**1.1. Generalized Linear Models**

The following are a set of methods intended for regression in which the target value is expected to be a linear combination of the input variables. In mathematical notion, if \hat{y} is the predicted value.

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

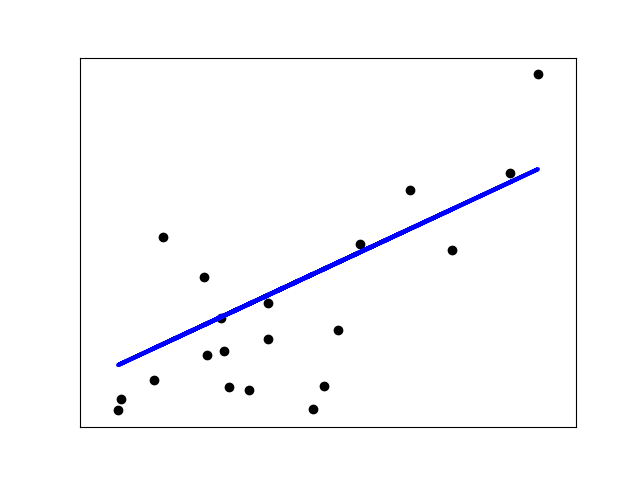
Across the module, we designate the vector w = (w_1,
..., w_p) as coef\_ and w_0 as intercept\_.

To perform classification with generalized linear models, see [Logistic regression](http://scikit-learn.org/stable/modules/linear_model.html#logistic-regression).

**1.1.1. Ordinary Least Squares**

[**LinearRegression**](http://scikit-learn.org/stable/modules/generated/sklearn.linear_model.LinearRegression.html#sklearn.linear_model.LinearRegression) fits a linear model with coefficients w = (w_1, ..., w_p) to minimize the residual sum of squares between the observed responses in the dataset, and the responses predicted by the linear approximation. Mathematically it solves a problem of the form:

\underset{w}{min\,} {|| X w - y||_2}^2

[](http://scikit-learn.org/stable/auto_examples/linear_model/plot_ols.html)

[**LinearRegression**](http://scikit-learn.org/stable/modules/generated/sklearn.linear_model.LinearRegression.html#sklearn.linear_model.LinearRegression) will take in its fit method arrays X, y and will store the coefficients w of the linear model in its coef\_member:

>>>

**>>> from** **sklearn** **import** linear\_model

**>>>** reg = linear\_model.LinearRegression()

**>>>** reg.fit ([[0, 0], [1, 1], [2, 2]], [0, 1, 2])

LinearRegression(copy\_X=True, fit\_intercept=True, n\_jobs=1, normalize=False)

**>>>** reg.coef\_

array([ 0.5, 0.5])

However, coefficient estimates for Ordinary Least Squares rely on the independence of the model terms. When terms are correlated and the columns of the design matrix X have an approximate linear dependence, the design matrix becomes close to singular and as a result, the least-squares estimate becomes highly sensitive to random errors in the observed response, producing a large variance. This situation of *multicollinearity* can arise, for example, when data are collected without an experimental design.

**Examples:**

*# Code source: Jaques Grobler*

*# License: BSD 3 clause*

**import** **matplotlib.pyplot** **as** **plt**

**import** **numpy** **as** **np**

**from** **sklearn** **import** datasets, linear\_model

**from** **sklearn.metrics** **import** [mean\_squared\_error](http://scikit-learn.org/stable/modules/generated/sklearn.metrics.mean_squared_error.html#sklearn.metrics.mean_squared_error), [r2\_score](http://scikit-learn.org/stable/modules/generated/sklearn.metrics.r2_score.html#sklearn.metrics.r2_score)

*# Load the diabetes dataset*

diabetes = [datasets.load\_diabetes](http://scikit-learn.org/stable/modules/generated/sklearn.datasets.load_diabetes.html#sklearn.datasets.load_diabetes)()

*# Use only one feature*

diabetes\_X = diabetes.data[:, [np.newaxis](http://docs.scipy.org/doc/numpy-1.8.1/reference/arrays.indexing.html#numpy.newaxis), 2]

*# Split the data into training/testing sets*

diabetes\_X\_train = diabetes\_X[:-20]

diabetes\_X\_test = diabetes\_X[-20:]

*# Split the targets into training/testing sets*

diabetes\_y\_train = diabetes.target[:-20]

diabetes\_y\_test = diabetes.target[-20:]

*# Create linear regression object*

regr = [linear\_model.LinearRegression](http://scikit-learn.org/stable/modules/generated/sklearn.linear_model.LinearRegression.html#sklearn.linear_model.LinearRegression)()

*# Train the model using the training sets*

regr.fit(diabetes\_X\_train, diabetes\_y\_train)

*# Make predictions using the testing set*

diabetes\_y\_pred = regr.predict(diabetes\_X\_test)

*# The coefficients*

**print**('Coefficients: **\n**', regr.coef\_)

*# The mean squared error*

**print**("Mean squared error: *%.2f*"

% [mean\_squared\_error](http://scikit-learn.org/stable/modules/generated/sklearn.metrics.mean_squared_error.html#sklearn.metrics.mean_squared_error)(diabetes\_y\_test, diabetes\_y\_pred))

*# Explained variance score: 1 is perfect prediction*

**print**('Variance score: *%.2f*' % [r2\_score](http://scikit-learn.org/stable/modules/generated/sklearn.metrics.r2_score.html#sklearn.metrics.r2_score)(diabetes\_y\_test, diabetes\_y\_pred))

*# Plot outputs*

[plt.scatter](http://matplotlib.org/api/_as_gen/matplotlib.pyplot.scatter.html#matplotlib.pyplot.scatter)(diabetes\_X\_test, diabetes\_y\_test, color='black')

[plt.plot](http://matplotlib.org/api/_as_gen/matplotlib.pyplot.plot.html#matplotlib.pyplot.plot)(diabetes\_X\_test, diabetes\_y\_pred, color='blue', linewidth=3)

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

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

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

**1.1.1.1. Ordinary Least Squares Complexity**

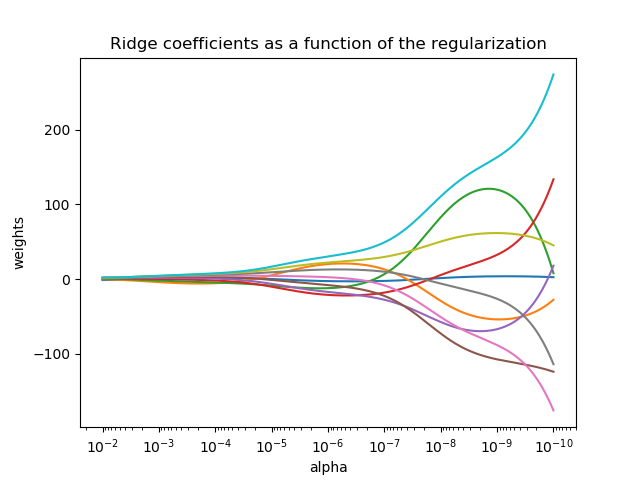
This method computes the least squares solution using a singular value decomposition of X. If X is a matrix of size (n, p) this method has a cost of O(n p^2), assuming that n \geq p.

**1.1.2. Ridge Regression**

[**Ridge**](http://scikit-learn.org/stable/modules/generated/sklearn.linear_model.Ridge.html#sklearn.linear_model.Ridge) regression addresses some of the problems of [Ordinary Least Squares](http://scikit-learn.org/stable/modules/linear_model.html#ordinary-least-squares) by imposing a penalty on the size of coefficients. The ridge coefficients minimize a penalized residual sum of squares,

\underset{w}{min\,} {{|| X w - y||_2}^2 + \alpha {||w||_2}^2}

Here, \alpha \geq 0 is a complexity parameter that controls the amount of shrinkage: the larger the value of \alpha, the greater the amount of shrinkage and thus the coefficients become more robust to collinearity.

[](http://scikit-learn.org/stable/auto_examples/linear_model/plot_ridge_path.html)

As with other linear models, [**Ridge**](http://scikit-learn.org/stable/modules/generated/sklearn.linear_model.Ridge.html#sklearn.linear_model.Ridge) will take in its fit method arrays X, y and will store the coefficients w of the linear model in its coef\_ member:

>>>

**>>> from** **sklearn** **import** linear\_model

**>>>** reg = linear\_model.Ridge (alpha = .5)

**>>>** reg.fit ([[0, 0], [0, 0], [1, 1]], [0, .1, 1])

Ridge(alpha=0.5, copy\_X=True, fit\_intercept=True, max\_iter=None,

normalize=False, random\_state=None, solver='auto', tol=0.001)

**>>>** reg.coef\_

array([ 0.34545455, 0.34545455])

**>>>** reg.intercept\_

0.13636...

**Examples:**

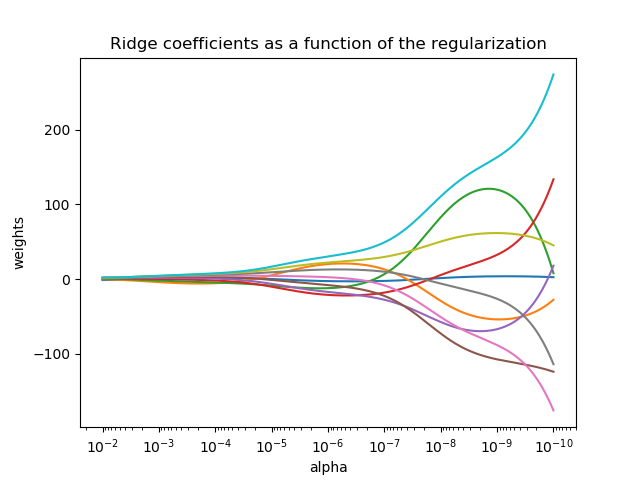
* [Plot Ridge coefficients as a function of the regularization](http://scikit-learn.org/stable/auto_examples/linear_model/plot_ridge_path.html#sphx-glr-auto-examples-linear-model-plot-ridge-path-py)

Shows the effect of collinearity in the coefficients of an estimator.

[**Ridge**](http://scikit-learn.org/stable/modules/generated/sklearn.linear_model.Ridge.html#sklearn.linear_model.Ridge) Regression is the estimator used in this example. Each color represents a different feature of the coefficient vector, and this is displayed as a function of the regularization parameter.

This example also shows the usefulness of applying Ridge regression to highly ill-conditioned matrices. For such matrices, a slight change in the target variable can cause huge variances in the calculated weights. In such cases, it is useful to set a certain regularization (alpha) to reduce this variation (noise).

When alpha is very large, the regularization effect dominates the squared loss function and the coefficients tend to zero. At the end of the path, as alpha tends toward zero and the solution tends towards the ordinary least squares, coefficients exhibit big oscillations. In practise it is necessary to tune alpha in such a way that a balance is maintained between both.



