# **Support vector machine**

From Wikipedia, the free encyclopedia

*Not to be confused with*[*Secure Virtual Machine*](https://en.wikipedia.org/wiki/Secure_Virtual_Machine)*.*

In [machine learning](https://en.wikipedia.org/wiki/Machine_learning), **support vector machines** (**SVMs**, also **support vector networks**[[1]](https://en.wikipedia.org/wiki/Support_vector_machine#cite_note-CorinnaCortes-1)) are [supervised learning](https://en.wikipedia.org/wiki/Supervised_learning) models with associated learning [algorithms](https://en.wikipedia.org/wiki/Algorithm) that analyze data used for [classification](https://en.wikipedia.org/wiki/Statistical_classification) and [regression analysis](https://en.wikipedia.org/wiki/Regression_analysis). Given a set of training examples, each marked as belonging to one or the other of two categories, an SVM training algorithm builds a model that assigns new examples to one category or the other, making it a non-[probabilistic](https://en.wikipedia.org/wiki/Probabilistic_classification" \o "Probabilistic classification)[binary](https://en.wikipedia.org/wiki/Binary_classifier) [linear classifier](https://en.wikipedia.org/wiki/Linear_classifier). An SVM model is a representation of the examples as points in space, mapped so that the examples of the separate categories are divided by a clear gap that is as wide as possible. New examples are then mapped into that same space and predicted to belong to a category based on which side of the gap they fall.

# ***k*-nearest neighbors algorithm**

From Wikipedia, the free encyclopedia

In [pattern recognition](https://en.wikipedia.org/wiki/Pattern_recognition), the ***k*-nearest neighbors algorithm** (***k*-NN**) is a [non-parametric](https://en.wikipedia.org/wiki/Non-parametric_statistics) method used for [classification](https://en.wikipedia.org/wiki/Statistical_classification) and [regression](https://en.wikipedia.org/wiki/Regression_analysis).[[1]](https://en.wikipedia.org/wiki/K-nearest_neighbors_algorithm#cite_note-1) In both cases, the input consists of the *k* closest training examples in the [feature space](https://en.wikipedia.org/wiki/Feature_space). The output depends on whether *k*-NN is used for classification or regression:

* In *k-NN classification*, the output is a class membership. An object is classified by a majority vote of its neighbors, with the object being assigned to the class most common among its *k* nearest neighbors (*k* is a positive [integer](https://en.wikipedia.org/wiki/Integer), typically small). If *k* = 1, then the object is simply assigned to the class of that single nearest neighbor.
* In *k-NN regression*, the output is the property value for the object. This value is the average of the values of its *k* nearest neighbors.

*k*-NN is a type of [instance-based learning](https://en.wikipedia.org/wiki/Instance-based_learning), or [lazy learning](https://en.wikipedia.org/wiki/Lazy_learning), where the function is only approximated locally and all computation is deferred until classification. The *k*-NN algorithm is among the simplest of all [machine learning](https://en.wikipedia.org/wiki/Machine_learning) algorithms.

**Gaussian Processes (GP)** are a generic supervised learning method designed to solve *regression* and *probabilistic classification* problems.

The advantages of Gaussian processes are:

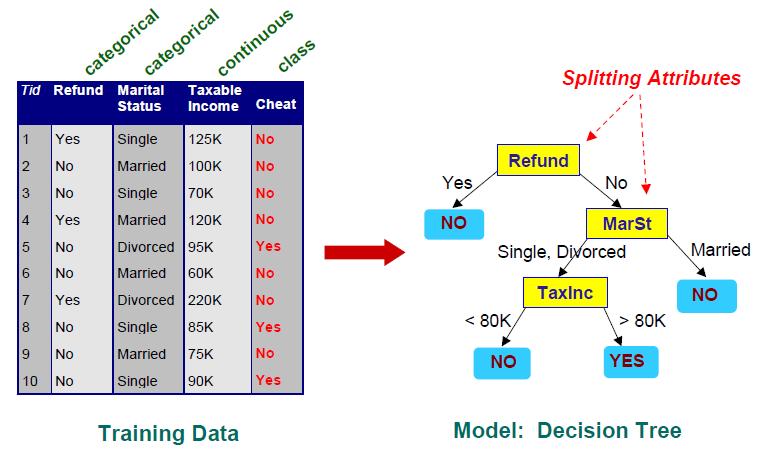
* The prediction interpolates the observations (at least for regular kernels).
* The prediction is probabilistic (Gaussian) so that one can compute empirical confidence intervals and decide based on those if one should refit (online fitting, adaptive fitting) the prediction in some region of interest.
* Versatile: different [kernels](http://scikit-learn.org/stable/modules/gaussian_process.html#gp-kernels) can be specified. Common kernels are provided, but it is also possible to specify custom kernels.

