Linear Regression

From Scikit-learn

The following are a set of methods intended for regression in which

the target value is expected to be a linear combination of the features.

In mathematical notation, if

\(\hat{y}\)

is the predicted

value.

\[\hat{y}(w, x) = w_0 + w_1 x_1 + ... + w_p x_p\]

Across the module, we designate the vector

\(w = (w_1,

..., w_p)\)

as

coef_

and

\(w_0\)

as

intercept_

.

To perform classification with generalized linear models, see

Logistic regression

.

From

Instatistics,linear regressionis amodelthat estimates the relationship between ascalarresponse (dependent variable) and one or more explanatory variables (regressororindependent variable). A model with exactly one explanatory variable is asimple linear regression; a model with two or more explanatory variables is amultiple linear regression.[1]This term is distinct

frommultivariate linear regression, which predicts multiplecorrelateddependent variables rather than a single dependent variable.[2]

In linear regression, the relationships are modeled usinglinear predictor functionswhose unknown modelparametersareestimatedfrom thedata. Most commonly, theconditional meanof the response given the values of the explanatory variables (or predictors) is assumed to be anaffine functionof those values; less commonly, the conditionalmedianor some otherquantileis used. Like all forms ofregression analysis, linear regression focuses on theconditional probability distributionof the response given the values of the predictors, rather than on thejoint probability distributionof all of these variables, which is the domain ofmultivariate analysis.

Linear regression is also a type ofmachine learningalgorithm, more specifically asupervisedalgorithm, that learns from the labelled datasets and maps the data points to the most optimized linear functions that can be used for prediction on new datasets.[3]

Linear regression was the first type of regression analysis to be studied rigorously, and to be used extensively in practical applications.[4]This is because models which depend linearly on their unknown parameters are easier to fit than models which are non-linearly related to their parameters and because the statistical properties of the resulting estimators are easier to determine.

Linear regression has many practical uses. Most applications fall into one of the following two broad categories:

Linear regression models are often fitted using theleast squaresapproach, but they may also be fitted in other ways, such as by minimizing the "lack of fit" in some othernorm(as withleast absolute deviationsregression), or by minimizing a penalized version of the least squarescost functionas inridge regression(L2-norm penalty) andlasso(L1-norm penalty). Use of theMean Squared Error(MSE) as the cost on a dataset that has many large outliers, can result in a model that fits the outliers more than the true data due to the higher importance assigned by MSE to large errors. So, cost functions that are robust to outliers should be used if the dataset has many largeoutliers. Conversely, the least squares approach can be used to fit models that are

not linear models. Thus, although the terms "least squares" and "linear model" are closely linked, they are not synonymous.

Given adata set{yi,xi1,...,xip}i=1n{\displaystyle \{y_{i},\,x_{i1},\ldots ,x_{ip}\}_{i=1}^{n}}ofnstatistical units, a linear regression model assumes that the relationship between the dependent variableyand the vector of regressorsxislinear. This relationship is modeled through adisturbance termorerror variableε—an unobservedrandom variablethat adds "noise" to the linear relationship between the dependent variable and regressors. Thus the model takes the formyi=β0+β1xi1+⋯+βpxip+εi=xiTβ+εi,i=1,...,n,{\displaystyle y_{i}=\beta _{0}+\beta _{1}x_{i1}+\cdots +\beta _{p}x_{ip}+\varepsilon _{i}=\mathbf {x} _{i}^{\mathsf {T}}{\boldsymbol {\beta }}+\varepsilon _{i},\qquad i=1,\ldots ,n,}whereTdenotes thetranspose, so thatxiTβis theinner productbetweenvectorsxiandβ.

Often thesenequations are stacked together and written inmatrix notationas

where

Fitting a linear model to a given data set usually requires estimating the regression coefficientsβ{\displaystyle {\boldsymbol {\beta }}}such that the error termε=y−Xβ{\displaystyle {\boldsymbol {\varepsilon }}=\mathbf {y} -\mathbf {X} {\boldsymbol {\beta }}}is minimized. For example, it is common to use the sum of squared errors‖ε‖22{\displaystyle \|{\boldsymbol {\varepsilon }}\|_{2}^{2}}as a measure ofε{\displaystyle {\boldsymbol {\varepsilon }}}for minimization.

Logistic Regression

From Scikit-learn

The following are a set of methods intended for regression in which

the target value is expected to be a linear combination of the features.

In mathematical notation, if

\(\hat{y}\)

is the predicted

value.

$$\hat{y}(w, x) = w_0 + w_1 x_1 + ... + w_p x_p$$

Across the module, we designate the vector

$w = (w_1,$

..., $w_p)$

as

coef_

and

w_0

as

intercept_

.

To perform classification with generalized linear models, see

Logistic regression

.

From

Instatistics, alogistic model(orlogit model) is astatistical modelthat models thelog-oddsof an event as alinear combinationof one or moreindependent variables. Inregression analysis,logistic regression[1](orlogit regression)estimatesthe parameters of a logistic model (the coefficients in the linear or non linear combinations). In binary logistic regression there is a singlebinarydependent variable, coded by anindicator variable, where the two values are labeled "0" and "1", while theindependent variablescan each be a binary variable (two classes, coded by an indicator variable) or acontinuous variable(any real value). The corresponding probability of the value labeled "1" can vary between 0 (certainly the value "0") and 1 (certainly the value "1"), hence the labeling;[2]the function that converts log-odds to probability is thelogistic function, hence the name. Theunit of measurementfor the log-odds scale is called

alogit, fromlogistic unit, hence the alternative names. See§ Backgroundand§ Definitionfor formal mathematics, and§ Examplefor a worked example.

Binary variables are widely used in statistics to model the probability of a certain class or event taking place, such as the probability of a team winning, of a patient being healthy, etc. (see§ Applications), and the logistic model has been the most commonly used model forbinary regressionsince about 1970.[3]Binary variables can be generalized tocategorical variableswhen there are more than two possible values (e.g. whether an image is of a cat, dog, lion, etc.), and the binary logistic regression generalized tomultinomial logistic regression. If the multiple categories areordered, one can use theordinal logistic regression(for example the proportional odds ordinal logistic model[4]). See§ Extensionsfor further extensions. The logistic regression model itself simply models probability of output in terms of input and does not performstatistical classification(it is not a classifier), though it can be used to make a classifier, for instance by choosing a cutoff value and classifying inputs with probability greater than the cutoff as one class, below the cutoff as the other; this is a common way to make abinary classifier.

Analogous linear models for binary variables with a differentsigmoid functioninstead of the logistic function (to convert the linear combination to a probability) can also be used, most notably theprobit model; see§ Alternatives. The defining characteristic of the logistic model is that increasing one of the independent variables multiplicatively scales the odds of the given outcome at aconstantrate, with each independent variable having its own parameter; for a binary dependent variable this generalizes theodds ratio. More abstractly, the logistic function is thenatural parameterfor theBernoulli distribution, and in this sense is the "simplest" way to convert a real number to a probability. In particular, it maximizes entropy (minimizes added information), and in this sense makes the fewest assumptions of the data being modeled; see§ Maximum entropy.

The parameters of a logistic regression are most commonly estimated bymaximum-likelihood estimation(MLE). This does not have a closed-form expression, unlikelinear least squares;

see§ Model fitting. Logistic regression by MLE plays a similarly basic role for binary or categorical responses as linear regression byordinary least squares(OLS) plays forscalarresponses: it is a simple, well-analyzed baseline model; see§ Comparison with linear regressionfor discussion. The logistic regression as a general statistical model was originally developed and popularized primarily byJoseph Berkson,[5]beginning inBerkson (1944), where he coined "logit"; see§ History.

