[1 Naive Bayes 3](#_Toc497989980)

[1.1 The overview of Naïve Bayes class library in scikit-learn 3](#_Toc497989981)

[1.2 Gaussian Naive Bayes 3](#_Toc497989982)

[1.3 Multinomial Naive Bayes 3](#_Toc497989983)

[1.4 Bernoulli Naive Bayes 4](#_Toc497989984)

[2 Trees 5](#_Toc497989985)

[2.1. Decision Stump 5](#_Toc497989986)

[2.2. LMT: logistic model trees 6](#_Toc497989987)

[2.3. J48 6](#_Toc497989988)

[2.4. M5P 7](#_Toc497989989)

[2.5. RandomForest 7](#_Toc497989990)

[2.6. RandomTree 8](#_Toc497989991)

[2.7. REPTree 8](#_Toc497989992)

[3. Rules: 8](#_Toc497989993)

[3.1 JRip 8](#_Toc497989994)

[3.2 DecisionTable 8](#_Toc497989995)

[3.3 M5Rules 8](#_Toc497989996)

[3.4 OneR 8](#_Toc497989997)

[3.5 Part: obtains rules from partial decision trees 8](#_Toc497989998)

[4. Functions: 9](#_Toc497989999)

[4.1. LinearRegression 9](#_Toc497990000)

[4.2. Logistic 9](#_Toc497990001)

[4.3. SGD: 10](#_Toc497990002)

[4.4. SimpleLinearRegression 11](#_Toc497990003)

[4.5. SimpleLogistic: generates a degenerate logistic model tree 11](#_Toc497990004)

[4.6. SMO: implements the sequential minimal optimization algorithm for training a support vector classifier 11](#_Toc497990005)

[4.7. SMOreg: implements the sequential minimal optimization algorithm for learning a support vector regression model 11](#_Toc497990006)

[5.0 k-NN based: 11](#_Toc497990007)

[5.1. IBk: k nearest neighbor classifier 11](#_Toc497990008)

[5.2. KStar: is a nearest-neighbor method with generalized distance function based on transformations 12](#_Toc497990009)

[6. Neural Networks: 12](#_Toc497990010)

[6.1. MultilayerPerceptron 12](#_Toc497990011)

[7. Ensemble Methods 12](#_Toc497990012)

[7.1. Stacking 2 12](#_Toc497990013)

[7.2. Vote 2 12](#_Toc497990014)

[7.3. LWL 5 12](#_Toc497990015)

[7.4. AdaBoostM1 6 12](#_Toc497990016)

[7.5. AdditiveRegression 4 15](#_Toc497990017)

[7.6. AttributeSelectedClassifier 2 15](#_Toc497990018)

[7.7. Bagging 4 15](#_Toc497990019)

[7.8. RandomCommittee 2 16](#_Toc497990020)

[7.9. RandomSubSpace 3 16](#_Toc497990021)

[8. Attribute Selection Methods 16](#_Toc497990022)

[8.1. BestFirst 2 16](#_Toc497990023)

[8.2. GreedyStepwise 4 16](#_Toc497990024)

#### Naive Bayes

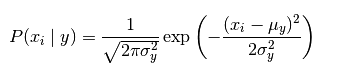
##### The overview of Naïve Bayes class library in scikit-learn

Naive Bayes is a relatively simple algorithm, you can use it simply in scikt-learn , There are three Naive Bayesian classification lgorithms:GaussianNB, MultinomialNB and BernoulliNB. GaussianNB is a naive Bayes that is a priori Gaussian distribution, MultinomialNB is a naive Bayes that is prior known as a polynomial distribution, and BernoulliNB is a naive Bayes that is priori known as Bernoulli distribution.

The classification of these three categories for different scenarios, in general, if the distribution of the sample features are mostly continuous values, the use of GaussianNB. MultinomialNB is a good fit when the majority of sample features are multivariate and discrete. If the sample features are binary or sparse multivariate discrete values, BernoulliNB should be used.

##### Gaussian Naive Bayes

GaussianNB implements the Gaussian Naive Bayes algorithm for classification. The likelihood of the features is assumed to be Gaussian:



The parameters and   are estimated using maximum likelihood.