The disadvantages of Gaussian processes include:

* They are not sparse, i.e., they use the whole samples/features information to perform the prediction.
* They lose efficiency in high dimensional spaces – namely when the number of features exceeds a few dozens.

## **Decision Tree Based Method**

The decision tree classifiers organized a series of test questions and conditions in a tree structure. The following figure [ 1 ] shows a example decision tree for predictin whether the person cheats. In the decision tree, the root and internal nodes contain attribute test conditions to separate recordes that have different characteristics. All the terminal node is assigned a class lable Yes or No.



**Random forests** or random decision forests[[1]](https://en.wikipedia.org/wiki/Random_forest#cite_note-ho1995-1)[[2]](https://en.wikipedia.org/wiki/Random_forest#cite_note-ho1998-2) are an [ensemble learning](https://en.wikipedia.org/wiki/Ensemble_learning) method for [classification](https://en.wikipedia.org/wiki/Statistical_classification), [regression](https://en.wikipedia.org/wiki/Regression_analysis) and other tasks, that operate by constructing a multitude of [decision trees](https://en.wikipedia.org/wiki/Decision_tree_learning) at training time and outputting the class that is the [mode](https://en.wikipedia.org/wiki/Mode_(statistics)) of the classes (classification) or mean prediction (regression) of the individual trees. Random decision forests correct for decision trees' habit of [overfitting](https://en.wikipedia.org/wiki/Overfitting) to their training set.[[3]](https://en.wikipedia.org/wiki/Random_forest#cite_note-elemstatlearn-3):587–588

# **Naive Bayes classifier - Gausian NB**

From Wikipedia, the free encyclopedia

In [machine learning](https://en.wikipedia.org/wiki/Machine_learning), **naive Bayes classifiers** are a family of simple [probabilistic classifiers](https://en.wikipedia.org/wiki/Probabilistic_classifier) based on applying [Bayes' theorem](https://en.wikipedia.org/wiki/Bayes%27_theorem) with strong (naive) [independence](https://en.wikipedia.org/wiki/Statistical_independence) assumptions between the features.

Naive Bayes has been studied extensively since the 1950s. It was introduced under a different name into the [text retrieval](https://en.wikipedia.org/wiki/Information_retrieval) community in the early 1960s,[[1]](https://en.wikipedia.org/wiki/Naive_Bayes_classifier#cite_note-aima-1):488 and remains a popular (baseline) method for [text categorization](https://en.wikipedia.org/wiki/Text_categorization), the problem of judging documents as belonging to one category or the other (such as [spam or legitimate](https://en.wikipedia.org/wiki/Spam_filtering), sports or politics, etc.) with [word frequencies](https://en.wikipedia.org/wiki/Bag_of_words) as the features. With appropriate pre-processing, it is competitive in this domain with more advanced methods including [support vector machines](https://en.wikipedia.org/wiki/Support_vector_machine).[[2]](https://en.wikipedia.org/wiki/Naive_Bayes_classifier#cite_note-rennie-2) It also finds application in automatic [medical diagnosis](https://en.wikipedia.org/wiki/Medical_diagnosis).[[3]](https://en.wikipedia.org/wiki/Naive_Bayes_classifier#cite_note-rish-3)

# **Quadratic classifier**

From Wikipedia, the free encyclopedia

*This article is about machine learning and statistical classification. For other uses of the word "quadratic" in mathematics, see*[*Quadratic (disambiguation)*](https://en.wikipedia.org/wiki/Quadratic_(disambiguation))*.*

|  |  |
| --- | --- |
| [https://upload.wikimedia.org/wikipedia/en/thumb/9/99/Question_book-new.svg/50px-Question_book-new.svg.png](https://en.wikipedia.org/wiki/File:Question_book-new.svg) | This article **needs additional citations for**[**verification**](https://en.wikipedia.org/wiki/Wikipedia:Verifiability). Please help [improve this article](https://en.wikipedia.org/w/index.php?title=Quadratic_classifier&action=edit) by [adding citations to reliable sources](https://en.wikipedia.org/wiki/Help:Introduction_to_referencing_with_Wiki_Markup/1). Unsourced material may be challenged and removed. *(December 2009)(*[*Learn how and when to remove this template message*](https://en.wikipedia.org/wiki/Help:Maintenance_template_removal)*)* |

A **quadratic classifier** is used in [machine learning](https://en.wikipedia.org/wiki/Machine_learning) and [statistical classification](https://en.wikipedia.org/wiki/Statistical_classification) to separate measurements of two or more classes of objects or events by a [quadric](https://en.wikipedia.org/wiki/Quadric) surface. It is a more general version of the [linear classifier](https://en.wikipedia.org/wiki/Linear_classifier).