Logistic regression is used in various fields, including machine learning, most medical fields, and social sciences. For example, the Trauma and Injury Severity Score (TRISS), which is widely used to predict mortality in injured patients, was originally developed by Boydet al.using logistic regression.[6]Many other medical scales used to assess severity of a patient have been developed using logistic regression.[7][8][9][10]Logistic regression may be used to predict the risk of developing a given disease (e.g.diabetes;coronary heart disease), based on observed characteristics of the patient (age, sex,body mass index, results of variousblood tests, etc.).[11][12]Another example might be to predict whether a Nepalese voter will vote Nepali Congress or Communist Party of Nepal or Any Other Party, based on age, income, sex, race, state of residence, votes in previous elections, etc.[13]The technique can also be used inengineering, especially for predicting the probability of failure of a given process, system or product.[14][15]It is also used inmarketingapplications such as prediction of a customer's propensity to purchase a product or halt a subscription, etc.[16]Ineconomics, it can be used to predict the likelihood of a person ending up in the labor force, and a business application would be to predict the likelihood of a homeowner defaulting on amortgage.Conditional random fields, an extension of logistic regression to sequential data, are used innatural language processing. Disaster planners and engineers rely on these models to predict decisions taken by householders or building occupants in small-scale and large-scales evacuations, such as building fires, wildfires, hurricanes among others.[17][18][19]These models help in the development of reliabledisaster managing plansand safer design for thebuilt environment.

Logistic regression is asupervised machine learningalgorithm widely used forbinary

classificationtasks, such as identifying whether an email is spam or not and diagnosing diseases by assessing the presence or absence of specific conditions based on patient test results. This approach utilizes the logistic (or sigmoid) function to transform a linear combination of input features into a probability value ranging between 0 and 1. This probability indicates the likelihood that a given input corresponds to one of two predefined categories. The essential mechanism of logistic regression is grounded in the logistic function's ability to model the probability of binary outcomes accurately. With its distinctive S-shaped curve, the logistic function effectively maps any real-valued number to a value within the 0 to 1 interval. This feature renders it particularly suitable for binary classification tasks, such as sorting emails into "spam" or "not spam". By calculating the probability that the dependent variable will be categorized into a specific group, logistic regression provides a probabilistic framework that supports informed decision-making.[20]

As a simple example, we can use a logistic regression with one explanatory variable and two categories to answer the following question:

A group of 20 students spends between 0 and 6 hours studying for an exam. How does the number of hours spent studying affect the probability of the student passing the exam?

The reason for using logistic regression for this problem is that the values of the dependent variable, pass and fail, while represented by "1" and "0", are notcardinal numbers. If the problem was changed so that pass/fail was replaced with the grade 0–100 (cardinal numbers), then simpleregression analysiscould be used.

The table shows the number of hours each student spent studying, and whether they passed (1) or failed (0).

Decision Tree

From Scikit-learn

Decision Trees (DTs)

are a non-parametric supervised learning method used

for

classification

and

regression

. The goal is to create a model that predicts the value of a

target variable by learning simple decision rules inferred from the data

features. A tree can be seen as a piecewise constant approximation.

For instance, in the example below, decision trees learn from data to

approximate a sine curve with a set of if-then-else decision rules. The deeper

the tree, the more complex the decision rules and the fitter the model.

Some advantages of decision trees are:

Simple to understand and to interpret. Trees can be visualized.

Requires little data preparation. Other techniques often require data

normalization, dummy variables need to be created and blank values to

be removed. Some tree and algorithm combinations support

missing values

.

The cost of using the tree (i.e., predicting data) is logarithmic in the

number of data points used to train the tree.

Able to handle both numerical and categorical data. However, the scikit-learn

implementation does not support categorical variables for now. Other

techniques are usually specialized in analyzing datasets that have only one type

of variable. See

algorithms

for more

information.

Able to handle multi-output problems.

Uses a white box model. If a given situation is observable in a model, the explanation for the condition is easily explained by boolean logic. By contrast, in a black box model (e.g., in an artificial neural network), results may be more difficult to interpret.

Possible to validate a model using statistical tests. That makes it possible to account for the reliability of the model.

Performs well even if its assumptions are somewhat violated by the true model from which the data were generated.

Simple to understand and to interpret. Trees can be visualized.

Requires little data preparation. Other techniques often require data normalization, dummy variables need to be created and blank values to be removed. Some tree and algorithm combinations support missing values

.

The cost of using the tree (i.e., predicting data) is logarithmic in the number of data points used to train the tree.

Able to handle both numerical and categorical data. However, the scikit-learn implementation does not support categorical variables for now. Other techniques are usually specialized in analyzing datasets that have only one type of variable. See

algorithms

for more

information.

Able to handle multi-output problems.

Uses a white box model. If a given situation is observable in a model, the explanation for the condition is easily explained by boolean logic. By contrast, in a black box model (e.g., in an artificial neural

network), results may be more difficult to interpret.

Possible to validate a model using statistical tests. That makes it possible to account for the reliability of the model.

Performs well even if its assumptions are somewhat violated by the true model from which the data were generated.

The disadvantages of decision trees include:

Decision-tree learners can create over-complex trees that do not generalize the data well. This is called overfitting. Mechanisms such as pruning, setting the minimum number of samples required at a leaf node or setting the maximum depth of the tree are necessary to avoid this problem.

Decision trees can be unstable because small variations in the data might result in a completely different tree being generated. This problem is mitigated by using decision trees within an ensemble.

Predictions of decision trees are neither smooth nor continuous, but piecewise constant approximations as seen in the above figure. Therefore, they are not good at extrapolation.

The problem of learning an optimal decision tree is known to be NP-complete under several aspects of optimality and even for simple concepts. Consequently, practical decision-tree learning algorithms are based on heuristic algorithms such as the greedy algorithm where locally optimal decisions are made at each node. Such algorithms cannot guarantee to return the globally optimal decision tree. This can be mitigated by training multiple trees in an ensemble learner, where the features and samples are randomly sampled with replacement.

There are concepts that are hard to learn because decision trees

do not express them easily, such as XOR, parity or multiplexer problems.

Decision tree learners create biased trees if some classes dominate.
It is therefore recommended to balance the dataset prior to fitting
with the decision tree.

Decision-tree learners can create over-complex trees that do not
generalize the data well. This is called overfitting. Mechanisms
such as pruning, setting the minimum number of samples required
at a leaf node or setting the maximum depth of the tree are
necessary to avoid this problem.

Decision trees can be unstable because small variations in the
data might result in a completely different tree being generated.
This problem is mitigated by using decision trees within an
ensemble.

Predictions of decision trees are neither smooth nor continuous, but
piecewise constant approximations as seen in the above figure. Therefore,
they are not good at extrapolation.

The problem of learning an optimal decision tree is known to be
NP-complete under several aspects of optimality and even for simple
concepts. Consequently, practical decision-tree learning algorithms
are based on heuristic algorithms such as the greedy algorithm where
locally optimal decisions are made at each node. Such algorithms
cannot guarantee to return the globally optimal decision tree. This
can be mitigated by training multiple trees in an ensemble learner,
where the features and samples are randomly sampled with replacement.

There are concepts that are hard to learn because decision trees
do not express them easily, such as XOR, parity or multiplexer problems.

Decision tree learners create biased trees if some classes dominate.

It is therefore recommended to balance the dataset prior to fitting

with the decision tree.

From

Decision tree learningis asupervised learningapproach used instatistics,data miningandmachine learning. In this formalism, a classification or regressiondecision treeis used as apredictive modelto draw conclusions about a set of observations.

Tree models where the target variable can take a discrete set of values are calledclassificationtrees; in these tree structures,leavesrepresent class labels and branches representconjunctionsof features that lead to those class labels. Decision trees where the target variable can take continuous values (typicallyreal numbers) are calledregressiontrees. More generally, the concept of regression tree can be extended to any kind of object equipped with pairwise dissimilarities such as categorical sequences.[1]

Decision trees are among the most popular machine learning algorithms given their intelligibility and simplicity.[2]

In decision analysis, a decision tree can be used to visually and explicitly represent decisions anddecision making. Indata mining, a decision tree describes data (but the resulting classification tree can be an input for decision making).

Decision tree learning is a method commonly used in data mining.[3]The goal is to create a model that predicts the value of a target variable based on several input variables.