1. **import** numpy as np
2. X = np.array([[-1, -1], [-2, -1], [-3, -2], [1, 1], [2, 1], [3, 2]])
3. Y = np.array([1, 1, 1, 2, 2, 2])
4. **from** sklearn.naive\_bayes **import** GaussianNB
5. clf = GaussianNB().fit(X, Y)
6. **print** clf.predict([[-0.8,-1]])
7. clf\_pf = GaussianNB().partial\_fit(X, Y, np.unique(Y))
8. **print** clf\_pf.predict([[-0.8,-1]])

result:

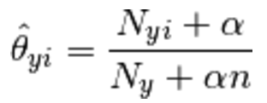


In addition, GaussianNB has an important function, partial\_fit method, this method is generally used when the training data is very large, it can not be fully loaded into memory at a time. At this time we can divide the data into a number of points, then repeat call partial\_fit step by step, MultinomialNB and BernoulliNB mentioned later also have similar functions.

##### Multinomial Naive Bayes

[MultinomialNB](http://scikit-learn.org/stable/modules/generated/sklearn.naive_bayes.MultinomialNB.html#sklearn.naive_bayes.MultinomialNB) implements the naive Bayes algorithm for multinomially distributed data, and is one of the two classic naive Bayes variants used in text classification (where the data are typically represented as word vector counts, although tf-idf vectors are also known to work well in practice). The distribution is parametrized by vectors  for each class , where  is the number of features (in text classification, the size of the vocabulary) and  is the probability  of feature   appearing in a sample belonging to class .

The parameters  is estimated by a smoothed version of maximum likelihood, i.e. relative frequency counting:



1. **import** numpy as np
2. X = np.random.randint(5, size=(6, 100))
3. y = np.array([1, 2, 3, 4, 5, 6])
4. **from** sklearn.naive\_bayes **import** MultinomialNB
5. clf = MultinomialNB().fit(X, y)
6. **print** clf.predict(X[2:3])

result : [3]

##### Bernoulli Naive Bayes

BernoulliNB implements a naive Bayesian training and classification algorithm that is based on the distribution of multivariate Bernoulli distributions; for example, there may be multiple features, but each is assumed to be a binary value (Bernoulli, Boolean) variable. Therefore, samples of such requirements are represented as eigenvectors of binary values; a bernoullinb instance can be entered if given to any other type of data (depending on the binarization parameter)

The decision rule for Bernoulli naive Bayes is based on:



In the case of text categorization, the word appearance vector (rather than the word count vector) can be used to train and use the classification. bernoullinb might perform better on some datasets, especially those short ones. If time permits, two models are recommended for evaluation.

1. **import** numpy as np
2. X = np.random.randint(2, size=(6, 100))
3. Y = np.array([1, 2, 3, 4, 4, 5])
4. **from** sklearn.naive\_bayes **import** BernoulliNB
5. clf = BernoulliNB()
6. clf.fit(X, Y)
7. BernoulliNB(alpha=1.0, binarize=0.0, class\_prior=None, fit\_prior=True)
8. **print**(clf.predict(X[2:3]))

result : [3]

more about Naïve Bayes please refer to <http://scikit-learn.org/stable/modules/naive_bayes.html#gaussian-naive-bayes>

#### Trees

##### 2.1. Decision Stump

Decision Stump is an extremely simple but very effective machine learning algorithm. It is in the mechanism and decision tree algorithm is very close, but the structure is simple and many, like the shape of only two branches of the tree

For example, we can determine whether a person's sex is determined by the two characteristics of beard and height, and it is clear whether a beard may be more accurate in determining sex than height, so we will give this feature more Big weight, for example, we set the weight to 0.8: 0.2. This is more accurate than the 0.5: 0.5 weight

Then we build the decision tree. Our decision tree to achieve the main two functions, one is to find out the results of the greatest impact on the eigenvalues. Another function is to find this feature worth the threshold. For example, the threshold is d, when the eigenvalue is greater than d the result is 1, when the eigenvalue is less than d result is 0.

