf_lr.fit(Xtr, Ytr)

y_pred = clf_lr.predict(Xte)

acc = accuracy_score(Yte, y_pred)
macro_f1 = f1_score(Yte, y_pred, average='macro')
micro_f1 = f1_score(Yte, y_pred, average='micro')

print(acc, macro_f1, micro_f1)
```

```
0.6889272437599575 0.6778761181105242 0.6889272437599575
```

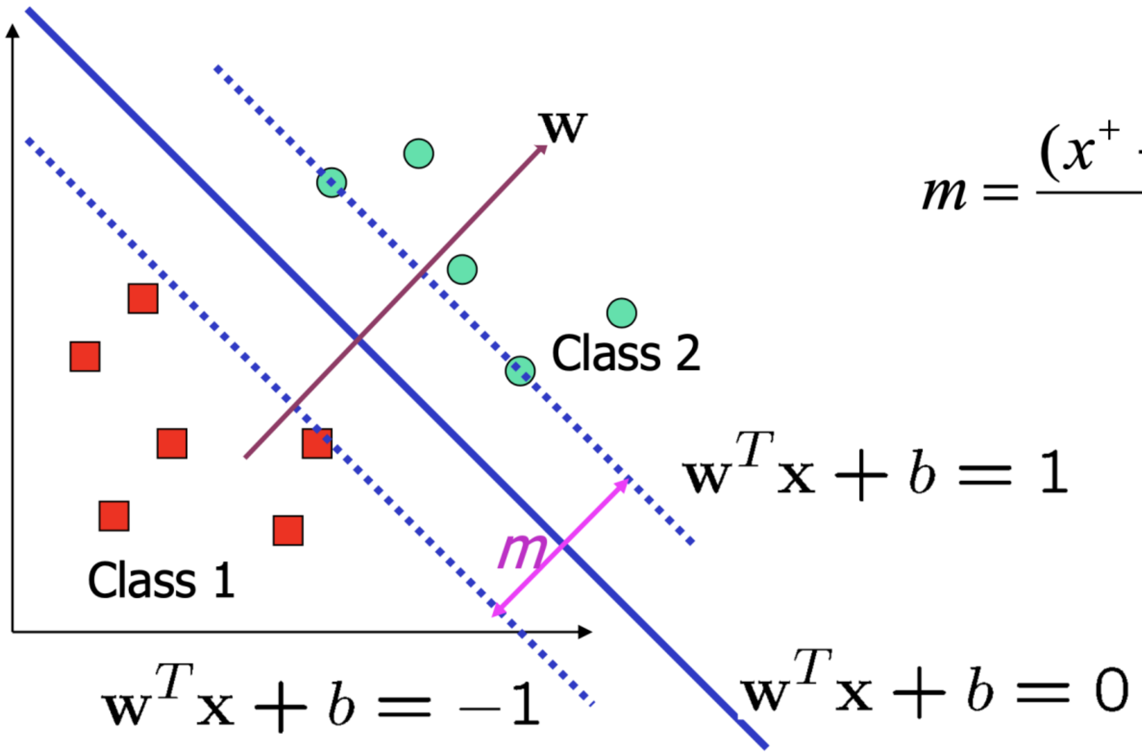
K-nearest neighbor method

- Properties
  - + Training is very fast (lazy training)
  - + Easy to understand and implement

  - - Testing is slow
  - - Curse of dimensionality
  - - Need adequate distance measure

More supervised learning models

- Support vector machine (SVM)
