sklearn.linear_model.LogisticRegression

» *class* 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*) [source]

Logistic Regression (aka logit, MaxEnt) classifier.

In the multiclass case, the training algorithm uses the one-vs-rest (OvR) scheme if the ‘multi_class’ option is set to ‘ovr’, and uses the cross- entropy loss if the ‘multi_class’ option is set to ‘multinomial’. (Currently the ‘multinomial’ option is supported only by the ‘lbfgs’, ‘sag’ and ‘newton-cg’ solvers.)

This class implements regularized logistic regression using the ‘liblinear’ library, ‘newton-cg’, ‘sag’ and ‘lbfgs’ solvers. It can handle both dense and sparse input. Use C-ordered arrays or CSR matrices containing 64-bit floats for optimal performance; any other input format will be converted (and copied).

The ‘newton-cg’, ‘sag’, and ‘lbfgs’ solvers support only L2 regularization with primal formulation. The ‘liblinear’ solver supports both L1 and L2 regularization, with a dual formulation only for the L2 penalty.

Read more in the User Guide.

Parameters:

penalty : str, ‘l1’ or ‘l2’, default: ‘l2’

Used to specify the norm used in the penalization. The ‘newton-cg’, ‘sag’ and ‘lbfgs’ solvers support only l2 penalties.

dual : bool, default: False

Dual or primal formulation. Dual formulation is only implemented for l2 penalty with liblinear solver. Prefer dual=False when n_samples > n_features.

C : float, default: 1.0

Inverse of regularization strength; must be a positive float. Like in support vector machines, smaller values specify stronger regularization.

fit_intercept : bool, default: True

Specifies if a constant (a.k.a. bias or intercept) should be added to the decision function.

intercept_scaling : float, default 1.

Useful only when the solver ‘liblinear’ is used and self.fit_intercept is set to True. In this case, x becomes [x, self.intercept_scaling], i.e. a “synthetic” feature with constant value equal to intercept_scaling is

appended to the instance vector. The intercept becomes `intercept_scaling * synthetic_feature_weight`.

Note! the synthetic feature weight is subject to l1/l2 regularization as all other features. To lessen the effect of regularization on synthetic feature weight (and therefore on the intercept) intercept_scaling has to be increased.

class_weight : dict or 'balanced', default: None

»

Weights associated with classes in the form `{class_label: weight}`. If not given, all classes are supposed to have weight one.

The "balanced" mode uses the values of y to automatically adjust weights inversely proportional to class frequencies in the input data as `n_samples / (n_classes * np.bincount(y))`.

Note that these weights will be multiplied with sample_weight (passed through the fit method) if sample_weight is specified.

New in version 0.17: class_weight='balanced' instead of deprecated *class_weight='auto'*.

max_iter : int, default: 100

Useful only for the newton-cg, sag and lbfgs solvers. Maximum number of iterations taken for the solvers to converge.

random_state : int seed, RandomState instance, default: None

The seed of the pseudo random number generator to use when shuffling the data. Used only in solvers 'sag' and 'liblinear'.

solver : {'newton-cg', 'lbfgs', 'liblinear', 'sag'}, default: 'liblinear'

Algorithm to use in the optimization problem.

- For small datasets, 'liblinear' is a good choice, whereas 'sag' is faster for large ones.
- For multiclass problems, only 'newton-cg', 'sag' and 'lbfgs' handle multinomial loss; 'liblinear' is limited to one-versus-rest schemes.
- 'newton-cg', 'lbfgs' and 'sag' only handle L2 penalty.

Note that 'sag' fast convergence is only guaranteed on features with approximately the same scale. You can preprocess the data with a scaler from sklearn.preprocessing.

New in version 0.17: Stochastic Average Gradient descent solver.

tol : float, default: 1e-4

Tolerance for stopping criteria.

multi_class : str, {'ovr', 'multinomial'}, default: 'ovr'

Multiclass option can be either 'ovr' or 'multinomial'. If the option chosen is 'ovr', then a binary problem is fit for each label. Else the loss minimised is the multinomial loss fit across the entire probability distribution. Works only for the 'newton-cg', 'sag' and 'lbfgs' solver.