## Naive Bayes

Naive Bayes uses Bayes Theorem to model the conditional relationship of each attribute to the class variable.

This recipe shows the fitting of an Naive Bayes model to the iris dataset.



|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16 | # Gaussian Naive Bayes  from sklearn import datasets  from sklearn import metrics  from sklearn.naive\_bayes import GaussianNB  # load the iris datasets  dataset = datasets.load\_iris()  # fit a Naive Bayes model to the data  model = GaussianNB()  model.fit(dataset.data, dataset.target)  print(model)  # make predictions  expected = dataset.target  predicted = model.predict(dataset.data)  # summarize the fit of the model  print(metrics.classification\_report(expected, predicted))  print(metrics.confusion\_matrix(expected, predicted)) |

For more information see the [API reference for the Gaussian Naive Bayes](http://scikit-learn.org/stable/modules/generated/sklearn.naive_bayes.GaussianNB.html#sklearn.naive_bayes.GaussianNB) for details on configuring the algorithm parameters. Also see the [Naive Bayes section of the user guide](http://scikit-learn.org/stable/modules/naive_bayes.html#naive-bayes).

## k-Nearest Neighbor

The k-Nearest Neighbor (kNN) method makes predictions by locating similar cases to a given data instance (using a similarity function) and returning the average or majority of the most similar data instances. The kNN algorithm can be used for classification or regression.

This recipe shows use of the kNN model to make predictions for the iris dataset.



|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16 | # k-Nearest Neighbor  from sklearn import datasets  from sklearn import metrics  from sklearn.neighbors import KNeighborsClassifier  # load iris the datasets  dataset = datasets.load\_iris()  # fit a k-nearest neighbor model to the data  model = KNeighborsClassifier()  model.fit(dataset.data, dataset.target)  print(model)  # make predictions  expected = dataset.target  predicted = model.predict(dataset.data)  # summarize the fit of the model  print(metrics.classification\_report(expected, predicted))  print(metrics.confusion\_matrix(expected, predicted)) |

For more information see the [API reference for the k-Nearest Neighbor](http://scikit-learn.org/stable/modules/generated/sklearn.neighbors.KNeighborsClassifier.html#sklearn.neighbors.KNeighborsClassifier) for details on configuring the algorithm parameters. Also see the [k-Nearest Neighbor section of the user guide](http://scikit-learn.org/stable/modules/neighbors.html#neighbors).

## Classification and Regression Trees

Classification and Regression Trees (CART) are constructed from a dataset by making splits that best separate the data for the classes or predictions being made. The CART algorithm can be used for classification or regression.

This recipe shows use of the CART model to make predictions for the iris dataset.



|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16 | # Decision Tree Classifier  from sklearn import datasets  from sklearn import metrics  from sklearn.tree import DecisionTreeClassifier  # load the iris datasets  dataset = datasets.load\_iris()  # fit a CART model to the data  model = DecisionTreeClassifier()  model.fit(dataset.data, dataset.target)  print(model)  # make predictions  expected = dataset.target  predicted = model.predict(dataset.data)  # summarize the fit of the model  print(metrics.classification\_report(expected, predicted))  print(metrics.confusion\_matrix(expected, predicted)) |

For more information see the [API reference for CART](http://scikit-learn.org/stable/modules/generated/sklearn.tree.DecisionTreeClassifier.html#sklearn.tree.DecisionTreeClassifier) for details on configuring the algorithm parameters. Also see the [Decision Tree section of the user guide](http://scikit-learn.org/stable/modules/tree.html#tree).

## Support Vector Machines

Support Vector Machines (SVM) are a method that uses points in a transformed problem space that best separate classes into two groups. Classification for multiple classes is supported by a one-vs-all method. SVM also supports regression by modeling the function with a minimum amount of allowable error.

This recipe shows use of the SVM model to make predictions for the iris dataset.



|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16 | # Support Vector Machine  from sklearn import datasets  from sklearn import metrics  from sklearn.svm import SVC  # load the iris datasets  dataset = datasets.load\_iris()  # fit a SVM model to the data  model = SVC()  model.fit(dataset.data, dataset.target)  print(model)  # make predictions  expected = dataset.target  predicted = model.predict(dataset.data)  # summarize the fit of the model  print(metrics.classification\_report(expected, predicted))  print(metrics.confusion\_matrix(expected, predicted)) |

For more information see the [API reference for SVM](http://scikit-learn.org/stable/modules/generated/sklearn.svm.SVC.html#sklearn.svm.SVC) for details on configuring the algorithm parameters. Also see the [SVM section of the user guide](http://scikit-learn.org/stable/modules/svm.html#svm).