Build decision tree

First look at the data set, is a matrix of two eigenvalues.

def loadSimpData():

datMat = matrix([[ 1. , 2.1], [ 2. , 1.1], [ 1.3, 1. ], [ 1. , 1. ], [ 2. , 1. ]])

classLabels = [1.0, 1.0, -1.0, -1.0, 1.0]

return datMat,classLabels

Followed by the tree's classification function. This function is used in the following loop, the effect is very simple, that is, compared to the eigenvalues and objective functions of each column, return the results of the comparison. The four parameters are (input matrix, column, threshold, lt or gt)

def stumpClassify(dataMatrix,dimen,threshVal,threshIneq):#just classify the data

retArray = ones((shape(dataMatrix)[0],1))

if threshIneq == 'lt':

retArray[dataMatrix[:,dimen] <= threshVal] = -1.0

else:

retArray[dataMatrix[:,dimen] > threshVal] = -1.0

return retArray

Finally, the binary tree function is constructed, and the optimal eigenvalue and its threshold are obtained by cyclic comparison. D is the weight of the initial matrix.

def buildStump(dataArr,classLabels,D):

dataMatrix = mat(dataArr); labelMat = mat(classLabels).T

m,n = shape(dataMatrix)

numSteps = 10.0; bestStump = {}; bestClasEst = mat(zeros((m,1)))

minError = inf #init error sum, to +infinity

for i in range(n):#loop over all dimensions

rangeMin = dataMatrix[:,i].min(); rangeMax = dataMatrix[:,i].max();

stepSize = (rangeMax-rangeMin)/numSteps

for j in range(-1,int(numSteps)+1):#loop over all range in current dimension

for inequal in ['lt', 'gt']: #go over less than and greater than

threshVal = (rangeMin + float(j) \* stepSize)

predictedVals = stumpClassify(dataMatrix,i,threshVal,inequal)#call stump classify with i, j, lessThan

errArr = mat(ones((m,1)))

errArr[predictedVals == labelMat] = 0

weightedError = D.T\*errArr #calc total error multiplied by D

#print "split: dim %d, thresh %.2f, thresh ineqal: %s, the weighted error is %.3f" % (i, threshVal, inequal, weightedError)

if weightedError < minError:

minError = weightedError

bestClasEst = predictedVals.copy()

bestStump['dim'] = i

bestStump['thresh'] = threshVal

bestStump['ineq'] = inequal

return bestStump,minError,bestClasEst

result

When we assume that the initial weights are the same (5 rows of data are both 0.2), get the result：{'dim': 0, 'ineq': 'lt', 'thresh': 1.3} - the first eigenvalue is the largest, the threshold is 1.3，[0.2]] - the error rate of 0.2, one of the five wrong

#### 2.2. LMT: logistic model trees

##### 2.3. J48

Tree algorithms: ID3, C4.5, C5.0 and CART，What are all the various decision tree algorithms and how do they differ from each other? Which one is implemented in scikit-learn?

ID3 (Iterative Dichotomiser 3) was developed in 1986 by Ross Quinlan. The algorithm creates a multiway tree, finding for each node (i.e. in a greedy manner) the categorical feature that will yield the largest information gain for categorical targets. Trees are grown to their maximum size and then a pruning step is usually applied to improve the ability of the tree to generalise to unseen data.

C4.5 is the successor to ID3 and removed the restriction that features must be categorical by dynamically defining a discrete attribute (based on numerical variables) that partitions the continuous attribute value into a discrete set of intervals. C4.5 converts the trained trees (i.e. the output of the ID3 algorithm) into sets of if-then rules. These accuracy of each rule is then evaluated to determine the order in which they should be applied. Pruning is done by removing a rule’s precondition if the accuracy of the rule improves without it.

C5.0 is Quinlan’s latest version release under a proprietary license. It uses less memory and builds smaller rulesets than C4.5 while being more accurate.

CART (Classification and Regression Trees) is very similar to C4.5, but it differs in that it supports numerical target variables (regression) and does not compute rule sets. CART constructs binary trees using the feature and threshold that yield the largest information gain at each node.

scikit-learn uses an optimised version of the CART algorithm.