*# Author: Fabian Pedregosa -- <fabian.pedregosa@inria.fr>*

*# License: BSD 3 clause*

**print**(\_\_doc\_\_)

**import** **numpy** **as** **np**

**import** **matplotlib.pyplot** **as** **plt**

**from** **sklearn** **import** linear\_model

*# X is the 10x10 Hilbert matrix*

X = 1. / ([np.arange](http://docs.scipy.org/doc/numpy-1.8.1/reference/generated/numpy.arange.html#numpy.arange)(1, 11) + [np.arange](http://docs.scipy.org/doc/numpy-1.8.1/reference/generated/numpy.arange.html#numpy.arange)(0, 10)[:, [np.newaxis](http://docs.scipy.org/doc/numpy-1.8.1/reference/arrays.indexing.html#numpy.newaxis)])

y = [np.ones](http://docs.scipy.org/doc/numpy-1.8.1/reference/generated/numpy.ones.html#numpy.ones)(10)

*# #############################################################################*

*# Compute paths*

n\_alphas = 200

alphas = [np.logspace](http://docs.scipy.org/doc/numpy-1.8.1/reference/generated/numpy.logspace.html#numpy.logspace)(-10, -2, n\_alphas)

coefs = []

**for** a **in** alphas:

ridge = [linear\_model.Ridge](http://scikit-learn.org/stable/modules/generated/sklearn.linear_model.Ridge.html#sklearn.linear_model.Ridge)(alpha=a, fit\_intercept=False)

ridge.fit(X, y)

coefs.append(ridge.coef\_)

*# #############################################################################*

*# Display results*

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

ax.plot(alphas, coefs)

ax.set\_xscale('log')

ax.set\_xlim(ax.get\_xlim()[::-1]) *# reverse axis*

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

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

[plt.title](http://matplotlib.org/api/_as_gen/matplotlib.pyplot.title.html#matplotlib.pyplot.title)('Ridge coefficients as a function of the regularization')

[plt.axis](http://matplotlib.org/api/axis_api.html#matplotlib.axis)('tight')

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

**1.1.2.2. Setting the regularization parameter: generalized Cross-Validation**

[**RidgeCV**](http://scikit-learn.org/stable/modules/generated/sklearn.linear_model.RidgeCV.html#sklearn.linear_model.RidgeCV) implements ridge regression with built-in cross-validation of the alpha parameter. The object works in the same way as GridSearchCV except that it defaults to Generalized Cross-Validation (GCV), an efficient form of leave-one-out cross-validation:

>>>

**>>> from** **sklearn** **import** linear\_model

**>>>** reg = linear\_model.RidgeCV(alphas=[0.1, 1.0, 10.0])

**>>>** reg.fit([[0, 0], [0, 0], [1, 1]], [0, .1, 1])

RidgeCV(alphas=[0.1, 1.0, 10.0], cv=None, fit\_intercept=True, scoring=None,

normalize=False)

**>>>** reg.alpha\_

0.1

**1.1.3. Lasso**

The [**Lasso**](http://scikit-learn.org/stable/modules/generated/sklearn.linear_model.Lasso.html#sklearn.linear_model.Lasso) is a linear model that estimates sparse coefficients. It is useful in some contexts due to its tendency to prefer solutions with fewer parameter values, effectively reducing the number of variables upon which the given solution is dependent. For this reason, the Lasso and its variants are fundamental to the field of compressed sensing. Under certain conditions, it can recover the exact set of non-zero weights (see [Compressive sensing: tomography reconstruction with L1 prior (Lasso)](http://scikit-learn.org/stable/auto_examples/applications/plot_tomography_l1_reconstruction.html#sphx-glr-auto-examples-applications-plot-tomography-l1-reconstruction-py)).

Mathematically, it consists of a linear model trained with \ell_1 prior as regularizer. The objective function to minimize is:

\underset{w}{min\,} { \frac{1}{2n_{samples}} ||X w - y||_2 ^ 2 + \alpha ||w||_1}

The lasso estimate thus solves the minimization of the least-squares penalty with \alpha ||w||_1 added, where \alpha is a constant and ||w||_1 is the \ell_1-norm of the parameter vector.

The implementation in the class [**Lasso**](http://scikit-learn.org/stable/modules/generated/sklearn.linear_model.Lasso.html#sklearn.linear_model.Lasso) uses coordinate descent as the algorithm to fit the coefficients. See [Least Angle Regression](http://scikit-learn.org/stable/modules/linear_model.html#least-angle-regression) for another implementation:

>>>

**>>> from** **sklearn** **import** linear\_model

**>>>** reg = linear\_model.Lasso(alpha = 0.1)

**>>>** reg.fit([[0, 0], [1, 1]], [0, 1])

Lasso(alpha=0.1, copy\_X=True, fit\_intercept=True, max\_iter=1000,

normalize=False, positive=False, precompute=False, random\_state=None,

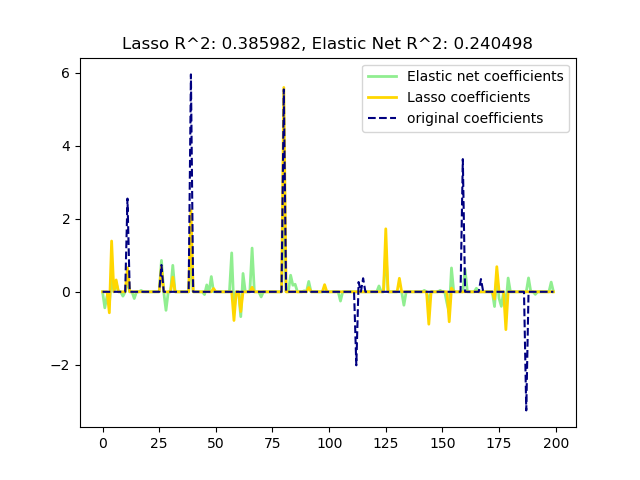
selection='cyclic', tol=0.0001, warm\_start=False)

**>>>** reg.predict([[1, 1]])

array([ 0.8])

Also useful for lower-level tasks is the function [**lasso\_path**](http://scikit-learn.org/stable/modules/generated/sklearn.linear_model.lasso_path.html#sklearn.linear_model.lasso_path) that computes the coefficients along the full path of possible values.

**Examples:**

* [Lasso and Elastic Net for Sparse Signals](http://scikit-learn.org/stable/auto_examples/linear_model/plot_lasso_and_elasticnet.html#sphx-glr-auto-examples-linear-model-plot-lasso-and-elasticnet-py)
* Estimates Lasso and Elastic-Net regression models on a manually generated sparse signal corrupted with an additive noise. Estimated coefficients are compared with the ground-truth.
* 
* Out:
* [Lasso](http://scikit-learn.org/stable/modules/generated/sklearn.linear_model.Lasso.html#sklearn.linear_model.Lasso)(alpha=0.1, copy\_X=**True**, fit\_intercept=**True**, max\_iter=1000,
* normalize=**False**, positive=**False**, precompute=**False**, random\_state=**None**,
* selection='cyclic', tol=0.0001, warm\_start=**False**)
* r^2 on test data : 0.385982
* [ElasticNet](http://scikit-learn.org/stable/modules/generated/sklearn.linear_model.ElasticNet.html#sklearn.linear_model.ElasticNet)(alpha=0.1, copy\_X=**True**, fit\_intercept=**True**, l1\_ratio=0.7,
* max\_iter=1000, normalize=**False**, positive=**False**, precompute=**False**,
* random\_state=**None**, selection='cyclic', tol=0.0001, warm\_start=**False**)
* r^2 on test data : 0.240498
* **print**(\_\_doc\_\_)
* **import** **numpy** **as** **np**
* **import** **matplotlib.pyplot** **as** **plt**
* **from** **sklearn.metrics** **import** [r2\_score](http://scikit-learn.org/stable/modules/generated/sklearn.metrics.r2_score.html#sklearn.metrics.r2_score)
* *# #############################################################################*
* *# Generate some sparse data to play with*
* [np.random.seed](http://docs.scipy.org/doc/numpy-1.8.1/reference/generated/numpy.random.seed.html#numpy.random.seed)(42)
* n\_samples, n\_features = 50, 200
* X = [np.random.randn](http://docs.scipy.org/doc/numpy-1.8.1/reference/generated/numpy.random.randn.html#numpy.random.randn)(n\_samples, n\_features)
* coef = 3 \* [np.random.randn](http://docs.scipy.org/doc/numpy-1.8.1/reference/generated/numpy.random.randn.html#numpy.random.randn)(n\_features)
* inds = [np.arange](http://docs.scipy.org/doc/numpy-1.8.1/reference/generated/numpy.arange.html#numpy.arange)(n\_features)
* [np.random.shuffle](http://docs.scipy.org/doc/numpy-1.8.1/reference/generated/numpy.random.shuffle.html#numpy.random.shuffle)(inds)
* coef[inds[10:]] = 0 *# sparsify coef*
* y = [np.dot](http://docs.scipy.org/doc/numpy-1.8.1/reference/generated/numpy.dot.html#numpy.dot)(X, coef)
* *# add noise*
* y += 0.01 \* [np.random.normal](http://docs.scipy.org/doc/numpy-1.8.1/reference/generated/numpy.random.normal.html#numpy.random.normal)(size=n\_samples)
* *# Split data in train set and test set*
* n\_samples = X.shape[0]
* X\_train, y\_train = X[:n\_samples // 2], y[:n\_samples // 2]
* X\_test, y\_test = X[n\_samples // 2:], y[n\_samples // 2:]
* *# #############################################################################*
* *# Lasso*
* **from** **sklearn.linear\_model** **import** [Lasso](http://scikit-learn.org/stable/modules/generated/sklearn.linear_model.Lasso.html#sklearn.linear_model.Lasso)
* alpha = 0.1
* lasso = [Lasso](http://scikit-learn.org/stable/modules/generated/sklearn.linear_model.Lasso.html#sklearn.linear_model.Lasso)(alpha=alpha)
* y\_pred\_lasso = lasso.fit(X\_train, y\_train).predict(X\_test)
* r2\_score\_lasso = [r2\_score](http://scikit-learn.org/stable/modules/generated/sklearn.metrics.r2_score.html#sklearn.metrics.r2_score)(y\_test, y\_pred\_lasso)
* **print**(lasso)
* **print**("r^2 on test data : *%f*" % r2\_score\_lasso)
* *# #############################################################################*
* *# ElasticNet*
* **from** **sklearn.linear\_model** **import** [ElasticNet](http://scikit-learn.org/stable/modules/generated/sklearn.linear_model.ElasticNet.html#sklearn.linear_model.ElasticNet)
* enet = [ElasticNet](http://scikit-learn.org/stable/modules/generated/sklearn.linear_model.ElasticNet.html#sklearn.linear_model.ElasticNet)(alpha=alpha, l1\_ratio=0.7)
* y\_pred\_enet = enet.fit(X\_train, y\_train).predict(X\_test)
* r2\_score\_enet = [r2\_score](http://scikit-learn.org/stable/modules/generated/sklearn.metrics.r2_score.html#sklearn.metrics.r2_score)(y\_test, y\_pred\_enet)
* **print**(enet)
* **print**("r^2 on test data : *%f*" % r2\_score\_enet)
* [plt.plot](http://matplotlib.org/api/_as_gen/matplotlib.pyplot.plot.html#matplotlib.pyplot.plot)(enet.coef\_, color='lightgreen', linewidth=2,
* label='Elastic net coefficients')
* [plt.plot](http://matplotlib.org/api/_as_gen/matplotlib.pyplot.plot.html#matplotlib.pyplot.plot)(lasso.coef\_, color='gold', linewidth=2,
* label='Lasso coefficients')
* [plt.plot](http://matplotlib.org/api/_as_gen/matplotlib.pyplot.plot.html#matplotlib.pyplot.plot)(coef, '--', color='navy', label='original coefficients')
* [plt.legend](http://matplotlib.org/api/legend_api.html#matplotlib.legend)(loc='best')
* [plt.title](http://matplotlib.org/api/_as_gen/matplotlib.pyplot.title.html#matplotlib.pyplot.title)("Lasso R^2: *%f*, Elastic Net R^2: *%f*"
* % (r2\_score\_lasso, r2\_score\_enet))
* [plt.show](http://matplotlib.org/api/_as_gen/matplotlib.pyplot.show.html#matplotlib.pyplot.show)()

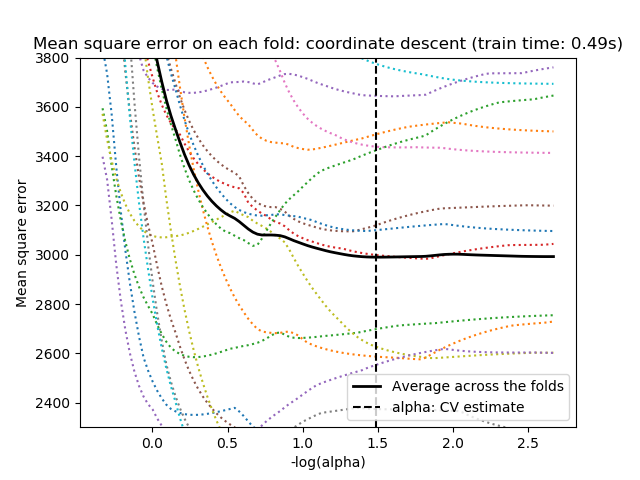
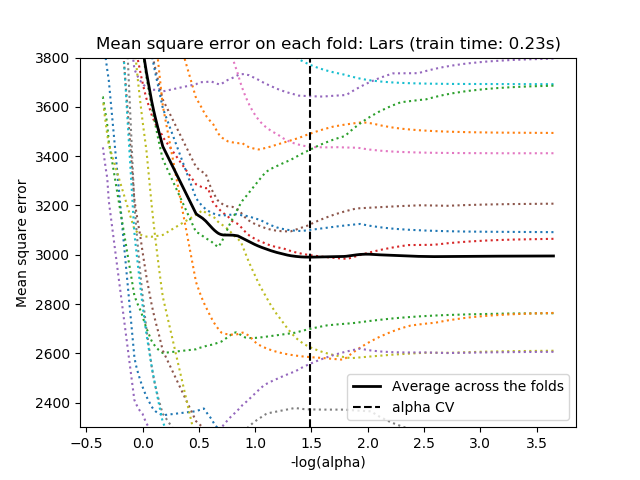
**1.1.3.1. Setting regularization parameter**

The alpha parameter controls the degree of sparsity of the coefficients estimated.