  - Learn a hyperplane such that the distance from it to the nearest data point on each side is maximized



$$m = \frac{(x^+ - x^-) \cdot w}{|w|} = \frac{2}{|w|}$$

Fig: PITT ECE 2195

More supervised learning models

- Let $\{x_1, \ldots, x_n\}$ be our data set and let $y_i \in \{1,-1\}$ be the class label of x_i
- The decision boundary should classify all points correctly
  $\Rightarrow \quad y_i(\mathbf{w}^T\mathbf{x}_i + b) \geq 1, \qquad \forall i$
- The decision boundary can be found by solving the following constrained optimization problem

$$\text{Minimize } \frac{1}{2}||\mathbf{w}||^2$$
$$\text{subject to } y_i(\mathbf{w}^T\mathbf{x}_i + b) \geq 1 \qquad \forall i$$

Fig: PITT ECE 2195

More supervised learning models

- Naïve Bayes Classifier

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

More supervised learning models

- Decision t
    - A tree-l
    - Each in
    - Each br
    - Each le

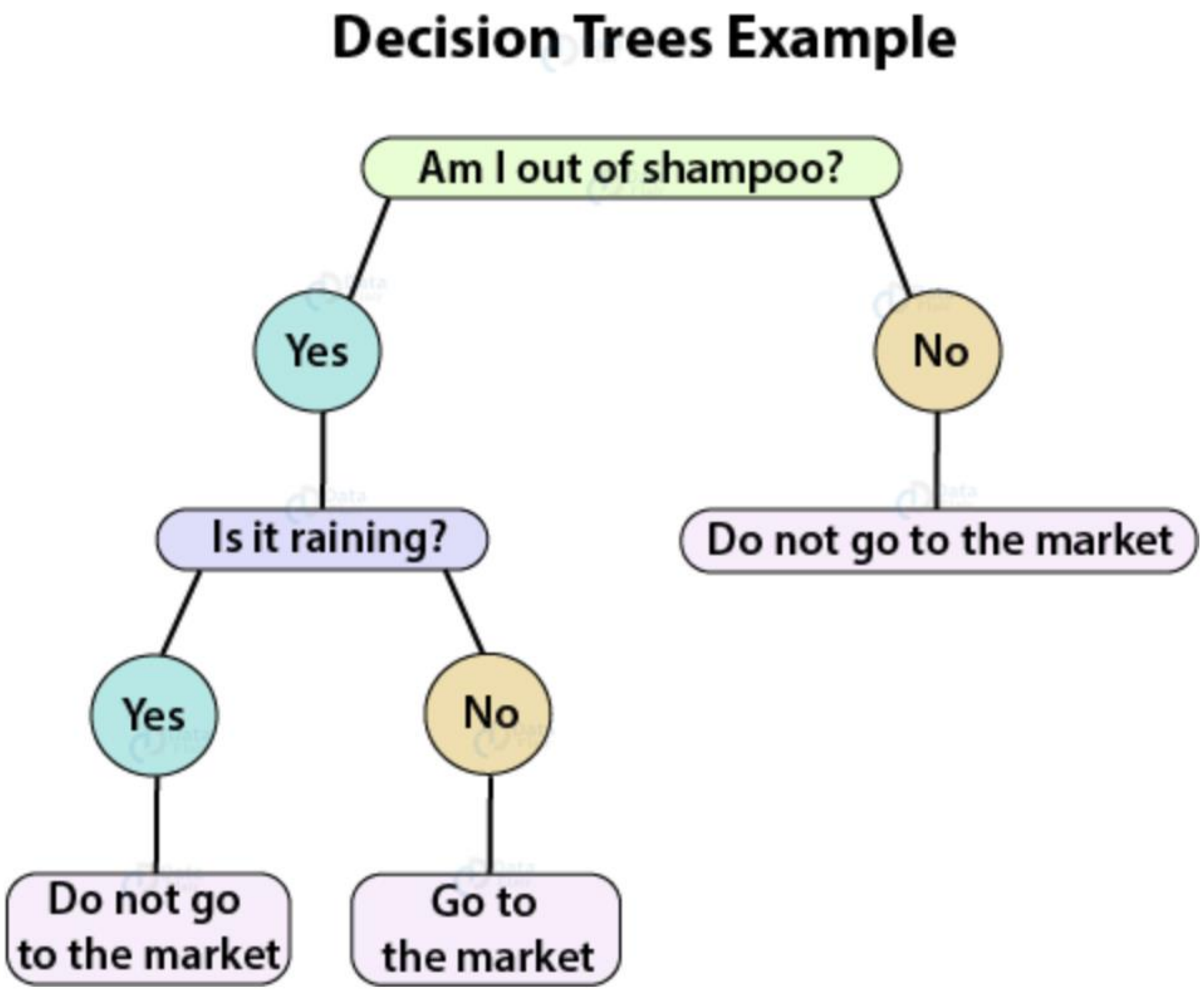
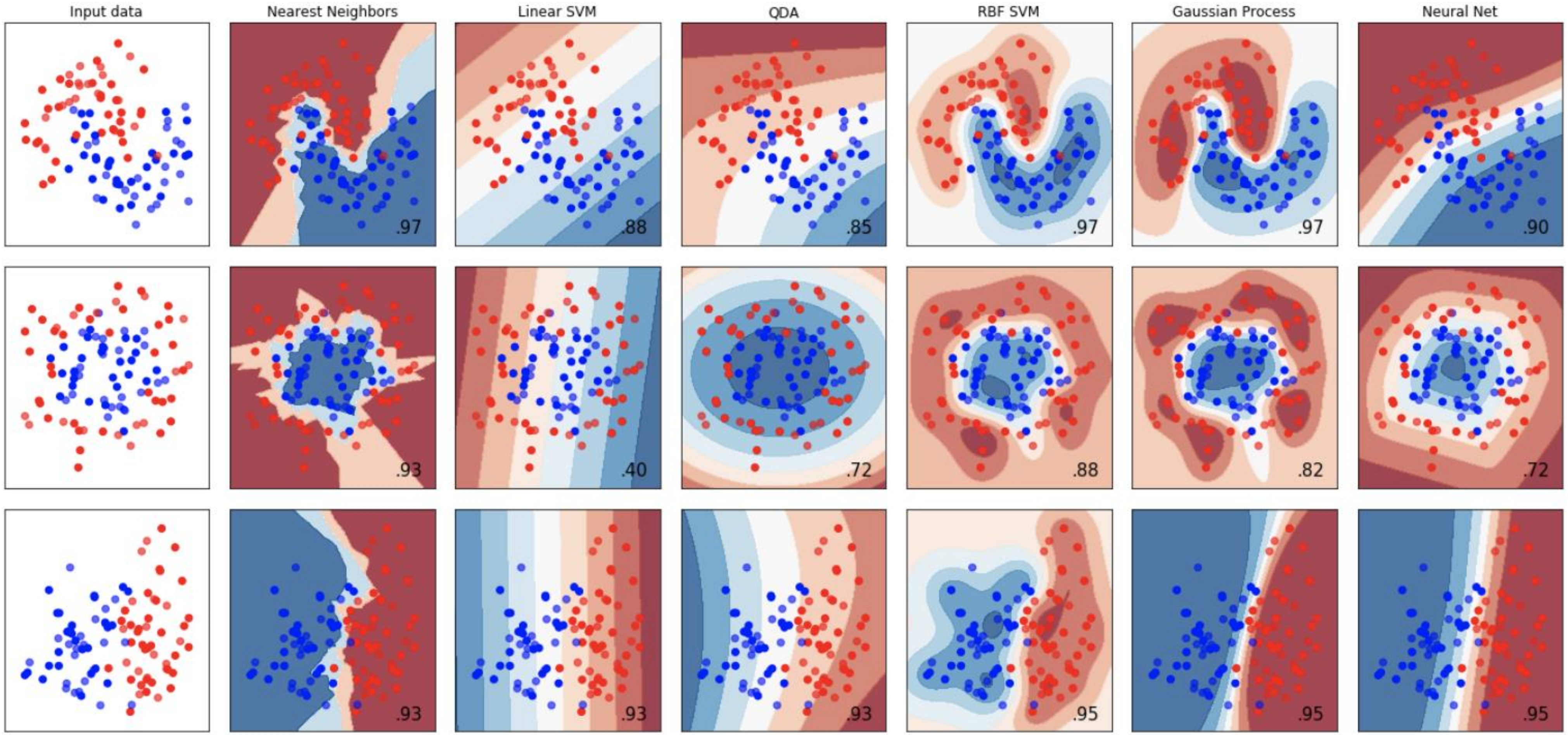


Fig:https://data-flair.training/blogs/r-decision-trees/

More supervised learning models



https://scikit-learn.org/stable/auto_examples/classification/plot_classifier_comparison.html#sphx-glr-auto-examples-classification-plot-classifier-comparison-py

More supervised learning models

- Textbook:
  - Pattern Recognition and Machine Learning