*class sklearn.tree.DecisionTreeClassifier(criterion=’gini’, splitter=’best’, max\_depth=None, min\_samples\_split=2, min\_samples\_leaf=1, min\_weight\_fraction\_leaf=0.0, max\_features=None, random\_state=None, max\_leaf\_nodes=None, min\_impurity\_decrease=0.0, min\_impurity\_split=None, class\_weight=None, presort=False)*

The function to measure the quality of a split. Supported criteria are “gini” for the Gini impurity and “entropy” for the information gain.

Example：

>>> from sklearn import tree

>>> X = [[0, 0], [1, 1]]

>>> Y = [0, 1]

// The function to measure the quality of a split. Supported criteria are “gini” for the Gini impurity and “entropy” for the information gain.

>>> clf = tree.DecisionTreeClassifier()

>>> clf = clf.fit(X, Y)

>>> clf.predict([[2., 2.]])

array([1])

More details:

<http://scikit-learn.org/stable/modules/generated/sklearn.tree.DecisionTreeClassifier.html#examples-using-sklearn-tree-decisiontreeclassifier>

#### 2.4. M5P

##### 2.5. RandomForest

A random forest is a meta estimator that fits a n

umber of decision tree classifiers on various sub-samples of the dataset and use averaging to improve the predictive accuracy and control over-fitting. The sub-sample size is always the same as the original input sample size but the samples are drawn with replacement if bootstrap=True (default).

Class: RandomForestClassifier

Constructor：sklearn.ensemble.RandomForestClassifier(n\_estimators=10

, criterion='gini'

, max\_depth=None

, min\_samples\_split=2

, min\_samples\_leaf=1

, max\_features='auto'

, max\_leaf\_nodes=None

, bootstrap=True

, oob\_score=False

, n\_jobs=1

, random\_state=None

, verbose=0

, min\_density=None

, compute\_importances=None)

Examples：

>>> from sklearn.ensemble import RandomForestClassifier

>>> from sklearn.datasets import make\_classification

>>>

>>> X, y = make\_classification(n\_samples=1000, n\_features=4,

... n\_informative=2, n\_redundant=0,

... random\_state=0, shuffle=False)

>>> clf = RandomForestClassifier(max\_depth=2, random\_state=0)

>>> clf.fit(X, y)

RandomForestClassifier(bootstrap=True, class\_weight=None, criterion='gini',

max\_depth=2, max\_features='auto', max\_leaf\_nodes=None,

min\_impurity\_decrease=0.0, min\_impurity\_split=None,

min\_samples\_leaf=1, min\_samples\_split=2,

min\_weight\_fraction\_leaf=0.0, n\_estimators=10, n\_jobs=1,

oob\_score=False, random\_state=0, verbose=0, warm\_start=False)

>>> print(clf.feature\_importances\_)

[ 0.17287856 0.80608704 0.01884792 0.00218648]

>>> print(clf.predict([[0, 0, 0, 0]]))

[1]

More details:

<http://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestClassifier.html>

##### 2.6. RandomTree

An unsupervised transformation of a dataset to a high-dimensional sparse representation. A datapoint is coded according to which leaf of each tree it is sorted into. Using a one-hot encoding of the leaves, this leads to a binary coding with as many ones as there are trees in the forest.

class sklearn.ensemble.RandomTreesEmbedding(n\_estimators=10, max\_depth=5, min\_samples\_split=2, min\_samples\_leaf=1, min\_weight\_fraction\_leaf=0.0, max\_leaf\_nodes=None, min\_impurity\_decrease=0.0, min\_impurity\_split=None, sparse\_output=True, n\_jobs=1, random\_state=None, verbose=0, warm\_start=False)

More details:

<http://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestClassifier.html>

##### 2.7. REPTree

#### 3. Rules:

##### 3.1 JRip

##### 3.2 DecisionTable

##### 3.3 M5Rules

##### 3.4 OneR

##### 3.5 Part: obtains rules from partial decision trees

#### 4. Functions:

##### 4.1. LinearRegression

Class: LinearRegression

Constructor：sklearn.linear\_model.LinearRegression(fit\_intercept=True, normalize=False, copy\_X=True, n\_jobs=1)

Examples：

>>>from sklearn import linear\_model

>>>X= [[0, 0], [1, 1], [2, 2]]

>>>y = [0, 1, 2]

>>>clf = linear\_model.LinearRegression()

>>>clf.fit(X, y)

>>> print (clf.coef\_)

[ 0.5 0.5]

>>>print (clf.intercept\_)

1.11022302463e-16

>>>print (clf.predict([[3, 3]]))

[ 3.]