A decision tree is a simple representation for classifying examples. For this section, assume that all of the inputfeatureshave finite discrete domains, and there is a single target feature called the "classification". Each element of the domain of the classification is called aclass.

A decision tree or a classification tree is a tree in which each internal (non-leaf) node is labeled with an input feature. The arcs coming from a node labeled with an input feature are labeled with each of the possible values of the target feature or the arc leads to a subordinate decision node on a different input feature. Each leaf of the tree is labeled with a class or a probability distribution over the classes, signifying that the data set has been classified by the tree into

either a specific class, or into a particular probability distribution (which, if the decision tree is well-constructed, is skewed towards certain subsets of classes).

A tree is built by splitting the sourceset, constituting the root node of the tree, into subsets—which constitute the successor children. The splitting is based on a set of splitting rules based on classification features.[4]This process is repeated on each derived subset in a recursive manner calledrecursive partitioning.

Therecursionis completed when the subset at a node has all the same values of the target variable, or when splitting no longer adds value to the predictions. This process oftop-down induction of decision trees(TDIDT)[5]is an example of agreedy algorithm, and it is by far the most common strategy for learning decision trees from data.[6]

Indata mining, decision trees can be described also as the combination of mathematical and computational techniques to aid the description, categorization and generalization of a given set of data.

Data comes in records of the form:

The dependent variable,Y{\displaystyle Y}, is the target variable that we are trying to understand, classify or generalize. The vectorx{\displaystyle {\textbf {x}}}is composed of the features,x1,x2,x3{\displaystyle x_{1},x_{2},x_{3}}etc., that are used for that task.

Random Forest

From Scikit-learn

Ensemble methods

combine the predictions of several

base estimators built with a given learning algorithm in order to improve

generalizability / robustness over a single estimator.

Two very famous examples of ensemble methods are

gradient-boosted trees

and

random forests

.

More generally, ensemble models can be applied to any base learner beyond

trees, in averaging methods such as

Bagging methods

,

model stacking

, or

Voting

, or in

boosting, as

AdaBoost

.

Fron

Random forestsorrandom decision forestsis anensemble learningmethod forclassification,regressionand other tasks that works by creating a multitude ofdecision treesduring training. For classification tasks, the output of the random forest is the class selected by most trees. For regression tasks, the output is the average of the predictions of the trees.[1][2]Random forests correct for decision trees' habit ofoverfittingto theirtraining set.[3]: 587–588

The first algorithm for random decision forests was created in 1995 byTin Kam Ho[1]using therandom subspace method,[2]which, in Ho's formulation, is a way to implement the "stochastic discrimination" approach to classification proposed by Eugene Kleinberg.[4][5][6]

An extension of the algorithm was developed byLeo Breiman[7]andAdele Cutler,[8]who registered[9]"Random Forests" as atrademarkin 2006 (as of 2019[update], owned byMinitab, Inc.).[10]The extension combines Breiman's "bagging" idea and random selection of features, introduced first by Ho[1]and later independently by Amit andGeman[11]in order to construct a

collection of decision trees with controlled variance.

The general method of random decision forests was first proposed by Salzberg and Heath in 1993,[12]with a method that used a randomized decision tree algorithm to create multiple trees and then combine them using majority voting. This idea was developed further by Ho in 1995.[1]Ho established that forests of trees splitting with oblique hyperplanes can gain accuracy as they grow without suffering from overtraining, as long as the forests are randomly restricted to be sensitive to only selectedfeaturedimensions. A subsequent work along the same lines[2]concluded that other splitting methods behave similarly, as long as they are randomly forced to be insensitive to some feature dimensions. This observation that a more complex classifier (a larger forest) gets more accurate nearly monotonically is in sharp contrast to the common belief that the complexity of a classifier can only grow to a certain level of accuracy before being hurt by overfitting. The explanation of the forest method's resistance to overtraining can be found in Kleinberg's theory of stochastic discrimination.[4][5][6]

The early development of Breiman's notion of random forests was influenced by the work of Amit and Geman[11]who introduced the idea of searching over a random subset of the available decisions when splitting a node, in the context of growing a singletree. The idea of random subspace selection from Ho[2]was also influential in the design of random forests. This method grows a forest of trees, and introduces variation among the trees by projecting the training data into a randomly chosensubspacebefore fitting each tree or each node. Finally, the idea of randomized node optimization, where the decision at each node is selected by a randomized procedure, rather than a deterministic optimization was first introduced byThomas G. Dietterich.[13]

The proper introduction of random forests was made in a paper byLeo Breiman.[7]This paper describes a method of building a forest of uncorrelated trees using aCARTlike procedure, combined with randomized node optimization andbagging. In addition, this paper combines several ingredients, some previously known and some novel, which form the basis of the modern practice of random forests, in particular:

The report also offers the first theoretical result for random forests in the form of a bound on thegeneralization errorwhich depends on the strength of the trees in the forest and theircorrelation.

Decision trees are a popular method for various machine learning tasks. Tree learning is almost "an off-the-shelf procedure for data mining", sayHastieet al., "because it is invariant under scaling and various other transformations of feature values, is robust to inclusion of irrelevant features, and produces inspectable models. However, they are seldom accurate".[3]:352

In particular, trees that are grown very deep tend to learn highly irregular patterns: theyoverfittheir training sets, i.e. havelow bias, but very high variance. Random forests are a way of averaging multiple deep decision trees, trained on different parts of the same training set, with the goal of reducing the variance.[3]:587–588This comes at the expense of a small increase in the bias and some loss of interpretability, but generally greatly boosts the performance in the final model.

The training algorithm for random forests applies the general technique ofbootstrap aggregating, or bagging, to tree learners. Given a training setX=x1, ...,xnwith responsesY=y1, ...,yn, bagging repeatedly (Btimes) selects arandom sample with replacementof the training set and fits trees to these samples:

SVM

From Scikit-learn

Support vector machines (SVMs)

are a set of supervised learning

methods used for

classification

,

regression

and

outliers detection

.

The advantages of support vector machines are:

Effective in high dimensional spaces.

Still effective in cases where number of dimensions is greater

than the number of samples.

Uses a subset of training points in the decision function (called

support vectors), so it is also memory efficient.

Versatile: different

Kernel functions

can be

specified for the decision function. Common kernels are

provided, but it is also possible to specify custom kernels.

Effective in high dimensional spaces.

Still effective in cases where number of dimensions is greater

than the number of samples.

Uses a subset of training points in the decision function (called

support vectors), so it is also memory efficient.

Versatile: different

Kernel functions

can be

specified for the decision function. Common kernels are

provided, but it is also possible to specify custom kernels.

The disadvantages of support vector machines include:

If the number of features is much greater than the number of

samples, avoid over-fitting in choosing

Kernel functions

and regularization

term is crucial.

SVMs do not directly provide probability estimates, these are

calculated using an expensive five-fold cross-validation

(see

Scores and probabilities

, below).

If the number of features is much greater than the number of

samples, avoid over-fitting in choosing

Kernel functions

and regularization

term is crucial.

SVMs do not directly provide probability estimates, these are

calculated using an expensive five-fold cross-validation

(see

Scores and probabilities

, below).

The support vector machines in scikit-learn support both dense

(

numpy.ndarray

and convertible to that by

numpy.asarray

) and

sparse (any

scipy.sparse

) sample vectors as input. However, to use

an SVM to make predictions for sparse data, it must have been fit on such

data. For optimal performance, use C-ordered

numpy.ndarray

(dense) or

scipy.sparse.csr_matrix

(sparse) with

dtype=float64

.

From

Inmachine learning,support vector machines(SVMs, alsosupport vector networks[1]) aresupervisedmax-marginmodels with associated learningalgorithmsthat analyze data forclassificationandregression analysis. Developed atAT&T Bell Laboratories,[1][2]SVMs are one of the most studied models, being based on statistical learning frameworks ofVC theoryproposed byVapnik(1982, 1995) andChervonenkis(1974).