1.1.3.1.1. Using cross-validation

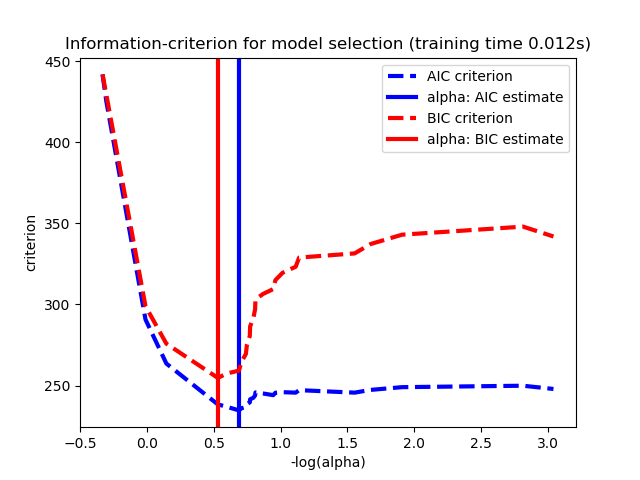
scikit-learn exposes objects that set the Lasso alpha parameter by cross-validation: [**LassoCV**](http://scikit-learn.org/stable/modules/generated/sklearn.linear_model.LassoCV.html#sklearn.linear_model.LassoCV) and [**LassoLarsCV**](http://scikit-learn.org/stable/modules/generated/sklearn.linear_model.LassoLarsCV.html#sklearn.linear_model.LassoLarsCV).[**LassoLarsCV**](http://scikit-learn.org/stable/modules/generated/sklearn.linear_model.LassoLarsCV.html#sklearn.linear_model.LassoLarsCV) is based on the [Least Angle Regression](http://scikit-learn.org/stable/modules/linear_model.html#least-angle-regression) algorithm explained below.

For high-dimensional datasets with many collinear regressors, [**LassoCV**](http://scikit-learn.org/stable/modules/generated/sklearn.linear_model.LassoCV.html#sklearn.linear_model.LassoCV) is most often preferable. However, [**LassoLarsCV**](http://scikit-learn.org/stable/modules/generated/sklearn.linear_model.LassoLarsCV.html#sklearn.linear_model.LassoLarsCV)has the advantage of exploring more relevant values of alpha parameter, and if the number of samples is very small compared to the number of features, it is often faster than [**LassoCV**](http://scikit-learn.org/stable/modules/generated/sklearn.linear_model.LassoCV.html#sklearn.linear_model.LassoCV).

**[](http://scikit-learn.org/stable/auto_examples/linear_model/plot_lasso_model_selection.html) [](http://scikit-learn.org/stable/auto_examples/linear_model/plot_lasso_model_selection.html)**

1.1.3.1.2. Information-criteria based model selection

Alternatively, the estimator [**LassoLarsIC**](http://scikit-learn.org/stable/modules/generated/sklearn.linear_model.LassoLarsIC.html#sklearn.linear_model.LassoLarsIC) proposes to use the Akaike information criterion (AIC) and the Bayes Information criterion (BIC). It is a computationally cheaper alternative to find the optimal value of alpha as the regularization path is computed only once instead of k+1 times when using k-fold cross-validation. However, such criteria needs a proper estimation of the degrees of freedom of the solution, are derived for large samples (asymptotic results) and assume the model is correct, i.e. that the data are actually generated by this model. They also tend to break when the problem is badly conditioned (more features than samples).

[](http://scikit-learn.org/stable/auto_examples/linear_model/plot_lasso_model_selection.html)

**Examples:**

* [Lasso model selection: Cross-Validation / AIC / BIC](http://scikit-learn.org/stable/auto_examples/linear_model/plot_lasso_model_selection.html#sphx-glr-auto-examples-linear-model-plot-lasso-model-selection-py)

Use the Akaike information criterion (AIC), the Bayes Information criterion (BIC) and cross-validation to select an optimal value of the regularization parameter alpha of the [Lasso](http://scikit-learn.org/stable/modules/linear_model.html#lasso) estimator.

Results obtained with LassoLarsIC are based on AIC/BIC criteria.

Information-criterion based model selection is very fast, but it relies on a proper estimation of degrees of freedom, are derived for large samples (asymptotic results) and assume the model is correct, i.e. that the data are actually generated by this model. They also tend to break when the problem is badly conditioned (more features than samples).

For cross-validation, we use 20-fold with 2 algorithms to compute the Lasso path: coordinate descent, as implemented by the LassoCV class, and Lars (least angle regression) as implemented by the LassoLarsCV class. Both algorithms give roughly the same results. They differ with regards to their execution speed and sources of numerical errors.

Lars computes a path solution only for each kink in the path. As a result, it is very efficient when there are only of few kinks, which is the case if there are few features or samples. Also, it is able to compute the full path without setting any meta parameter. On the opposite, coordinate descent compute the path points on a pre-specified grid (here we use the default). Thus it is more efficient if the number of grid points is smaller than the number of kinks in the path. Such a strategy can be interesting if the number of features is really large and there are enough samples to select a large amount. In terms of numerical errors, for heavily correlated variables, Lars will accumulate more errors, while the coordinate descent algorithm will only sample the path on a grid.

Note how the optimal value of alpha varies for each fold. This illustrates why nested-cross validation is necessary when trying to evaluate the performance of a method for which a parameter is chosen by cross-validation: this choice of parameter may not be optimal for unseen data.

Out:

Computing regularization path using the coordinate descent lasso...

Computing regularization path using the Lars lasso...

**print**(\_\_doc\_\_)

*# Author: Olivier Grisel, Gael Varoquaux, Alexandre Gramfort*

*# License: BSD 3 clause*

**import** **time**

**import** **numpy** **as** **np**

**import** **matplotlib.pyplot** **as** **plt**

**from** **sklearn.linear\_model** **import** [LassoCV](http://scikit-learn.org/stable/modules/generated/sklearn.linear_model.LassoCV.html#sklearn.linear_model.LassoCV), [LassoLarsCV](http://scikit-learn.org/stable/modules/generated/sklearn.linear_model.LassoLarsCV.html#sklearn.linear_model.LassoLarsCV), [LassoLarsIC](http://scikit-learn.org/stable/modules/generated/sklearn.linear_model.LassoLarsIC.html#sklearn.linear_model.LassoLarsIC)

**from** **sklearn** **import** datasets

diabetes = [datasets.load\_diabetes](http://scikit-learn.org/stable/modules/generated/sklearn.datasets.load_diabetes.html#sklearn.datasets.load_diabetes)()

X = diabetes.data

y = diabetes.target

rng = [np.random.RandomState](http://docs.scipy.org/doc/numpy-1.8.1/reference/generated/numpy.random.RandomState.wald.html#numpy.random.RandomState)(42)

X = [np.c\_](http://docs.scipy.org/doc/numpy-1.8.1/reference/generated/numpy.c_.html#numpy.c_)[X, rng.randn(X.shape[0], 14)] *# add some bad features*

*# normalize data as done by Lars to allow for comparison*

X /= [np.sqrt](http://docs.scipy.org/doc/numpy-1.8.1/reference/generated/numpy.sqrt.html#numpy.sqrt)([np.sum](http://docs.scipy.org/doc/numpy-1.8.1/reference/generated/numpy.sum.html#numpy.sum)(X \*\* 2, axis=0))

*# #############################################################################*

*# LassoLarsIC: least angle regression with BIC/AIC criterion*

model\_bic = [LassoLarsIC](http://scikit-learn.org/stable/modules/generated/sklearn.linear_model.LassoLarsIC.html#sklearn.linear_model.LassoLarsIC)(criterion='bic')

t1 = time.time()

model\_bic.fit(X, y)

t\_bic = time.time() - t1

alpha\_bic\_ = model\_bic.alpha\_

model\_aic = [LassoLarsIC](http://scikit-learn.org/stable/modules/generated/sklearn.linear_model.LassoLarsIC.html#sklearn.linear_model.LassoLarsIC)(criterion='aic')

model\_aic.fit(X, y)

alpha\_aic\_ = model\_aic.alpha\_

**def** plot\_ic\_criterion(model, name, color):

alpha\_ = model.alpha\_

alphas\_ = model.alphas\_

criterion\_ = model.criterion\_

[plt.plot](http://matplotlib.org/api/_as_gen/matplotlib.pyplot.plot.html#matplotlib.pyplot.plot)(-[np.log10](http://docs.scipy.org/doc/numpy-1.8.1/reference/generated/numpy.log10.html#numpy.log10)(alphas\_), criterion\_, '--', color=color,

linewidth=3, label='*%s* criterion' % name)

[plt.axvline](http://matplotlib.org/api/_as_gen/matplotlib.pyplot.axvline.html#matplotlib.pyplot.axvline)(-[np.log10](http://docs.scipy.org/doc/numpy-1.8.1/reference/generated/numpy.log10.html#numpy.log10)(alpha\_), color=color, linewidth=3,

label='alpha: *%s* estimate' % name)

[plt.xlabel](http://matplotlib.org/api/_as_gen/matplotlib.pyplot.xlabel.html#matplotlib.pyplot.xlabel)('-log(alpha)')

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

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

plot\_ic\_criterion(model\_aic, 'AIC', 'b')

plot\_ic\_criterion(model\_bic, 'BIC', 'r')

[plt.legend](http://matplotlib.org/api/legend_api.html#matplotlib.legend)()

[plt.title](http://matplotlib.org/api/_as_gen/matplotlib.pyplot.title.html#matplotlib.pyplot.title)('Information-criterion for model selection (training time *%.3f*s)'

% t\_bic)

*# #############################################################################*

*# LassoCV: coordinate descent*

*# Compute paths*

**print**("Computing regularization path using the coordinate descent lasso...")

t1 = time.time()

model = [LassoCV](http://scikit-learn.org/stable/modules/generated/sklearn.linear_model.LassoCV.html#sklearn.linear_model.LassoCV)(cv=20).fit(X, y)

t\_lasso\_cv = time.time() - t1

*# Display results*

m\_log\_alphas = -[np.log10](http://docs.scipy.org/doc/numpy-1.8.1/reference/generated/numpy.log10.html#numpy.log10)(model.alphas\_)

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

ymin, ymax = 2300, 3800

[plt.plot](http://matplotlib.org/api/_as_gen/matplotlib.pyplot.plot.html#matplotlib.pyplot.plot)(m\_log\_alphas, model.mse\_path\_, ':')

[plt.plot](http://matplotlib.org/api/_as_gen/matplotlib.pyplot.plot.html#matplotlib.pyplot.plot)(m\_log\_alphas, model.mse\_path\_.mean(axis=-1), 'k',

label='Average across the folds', linewidth=2)