More details:

<http://scikit-learn.org/stable/modules/generated/sklearn.linear_model.LinearRegression.html>

<http://scikit-learn.org/stable/auto_examples/linear_model/plot_ols.html>

##### 4.2. Logistic

Class: LinearRegression

Constructor：sklearn.linear\_model.LogisticRegression(penalty=’l2’, dual=False, tol=0.0001, C=1.0, fit\_intercept=True, intercept\_scaling=1, class\_weight=None, random\_state=None, solver=’liblinear’, max\_iter=100, multi\_class=’ovr’, verbose=0, warm\_start=False, n\_jobs=1)

Examples：

clf\_l1\_LR = [LogisticRegression](http://scikit-learn.org/stable/modules/generated/sklearn.linear_model.LogisticRegression.html#sklearn.linear_model.LogisticRegression)(C=C, penalty='l1', tol=0.01)

clf\_l2\_LR = [LogisticRegression](http://scikit-learn.org/stable/modules/generated/sklearn.linear_model.LogisticRegression.html#sklearn.linear_model.LogisticRegression)(C=C, penalty='l2', tol=0.01)

clf\_l1\_LR.fit(X, y)

clf\_l2\_LR.fit(X, y)

More details:

<http://scikit-learn.org/stable/modules/generated/sklearn.linear_model.LogisticRegression.html>

##### 4.3. SGD:

Stochastic Gradient Descent (SGD) is a simple yet very efficient approach to discriminative learning of linear classifiers under convex loss functions such as (linear) Support Vector Machines and Logistic Regression. Even though SGD has been around in the machine learning community for a long time, it has received a considerable amount of attention just recently in the context of large-scale learning.SGD has been uccessfully applied to large-scale and sparse machine learning problems often encountered in text classification and natural language processing. Given that the data is sparse, the classifiers in this module easily scale to problems with more than 10^5 training examples and more than 10^5 features.

The advantages of Stochastic Gradient Descent are:

• Efficiency.

• Ease of implementation (lots of opportunities for code tuning).

The disadvantages of Stochastic Gradient Descent include:

• SGD requires a number of hyperparameters such as the regularization parameter and

the number of iterations.

• SGD is sensitive to feature scaling.

>>> from sklearn.linear\_model import SGDClassifier

>>> X = [[0., 0.], [1., 1.]]

>>> y = [0, 1]

>>> clf = SGDClassifier(loss="hinge", penalty="l2")

>>> clf.fit(X, y)

SGDClassifier(alpha=0.0001, average=False, class\_weight=None, epsilon=0.1,

eta0=0.0, fit\_intercept=True, l1\_ratio=0.15,

learning\_rate='optimal', loss='hinge', max\_iter=None, n\_iter=None,

n\_jobs=1, penalty='l2', power\_t=0.5, random\_state=None,

shuffle=True, tol=None, verbose=0, warm\_start=False)

>>> clf.predict([[2., 2.]])

array([1])

More details:

<http://scikit-learn.org/stable/modules/sgd.html>

##### 4.4. SimpleLinearRegression

Linear regression is one of the basic algorithms in data mining. In a sense, linear regression has begun to come into play when learning a function, except that no error term is involved. In fact, the idea of linear regression is to solve a set of equations and obtain the regression function. However, after the error term appears, the solution to the equation is changed, and the least squares method is generally used to calculate it.