In addition to performinglinear classification, SVMs can efficiently perform non-linear classification using thekernel trick, representing the data only through a set of pairwise similarity comparisons between the original data points using a kernel function, which transforms them into coordinates in a higher-dimensionalfeature space. Thus, SVMs use the kernel trick to implicitly map their inputs into high-dimensional feature spaces, where linear classification can be performed.[3]Being max-margin models, SVMs are resilient to noisy data (e.g., misclassified examples). SVMs can also be used forregressiontasks, where the objective becomesε{\displaystyle \epsilon }-sensitive.

The support vector clustering[4]algorithm, created byHava SiegelmannandVladimir Vapnik, applies the statistics of support vectors, developed in the support vector machines algorithm, to categorize unlabeled data.[citation needed]These data sets requireunsupervised learningapproaches, which attempt to find naturalclustering of the datainto groups, and then to map new data according to these clusters.

The popularity of SVMs is likely due to their amenability to theoretical analysis, and their

flexibility in being applied to a wide variety of tasks, includingstructured predictionproblems. It is not clear that SVMs have better predictive performance than other linear models, such aslogistic regressionandlinear regression.[5]

Classifying datais a common task inmachine learning.

Suppose some given data points each belong to one of two classes, and the goal is to decide which class anewdata pointwill be in. In the case of support vector machines, a data point is viewed as ap{\displaystyle p}-dimensional vector (a list ofp{\displaystyle p}numbers), and we want to know whether we can separate such points with a(p−1){\displaystyle (p-1)}-dimensionalhyperplane. This is called alinear classifier. There are many hyperplanes that might classify the data. One reasonable choice as the best hyperplane is the one that represents the largest separation, ormargin, between the two classes. So we choose the hyperplane so that the distance from it to the nearest data point on each side is maximized. If such a hyperplane exists, it is known as themaximum-margin hyperplaneand the linear classifier it defines is known as amaximum-margin classifier; or equivalently, theperceptron of optimal stability.[6]

More formally, a support vector machine constructs ahyperplaneor set of hyperplanes in a high or infinite-dimensional space, which can be used forclassification,regression, or other tasks like outliers detection.[7]Intuitively, a good separation is achieved by the hyperplane that has the largest distance to the nearest training-data point of any class (so-called functional margin), since in general the larger the margin, the lower thegeneralization errorof the classifier.[8]A lowergeneralization errormeans that the implementer is less likely to experienceoverfitting.

Whereas the original problem may be stated in a finite-dimensional space, it often happens that the sets to discriminate are notlinearly separablein that space. For this reason, it was proposed[9]that the original finite-dimensional space be mapped into a much higher-dimensional space, presumably making the separation easier in that space. To keep the computational load reasonable, the mappings used by SVM schemes are designed to ensure thatdot productsof pairs of input data vectors may be computed easily in terms of the variables

in the original space, by defining them in terms of akernel functionk(x,y){\displaystyle k(x,y)}selected to suit the problem.[10]The hyperplanes in the higher-dimensional space are defined as the set of points whose dot product with a vector in that space is constant, where such a set of vectors is an orthogonal (and thus minimal) set of vectors that defines a hyperplane. The vectors defining the hyperplanes can be chosen to be linear combinations with parametersαi{\displaystyle \alpha _{i}}of images offeature vectorsxi{\displaystyle x_{i}}that occur in the data base. With this choice of a hyperplane, the pointsx{\displaystyle x}in thefeature spacethat are mapped into the hyperplane are defined by the relation∑iαik(xi,x)=constant.{\displaystyle \textstyle \sum _{i}\alpha _{i}k(x_{i},x)={\text{constant}}.}Note that ifk(x,y){\displaystyle k(x,y)}becomes small asy{\displaystyle y}grows further away fromx{\displaystyle x}, each term in the sum measures the degree of closeness of the test pointx{\displaystyle x}to the corresponding data base pointxi{\displaystyle x_{i}}. In this way, the sum of kernels above can be used to measure the relative nearness of each test point to the data points originating in one or the other of the sets to be discriminated. Note the fact that the set of pointsx{\displaystyle x}mapped into any hyperplane can be quite convoluted as a result, allowing much more complex discrimination between sets that are not convex at all in the original space.

SVMs can be used to solve various real-world problems:

The original SVM algorithm was invented byVladimir N. VapnikandAlexey Ya. Chervonenkisin 1964.[citation needed]In 1992, Bernhard Boser,Isabelle GuyonandVladimir Vapniksuggested a way to create nonlinear classifiers by applying thekernel trickto maximum-margin hyperplanes.[9]The "soft margin" incarnation, as is commonly used in software packages, was proposed byCorinna Cortesand Vapnik in 1993 and published in 1995.[1]

We are given a training dataset ofn{\displaystyle n}points of the form(x1,y1),...,(xn,yn),{\displaystyle (\mathbf {x} _{1},y_{1}),\ldots ,(\mathbf {x} _{n},y_{n}),}where theyi{\displaystyle y_{i}}are either 1 or −1, each indicating the class to which the pointxi{\displaystyle \mathbf {x} _{i}}belongs. Eachxi{\displaystyle \mathbf {x}

_{i}}is ap{\displaystyle p}-dimensionalrealvector. We want to find the "maximum-margin hyperplane" that divides the group of pointsxi{\displaystyle \mathbf {x} _{i}}for whichyi=1{\displaystyle y_{i}=1}from the group of points for whichyi=−1{\displaystyle y_{i}=-1}, which is defined so that the distance between the hyperplane and the nearest pointxi{\displaystyle \mathbf {x} _{i}}from either group is maximized.

KNN

From Scikit-learn

sklearn.neighbors

provides functionality for unsupervised and

supervised neighbors-based learning methods. Unsupervised nearest neighbors

is the foundation of many other learning methods,

notably manifold learning and spectral clustering. Supervised neighbors-based

learning comes in two flavors:

classification

for data with

discrete labels, and

regression

for data with continuous labels.

The principle behind nearest neighbor methods is to find a predefined number

of training samples closest in distance to the new point, and

predict the label from these. The number of samples can be a user-defined

constant (k-nearest neighbor learning), or vary based

on the local density of points (radius-based neighbor learning).

The distance can, in general, be any metric measure: standard Euclidean

distance is the most common choice.

Neighbors-based methods are known as

non-generalizing

machine

learning methods, since they simply “remember” all of its training data

(possibly transformed into a fast indexing structure such as a

Ball Tree

or

KD Tree

).

Despite its simplicity, nearest neighbors has been successful in a

large number of classification and regression problems, including

handwritten digits and satellite image scenes. Being a non-parametric method,

it is often successful in classification situations where the decision

boundary is very irregular.

The classes in

sklearn.neighbors

can handle either NumPy arrays or

scipy.sparse

matrices as input. For dense matrices, a large number of

possible distance metrics are supported. For sparse matrices, arbitrary

Minkowski metrics are supported for searches.

There are many learning routines which rely on nearest neighbors at their

core. One example is

kernel density estimation

,

discussed in the

density estimation

section.

Fron

Instatistics, thek-nearest neighbors algorithm(k-NN) is anon-parametricsupervised learningmethod. It was first developed byEvelyn FixandJoseph Hodgesin 1951,[1]and later expanded byThomas Cover.[2]Most often, it is used forclassification, as ak-NN classifier, the output of which is a class membership. An object is classified by a plurality vote of its neighbors, with the object being assigned to the class most common among itsknearest neighbors (kis a positiveinteger, typically small). Ifk= 1, then the object is simply assigned to the class of that single nearest neighbor.

Thek-NN algorithm can also be generalized forregression. Ink-NN regression, also known asnearest neighbor smoothing, the output is the property value for the object. This value is the average of the values ofknearest neighbors. Ifk= 1, then the output is simply assigned to the value of that single nearest neighbor, also known asnearest neighbor interpolation.

For both classification and regression, a useful technique can be to assign weights to the contributions of the neighbors, so that nearer neighbors contribute more to the average than distant ones. For example, a common weighting scheme consists of giving each neighbor a weight of 1/d, wheredis the distance to the neighbor.[3]

The input consists of thekclosest training examples in adata set.

The neighbors are taken from a set of objects for which the class (fork-NN classification) or the object property value (fork-NN regression) is known. This can be thought of as the training set for the algorithm, though no explicit training step is required.