»

New in version 0.18: Stochastic Average Gradient descent solver for 'multinomial' case.

verbose : int, default: 0

For the liblinear and lbfgs solvers set verbose to any positive number for verbosity.

warm_start : bool, default: False

When set to True, reuse the solution of the previous call to fit as initialization, otherwise, just erase the previous solution. Useless for liblinear solver.

New in version 0.17: warm_start to support *lbfgs*, *newton-cg*, *sag* solvers.

n_jobs : int, default: 1

Number of CPU cores used during the cross-validation loop. If given a value of -1, all cores are used.

Attributes:

coef_ : array, shape (n_classes, n_features)

Coefficient of the features in the decision function.

intercept_ : array, shape (n_classes,)

Intercept (a.k.a. bias) added to the decision function. If *fit_intercept* is set to False, the intercept is set to zero.

n_iter_ : array, shape (n_classes,) or (1,)

Actual number of iterations for all classes. If binary or multinomial, it returns only 1 element. For liblinear solver, only the maximum number of iteration across all classes is given.

See also:

`SGDClassifier`

incrementally trained logistic regression (when given the parameter `loss="log"`).

`sklearn.svm.LinearSVC`

learns SVM models using the same algorithm.

Notes

The underlying C implementation uses a random number generator to select features when fitting the model. It is thus not uncommon, to have slightly different results for the same input data. If that happens, try with a smaller tol parameter.

Predict output may not match that of standalone liblinear in certain cases. See differences from liblinear in the narrative documentation.

»

References

LIBLINEAR – A Library for Large Linear Classification
http://www.csie.ntu.edu.tw/~cjlin/liblinear/

SAG – Mark Schmidt, Nicolas Le Roux, and Francis Bach
Minimizing Finite Sums with the Stochastic Average Gradient https://hal.inria.fr/hal-00860051/document

Hsiang-Fu Yu, Fang-Lan Huang, Chih-Jen Lin (2011). Dual coordinate descent
methods for logistic regression and maximum entropy models. Machine Learning 85(1-2):41-75.
http://www.csie.ntu.edu.tw/~cjlin/papers/maxent_dual.pdf

Methods

decision_function(X)	Predict confidence scores for samples.
densify()	Convert coefficient matrix to dense array format.
fit(X, y[, sample_weight])	Fit the model according to the given training data.
fit_transform(X[, y])	Fit to data, then transform it.
get_params([deep])	Get parameters for this estimator.
predict(X)	Predict class labels for samples in X.
predict_log_proba(X)	Log of probability estimates.
predict_proba(X)	Probability estimates.
score(X, y[, sample_weight])	Returns the mean accuracy on the given test data and labels.
set_params(**params)	Set the parameters of this estimator.
sparsify()	Convert coefficient matrix to sparse format.
transform(*args, **kwargs)	DEPRECATED: Support to use estimators as feature selectors will be removed in version 0.19.

__init__(*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*) [source]

decision_function(*X*) [source]

Predict confidence scores for samples.

The confidence score for a sample is the signed distance of that sample to the hyperplane.

Parameters: **X** : {array-like, sparse matrix}, shape = (n_samples, n_features)

Samples.

Returns: **array, shape=(n_samples,) if n_classes == 2 else (n_samples, n_classes)** :

Confidence scores per (sample, class) combination. In the binary case, confidence score for self.classes_[1] where >0 means this class would be predicted.

»

densify() [source]

Convert coefficient matrix to dense array format.

Converts the `coef_` member (back) to a numpy.ndarray. This is the default format of `coef_` and is required for fitting, so calling this method is only required on models that have previously been sparsified; otherwise, it is a no-op.

Returns: **self: estimator** :

fit(*X*, *y*, *sample_weight=None*) [source]

Fit the model according to the given training data.