[plt.axvline](http://matplotlib.org/api/_as_gen/matplotlib.pyplot.axvline.html#matplotlib.pyplot.axvline)(-[np.log10](http://docs.scipy.org/doc/numpy-1.8.1/reference/generated/numpy.log10.html#numpy.log10)(model.alpha\_), linestyle='--', color='k',

label='alpha: CV estimate')

[plt.legend](http://matplotlib.org/api/legend_api.html#matplotlib.legend)()

[plt.xlabel](http://matplotlib.org/api/_as_gen/matplotlib.pyplot.xlabel.html#matplotlib.pyplot.xlabel)('-log(alpha)')

[plt.ylabel](http://matplotlib.org/api/_as_gen/matplotlib.pyplot.ylabel.html#matplotlib.pyplot.ylabel)('Mean square error')

[plt.title](http://matplotlib.org/api/_as_gen/matplotlib.pyplot.title.html#matplotlib.pyplot.title)('Mean square error on each fold: coordinate descent '

'(train time: *%.2f*s)' % t\_lasso\_cv)

[plt.axis](http://matplotlib.org/api/axis_api.html#matplotlib.axis)('tight')

[plt.ylim](http://matplotlib.org/api/_as_gen/matplotlib.pyplot.ylim.html#matplotlib.pyplot.ylim)(ymin, ymax)

*# #############################################################################*

*# LassoLarsCV: least angle regression*

*# Compute paths*

**print**("Computing regularization path using the Lars lasso...")

t1 = time.time()

model = [LassoLarsCV](http://scikit-learn.org/stable/modules/generated/sklearn.linear_model.LassoLarsCV.html#sklearn.linear_model.LassoLarsCV)(cv=20).fit(X, y)

t\_lasso\_lars\_cv = time.time() - t1

*# Display results*

m\_log\_alphas = -[np.log10](http://docs.scipy.org/doc/numpy-1.8.1/reference/generated/numpy.log10.html#numpy.log10)(model.cv\_alphas\_)

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

[plt.plot](http://matplotlib.org/api/_as_gen/matplotlib.pyplot.plot.html#matplotlib.pyplot.plot)(m\_log\_alphas, model.mse\_path\_, ':')

[plt.plot](http://matplotlib.org/api/_as_gen/matplotlib.pyplot.plot.html#matplotlib.pyplot.plot)(m\_log\_alphas, model.mse\_path\_.mean(axis=-1), 'k',

label='Average across the folds', linewidth=2)

[plt.axvline](http://matplotlib.org/api/_as_gen/matplotlib.pyplot.axvline.html#matplotlib.pyplot.axvline)(-[np.log10](http://docs.scipy.org/doc/numpy-1.8.1/reference/generated/numpy.log10.html#numpy.log10)(model.alpha\_), linestyle='--', color='k',

label='alpha CV')

[plt.legend](http://matplotlib.org/api/legend_api.html#matplotlib.legend)()

[plt.xlabel](http://matplotlib.org/api/_as_gen/matplotlib.pyplot.xlabel.html#matplotlib.pyplot.xlabel)('-log(alpha)')

[plt.ylabel](http://matplotlib.org/api/_as_gen/matplotlib.pyplot.ylabel.html#matplotlib.pyplot.ylabel)('Mean square error')

[plt.title](http://matplotlib.org/api/_as_gen/matplotlib.pyplot.title.html#matplotlib.pyplot.title)('Mean square error on each fold: Lars (train time: *%.2f*s)'

% t\_lasso\_lars\_cv)

[plt.axis](http://matplotlib.org/api/axis_api.html#matplotlib.axis)('tight')

[plt.ylim](http://matplotlib.org/api/_as_gen/matplotlib.pyplot.ylim.html#matplotlib.pyplot.ylim)(ymin, ymax)

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

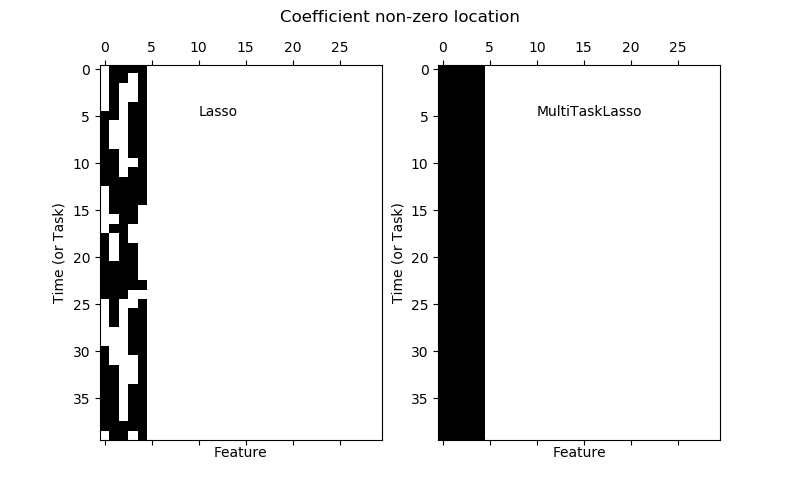
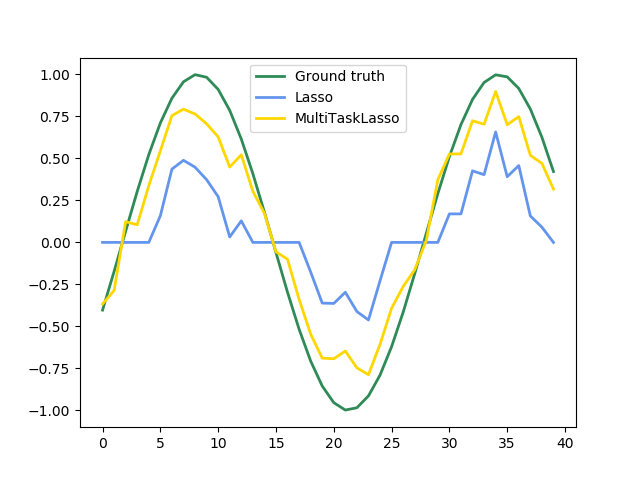
1.1.3.1.3. Comparison with the regularization parameter of SVM

The equivalence between alpha and the regularization parameter of SVM, C is given by alpha = 1 / C or alpha = 1 / (n\_samples \* C), depending on the estimator and the exact objective function optimized by the model.

**1.1.4. Multi-task Lasso**

The [**MultiTaskLasso**](http://scikit-learn.org/stable/modules/generated/sklearn.linear_model.MultiTaskLasso.html#sklearn.linear_model.MultiTaskLasso) is a linear model that estimates sparse coefficients for multiple regression problems jointly: y is a 2D array, of shape (n\_samples, n\_tasks). The constraint is that the selected features are the same for all the regression problems, also called tasks.

The following figure compares the location of the non-zeros in W obtained with a simple Lasso or a MultiTaskLasso. The Lasso estimates yields scattered non-zeros while the non-zeros of the MultiTaskLasso are full columns.

**[](http://scikit-learn.org/stable/auto_examples/linear_model/plot_multi_task_lasso_support.html) [](http://scikit-learn.org/stable/auto_examples/linear_model/plot_multi_task_lasso_support.html)**

**Fitting a time-series model, imposing that any active feature be active at all times.**

**Examples:**

* [Joint feature selection with multi-task Lasso](http://scikit-learn.org/stable/auto_examples/linear_model/plot_multi_task_lasso_support.html#sphx-glr-auto-examples-linear-model-plot-multi-task-lasso-support-py)
* *# Author: Alexandre Gramfort <alexandre.gramfort@inria.fr>*
* *# License: BSD 3 clause*
* **import** **matplotlib.pyplot** **as** **plt**
* **import** **numpy** **as** **np**
* **from** **sklearn.linear\_model** **import** [MultiTaskLasso](http://scikit-learn.org/stable/modules/generated/sklearn.linear_model.MultiTaskLasso.html#sklearn.linear_model.MultiTaskLasso), [Lasso](http://scikit-learn.org/stable/modules/generated/sklearn.linear_model.Lasso.html#sklearn.linear_model.Lasso)
* rng = [np.random.RandomState](http://docs.scipy.org/doc/numpy-1.8.1/reference/generated/numpy.random.RandomState.wald.html#numpy.random.RandomState)(42)
* *# Generate some 2D coefficients with sine waves with random frequency and phase*
* n\_samples, n\_features, n\_tasks = 100, 30, 40
* n\_relevant\_features = 5
* coef = [np.zeros](http://docs.scipy.org/doc/numpy-1.8.1/reference/generated/numpy.zeros.html#numpy.zeros)((n\_tasks, n\_features))
* times = [np.linspace](http://docs.scipy.org/doc/numpy-1.8.1/reference/generated/numpy.linspace.html#numpy.linspace)(0, 2 \* np.pi, n\_tasks)
* **for** k **in** range(n\_relevant\_features):
* coef[:, k] = [np.sin](http://docs.scipy.org/doc/numpy-1.8.1/reference/generated/numpy.sin.html#numpy.sin)((1. + rng.randn(1)) \* times + 3 \* rng.randn(1))
* X = rng.randn(n\_samples, n\_features)
* Y = [np.dot](http://docs.scipy.org/doc/numpy-1.8.1/reference/generated/numpy.dot.html#numpy.dot)(X, coef.T) + rng.randn(n\_samples, n\_tasks)
* coef\_lasso\_ = [np.array](http://docs.scipy.org/doc/numpy-1.8.1/reference/generated/numpy.array.html#numpy.array)([[Lasso](http://scikit-learn.org/stable/modules/generated/sklearn.linear_model.Lasso.html#sklearn.linear_model.Lasso)(alpha=0.5).fit(X, y).coef\_ **for** y **in** Y.T])
* coef\_multi\_task\_lasso\_ = [MultiTaskLasso](http://scikit-learn.org/stable/modules/generated/sklearn.linear_model.MultiTaskLasso.html#sklearn.linear_model.MultiTaskLasso)(alpha=1.).fit(X, Y).coef\_
* *# #############################################################################*
* *# Plot support and time series*
* fig = [plt.figure](http://matplotlib.org/api/_as_gen/matplotlib.figure.AxesStack.html#matplotlib.figure)(figsize=(8, 5))
* [plt.subplot](http://matplotlib.org/api/_as_gen/matplotlib.pyplot.subplot.html#matplotlib.pyplot.subplot)(1, 2, 1)
* [plt.spy](http://matplotlib.org/api/_as_gen/matplotlib.pyplot.spy.html#matplotlib.pyplot.spy)(coef\_lasso\_)
* [plt.xlabel](http://matplotlib.org/api/_as_gen/matplotlib.pyplot.xlabel.html#matplotlib.pyplot.xlabel)('Feature')
* [plt.ylabel](http://matplotlib.org/api/_as_gen/matplotlib.pyplot.ylabel.html#matplotlib.pyplot.ylabel)('Time (or Task)')
* [plt.text](http://matplotlib.org/api/text_api.html#matplotlib.text)(10, 5, 'Lasso')
* [plt.subplot](http://matplotlib.org/api/_as_gen/matplotlib.pyplot.subplot.html#matplotlib.pyplot.subplot)(1, 2, 2)
* [plt.spy](http://matplotlib.org/api/_as_gen/matplotlib.pyplot.spy.html#matplotlib.pyplot.spy)(coef\_multi\_task\_lasso\_)
* [plt.xlabel](http://matplotlib.org/api/_as_gen/matplotlib.pyplot.xlabel.html#matplotlib.pyplot.xlabel)('Feature')
* [plt.ylabel](http://matplotlib.org/api/_as_gen/matplotlib.pyplot.ylabel.html#matplotlib.pyplot.ylabel)('Time (or Task)')
* [plt.text](http://matplotlib.org/api/text_api.html#matplotlib.text)(10, 5, 'MultiTaskLasso')
* fig.suptitle('Coefficient non-zero location')
* feature\_to\_plot = 0
* [plt.figure](http://matplotlib.org/api/_as_gen/matplotlib.figure.AxesStack.html#matplotlib.figure)()
* lw = 2
* [plt.plot](http://matplotlib.org/api/_as_gen/matplotlib.pyplot.plot.html#matplotlib.pyplot.plot)(coef[:, feature\_to\_plot], color='seagreen', linewidth=lw,
* label='Ground truth')
* [plt.plot](http://matplotlib.org/api/_as_gen/matplotlib.pyplot.plot.html#matplotlib.pyplot.plot)(coef\_lasso\_[:, feature\_to\_plot], color='cornflowerblue', linewidth=lw,
* label='Lasso')
* [plt.plot](http://matplotlib.org/api/_as_gen/matplotlib.pyplot.plot.html#matplotlib.pyplot.plot)(coef\_multi\_task\_lasso\_[:, feature\_to\_plot], color='gold', linewidth=lw,
* label='MultiTaskLasso')
* [plt.legend](http://matplotlib.org/api/legend_api.html#matplotlib.legend)(loc='upper center')
* [plt.axis](http://matplotlib.org/api/axis_api.html#matplotlib.axis)('tight')
* [plt.ylim](http://matplotlib.org/api/_as_gen/matplotlib.pyplot.ylim.html#matplotlib.pyplot.ylim)([-1.1, 1.1])
* [plt.show](http://matplotlib.org/api/_as_gen/matplotlib.pyplot.show.html#matplotlib.pyplot.show)()