A peculiarity (sometimes even a disadvantage) of thek-NN algorithm is its sensitivity to the local structure of the data.

Ink-NN classification the function is only approximated locally and all computation is deferred until function evaluation. Since this algorithm relies on distance, if the features represent different physical units or come in vastly different scales, then feature-wisenormalizingof the training data can greatly improve its accuracy.[4]

Suppose we have pairs(X1,Y1),(X2,Y2),...,(Xn,Yn){\displaystyle

(X_{1},Y_{1}),(X_{2},Y_{2}),\dots ,(X_{n},Y_{n})}taking values inRd×{1,2}{\displaystyle \mathbb {R} ^{d}\times \{1,2\}}, whereYis the class label ofX, so thatX|Y=r∼Pr{\displaystyle X|Y=r\sim P_{r}}forr=1,2{\displaystyle r=1,2}(and probability distributionsPr{\displaystyle P_{r}}). Given some norm‖·‖{\displaystyle \|\cdot \|}onRd{\displaystyle \mathbb {R} ^{d}}and a pointx∈Rd{\displaystyle x\in \mathbb {R} ^{d}}, let(X(1),Y(1)),...,(X(n),Y(n)){\displaystyle (X_{(1)},Y_{(1)}),\dots ,(X_{(n)},Y_{(n)})}be a reordering of the training data such that‖X(1)−x‖≤⋯≤‖X(n)−x‖{\displaystyle \|X_{(1)}-x\|\leq \dots \leq \|X_{(n)}-x\|}.

The training examples are vectors in a multidimensional feature space, each with a class label. The training phase of the algorithm consists only of storing thefeature vectorsand class labels of the training samples.

In the classification phase,kis a user-defined constant, and an unlabeled vector (a query or test point) is classified by assigning the label which is most frequent among thektraining samples nearest to that query point.

A commonly used distance metric forcontinuous variablesisEuclidean distance. For discrete variables, such as for text classification, another metric can be used, such as theoverlap metric(orHamming distance). In the context of gene expression microarray data, for example,k-NN has been employed with correlation coefficients, such as Pearson and Spearman, as a metric.[5]Often, the classification accuracy ofk-NN can be improved significantly if the distance metric is learned with specialized algorithms such asLarge Margin Nearest NeighbororNeighbourhood components analysis.

A drawback of the basic "majority voting" classification occurs when the class distribution is skewed. That is, examples of a more frequent class tend to dominate the prediction of the new example, because they tend to be common among theknearest neighbors due to their large number.[7]One way to overcome this problem is to weight the classification, taking into account the distance from the test point to each of itsknearest neighbors. The class (or value, in regression problems) of each of theknearest points is multiplied by a weight proportional to the

inverse of the distance from that point to the test point. Another way to overcome skew is by abstraction in data representation. For example, in aself-organizing map(SOM), each node is a representative (a center) of a cluster of similar points, regardless of their density in the original training data.K-NN can then be applied to the SOM.

Ensemble

From Scikit-learn

Ensemble methods

combine the predictions of several

base estimators built with a given learning algorithm in order to improve

generalizability / robustness over a single estimator.

Two very famous examples of ensemble methods are

gradient-boosted trees

and

random forests

.

More generally, ensemble models can be applied to any base learner beyond

trees, in averaging methods such as

Bagging methods

,

model stacking

, or

Voting

, or in

boosting, as

AdaBoost

.

Fron

Instatisticsandmachine learning,ensemble methodsuse multiple learning algorithms to obtain betterpredictive performancethan could be obtained from any of the constituent learning algorithms alone.[1][2][3]Unlike astatistical ensemblein statistical mechanics, which is usually infinite, a machine learning ensemble consists of only a concrete finite set of alternative models, but typically allows for much more flexible structure to exist among those alternatives.

Supervised learningalgorithms search through ahypothesisspace to find a suitable hypothesis that will make good predictions with a particular problem.[4]Even if this space contains hypotheses that are very well-suited for a particular problem, it may be very difficult to find a good one. Ensembles combine multiple hypotheses to form one which should be theoretically better.

Ensemble learningtrains two or more machine learning algorithms on a specificclassificationorregressiontask. The algorithms within the ensemble model are generally referred as "base models", "base learners", or "weak learners" in literature. These base models can be constructed using a single modelling algorithm, or several different algorithms. The idea is to train a diverse set of weak models on the same modelling task, such that the outputs of each weak learner have poor predictive ability (i.e., highbias), and among all weak learners, the outcome and error values exhibit highvariance. Fundamentally, an ensemble learning model trains at least two high-bias (weak) and high-variance (diverse) models to be combined into a better-performing model. The set of weak models — which would not produce satisfactory predictive results individually — are combined or averaged to produce a single, high performing, accurate, and low-variance model to fit the task as required.

Ensemble learning typically refers to bagging (bootstrap aggregating),boostingor stacking/blending techniques to induce high variance among the base models. Bagging creates diversity by generating random samples from the training observations and fitting the same model to each different sample — also known ashomogeneous parallel ensembles. Boosting follows an iterative process by sequentially training each base model on the up-weighted errors

of the previous base model, producing an additive model to reduce the final model errors — also known assequential ensemble learning. Stacking or blending consists of different base models, each trained independently (i.e. diverse/high variance) to be combined into the ensemble model — producing aheterogeneous parallel ensemble. Common applications of ensemble learning includerandom forests(an extension of bagging), Boosted Tree models, andGradient BoostedTree Models. Models in applications of stacking are generally more task-specific — such as combining clustering techniques with other parametric and/or non-parametric techniques.[5]

The broader term Multiple Classifier Systems (MCS) encompasses not only ensemble methods built from identical base learners (homogeneous ensembles), but also extends to the hybridization of hypotheses generated from diverse base learning algorithms, such as combining decision trees with neural networks or support vector machines. This heterogeneous approach, often termed hybrid ensembles, aims to capitalize on the complementary strengths of each learner type, thereby improving predictive accuracy and robustness across complex, high-dimensional data domains.[6]

Evaluating the prediction of an ensemble typically requires more computation than evaluating the prediction of a single model. In one sense, ensemble learning may be thought of as a way to compensate for poor learning algorithms by performing a lot of extra computation. On the other hand, the alternative is to do a lot more learning with one non-ensemble model. An ensemble may be more efficient at improving overall accuracy for the same increase in compute, storage, or communication resources by using that increase on two or more methods, than would have been improved by increasing resource use for a single method. Fast algorithms such asdecision treesare commonly used in ensemble methods (e.g., random forests), although slower algorithms can benefit from ensemble techniques as well.

By analogy, ensemble techniques have been used also inunsupervised learningscenarios, for example inconsensus clusteringor inanomaly detection.

Empirically, ensembles tend to yield better results when there is a significant diversity among

the models.[7][8]Many ensemble methods, therefore, seek to promote diversity among the models they combine.[9][10]Although perhaps non-intuitive, more random algorithms (like random decision trees) can be used to produce a stronger ensemble than very deliberate algorithms (like entropy-reducing decision trees).[11]Using a variety of strong learning algorithms, however, has been shown to be more effective than using techniques that attempt todumb-downthe models in order to promote diversity.[12]It is possible to increase diversity in the training stage of the model using correlation for regression tasks[13]or using information measures such as cross entropy for classification tasks.[14]

Theoretically, one can justify the diversity concept because the lower bound of the error rate of an ensemble system can be decomposed into accuracy, diversity, and the other term.[15]

Ensemble learning, including both regression and classification tasks, can be explained using a geometric framework.[16]Within this framework, the output of each individual classifier or regressor for the entire dataset can be viewed as a point in a multi-dimensional space. Additionally, the target result is also represented as a point in this space, referred to as the "ideal point."

Clustering

From Scikit-learn

Clustering

of

unlabeled data can be performed with the module

sklearn.cluster

.

Each clustering algorithm comes in two variants: a class, that implements

the

fit

method to learn the clusters on train data, and a function,

that, given train data, returns an array of integer labels corresponding to the different clusters. For the class, the labels over the training data can be found in the labels_ attribute.

