Supervised learning

Aprendizaje Automático para la Robótica
Máster Universitario en Ingeniería Industrial

Departamento de Automática

Objectives

1. Extend supervised learning algorithms
2. Apply supervised learning to real-world problems

Bibliography

- Müller, Andreas C., Guido, Sarah. *Introduction to Machine Learning with Python*. O'Reilly. 2016

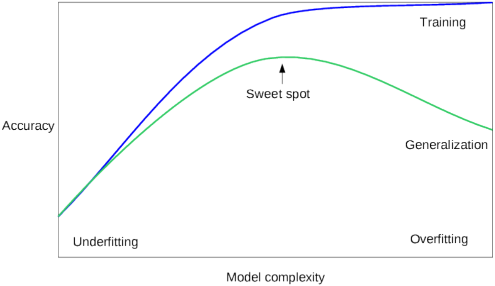
All figures have been taken from
`https://github.com/amueller/introduction_to_ml_with_python/blob/master/02-supervised-learning.ipynb`

Table of Contents

Generalization, overfitting and underfitting

Generalization: accurate predictions on unseen data

- i.e. there is no overfitting neither underfitting
- Depends on model complexity and data variability



(Source)

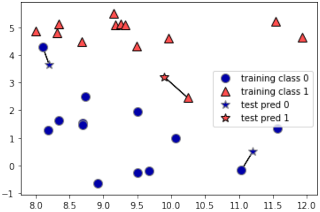
k-Nearest Neighbors

k-NN classification (I)

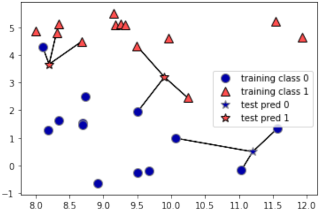
k-NN (k-Nearest Neighbors): Likely, the simplest classifier

- Given a data point, it takes its k closests neighbors
- Same prediction than its neighbors

1-NN



3-NN



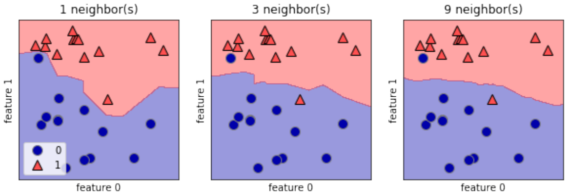
k-NN does not generate a model

- The whole dataset must be stored

k uses to be an odd number (1-NN, 3-NN, 5-NN, ...)

k-Nearest Neighbors

k-NN classification (II)



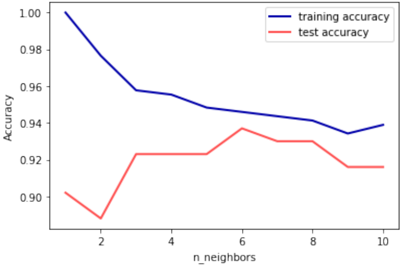
k determines the model complexity

- Smoother boundaries in larger k values
- Model complexity decreases with k
- If k equals the number of samples, k-NN always predicts the most frequent class

How to figure out the best k?

k-Nearest Neighbors

k-NN classification (III)



k-Nearest Neighbors classifier

Scikit-learn

`sklearn.neighbors.KNeighborsClassifier`

Constructor arguments:

- `n_neighbors`: int, default=5
- `metric`: string, default='minkowski'
- `p`: int, default=2 ($p = 1$ Manhatan distance, $p = 2$ euclidean distance)

Attributes:

- `classes_`: ndarray (n_samples)

Methods: `fit()`, `predict()`

(Scikit-Learn reference)

k-Nearest Neighbors

kNN regression (I)

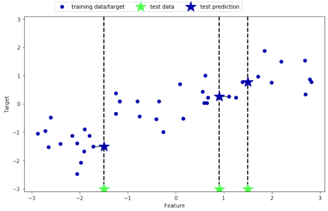
k-NN regression

Given a data point

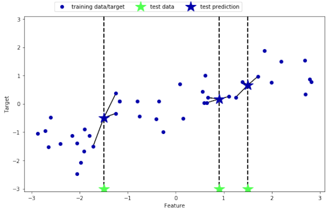
1. Take the *k* closest data points
2. Predict same target value (1-NN) or averate target value (k-NN)

Performace is measured with a regression metric, by default, R^2

1-NN

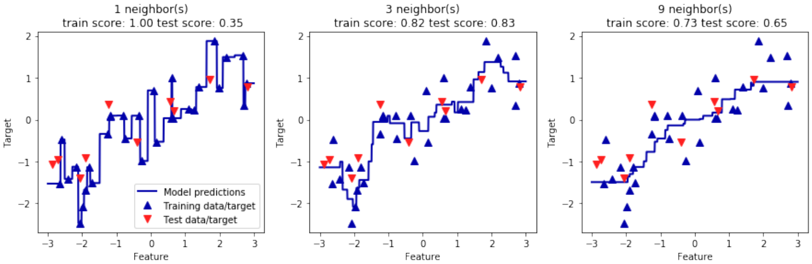


3-NN



k-Nearest Neighbors

kNN regression (II)



k determines boundary smoothness

1. With $k = 1$, prediction visits all data points
2. With large k values, fit is worse

k-Nearest Neighbors regressor

Scikit-learn

`sklearn.neighbors.KNeighborsRegressor`

Constructor arguments:

- `n_neighbors`: int, default=5
- `metric`: string, default='minkowski'
- `p`: int, default=2 ($p = 1$ Manhatan distance, $p = 2$ euclidean distance)

Attributes:

Methods: `fit()`, `predict()`

(Scikit-Learn reference)

k-Nearest Neighbors

Summary

Hyperparameters	Advantages	Disadvantages
k	Simple	Slow with large datasets
Distance	Baseline	Bad performance with hundreds or more attributes
		No model
		Dataset must be stored in memory