Mathematically, it consists of a linear model trained with a mixed \ell_1 \ell_2 prior as regularizer. The objective function to minimize is:

\underset{w}{min\,} { \frac{1}{2n_{samples}} ||X W - Y||_{Fro} ^ 2 + \alpha ||W||_{21}}

where Fro indicates the Frobenius norm:

||A||_{Fro} = \sqrt{\sum_{ij} a_{ij}^2}

and \ell_1 \ell_2 reads:

||A||_{2 1} = \sum_i \sqrt{\sum_j a_{ij}^2}

The implementation in the class [**MultiTaskLasso**](http://scikit-learn.org/stable/modules/generated/sklearn.linear_model.MultiTaskLasso.html#sklearn.linear_model.MultiTaskLasso) uses coordinate descent as the algorithm to fit the coefficients.

**1.1.5. Elastic Net**

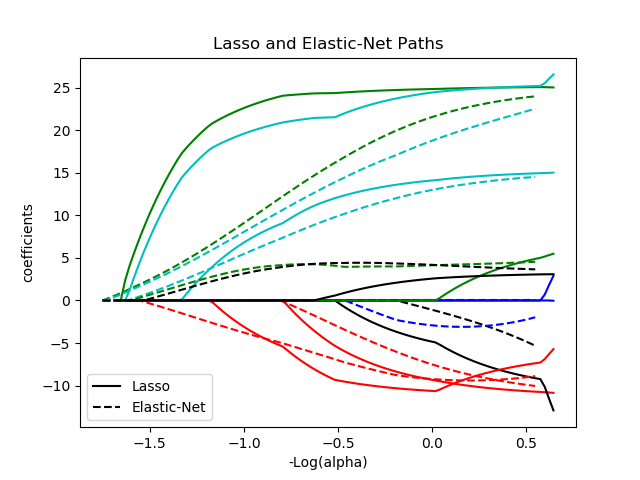
[**ElasticNet**](http://scikit-learn.org/stable/modules/generated/sklearn.linear_model.ElasticNet.html#sklearn.linear_model.ElasticNet) is a linear regression model trained with L1 and L2 prior as regularizer. This combination allows for learning a sparse model where few of the weights are non-zero like [**Lasso**](http://scikit-learn.org/stable/modules/generated/sklearn.linear_model.Lasso.html#sklearn.linear_model.Lasso), while still maintaining the regularization properties of [**Ridge**](http://scikit-learn.org/stable/modules/generated/sklearn.linear_model.Ridge.html#sklearn.linear_model.Ridge). We control the convex combination of L1 and L2 using the l1\_ratio parameter.

Elastic-net is useful when there are multiple features which are correlated with one another. Lasso is likely to pick one of these at random, while elastic-net is likely to pick both.

A practical advantage of trading-off between Lasso and Ridge is it allows Elastic-Net to inherit some of Ridge’s stability under rotation.

The objective function to minimize is in this case

\underset{w}{min\,} { \frac{1}{2n_{samples}} ||X w - y||_2 ^ 2 + \alpha \rho ||w||_1 +
\frac{\alpha(1-\rho)}{2} ||w||_2 ^ 2}

[](http://scikit-learn.org/stable/auto_examples/linear_model/plot_lasso_coordinate_descent_path.html)

The class [**ElasticNetCV**](http://scikit-learn.org/stable/modules/generated/sklearn.linear_model.ElasticNetCV.html#sklearn.linear_model.ElasticNetCV) can be used to set the parameters alpha (\alpha) and l1\_ratio (\rho) by cross-validation.

**1.1.6. Multi-task Elastic Net**

The [**MultiTaskElasticNet**](http://scikit-learn.org/stable/modules/generated/sklearn.linear_model.MultiTaskElasticNet.html#sklearn.linear_model.MultiTaskElasticNet) is an elastic-net model that estimates sparse coefficients for multiple regression problems jointly: Y is a 2D array, of shape (n\_samples, n\_tasks). The constraint is that the selected features are the same for all the regression problems, also called tasks.

Mathematically, it consists of a linear model trained with a mixed \ell_1 \ell_2 prior and \ell_2 prior as regularizer. The objective function to minimize is:

\underset{W}{min\,} { \frac{1}{2n_{samples}} ||X W - Y||_{Fro}^2 + \alpha \rho ||W||_{2 1} +
\frac{\alpha(1-\rho)}{2} ||W||_{Fro}^2}

The implementation in the class [**MultiTaskElasticNet**](http://scikit-learn.org/stable/modules/generated/sklearn.linear_model.MultiTaskElasticNet.html#sklearn.linear_model.MultiTaskElasticNet) uses coordinate descent as the algorithm to fit the coefficients.

The class [**MultiTaskElasticNetCV**](http://scikit-learn.org/stable/modules/generated/sklearn.linear_model.MultiTaskElasticNetCV.html#sklearn.linear_model.MultiTaskElasticNetCV) can be used to set the parameters alpha (\alpha) and l1\_ratio (\rho) by cross-validation.

**1.1.7. Least Angle Regression**

Least-angle regression (LARS) is a regression algorithm for high-dimensional data, developed by Bradley Efron, Trevor Hastie, Iain Johnstone and Robert Tibshirani. LARS is similar to forward stepwise regression. At each step, it finds the predictor most correlated with the response. When there are multiple predictors having equal correlation, instead of continuing along the same predictor, it proceeds in a direction equiangular between the predictors.

The advantages of LARS are:

* It is numerically efficient in contexts where p >> n (i.e., when the number of dimensions is significantly greater than the number of points)
* It is computationally just as fast as forward selection and has the same order of complexity as an ordinary least squares.
* It produces a full piecewise linear solution path, which is useful in cross-validation or similar attempts to tune the model.
* If two variables are almost equally correlated with the response, then their coefficients should increase at approximately the same rate. The algorithm thus behaves as intuition would expect, and also is more stable.
* It is easily modified to produce solutions for other estimators, like the Lasso.

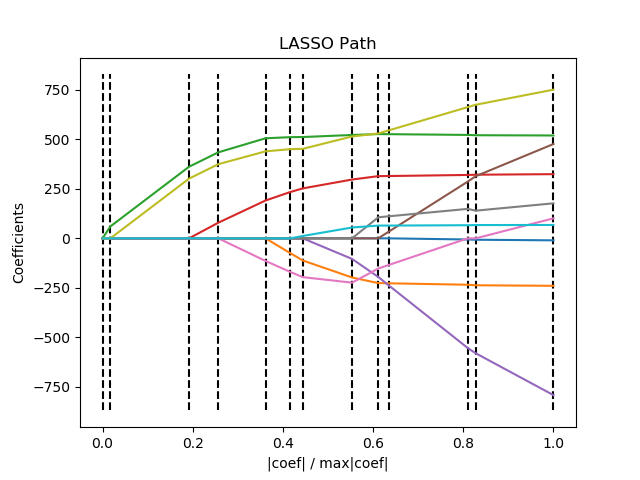
The disadvantages of the LARS method include:

* Because LARS is based upon an iterative refitting of the residuals, it would appear to be especially sensitive to the effects of noise. This problem is discussed in detail by Weisberg in the discussion section of the Efron et al. (2004) Annals of Statistics article.

The LARS model can be used using estimator [**Lars**](http://scikit-learn.org/stable/modules/generated/sklearn.linear_model.Lars.html#sklearn.linear_model.Lars), or its low-level implementation [**lars\_path**](http://scikit-learn.org/stable/modules/generated/sklearn.linear_model.lars_path.html#sklearn.linear_model.lars_path).

**1.1.8. LARS Lasso**

[**LassoLars**](http://scikit-learn.org/stable/modules/generated/sklearn.linear_model.LassoLars.html#sklearn.linear_model.LassoLars) is a lasso model implemented using the LARS algorithm, and unlike the implementation based on coordinate\_descent, this yields the exact solution, which is piecewise linear as a function of the norm of its coefficients.

[](http://scikit-learn.org/stable/auto_examples/linear_model/plot_lasso_lars.html)

>>>

**>>> from** **sklearn** **import** linear\_model

**>>>** reg = linear\_model.LassoLars(alpha=.1)

**>>>** reg.fit([[0, 0], [1, 1]], [0, 1])

LassoLars(alpha=0.1, copy\_X=True, eps=..., fit\_intercept=True,

fit\_path=True, max\_iter=500, normalize=True, positive=False,

precompute='auto', verbose=False)

**>>>** reg.coef\_

array([ 0.717157..., 0. ])

**Examples:**

* [Lasso path using LARS](http://scikit-learn.org/stable/auto_examples/linear_model/plot_lasso_lars.html#sphx-glr-auto-examples-linear-model-plot-lasso-lars-py)
* **import** **numpy** **as** **np**
* **import** **matplotlib.pyplot** **as** **plt**
* **from** **sklearn** **import** linear\_model
* **from** **sklearn** **import** datasets
* diabetes = [datasets.load\_diabetes](http://scikit-learn.org/stable/modules/generated/sklearn.datasets.load_diabetes.html#sklearn.datasets.load_diabetes)()
* X = diabetes.data
* y = diabetes.target
* **print**("Computing regularization path using the LARS ...")
* alphas, \_, coefs = [linear\_model.lars\_path](http://scikit-learn.org/stable/modules/generated/sklearn.linear_model.lars_path.html#sklearn.linear_model.lars_path)(X, y, method='lasso', verbose=True)
* xx = [np.sum](http://docs.scipy.org/doc/numpy-1.8.1/reference/generated/numpy.sum.html#numpy.sum)(np.abs(coefs.T), axis=1)
* xx /= xx[-1]
* [plt.plot](http://matplotlib.org/api/_as_gen/matplotlib.pyplot.plot.html#matplotlib.pyplot.plot)(xx, coefs.T)
* ymin, ymax = [plt.ylim](http://matplotlib.org/api/_as_gen/matplotlib.pyplot.ylim.html#matplotlib.pyplot.ylim)()
* [plt.vlines](http://matplotlib.org/api/_as_gen/matplotlib.pyplot.vlines.html#matplotlib.pyplot.vlines)(xx, ymin, ymax, linestyle='dashed')
* [plt.xlabel](http://matplotlib.org/api/_as_gen/matplotlib.pyplot.xlabel.html#matplotlib.pyplot.xlabel)('|coef| / max|coef|')
* [plt.ylabel](http://matplotlib.org/api/_as_gen/matplotlib.pyplot.ylabel.html#matplotlib.pyplot.ylabel)('Coefficients')
* [plt.title](http://matplotlib.org/api/_as_gen/matplotlib.pyplot.title.html#matplotlib.pyplot.title)('LASSO Path')
* [plt.axis](http://matplotlib.org/api/axis_api.html#matplotlib.axis)('tight')
* [plt.show](http://matplotlib.org/api/_as_gen/matplotlib.pyplot.show.html#matplotlib.pyplot.show)()

The Lars algorithm provides the full path of the coefficients along the regularization parameter almost for free, thus a common operation consist of retrieving the path with function [**lars\_path**](http://scikit-learn.org/stable/modules/generated/sklearn.linear_model.lars_path.html#sklearn.linear_model.lars_path)

**1.1.8.1. Mathematical formulation**

The algorithm is similar to forward stepwise regression, but instead of including variables at each step, the estimated parameters are increased in a direction equiangular to each one’s correlations with the residual.