Input data

One important thing to note is that the algorithms implemented in this module can take different kinds of matrix as input. All the methods accept standard data matrices of shape (n_samples, n_features) .

These can be obtained from the classes in the sklearn.feature_extraction module. For AffinityPropagation , SpectralClustering and DBSCAN one can also input similarity matrices of shape (n_samples, n_samples) . These can be obtained from the functions in the sklearn.metrics.pairwise module.

Fron

Cluster analysisorclusteringis the task of grouping a set of objects in such a way that objects in the same group (called acluster) are moresimilar(in some specific sense defined by the analyst) to each other than to those in other groups (clusters). It is a main task ofexploratory data analysis, and a common technique forstatisticaldata analysis, used in many fields, includingpattern recognition,image analysis,information retrieval,bioinformatics,data compression,computer graphicsandmachine learning.

Cluster analysis refers to a family of algorithms and tasks rather than one specificalgorithm. It can be achieved by various algorithms that differ significantly in their understanding of what constitutes a cluster and how to efficiently find them. Popular notions of clusters include groups with smalldistancesbetween cluster members, dense areas of the data space, intervals or particularstatistical distributions. Clustering can therefore be formulated as amulti-objective optimizationproblem. The appropriate clustering algorithm and parameter settings (including parameters such as thedistance functionto use, a density threshold or the number of expected clusters) depend on the individualdata setand intended use of the results. Cluster analysis as such is not an automatic task, but an iterative process ofknowledge discoveryor interactive multi-objective optimization that involves trial and failure. It is often necessary to modifydata preprocessingand model parameters until the result achieves the desired properties.

Besides the termclustering, there are a number of terms with similar meanings, includingautomaticclassification,numerical taxonomy,botryology(fromGreek:βότρυς'grape'),typological analysis, andcommunity detection. The subtle differences are often in the use of the results: while in data mining, the resulting groups are the matter of interest, in automatic classification the resulting discriminative power is of interest.

Cluster analysis originated in anthropology by Driver and Kroeber in 1932[1]and introduced to psychology byJoseph Zubinin 1938[2]andRobert Tryonin 1939[3]and famously used byCattellbeginning in 1943[4]for trait theory classification inpersonality psychology.

The notion of a "cluster" cannot be precisely defined, which is one of the reasons why there are so many clustering algorithms.[5]There is a common denominator: a group of data objects. However, different researchers employ different cluster models, and for each of these cluster models again different algorithms can be given. The notion of a cluster, as found by different algorithms, varies significantly in its properties. Understanding these "cluster models" is key to understanding the differences between the various algorithms. Typical cluster models include:

A "clustering" is essentially a set of such clusters, usually containing all objects in the data set. Additionally, it may specify the relationship of the clusters to each other, for example, a hierarchy of clusters embedded in each other. Clusterings can be roughly distinguished as:

There are also finer distinctions possible, for example:

As listed above, clustering algorithms can be categorized based on their cluster model. The following overview will only list the most prominent examples of clustering algorithms, as there are possibly over 100 published clustering algorithms. Not all provide models for their clusters and can thus not easily be categorized. An overview of algorithms explained in Wikipedia can be found in thelist of statistics algorithms.

There is no objectively "correct" clustering algorithm, but as it was noted, "clustering is in the eye of the beholder."[5]In fact, an axiomatic approach to clustering demonstrates that it is impossible for any clustering method to meet three fundamental properties simultaneously:scale invariance(results remain unchanged under proportional scaling of distances),richness(all possible partitions of the data can be achieved), andconsistencybetween distances and the clustering structure.[7]The most appropriate clustering algorithm for a particular problem often needs to be chosen experimentally, unless there is a mathematical reason to prefer one cluster model over another. An algorithm that is designed for one kind of model will generally fail on a data set that contains a radically different kind of model.[5]For example, k-means cannot find non-convex clusters.[5]Most traditional clustering methods assume the clusters exhibit a spherical, elliptical or convex shape.[8]

Connectivity-based clustering, also known ashierarchical clustering, is based on the core idea of

objects being more related to nearby objects than to objects farther away. These algorithms connect "objects" to form "clusters" based on their distance. A cluster can be described largely by the maximum distance needed to connect parts of the cluster. At different distances, different clusters will form, which can be represented using adendrogram, which explains where the common name "hierarchical clustering" comes from: these algorithms do not provide a single partitioning of the data set, but instead provide an extensive hierarchy of clusters that merge with each other at certain distances. In a dendrogram, the y-axis marks the distance at which the clusters merge, while the objects are placed along the x-axis such that the clusters don't mix.

PCA

From Scikit-learn

Principal component analysis (PCA).

Linear dimensionality reduction using Singular Value Decomposition of the

data to project it to a lower dimensional space. The input data is centered

but not scaled for each feature before applying the SVD.

It uses the LAPACK implementation of the full SVD or a randomized truncated

SVD by the method of Halko et al. 2009, depending on the shape of the input

data and the number of components to extract.

With sparse inputs, the ARPACK implementation of the truncated SVD can be

used (i.e. through

scipy.sparse.linalg.svds

). Alternatively, one

may consider

TruncatedSVD

where the data are not centered.

Notice that this class only supports sparse inputs for some solvers such as

“arpack” and “covariance_eigh”. See

TruncatedSVD

for an

alternative with sparse data.

For a usage example, see

Principal Component Analysis (PCA) on Iris Dataset

Read more in the

User Guide

.

Number of components to keep.

if n_components is not set all components are kept:

n_components

==

min

(

n_samples

,

n_features

)

n_components

==

min

(

n_samples

,

n_features

)

n_components

==

min

(

n_samples

,

n_features

)

If

n_components

==

'mle'

and

svd_solver

==

'full'

, Minka’s

MLE is used to guess the dimension. Use of

n_components

==

'mle'

will interpret

svd_solver

==

'auto'

as

svd_solver

==

'full'

.

If

0

<

n_components

<

1

and

svd_solver

==

'full'

, select the

number of components such that the amount of variance that needs to be

explained is greater than the percentage specified by n_components.

If

svd_solver

==

'arpack'

, the number of components must be

strictly less than the minimum of n_features and n_samples.

Hence, the None case results in:

n_components

==

min

(

n_samples

,

n_features

)

-

1

n_components

==

min

(

n_samples

,

n_features

)

-

1

n_components

==

min

(

n_samples

,

n_features

)

-

1

If False, data passed to fit are overwritten and running

fit(X).transform(X) will not yield the expected results,

use fit_transform(X) instead.

When True (False by default) the

components_

vectors are multiplied

by the square root of n_samples and then divided by the singular values

to ensure uncorrelated outputs with unit component-wise variances.

Whitening will remove some information from the transformed signal

(the relative variance scales of the components) but can sometime

improve the predictive accuracy of the downstream estimators by

making their data respect some hard-wired assumptions.

The solver is selected by a default ‘auto’ policy is based on

X.shape

and

n_components

: if the input data has fewer than 1000 features and

more than 10 times as many samples, then the “covariance_eigh”

solver is used. Otherwise, if the input data is larger than 500x500

and the number of components to extract is lower than 80% of the

smallest dimension of the data, then the more efficient

“randomized” method is selected. Otherwise the exact “full” SVD is

computed and optionally truncated afterwards.

Run exact full SVD calling the standard LAPACK solver via

scipy.linalg.svd

and select the components by postprocessing

Precompute the covariance matrix (on centered data), run a

classical eigenvalue decomposition on the covariance matrix

typically using LAPACK and select the components by postprocessing.

This solver is very efficient for n_samples >> n_features and small

n_features. It is, however, not tractable otherwise for large

n_features (large memory footprint required to materialize the

covariance matrix). Also note that compared to the “full” solver,

this solver effectively doubles the condition number and is

therefore less numerical stable (e.g. on input data with a large

range of singular values).

Run SVD truncated to

n_components

calling ARPACK solver via

scipy.sparse.linalg.svds

. It requires strictly

0

<

n_components

<

min(X.shape)

Run randomized SVD by the method of Halko et al.