Parameters: **X** : {array-like, sparse matrix}, shape (n_samples, n_features)

Training vector, where n_samples is the number of samples and n_features is the number of features.

y : array-like, shape (n_samples,)

Target vector relative to X.

sample_weight : array-like, shape (n_samples,) optional

Array of weights that are assigned to individual samples. If not provided, then each sample is given unit weight.

New in version 0.17: sample_weight support to LogisticRegression.

Returns: **self** : object

Returns self.

fit_transform(*X*, *y=None*, ***fit_params*) [source]

Fit to data, then transform it.

Fits transformer to X and y with optional parameters fit_params and returns a transformed version of X.

Parameters: **X** : numpy array of shape [n_samples, n_features]

Training set.

y : numpy array of shape [n_samples]

Target values.

Returns: **X_new** : numpy array of shape [n_samples, n_features_new]

Transformed array.

»

get_params(*deep=True*) [source]

Get parameters for this estimator.

Parameters: **deep** : boolean, optional

If True, will return the parameters for this estimator and contained subobjects that are estimators.

Returns: **params** : mapping of string to any

Parameter names mapped to their values.

predict(*X*) [source]

Predict class labels for samples in X.

Parameters: **X** : {array-like, sparse matrix}, shape = [n_samples, n_features]

Samples.

Returns: **C** : array, shape = [n_samples]

Predicted class label per sample.

predict_log_proba(*X*) [source]

Log of probability estimates.

The returned estimates for all classes are ordered by the label of classes.

Parameters: **X** : array-like, shape = [n_samples, n_features]

Returns: **T** : array-like, shape = [n_samples, n_classes]

Returns the log-probability of the sample for each class in the model, where classes are ordered as they are in `self.classes_`.

---

predict_proba(*X*) [source]

Probability estimates.

The returned estimates for all classes are ordered by the label of classes.

For a multi_class problem, if multi_class is set to be “multinomial” the softmax function is used to find the predicted probability of each class. Else use a one-vs-rest approach, i.e calculate the probability of each class assuming it to be positive using the logistic function. and normalize these values across all the classes.

Parameters: **X** : array-like, shape = [n_samples, n_features]

Returns: **T** : array-like, shape = [n_samples, n_classes]

Returns the probability of the sample for each class in the model, where classes are ordered as they are in `self.classes_`.

---

score(*X*, *y*, *sample_weight=None*) [source]

Returns the mean accuracy on the given test data and labels.

In multi-label classification, this is the subset accuracy which is a harsh metric since you require for each sample that each label set be correctly predicted.

Parameters: **X** : array-like, shape = (n_samples, n_features)

Test samples.

y : array-like, shape = (n_samples) or (n_samples, n_outputs)

True labels for X.

sample_weight : array-like, shape = [n_samples], optional

Sample weights.

Returns: **score** : float

Mean accuracy of self.predict(X) wrt. y.

---

set_params(***params*) [source]

Set the parameters of this estimator.

The method works on simple estimators as well as on nested objects (such as pipelines). The latter have parameters of the form `<component>__<parameter>` so that it's possible to update each component of a nested object.

Returns: **self** :

sparsify() [source]

»

Convert coefficient matrix to sparse format.

Converts the `coef_` member to a scipy.sparse matrix, which for L1-regularized models can be much more memory- and storage-efficient than the usual numpy.ndarray representation.

The `intercept_` member is not converted.

Returns: **self: estimator** :

Notes

For non-sparse models, i.e. when there are not many zeros in `coef_`, this may actually *increase* memory usage, so use this method with care. A rule of thumb is that the number of zero elements, which can be computed with `(coef_ == 0).sum()`, must be more than 50% for this to provide significant benefits.

After calling this method, further fitting with the partial_fit method (if any) will not work until you call densify.

transform(**args*, ***kwargs*) [source]

DEPRECATED: Support to use estimators as feature selectors will be removed in version 0.19. Use SelectFromModel instead.