Instead of giving a vector result, the LARS solution consists of a curve denoting the solution for each value of the L1 norm of the parameter vector. The full coefficients path is stored in the array coef\_path\_, which has size (n\_features, max\_features+1). The first column is always zero.

**1.1.9. Orthogonal Matching Pursuit (OMP)**

[**OrthogonalMatchingPursuit**](http://scikit-learn.org/stable/modules/generated/sklearn.linear_model.OrthogonalMatchingPursuit.html#sklearn.linear_model.OrthogonalMatchingPursuit) and [**orthogonal\_mp**](http://scikit-learn.org/stable/modules/generated/sklearn.linear_model.orthogonal_mp.html#sklearn.linear_model.orthogonal_mp) implements the OMP algorithm for approximating the fit of a linear model with constraints imposed on the number of non-zero coefficients (ie. the L 0 pseudo-norm).

Being a forward feature selection method like [Least Angle Regression](http://scikit-learn.org/stable/modules/linear_model.html#least-angle-regression), orthogonal matching pursuit can approximate the optimum solution vector with a fixed number of non-zero elements:

\text{arg\,min\,} ||y - X\gamma||_2^2 \text{ subject to } \
||\gamma||_0 \leq n_{nonzero\_coefs}

Alternatively, orthogonal matching pursuit can target a specific error instead of a specific number of non-zero coefficients. This can be expressed as:

\text{arg\,min\,} ||\gamma||_0 \text{ subject to } ||y-X\gamma||_2^2 \
\leq \text{tol}

OMP is based on a greedy algorithm that includes at each step the atom most highly correlated with the current residual. It is similar to the simpler matching pursuit (MP) method, but better in that at each iteration, the residual is recomputed using an orthogonal projection on the space of the previously chosen dictionary elements.

**Examples:**

* [Orthogonal Matching Pursuit](http://scikit-learn.org/stable/auto_examples/linear_model/plot_omp.html#sphx-glr-auto-examples-linear-model-plot-omp-py)
* **import** **matplotlib.pyplot** **as** **plt**
* **import** **numpy** **as** **np**
* **from** **sklearn.linear\_model** **import** [OrthogonalMatchingPursuit](http://scikit-learn.org/stable/modules/generated/sklearn.linear_model.OrthogonalMatchingPursuit.html#sklearn.linear_model.OrthogonalMatchingPursuit)
* **from** **sklearn.linear\_model** **import** [OrthogonalMatchingPursuitCV](http://scikit-learn.org/stable/modules/generated/sklearn.linear_model.OrthogonalMatchingPursuitCV.html#sklearn.linear_model.OrthogonalMatchingPursuitCV)
* **from** **sklearn.datasets** **import** [make\_sparse\_coded\_signal](http://scikit-learn.org/stable/modules/generated/sklearn.datasets.make_sparse_coded_signal.html#sklearn.datasets.make_sparse_coded_signal)
* n\_components, n\_features = 512, 100
* n\_nonzero\_coefs = 17
* *# generate the data*
* *###################*
* *# y = Xw*
* *# |x|\_0 = n\_nonzero\_coefs*
* y, X, w = [make\_sparse\_coded\_signal](http://scikit-learn.org/stable/modules/generated/sklearn.datasets.make_sparse_coded_signal.html#sklearn.datasets.make_sparse_coded_signal)(n\_samples=1,
* n\_components=n\_components,
* n\_features=n\_features,
* n\_nonzero\_coefs=n\_nonzero\_coefs,
* random\_state=0)
* idx, = w.nonzero()
* *# distort the clean signal*
* y\_noisy = y + 0.05 \* [np.random.randn](http://docs.scipy.org/doc/numpy-1.8.1/reference/generated/numpy.random.randn.html#numpy.random.randn)(len(y))
* *# plot the sparse signal*
* [plt.figure](http://matplotlib.org/api/_as_gen/matplotlib.figure.AxesStack.html#matplotlib.figure)(figsize=(7, 7))
* [plt.subplot](http://matplotlib.org/api/_as_gen/matplotlib.pyplot.subplot.html#matplotlib.pyplot.subplot)(4, 1, 1)
* [plt.xlim](http://matplotlib.org/api/_as_gen/matplotlib.pyplot.xlim.html#matplotlib.pyplot.xlim)(0, 512)
* [plt.title](http://matplotlib.org/api/_as_gen/matplotlib.pyplot.title.html#matplotlib.pyplot.title)("Sparse signal")
* [plt.stem](http://matplotlib.org/api/_as_gen/matplotlib.pyplot.stem.html#matplotlib.pyplot.stem)(idx, w[idx])
* *# plot the noise-free reconstruction*
* omp = [OrthogonalMatchingPursuit](http://scikit-learn.org/stable/modules/generated/sklearn.linear_model.OrthogonalMatchingPursuit.html#sklearn.linear_model.OrthogonalMatchingPursuit)(n\_nonzero\_coefs=n\_nonzero\_coefs)
* omp.fit(X, y)
* coef = omp.coef\_
* idx\_r, = coef.nonzero()
* [plt.subplot](http://matplotlib.org/api/_as_gen/matplotlib.pyplot.subplot.html#matplotlib.pyplot.subplot)(4, 1, 2)
* [plt.xlim](http://matplotlib.org/api/_as_gen/matplotlib.pyplot.xlim.html#matplotlib.pyplot.xlim)(0, 512)
* [plt.title](http://matplotlib.org/api/_as_gen/matplotlib.pyplot.title.html#matplotlib.pyplot.title)("Recovered signal from noise-free measurements")
* [plt.stem](http://matplotlib.org/api/_as_gen/matplotlib.pyplot.stem.html#matplotlib.pyplot.stem)(idx\_r, coef[idx\_r])
* *# plot the noisy reconstruction*
* omp.fit(X, y\_noisy)
* coef = omp.coef\_
* idx\_r, = coef.nonzero()
* [plt.subplot](http://matplotlib.org/api/_as_gen/matplotlib.pyplot.subplot.html#matplotlib.pyplot.subplot)(4, 1, 3)
* [plt.xlim](http://matplotlib.org/api/_as_gen/matplotlib.pyplot.xlim.html#matplotlib.pyplot.xlim)(0, 512)
* [plt.title](http://matplotlib.org/api/_as_gen/matplotlib.pyplot.title.html#matplotlib.pyplot.title)("Recovered signal from noisy measurements")
* [plt.stem](http://matplotlib.org/api/_as_gen/matplotlib.pyplot.stem.html#matplotlib.pyplot.stem)(idx\_r, coef[idx\_r])
* *# plot the noisy reconstruction with number of non-zeros set by CV*
* omp\_cv = [OrthogonalMatchingPursuitCV](http://scikit-learn.org/stable/modules/generated/sklearn.linear_model.OrthogonalMatchingPursuitCV.html#sklearn.linear_model.OrthogonalMatchingPursuitCV)()
* omp\_cv.fit(X, y\_noisy)
* coef = omp\_cv.coef\_
* idx\_r, = coef.nonzero()
* [plt.subplot](http://matplotlib.org/api/_as_gen/matplotlib.pyplot.subplot.html#matplotlib.pyplot.subplot)(4, 1, 4)
* [plt.xlim](http://matplotlib.org/api/_as_gen/matplotlib.pyplot.xlim.html#matplotlib.pyplot.xlim)(0, 512)
* [plt.title](http://matplotlib.org/api/_as_gen/matplotlib.pyplot.title.html#matplotlib.pyplot.title)("Recovered signal from noisy measurements with CV")
* [plt.stem](http://matplotlib.org/api/_as_gen/matplotlib.pyplot.stem.html#matplotlib.pyplot.stem)(idx\_r, coef[idx\_r])
* [plt.subplots\_adjust](http://matplotlib.org/api/_as_gen/matplotlib.pyplot.subplots_adjust.html#matplotlib.pyplot.subplots_adjust)(0.06, 0.04, 0.94, 0.90, 0.20, 0.38)
* [plt.suptitle](http://matplotlib.org/api/_as_gen/matplotlib.pyplot.suptitle.html#matplotlib.pyplot.suptitle)('Sparse signal recovery with Orthogonal Matching Pursuit',
* fontsize=16)
* [plt.show](http://matplotlib.org/api/_as_gen/matplotlib.pyplot.show.html#matplotlib.pyplot.show)()

**1.1.10. Bayesian Regression**

Bayesian regression techniques can be used to include regularization parameters in the estimation procedure: the regularization parameter is not set in a hard sense but tuned to the data at hand.

This can be done by introducing [uninformative priors](https://en.wikipedia.org/wiki/Non-informative_prior#Uninformative_priors) over the hyper parameters of the model. The \ell_{2} regularization used in [Ridge Regression](http://scikit-learn.org/stable/modules/linear_model.html#id2) is equivalent to finding a maximum a posteriori estimation under a Gaussian prior over the parameters wwith precision \lambda^{-1}. Instead of setting lambda manually, it is possible to treat it as a random variable to be estimated from the data.

To obtain a fully probabilistic model, the output y is assumed to be Gaussian distributed around X w:

p(y|X,w,\alpha) = \mathcal{N}(y|X w,\alpha)

Alpha is again treated as a random variable that is to be estimated from the data.

The advantages of Bayesian Regression are:

* It adapts to the data at hand.
* It can be used to include regularization parameters in the estimation procedure.

The disadvantages of Bayesian regression include:

* Inference of the model can be time consuming.

**1.1.10.1. Bayesian Ridge Regression**

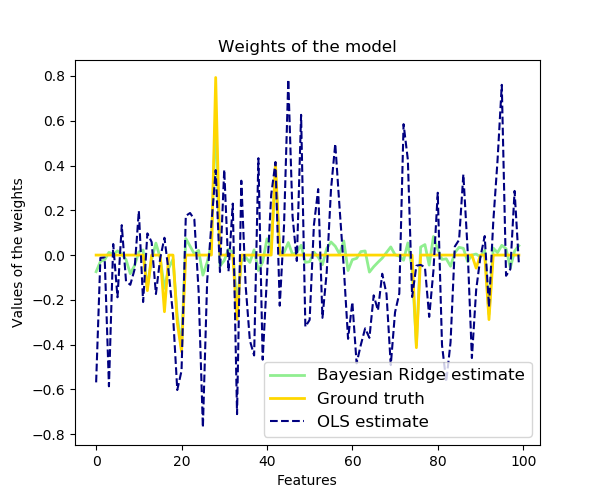
[**BayesianRidge**](http://scikit-learn.org/stable/modules/generated/sklearn.linear_model.BayesianRidge.html#sklearn.linear_model.BayesianRidge) estimates a probabilistic model of the regression problem as described above. The prior for the parameter w is given by a spherical Gaussian:

p(w|\lambda) =
\mathcal{N}(w|0,\lambda^{-1}\bold{I_{p}})

The priors over \alpha and \lambda are chosen to be [gamma distributions](https://en.wikipedia.org/wiki/Gamma_distribution), the conjugate prior for the precision of the Gaussian.

The resulting model is called *Bayesian Ridge Regression*, and is similar to the classical [**Ridge**](http://scikit-learn.org/stable/modules/generated/sklearn.linear_model.Ridge.html#sklearn.linear_model.Ridge). The parameters w, \alpha and \lambda are estimated jointly during the fit of the model. The remaining hyperparameters are the parameters of the gamma priors over \alpha and \lambda. These are usually chosen to be *non-informative*. The parameters are estimated by maximizing the *marginal log likelihood*.