Added in version 0.18.0.

Added in version 0.18.0.

Changed in version 1.5:

Added the ‘covariance_eigh’ solver.

Changed in version 1.5:

Added the ‘covariance_eigh’ solver.

Tolerance for singular values computed by svd_solver == ‘arpack’.

Must be of range [0.0, infinity).

Added in version 0.18.0.

Added in version 0.18.0.

Number of iterations for the power method computed by

svd_solver == ‘randomized’.

Must be of range [0, infinity).

Added in version 0.18.0.

Added in version 0.18.0.

This parameter is only relevant when

svd_solver="randomized"

.

It corresponds to the additional number of random vectors to sample the

range of

X

so as to ensure proper conditioning. See

randomized_svd

for more details.

Added in version 1.1.

Added in version 1.1.

Power iteration normalizer for randomized SVD solver.

Not used by ARPACK. See

randomized_svd

for more details.

Added in version 1.1.

Added in version 1.1.

Used when the ‘arpack’ or ‘randomized’ solvers are used. Pass an int

for reproducible results across multiple function calls.

See

Glossary

.

Added in version 0.18.0.

Added in version 0.18.0.

Principal axes in feature space, representing the directions of

maximum variance in the data. Equivalently, the right singular

vectors of the centered input data, parallel to its eigenvectors.

The components are sorted by decreasing

explained_variance_

.

The amount of variance explained by each of the selected components.

The variance estimation uses

n_samples

-

1

degrees of freedom.

Equal to n_components largest eigenvalues

of the covariance matrix of X.

Added in version 0.18.

Added in version 0.18.

Percentage of variance explained by each of the selected components.

If

n_components

is not set then all components are stored and the

sum of the ratios is equal to 1.0.

The singular values corresponding to each of the selected components.

The singular values are equal to the 2-norms of the

n_components

variables in the lower-dimensional space.

Added in version 0.19.

Added in version 0.19.

Per-feature empirical mean, estimated from the training set.

Equal to

X.mean(axis=0)

.

The estimated number of components. When n_components is set

to ‘mle’ or a number between 0 and 1 (with svd_solver == ‘full’) this

number is estimated from input data. Otherwise it equals the parameter

n_components, or the lesser value of n_features and n_samples

if n_components is None.

Number of samples in the training data.

The estimated noise covariance following the Probabilistic PCA model

from Tipping and Bishop 1999. See “Pattern Recognition and

Machine Learning” by C. Bishop, 12.2.1 p. 574 or

http://www.miketipping.com/papers/met-mppca.pdf

. It is required to

compute the estimated data covariance and score samples.

Equal to the average of (min(n_features, n_samples) - n_components)

smallest eigenvalues of the covariance matrix of X.

Number of features seen during

fit

.

Added in version 0.24.

Added in version 0.24.

Names of features seen during

fit

. Defined only when

X

has feature names that are all strings.

Added in version 1.0.

Added in version 1.0.

See also

KernelPCA

Kernel Principal Component Analysis.

SparsePCA

Sparse Principal Component Analysis.

TruncatedSVD

Dimensionality reduction using truncated SVD.

IncrementalPCA

Incremental Principal Component Analysis.

See also

Kernel Principal Component Analysis.

Sparse Principal Component Analysis.

Dimensionality reduction using truncated SVD.

Incremental Principal Component Analysis.

References

For n_components == ‘mle’, this class uses the method from:

Minka, T. P.. “Automatic choice of dimensionality for PCA”.

In NIPS, pp. 598-604

Implements the probabilistic PCA model from:

Tipping, M. E., and Bishop, C. M. (1999). “Probabilistic principal

component analysis”. Journal of the Royal Statistical Society:

Series B (Statistical Methodology), 61(3), 611-622.

via the score and score_samples methods.

For svd_solver == ‘arpack’, refer to

scipy.sparse.linalg.svds

.

For svd_solver == ‘randomized’, see:

Halko, N., Martinsson, P. G., and Tropp, J. A. (2011).

“Finding structure with randomness: Probabilistic algorithms for

constructing approximate matrix decompositions”.

SIAM review, 53(2), 217-288.

and also

Martinsson, P. G., Rokhlin, V., and Tygert, M. (2011).

“A randomized algorithm for the decomposition of matrices”.

Applied and Computational Harmonic Analysis, 30(1), 47-68.

Examples

>>>

import

numpy

as

np

>>>

from

sklearn.decomposition

import

PCA

>>>

X

=

np

.

array

([[

-

1

,

-

1

],

[

-

2

,

-

1

],

[

-

3

,

-

2

],

[

1

,

1

],

[

2

,

1

],

[

3

,

2

]])

>>>

pca

=

PCA

(

n_components

=

2

)

>>>

pca

.

fit

(

X

)

PCA(n_components=2)

>>>

print

(

pca

.

explained_variance_ratio_

)

[0.9924... 0.0075...]

>>>

print

(

pca

.

singular_values_

)

[6.30061... 0.54980...]

>>>

import

numpy

as

np

>>>

from

sklearn.decomposition

import

PCA

>>>

X

=

np

.

array

([[

-

1

,

-

1

],

[

-

2

,

-

1

],

[

-

3

,

-

2

],

[

1

,

1

],

[

2

,

1

],

[

3

,

2

]])

>>>

pca

=

PCA

(

n_components

=

2

)

>>>

pca

.

fit

(

X

)

PCA(n_components=2)

>>>

print

(

pca

.

explained_variance_ratio_

)

[0.9924... 0.0075...]

>>>

print

(

pca

.

singular_values_

)

[6.30061... 0.54980...]

>>>

import

numpy

as

np

>>>

from

sklearn.decomposition

import

PCA

>>>

X

=

np

.

array

([[

-

1

,

-

1

],

[

-

2

,

-

1

],

[

-

3

,

-

2

],

[

1

,

1

],

[

2

,

1

],

[

3

,

2

]])

>>>

pca

=

PCA

(

n_components

=

2

)

>>>

pca

.

fit

(

X

)

PCA(n_components=2)

>>>

print

(

pca

.

explained_variance_ratio_

)

[0.9924... 0.0075...]

>>>

print

(

pca

.

singular_values_

)

[6.30061... 0.54980...]

>>>

pca

=

PCA

(

n_components

=

2

,

svd_solver

=

'full'

)

>>>

pca

.

fit

(

X

)

PCA(n_components=2, svd_solver='full')

>>>

print

(

pca

.

explained_variance_ratio_

)

[0.9924... 0.00755...]

>>>

print

(

pca

.

singular_values_

)

[6.30061... 0.54980...]

>>>

pca

=

PCA

(

n_components

=

2

,

svd_solver

=

'full'

)

>>>

pca

.

fit

(

X

)

PCA(n_components=2, svd_solver='full')

>>>

print

(

pca

.

explained_variance_ratio_

)

[0.9924... 0.00755...]

>>>

print

(

pca

.

singular_values_

)

[6.30061... 0.54980...]

>>>

pca

=

PCA

(

n_components

=

2

,

svd_solver

=

'full'

)

>>>

pca

.

fit

(

X

)

PCA(n_components=2, svd_solver='full')

>>>

print

(

pca

.

explained_variance_ratio_

)

[0.9924... 0.00755...]

>>>

print

(

pca

.

singular_values_

)

[6.30061... 0.54980...]

>>>

pca

=

PCA

(

n_components

=

1

,

svd_solver

=

'arpack'

)

>>>

pca

.

fit

(

X

)

PCA(n_components=1, svd_solver='arpack')

>>>

print

(

pca

.

explained_variance_ratio_

)

[0.99244...]

>>>

print

(

pca

.

singular_values_

)

[6.30061...]

>>>

pca

=

PCA

(

n_components

=

1

,

svd_solver

=

'arpack'

)

>>>

pca

.

fit

(

X

)

PCA(n_components=1, svd_solver='arpack')

>>>

print

(

pca

.

explained_variance_ratio_

)

[0.99244...]