Linear models

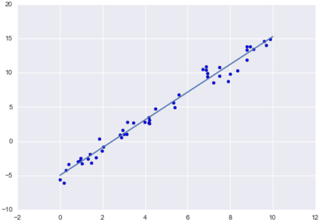
Linear model (I)

Linear model

$$y = a_0 + a_1x_1 + a_2x_2 + \cdots + a_nx_n$$

for a single feature $y = a_0 + a_1x_1$, where

- a_0 is the intercept
- a_1 is the slope



Lineal models assume a linear relationship among variables

- This limitation can be easely overcomed
- Surprisingly good results in high dimensional spaces

Different linear models for regression

- The difference lies in how a_i parameters are learned

Linear models (II)

Several methods to fit coefficients

- Ordinary Least Squares (OLS)
- Generalized Least Squares (GSL)
- Weighted Least Squares (WLS)
- Generalized Least Squares with AR Covariance Structure (GLSAR)

Regularization: Term that penalizes complexity

- L1 (Lasso regression)
- L2 (Ridge regression)
- ElasticNet: L1 and L2

Lasso

$\lambda \sum_j^n \beta_j^2$

Ridge

$\lambda \sum_j^n |\beta_j|$

ElasticNet

$\alpha \sum_j^n \beta_j^2 + (1-\alpha) \sum_j^n |\beta_j|$

Linear models

Scikit-learn

TODO

`sklearn.cluster.AgglomerativeClustering`

Constructor arguments:

- `linkage`: 'ward', 'complete', 'average', 'single'

Attributes:

- `n_clusters`: int
- `labels_`: ndarray (n_samples)

Methods: `fit()`, `fit_predict()`

(Scikit-Learn reference)

Linear models

Summary

Hyperparameters	Advantages	Disadvantages

Naive Bayes Classifiers

TODO

Naive Bayes Classifiers

Scikit-learn

`sklearn.cluster.AgglomerativeClustering`

Constructor arguments:

- `linkage`: 'ward', 'complete', 'average', 'single'

Attributes:

- `n_clusters`: int
- `labels_`: ndarray (n_samples)

Methods: `fit()`, `fit_predict()`

(Scikit-Learn reference)

Naive Bayes Classifiers

Summary

Hyperparameters	Advantages	Disadvantages

Decission Trees

TODO

Decission Trees

Scikit-learn

`sklearn.cluster.AgglomerativeClustering`

Constructor arguments:

- `linkage`: 'ward', 'complete', 'average', 'single'

Attributes:

- `n_clusters`: int
- `labels_`: ndarray (n_samples)

Methods: `fit()`, `fit_predict()`

(Scikit-Learn reference)

Decission Trees

Summary

Hyperparameters	Advantages	Disadvantages

Ensembles of Decision Trees

TODO

Ensembles of Decision Trees

Ensembles of Decision Trees : Scikit-learn

`sklearn.cluster.AgglomerativeClustering`

Constructor arguments:

- `linkage`: 'ward', 'complete', 'average', 'single'

Attributes:

- `n_clusters`: int
- `labels_`: ndarray (n_samples)

Methods: `fit()`, `fit_predict()`

(Scikit-Learn reference)

Ensembles of Decision Trees

Summary

Hyperparameters	Advantages	Disadvantages

Support Vector Machines

TODO

Support Vector Machines

Kernelized Support Vector Machines

TODO

Scikit-Learn

Support Vector Machines

Scikit-learn

`sklearn.cluster.AgglomerativeClustering`

Constructor arguments:

- `linkage`: 'ward', 'complete', 'average', 'single'

Attributes:

- `n_clusters`: int
- `labels_`: ndarray (n_samples)

Methods: `fit()`, `fit_predict()`

(Scikit-Learn reference)

Support Vector Machines

Summary

Hyperparameters	Advantages	Disadvantages

A

B

TODO

A

B: Scikit-learn

sklearn.cluster.AgglomerativeClustering

Constructor arguments:

- `linkage`: 'ward', 'complete', 'average', 'single'

Attributes:

- `n_clusters`: int
- `labels_`: ndarray (n_samples)

Methods: `fit()`, `fit_predict()`

(Scikit-Learn reference)

A

B: Summary

Hyperparameters	Advantages	Disadvantages

Algorithms

ARIMA (I)

AR: Autoregressive model

- Current observation depends on the last p observations
- Long term memory

MA: Moving Average model

- Current observation linearly depends on the last q innovations
- Short term memory

AR(p)

$$X_t = c + \sum_{i=1}^{p} \phi_i X_{t-1} + \epsilon_t$$

MA(q)

$$X_t = \mu + \epsilon_t + \theta_1 \epsilon_{t-1} + \ldots + \theta_q \epsilon_{t-q}$$

ARMA model = AR + MA

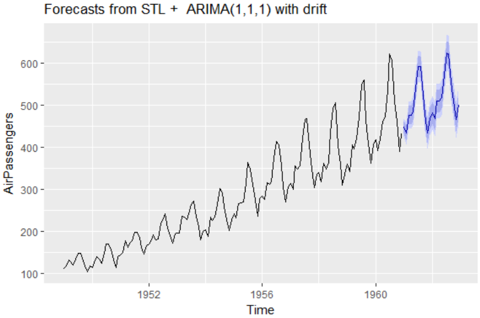
- ARMA(p, q): Two hyperparameters, p and q

Algorithms

ARIMA (II)

ARIMA = AR + i + MA (AR integrated MA)

- ARIMA(p, d, q)
- Three integer parameters: p, q and d (in practice, low order models)



(Source)

autoarima: search over p, q and d