By default \alpha_1 = \alpha_2 =  \lambda_1 = \lambda_2 = 10^{-6}.

[](http://scikit-learn.org/stable/auto_examples/linear_model/plot_bayesian_ridge.html)

Bayesian Ridge Regression is used for regression:

>>>

**>>> from** **sklearn** **import** linear\_model

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

**>>>** Y = [0., 1., 2., 3.]

**>>>** reg = linear\_model.BayesianRidge()

**>>>** reg.fit(X, Y)

BayesianRidge(alpha\_1=1e-06, alpha\_2=1e-06, compute\_score=False, copy\_X=True,

fit\_intercept=True, lambda\_1=1e-06, lambda\_2=1e-06, n\_iter=300,

normalize=False, tol=0.001, verbose=False)

After being fitted, the model can then be used to predict new values:

>>>

**>>>** reg.predict ([[1, 0.]])

array([ 0.50000013])

The weights w of the model can be access:

>>>

**>>>** reg.coef\_

array([ 0.49999993, 0.49999993])

Due to the Bayesian framework, the weights found are slightly different to the ones found by [Ordinary Least Squares](http://scikit-learn.org/stable/modules/linear_model.html#ordinary-least-squares). However, Bayesian Ridge Regression is more robust to ill-posed problem.

**Examples:**

* [Bayesian Ridge Regression](http://scikit-learn.org/stable/auto_examples/linear_model/plot_bayesian_ridge.html#sphx-glr-auto-examples-linear-model-plot-bayesian-ridge-py)

Computes a Bayesian Ridge Regression on a synthetic dataset.

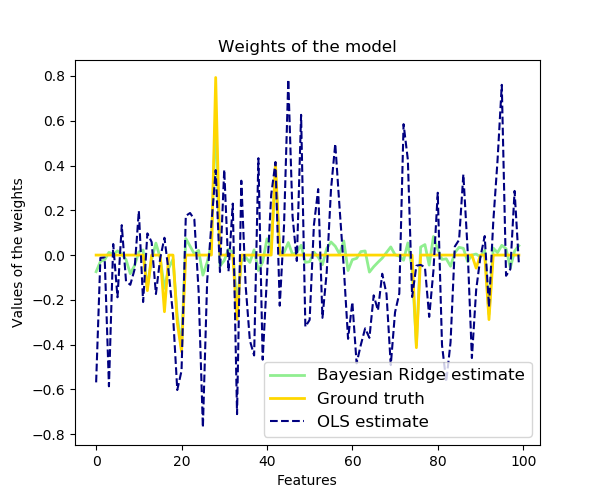
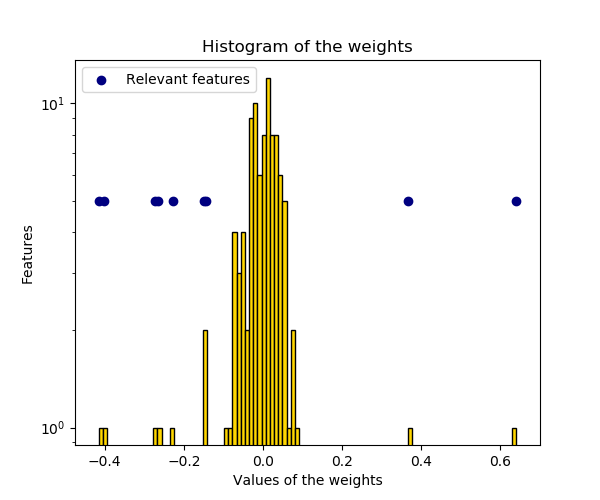
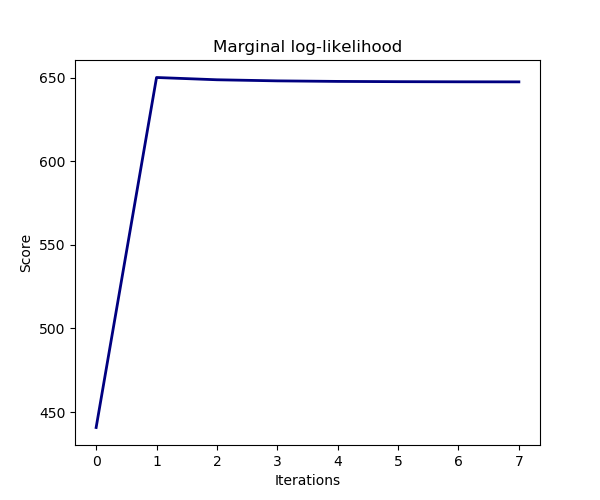
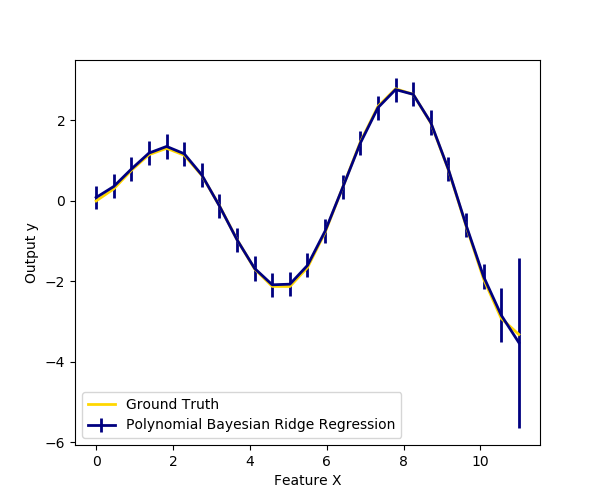
See [Bayesian Ridge Regression](http://scikit-learn.org/stable/modules/linear_model.html#bayesian-ridge-regression) for more information on the regressor.

Compared to the OLS (ordinary least squares) estimator, the coefficient weights are slightly shifted toward zeros, which stabilises them.

As the prior on the weights is a Gaussian prior, the histogram of the estimated weights is Gaussian.

The estimation of the model is done by iteratively maximizing the marginal log-likelihood of the observations.

We also plot predictions and uncertainties for Bayesian Ridge Regression for one dimensional regression using polynomial feature expansion. Note the uncertainty starts going up on the right side of the plot. This is because these test samples are outside of the range of the training samples.

* [](http://scikit-learn.org/stable/_images/sphx_glr_plot_bayesian_ridge_001.png)
* [](http://scikit-learn.org/stable/_images/sphx_glr_plot_bayesian_ridge_002.png)
* [](http://scikit-learn.org/stable/_images/sphx_glr_plot_bayesian_ridge_003.png)
* [](http://scikit-learn.org/stable/_images/sphx_glr_plot_bayesian_ridge_004.png)

**print**(\_\_doc\_\_)

**import** **numpy** **as** **np**

**import** **matplotlib.pyplot** **as** **plt**

**from** **scipy** **import** stats

**from** **sklearn.linear\_model** **import** [BayesianRidge](http://scikit-learn.org/stable/modules/generated/sklearn.linear_model.BayesianRidge.html#sklearn.linear_model.BayesianRidge), [LinearRegression](http://scikit-learn.org/stable/modules/generated/sklearn.linear_model.LinearRegression.html#sklearn.linear_model.LinearRegression)

*# #############################################################################*

*# Generating simulated data with Gaussian weights*

[np.random.seed](http://docs.scipy.org/doc/numpy-1.8.1/reference/generated/numpy.random.seed.html#numpy.random.seed)(0)

n\_samples, n\_features = 100, 100

X = [np.random.randn](http://docs.scipy.org/doc/numpy-1.8.1/reference/generated/numpy.random.randn.html#numpy.random.randn)(n\_samples, n\_features) *# Create Gaussian data*

*# Create weights with a precision lambda\_ of 4.*

lambda\_ = 4.

w = [np.zeros](http://docs.scipy.org/doc/numpy-1.8.1/reference/generated/numpy.zeros.html#numpy.zeros)(n\_features)

*# Only keep 10 weights of interest*

relevant\_features = [np.random.randint](http://docs.scipy.org/doc/numpy-1.8.1/reference/generated/numpy.random.randint.html#numpy.random.randint)(0, n\_features, 10)

**for** i **in** relevant\_features:

w[i] = stats.norm.rvs(loc=0, scale=1. / [np.sqrt](http://docs.scipy.org/doc/numpy-1.8.1/reference/generated/numpy.sqrt.html#numpy.sqrt)(lambda\_))

*# Create noise with a precision alpha of 50.*

alpha\_ = 50.

noise = stats.norm.rvs(loc=0, scale=1. / [np.sqrt](http://docs.scipy.org/doc/numpy-1.8.1/reference/generated/numpy.sqrt.html#numpy.sqrt)(alpha\_), size=n\_samples)

*# Create the target*

y = [np.dot](http://docs.scipy.org/doc/numpy-1.8.1/reference/generated/numpy.dot.html#numpy.dot)(X, w) + noise

*# #############################################################################*

*# Fit the Bayesian Ridge Regression and an OLS for comparison*

clf = [BayesianRidge](http://scikit-learn.org/stable/modules/generated/sklearn.linear_model.BayesianRidge.html#sklearn.linear_model.BayesianRidge)(compute\_score=True)

clf.fit(X, y)

ols = [LinearRegression](http://scikit-learn.org/stable/modules/generated/sklearn.linear_model.LinearRegression.html#sklearn.linear_model.LinearRegression)()

ols.fit(X, y)

*# #############################################################################*

*# Plot true weights, estimated weights, histogram of the weights, and*

*# predictions with standard deviations*

lw = 2

[plt.figure](http://matplotlib.org/api/_as_gen/matplotlib.figure.AxesStack.html#matplotlib.figure)(figsize=(6, 5))

[plt.title](http://matplotlib.org/api/_as_gen/matplotlib.pyplot.title.html#matplotlib.pyplot.title)("Weights of the model")

[plt.plot](http://matplotlib.org/api/_as_gen/matplotlib.pyplot.plot.html#matplotlib.pyplot.plot)(clf.coef\_, color='lightgreen', linewidth=lw,

label="Bayesian Ridge estimate")

[plt.plot](http://matplotlib.org/api/_as_gen/matplotlib.pyplot.plot.html#matplotlib.pyplot.plot)(w, color='gold', linewidth=lw, label="Ground truth")

[plt.plot](http://matplotlib.org/api/_as_gen/matplotlib.pyplot.plot.html#matplotlib.pyplot.plot)(ols.coef\_, color='navy', linestyle='--', label="OLS estimate")

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

[plt.ylabel](http://matplotlib.org/api/_as_gen/matplotlib.pyplot.ylabel.html#matplotlib.pyplot.ylabel)("Values of the weights")

[plt.legend](http://matplotlib.org/api/legend_api.html#matplotlib.legend)(loc="best", prop=dict(size=12))

[plt.figure](http://matplotlib.org/api/_as_gen/matplotlib.figure.AxesStack.html#matplotlib.figure)(figsize=(6, 5))

[plt.title](http://matplotlib.org/api/_as_gen/matplotlib.pyplot.title.html#matplotlib.pyplot.title)("Histogram of the weights")

[plt.hist](http://matplotlib.org/api/_as_gen/matplotlib.pyplot.hist.html#matplotlib.pyplot.hist)(clf.coef\_, bins=n\_features, color='gold', log=True,

edgecolor='black')

[plt.scatter](http://matplotlib.org/api/_as_gen/matplotlib.pyplot.scatter.html#matplotlib.pyplot.scatter)(clf.coef\_[relevant\_features], 5 \* [np.ones](http://docs.scipy.org/doc/numpy-1.8.1/reference/generated/numpy.ones.html#numpy.ones)(len(relevant\_features)),

color='navy', label="Relevant features")

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

[plt.xlabel](http://matplotlib.org/api/_as_gen/matplotlib.pyplot.xlabel.html#matplotlib.pyplot.xlabel)("Values of the weights")

[plt.legend](http://matplotlib.org/api/legend_api.html#matplotlib.legend)(loc="upper left")