##### 4.5. SimpleLogistic: generates a degenerate logistic model tree

##### 4.6. SMO: implements the sequential minimal optimization algorithm for training a support vector classifier

##### 4.7. SMOreg: implements the sequential minimal optimization algorithm for learning a support vector regression model

VotedPerceptron

#### 5.0 k-NN based:

##### 5.1. IBk: k nearest neighbor classifier

*class* sklearn.neighbors.KNeighborsClassifier(*n\_neighbors=5*, *weights=’uniform’*, *algorithm=’auto’*, *leaf\_size=30*, *p=2*, *metric=’minkowski’*, *metric\_params=None*, *n\_jobs=1*, *\*\*kwargs*)

Examples

>>> X = [[0], [1], [2], [3]]

>>> y = [0, 0, 1, 1]

>>> from sklearn.neighbors import KNeighborsClassifier

>>> neigh = KNeighborsClassifier(n\_neighbors=3)

>>> neigh.fit(X, y)

KNeighborsClassifier(...)

>>> print(neigh.predict([[1.1]]))

[0]

>>> print(neigh.predict\_proba([[0.9]]))

[[ 0.66666667 0.33333333]]

More details:

<http://scikit-learn.org/stable/modules/sgd.html>

##### 5.2. KStar: is a nearest-neighbor method with generalized distance function based on transformations

#### 6. Neural Networks:

##### 6.1. MultilayerPerceptron

Multi-layer Perceptron classifier.

This model optimizes the log-loss function using LBFGS or stochastic gradient descent.

*class* sklearn.neural\_network.MLPClassifier(*hidden\_layer\_sizes=(100*, *)*, *activation=’relu’*, *solver=’adam’*, *alpha=0.0001*, *batch\_size=’auto’*, *learning\_rate=’constant’*, *learning\_rate\_init=0.001*, *power\_t=0.5*, *max\_iter=200*, *shuffle=True*, *random\_state=None*, *tol=0.0001*, *verbose=False*, *warm\_start=False*, *momentum=0.9*, *nesterovs\_momentum=True*, *early\_stopping=False*, *validation\_fraction=0.1*, *beta\_1=0.9*, *beta\_2=0.999*, *epsilon=1e-08*)

More details:

<http://scikit-learn.org/stable/modules/generated/sklearn.neural_network.MLPClassifier.html>

#### 7. Ensemble Methods

##### 7.1. Stacking 2

##### 7.2. Vote 2

##### 7.3. LWL 5

##### 7.4. AdaBoostM1 6

Class:  AdaBoostClassifier

Constructor: sklearn.ensemble.AdaBoostClassifier(base\_estimator=None, n\_estimators=50, learning\_rate=1.0, algorithm=’SAMME.R’, random\_state=None)

Examples：

import numpy as np

import matplotlib.pyplot as plt

from sklearn import datasets

from sklearn.tree import [DecisionTreeClassifier](http://scikit-learn.org/stable/modules/generated/sklearn.tree.DecisionTreeClassifier.html#sklearn.tree.DecisionTreeClassifier)

from sklearn.metrics import [zero\_one\_loss](http://scikit-learn.org/stable/modules/generated/sklearn.metrics.zero_one_loss.html#sklearn.metrics.zero_one_loss)

from sklearn.ensemble import [AdaBoostClassifier](http://scikit-learn.org/stable/modules/generated/sklearn.ensemble.AdaBoostClassifier.html#sklearn.ensemble.AdaBoostClassifier)

n\_estimators = 400

# A learning rate of 1. may not be optimal for both SAMME and SAMME.R

learning\_rate = 1.

X, y = [datasets.make\_hastie\_10\_2](http://scikit-learn.org/stable/modules/generated/sklearn.datasets.make_hastie_10_2.html#sklearn.datasets.make_hastie_10_2)(n\_samples=12000, random\_state=1)

X\_test, y\_test = X[2000:], y[2000:]