>>>

print

(

pca

.

singular_values_

)

[6.30061...]

>>>

pca

=

PCA

(

n_components

=

1

,

svd_solver

=

'arpack'

)

>>>

pca

.

fit

(

X

)

PCA(n_components=1, svd_solver='arpack')

>>>

print

(

pca

.

explained_variance_ratio_

)

[0.99244...]

>>>

print

(

pca

.

singular_values_

)

[6.30061...]

Fit the model with X.

Training data, where

n_samples

is the number of samples

and

n_features

is the number of features.

Ignored.

Returns the instance itself.

Fit the model with X and apply the dimensionality reduction on X.

Training data, where

n_samples

is the number of samples

and

n_features

is the number of features.

Ignored.

Transformed values.

Notes

This method returns a Fortran-ordered array. To convert it to a

C-ordered array, use ‘np.ascontiguousarray’.

Compute data covariance with the generative model.

cov

=

components_.T

*

S**2

*

components_

+

sigma2

*

eye(n_features)

where S**2 contains the explained variances, and sigma2 contains the

noise variances.

Estimated covariance of data.

Get output feature names for transformation.

The feature names out will prefixed by the lowercased class name. For

example, if the transformer outputs 3 features, then the feature names

out are:

["class_name0",

"class_name1",

"class_name2"]

.

Only used to validate feature names with the names seen in

fit

.

Transformed feature names.

Get metadata routing of this object.

Please check

User Guide

on how the routing

mechanism works.

A

MetadataRequest

encapsulating

routing information.

Get parameters for this estimator.

If True, will return the parameters for this estimator and

contained subobjects that are estimators.

Parameter names mapped to their values.

Compute data precision matrix with the generative model.

Equals the inverse of the covariance but computed with

the matrix inversion lemma for efficiency.

Estimated precision of data.

Transform data back to its original space.

In other words, return an input

X_original

whose transform would be X.

New data, where

n_samples

is the number of samples

and

n_components

is the number of components.

Original data, where

n_samples

is the number of samples

and

n_features

is the number of features.

Notes

If whitening is enabled, inverse_transform will compute the

exact inverse operation, which includes reversing whitening.

Return the average log-likelihood of all samples.

See. “Pattern Recognition and Machine Learning”

by C. Bishop, 12.2.1 p. 574

or

http://www.miketipping.com/papers/met-mppca.pdf

The data.

Ignored.

Average log-likelihood of the samples under the current model.

Return the log-likelihood of each sample.

See. “Pattern Recognition and Machine Learning”

by C. Bishop, 12.2.1 p. 574

or

http://www.miketipping.com/papers/met-mppca.pdf

The data.

Log-likelihood of each sample under the current model.

Set output container.

See

Introducing the set_output API

for an example on how to use the API.

Configure output of

transform

and

fit_transform

.

"default"

: Default output format of a transformer

"pandas"

: DataFrame output

"polars"

: Polars output

None

: Transform configuration is unchanged

"default"

: Default output format of a transformer

"pandas"

: DataFrame output

"polars"

: Polars output

None

: Transform configuration is unchanged

Added in version 1.4:

"polars"

option was added.

Added in version 1.4:

"polars"

option was added.

Estimator instance.

Set the parameters of this estimator.

The method works on simple estimators as well as on nested objects

(such as

Pipeline

). The latter have

parameters of the form

<component>__<parameter>

so that it’s

possible to update each component of a nested object.

Estimator parameters.

Estimator instance.

Apply dimensionality reduction to X.

X is projected on the first principal components previously extracted

from a training set.

New data, where

n_samples

is the number of samples

and

n_features

is the number of features.

Projection of X in the first principal components, where

n_samples

is the number of samples and

n_components

is the number of the components.

From

Principal component analysis(PCA) is alineardimensionality reductiontechnique with applications inexploratory data analysis, visualization anddata preprocessing.

The data islinearly transformedonto a newcoordinate systemsuch that the directions (principal components) capturing the largest variation in the data can be easily identified.

Theprincipal componentsof a collection of points in areal coordinate spaceare a sequence ofp{\displaystyle p}unit vectors, where thei{\displaystyle i}-th vector is the direction of a line that best fits the data while beingorthogonalto the firsti−1{\displaystyle i-1}vectors. Here, a best-fitting line is defined as one that minimizes the average squaredperpendiculardistance from the points to the line. These directions (i.e., principal components) constitute anorthonormal basisin which different individual dimensions of the data arelinearly uncorrelated. Many studies use the first two principal components in order to plot the data in two dimensions and to visually identify clusters of closely related data points.[1]

Principal component analysis has applications in many fields such aspopulation genetics,microbiomestudies, andatmospheric science.[2]

When performing PCA, the first principal component of a set ofp{\displaystyle p}variables is the derived variable formed as a linear combination of the original variables that explains the most variance. The second principal component explains the most variance in what is left once the effect of the first component is removed, and we may proceed throughp{\displaystyle p}iterations until all the variance is explained. PCA is most commonly used when many of the variables are highly correlated with each other and it is desirable to reduce their number to anindependent set.

The first principal component can equivalently be defined as a direction that maximizes the variance of the projected data. Thei{\displaystyle i}-th principal component can be taken as a direction orthogonal to the firsti−1{\displaystyle i-1}principal components that maximizes the variance of the projected data.

For either objective, it can be shown that the principal components areeigenvectorsof the data'scovariance matrix. Thus, the principal components are often computed byeigendecompositionof the data covariance matrix orsingular value decompositionof the data matrix. PCA is the simplest of the true eigenvector-based multivariate analyses and is closely related tofactor analysis. Factor analysis typically incorporates more domain-specific assumptions about the underlying structure and solves eigenvectors of a slightly different matrix. PCA is also related tocanonical correlation analysis (CCA). CCA defines coordinate systems that optimally describe thecross-covariancebetween two datasets while PCA defines a neworthogonal coordinate systemthat optimally describes variance in a single dataset.[3][4][5][6]RobustandL1-norm-based variants of standard PCA have also been proposed.[7][8][9][6]

PCA was invented in 1901 byKarl Pearson,[10]as an analogue of theprincipal axis theoremin mechanics; it was later independently developed and named byHarold Hotellingin the 1930s.[11]Depending on the field of application, it is also named the discreteKarhunen–Loèvetransform (KLT) insignal processing, theHotellingtransform in multivariate quality control,proper orthogonal decomposition(POD) in mechanical engineering,singular value decomposition(SVD) ofX(invented in the last quarter of the 19th century[12]),eigenvalue decomposition(EVD) ofXTXin linear algebra,factor analysis(for a discussion of the differences between PCA and factor analysis see Ch. 7 of Jolliffe'sPrincipal Component Analysis),[13]Eckart–Young theorem(Harman, 1960), orempirical orthogonal functions(EOF) in meteorological science (Lorenz, 1956), empirical eigenfunction decomposition (Sirovich, 1987), quasiharmonic modes (Brooks et al., 1988),spectral decompositionin noise and vibration, andempirical modal analysisin structural dynamics.

PCA can be thought of as fitting ap-dimensionalellipsoidto the data, where each axis of the ellipsoid represents a principal component. If some axis of the ellipsoid is small, then the variance along that axis is also small.

To find the axes of the ellipsoid, we must first center the values of each variable in the dataset

on 0 by subtracting the mean of the variable's observed values from each of those values. These transformed values are used instead of the original observed values for each of the variables. Then, we compute thecovariance matrixof the data and calculate the eigenvalues and corresponding eigenvectors of this covariance matrix. Then we mustnormalizeeach of the orthogonal eigenvectors to turn them into unit vectors. Once this is done, each of the mutually-orthogonal unit eigenvectors can be interpreted as an axis of the ellipsoid fitted to the data. This choice of basis will transform the covariance matrix into a diagonalized form, in which the diagonal elements represent the variance of each axis. The proportion of the variance that each eigenvector represents can be calculated by dividing the eigenvalue corresponding to that eigenvector by the sum of all eigenvalues.

Biplotsandscree plots(degree ofexplained variance) are used to interpret findings of the PCA.