Reduce X to its most important features.

Uses `coef_` or `feature_importances_` to determine the most important features. For models with a `coef_` for each class, the absolute sum over the classes is used.

Parameters: **X** : array or scipy sparse matrix of shape [n_samples, n_features]

The input samples.

threshold
: *string, float or None, optional (default=None)*

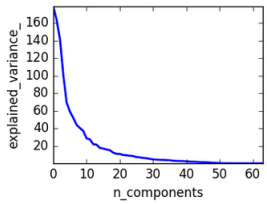
The threshold value to use for feature selection. Features whose importance is greater or equal are kept while the others are discarded. If "median" (resp. "mean"), then the threshold value is the median (resp. the mean) of the feature importances. A scaling factor (e.g., "1.25*mean") may also be used. If None and if available, the object attribute `threshold` is used. Otherwise, "mean" is used by default.

Returns: **X_r** : array of shape [n_samples, n_selected_features]

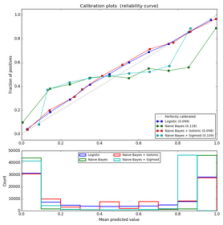
The input samples with only the selected features.

Examples using `sklearn.linear_model.LogisticRegression`

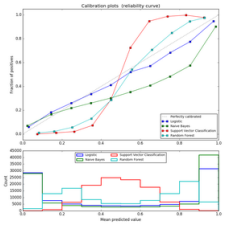
»



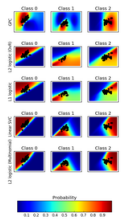
Pipelining: chaining a PCA and a logistic regression



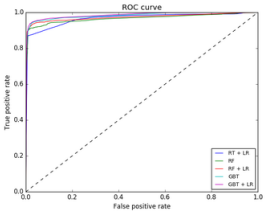
Probability Calibration curves



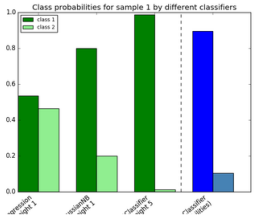
Comparison of Calibration of Classifiers



Plot classification probability

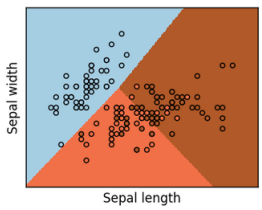


Feature transformations with ensembles of trees

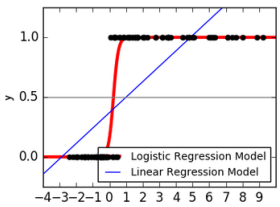


Plot class probabilities calculated by the VotingClassifier

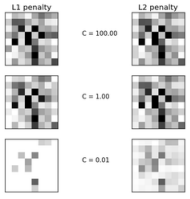
Digits Classification Exercise

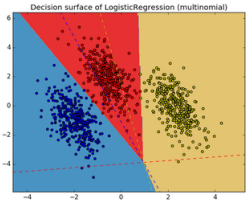


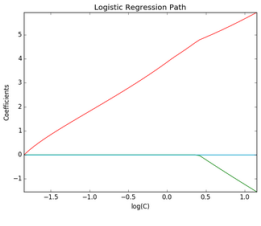
Logistic Regression 3-class Classifier



Logistic function





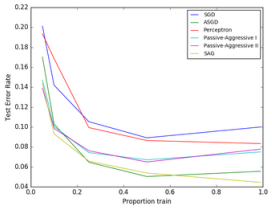


L1 Penalty and Sparsity in Logistic Regression

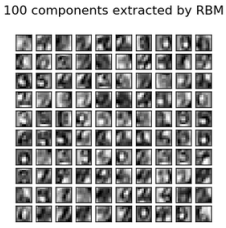
Plot multinomial and One-vs-Rest Logistic Regression

Path with L1- Logistic Regression

»



Comparing various online solvers



Restricted Boltzmann Machine features for digit classification