X\_train, y\_train = X[:2000], y[:2000]

dt\_stump = [DecisionTreeClassifier](http://scikit-learn.org/stable/modules/generated/sklearn.tree.DecisionTreeClassifier.html#sklearn.tree.DecisionTreeClassifier)(max\_depth=1, min\_samples\_leaf=1)

dt\_stump.fit(X\_train, y\_train)

dt\_stump\_err = 1.0 - dt\_stump.score(X\_test, y\_test)

dt = [DecisionTreeClassifier](http://scikit-learn.org/stable/modules/generated/sklearn.tree.DecisionTreeClassifier.html#sklearn.tree.DecisionTreeClassifier)(max\_depth=9, min\_samples\_leaf=1)

dt.fit(X\_train, y\_train)

dt\_err = 1.0 - dt.score(X\_test, y\_test)

ada\_discrete = [AdaBoostClassifier](http://scikit-learn.org/stable/modules/generated/sklearn.ensemble.AdaBoostClassifier.html#sklearn.ensemble.AdaBoostClassifier)(

base\_estimator=dt\_stump,

learning\_rate=learning\_rate,

n\_estimators=n\_estimators,

algorithm="SAMME")

ada\_discrete.fit(X\_train, y\_train)

ada\_real = [AdaBoostClassifier](http://scikit-learn.org/stable/modules/generated/sklearn.ensemble.AdaBoostClassifier.html#sklearn.ensemble.AdaBoostClassifier)(

base\_estimator=dt\_stump,

learning\_rate=learning\_rate,

n\_estimators=n\_estimators,

algorithm="SAMME.R")

ada\_real.fit(X\_train, y\_train)

fig = [plt.figure](http://matplotlib.org/api/_as_gen/matplotlib.figure.AxesStack.html#matplotlib.figure)()

ax = fig.add\_subplot(111)

ax.plot([1, n\_estimators], [dt\_stump\_err] \* 2, 'k-',

label='Decision Stump Error')

ax.plot([1, n\_estimators], [dt\_err] \* 2, 'k--',

label='Decision Tree Error')

ada\_discrete\_err = [np.zeros](http://docs.scipy.org/doc/numpy-1.8.1/reference/generated/numpy.zeros.html#numpy.zeros)((n\_estimators,))

for i, y\_pred in enumerate(ada\_discrete.staged\_predict(X\_test)):

ada\_discrete\_err[i] = [zero\_one\_loss](http://scikit-learn.org/stable/modules/generated/sklearn.metrics.zero_one_loss.html#sklearn.metrics.zero_one_loss)(y\_pred, y\_test)

ada\_discrete\_err\_train = [np.zeros](http://docs.scipy.org/doc/numpy-1.8.1/reference/generated/numpy.zeros.html#numpy.zeros)((n\_estimators,))

for i, y\_pred in enumerate(ada\_discrete.staged\_predict(X\_train)):

ada\_discrete\_err\_train[i] = [zero\_one\_loss](http://scikit-learn.org/stable/modules/generated/sklearn.metrics.zero_one_loss.html#sklearn.metrics.zero_one_loss)(y\_pred, y\_train)

ada\_real\_err = [np.zeros](http://docs.scipy.org/doc/numpy-1.8.1/reference/generated/numpy.zeros.html#numpy.zeros)((n\_estimators,))

for i, y\_pred in enumerate(ada\_real.staged\_predict(X\_test)):

ada\_real\_err[i] = [zero\_one\_loss](http://scikit-learn.org/stable/modules/generated/sklearn.metrics.zero_one_loss.html#sklearn.metrics.zero_one_loss)(y\_pred, y\_test)

ada\_real\_err\_train = [np.zeros](http://docs.scipy.org/doc/numpy-1.8.1/reference/generated/numpy.zeros.html#numpy.zeros)((n\_estimators,))

for i, y\_pred in enumerate(ada\_real.staged\_predict(X\_train)):

ada\_real\_err\_train[i] = [zero\_one\_loss](http://scikit-learn.org/stable/modules/generated/sklearn.metrics.zero_one_loss.html#sklearn.metrics.zero_one_loss)(y\_pred, y\_train)

ax.plot([np.arange](http://docs.scipy.org/doc/numpy-1.8.1/reference/generated/numpy.arange.html#numpy.arange)(n\_estimators) + 1, ada\_discrete\_err,

label='Discrete AdaBoost Test Error',

color='red')

ax.plot([np.arange](http://docs.scipy.org/doc/numpy-1.8.1/reference/generated/numpy.arange.html#numpy.arange)(n\_estimators) + 1, ada\_discrete\_err\_train,

label='Discrete AdaBoost Train Error',

color='blue')

ax.plot([np.arange](http://docs.scipy.org/doc/numpy-1.8.1/reference/generated/numpy.arange.html#numpy.arange)(n\_estimators) + 1, ada\_real\_err,

label='Real AdaBoost Test Error',

color='orange')

ax.plot([np.arange](http://docs.scipy.org/doc/numpy-1.8.1/reference/generated/numpy.arange.html#numpy.arange)(n\_estimators) + 1, ada\_real\_err\_train,

label='Real AdaBoost Train Error',

color='green')

ax.set\_ylim((0.0, 0.5))

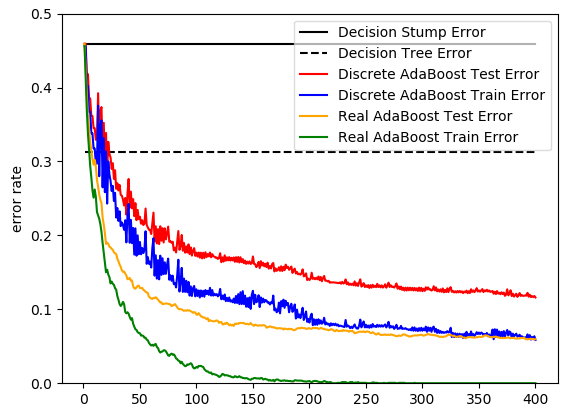
ax.set\_xlabel('n\_estimators')

ax.set\_ylabel('error rate')

leg = ax.legend(loc='upper right', fancybox=True)

leg.get\_frame().set\_alpha(0.7)

[plt.show](http://matplotlib.org/api/_as_gen/matplotlib.pyplot.show.html#matplotlib.pyplot.show)()



More details:

<http://scikit-learn.org/stable/modules/generated/sklearn.ensemble.AdaBoostClassifier.html>

##### 7.5. AdditiveRegression 4

##### 7.6. AttributeSelectedClassifier 2

##### 7.7. Bagging 4

Class: [BaggingClassifier](http://scikit-learn.org/stable/modules/generated/sklearn.ensemble.BaggingClassifier.html#sklearn.ensemble.BaggingClassifier)（Used for classification）, [BaggingRegressor](http://scikit-learn.org/stable/modules/generated/sklearn.ensemble.BaggingRegressor.html#sklearn.ensemble.BaggingRegressor)（Used for regression）

Constructor: sklearn.ensemble.BaggingClassifier(base\_estimator=None

, n\_estimators=10

, max\_samples=1.0

, max\_features=1.0

, bootstrap=True

, bootstrap\_features=False

, oob\_score=False

, n\_jobs=1

, random\_state=None, verbose=0)

Examples([BaggingClassifier](http://scikit-learn.org/stable/modules/generated/sklearn.ensemble.BaggingClassifier.html#sklearn.ensemble.BaggingClassifier))：

>>>from sklearn.ensemble import BaggingClassifier

>>>from sklearn.neighbors import KneighborsClassifier

>>>bagging=BaggingClassifier(KNeighborsClassifier(),max\_samples=0.5,max\_features=0.5)

More details:

<http://scikit-learn.org/stable/modules/generated/sklearn.ensemble.BaggingClassifier.html>

Constructor: sklearn.ensemble.BaggingClassifier(base\_estimator=None

, n\_estimators=10

, max\_samples=1.0

, max\_features=1.0

, bootstrap=True

, bootstrap\_features=False

, oob\_score=False

, n\_jobs=1

, random\_state=None, verbose=0)

More details:

<http://scikit-learn.org/stable/modules/generated/sklearn.ensemble.BaggingRegressor.html>

##### 7.8. RandomCommittee 2

##### 7.9. RandomSubSpace 3

#### 8. Attribute Selection Methods

##### 8.1. BestFirst 2

##### 8.2. GreedyStepwise 4