[plt.figure](http://matplotlib.org/api/_as_gen/matplotlib.figure.AxesStack.html#matplotlib.figure)(figsize=(6, 5))

[plt.title](http://matplotlib.org/api/_as_gen/matplotlib.pyplot.title.html#matplotlib.pyplot.title)("Marginal log-likelihood")

[plt.plot](http://matplotlib.org/api/_as_gen/matplotlib.pyplot.plot.html#matplotlib.pyplot.plot)(clf.scores\_, color='navy', linewidth=lw)

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

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

*# Plotting some predictions for polynomial regression*

**def** f(x, noise\_amount):

y = [np.sqrt](http://docs.scipy.org/doc/numpy-1.8.1/reference/generated/numpy.sqrt.html#numpy.sqrt)(x) \* [np.sin](http://docs.scipy.org/doc/numpy-1.8.1/reference/generated/numpy.sin.html#numpy.sin)(x)

noise = [np.random.normal](http://docs.scipy.org/doc/numpy-1.8.1/reference/generated/numpy.random.normal.html#numpy.random.normal)(0, 1, len(x))

**return** y + noise\_amount \* noise

degree = 10

X = [np.linspace](http://docs.scipy.org/doc/numpy-1.8.1/reference/generated/numpy.linspace.html#numpy.linspace)(0, 10, 100)

y = f(X, noise\_amount=0.1)

clf\_poly = [BayesianRidge](http://scikit-learn.org/stable/modules/generated/sklearn.linear_model.BayesianRidge.html#sklearn.linear_model.BayesianRidge)()

clf\_poly.fit([np.vander](http://docs.scipy.org/doc/numpy-1.8.1/reference/generated/numpy.vander.html#numpy.vander)(X, degree), y)

X\_plot = [np.linspace](http://docs.scipy.org/doc/numpy-1.8.1/reference/generated/numpy.linspace.html#numpy.linspace)(0, 11, 25)

y\_plot = f(X\_plot, noise\_amount=0)

y\_mean, y\_std = clf\_poly.predict([np.vander](http://docs.scipy.org/doc/numpy-1.8.1/reference/generated/numpy.vander.html#numpy.vander)(X\_plot, degree), return\_std=True)

[plt.figure](http://matplotlib.org/api/_as_gen/matplotlib.figure.AxesStack.html#matplotlib.figure)(figsize=(6, 5))

[plt.errorbar](http://matplotlib.org/api/_as_gen/matplotlib.pyplot.errorbar.html#matplotlib.pyplot.errorbar)(X\_plot, y\_mean, y\_std, color='navy',

label="Polynomial Bayesian Ridge Regression", linewidth=lw)

[plt.plot](http://matplotlib.org/api/_as_gen/matplotlib.pyplot.plot.html#matplotlib.pyplot.plot)(X\_plot, y\_plot, color='gold', linewidth=lw,

label="Ground Truth")

[plt.ylabel](http://matplotlib.org/api/_as_gen/matplotlib.pyplot.ylabel.html#matplotlib.pyplot.ylabel)("Output y")

[plt.xlabel](http://matplotlib.org/api/_as_gen/matplotlib.pyplot.xlabel.html#matplotlib.pyplot.xlabel)("Feature X")

[plt.legend](http://matplotlib.org/api/legend_api.html#matplotlib.legend)(loc="lower left")

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

**1.1.10.2. Automatic Relevance Determination - ARD**

[**ARDRegression**](http://scikit-learn.org/stable/modules/generated/sklearn.linear_model.ARDRegression.html#sklearn.linear_model.ARDRegression) is very similar to [Bayesian Ridge Regression](http://scikit-learn.org/stable/modules/linear_model.html#id10), but can lead to sparser weights w [[1]](http://scikit-learn.org/stable/modules/linear_model.html#id15) [[2]](http://scikit-learn.org/stable/modules/linear_model.html#id16). [**ARDRegression**](http://scikit-learn.org/stable/modules/generated/sklearn.linear_model.ARDRegression.html#sklearn.linear_model.ARDRegression)poses a different prior over w, by dropping the assumption of the Gaussian being spherical.

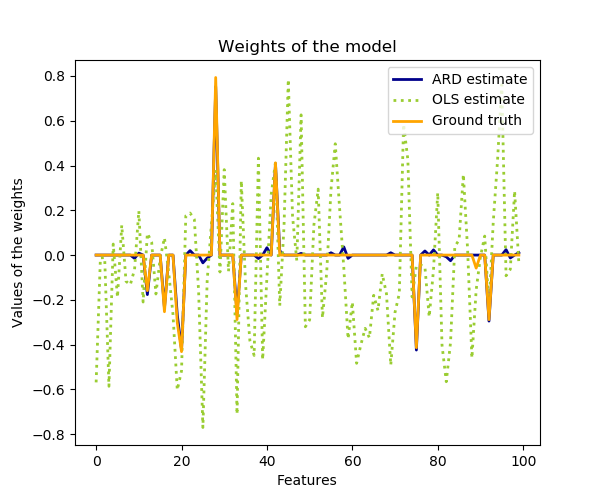
Instead, the distribution over w is assumed to be an axis-parallel, elliptical Gaussian distribution.

This means each weight w_{i} is drawn from a Gaussian distribution, centered on zero and with a precision \lambda_{i}:

p(w|\lambda) = \mathcal{N}(w|0,A^{-1})

with diag \; (A) = \lambda = \{\lambda_{1},...,\lambda_{p}\}.

In contrast to [Bayesian Ridge Regression](http://scikit-learn.org/stable/modules/linear_model.html#id10), each coordinate of w_{i} has its own standard deviation \lambda_i. The prior over all \lambda_i is chosen to be the same gamma distribution given by hyperparameters \lambda_1 and \lambda_2.

[](http://scikit-learn.org/stable/auto_examples/linear_model/plot_ard.html)

ARD is also known in the literature as *Sparse Bayesian Learning* and *Relevance Vector Machine* [[3]](http://scikit-learn.org/stable/modules/linear_model.html#id17) [[4]](http://scikit-learn.org/stable/modules/linear_model.html#id18).

**Examples:**

* [Automatic Relevance Determination Regression (ARD)](http://scikit-learn.org/stable/auto_examples/linear_model/plot_ard.html#sphx-glr-auto-examples-linear-model-plot-ard-py)
* **import** **numpy** **as** **np**
* **import** **matplotlib.pyplot** **as** **plt**
* **from** **scipy** **import** stats
* **from** **sklearn.linear\_model** **import** [ARDRegression](http://scikit-learn.org/stable/modules/generated/sklearn.linear_model.ARDRegression.html#sklearn.linear_model.ARDRegression), [LinearRegression](http://scikit-learn.org/stable/modules/generated/sklearn.linear_model.LinearRegression.html#sklearn.linear_model.LinearRegression)
* *# #############################################################################*
* *# Generating simulated data with Gaussian weights*
* *# Parameters of the example*
* [np.random.seed](http://docs.scipy.org/doc/numpy-1.8.1/reference/generated/numpy.random.seed.html#numpy.random.seed)(0)
* n\_samples, n\_features = 100, 100
* *# Create Gaussian data*
* X = [np.random.randn](http://docs.scipy.org/doc/numpy-1.8.1/reference/generated/numpy.random.randn.html#numpy.random.randn)(n\_samples, n\_features)
* *# Create weights with a precision lambda\_ of 4.*
* lambda\_ = 4.
* w = [np.zeros](http://docs.scipy.org/doc/numpy-1.8.1/reference/generated/numpy.zeros.html#numpy.zeros)(n\_features)
* *# Only keep 10 weights of interest*
* relevant\_features = [np.random.randint](http://docs.scipy.org/doc/numpy-1.8.1/reference/generated/numpy.random.randint.html#numpy.random.randint)(0, n\_features, 10)
* **for** i **in** relevant\_features:
* w[i] = stats.norm.rvs(loc=0, scale=1. / [np.sqrt](http://docs.scipy.org/doc/numpy-1.8.1/reference/generated/numpy.sqrt.html#numpy.sqrt)(lambda\_))
* *# Create noise with a precision alpha of 50.*
* alpha\_ = 50.
* noise = stats.norm.rvs(loc=0, scale=1. / [np.sqrt](http://docs.scipy.org/doc/numpy-1.8.1/reference/generated/numpy.sqrt.html#numpy.sqrt)(alpha\_), size=n\_samples)
* *# Create the target*
* y = [np.dot](http://docs.scipy.org/doc/numpy-1.8.1/reference/generated/numpy.dot.html#numpy.dot)(X, w) + noise
* *# #############################################################################*
* *# Fit the ARD Regression*
* clf = [ARDRegression](http://scikit-learn.org/stable/modules/generated/sklearn.linear_model.ARDRegression.html#sklearn.linear_model.ARDRegression)(compute\_score=True)
* clf.fit(X, y)
* ols = [LinearRegression](http://scikit-learn.org/stable/modules/generated/sklearn.linear_model.LinearRegression.html#sklearn.linear_model.LinearRegression)()
* ols.fit(X, y)
* *# #############################################################################*
* *# Plot the true weights, the estimated weights, the histogram of the*
* *# weights, and predictions with standard deviations*
* [plt.figure](http://matplotlib.org/api/_as_gen/matplotlib.figure.AxesStack.html#matplotlib.figure)(figsize=(6, 5))
* [plt.title](http://matplotlib.org/api/_as_gen/matplotlib.pyplot.title.html#matplotlib.pyplot.title)("Weights of the model")
* [plt.plot](http://matplotlib.org/api/_as_gen/matplotlib.pyplot.plot.html#matplotlib.pyplot.plot)(clf.coef\_, color='darkblue', linestyle='-', linewidth=2,
* label="ARD estimate")
* [plt.plot](http://matplotlib.org/api/_as_gen/matplotlib.pyplot.plot.html#matplotlib.pyplot.plot)(ols.coef\_, color='yellowgreen', linestyle=':', linewidth=2,
* label="OLS estimate")
* [plt.plot](http://matplotlib.org/api/_as_gen/matplotlib.pyplot.plot.html#matplotlib.pyplot.plot)(w, color='orange', linestyle='-', linewidth=2, label="Ground truth")
* [plt.xlabel](http://matplotlib.org/api/_as_gen/matplotlib.pyplot.xlabel.html#matplotlib.pyplot.xlabel)("Features")
* [plt.ylabel](http://matplotlib.org/api/_as_gen/matplotlib.pyplot.ylabel.html#matplotlib.pyplot.ylabel)("Values of the weights")
* [plt.legend](http://matplotlib.org/api/legend_api.html#matplotlib.legend)(loc=1)
* [plt.figure](http://matplotlib.org/api/_as_gen/matplotlib.figure.AxesStack.html#matplotlib.figure)(figsize=(6, 5))
* [plt.title](http://matplotlib.org/api/_as_gen/matplotlib.pyplot.title.html#matplotlib.pyplot.title)("Histogram of the weights")
* [plt.hist](http://matplotlib.org/api/_as_gen/matplotlib.pyplot.hist.html#matplotlib.pyplot.hist)(clf.coef\_, bins=n\_features, color='navy', log=True)
* [plt.scatter](http://matplotlib.org/api/_as_gen/matplotlib.pyplot.scatter.html#matplotlib.pyplot.scatter)(clf.coef\_[relevant\_features], 5 \* [np.ones](http://docs.scipy.org/doc/numpy-1.8.1/reference/generated/numpy.ones.html#numpy.ones)(len(relevant\_features)),
* color='gold', marker='o', label="Relevant features")
* [plt.ylabel](http://matplotlib.org/api/_as_gen/matplotlib.pyplot.ylabel.html#matplotlib.pyplot.ylabel)("Features")
* [plt.xlabel](http://matplotlib.org/api/_as_gen/matplotlib.pyplot.xlabel.html#matplotlib.pyplot.xlabel)("Values of the weights")
* [plt.legend](http://matplotlib.org/api/legend_api.html#matplotlib.legend)(loc=1)
* [plt.figure](http://matplotlib.org/api/_as_gen/matplotlib.figure.AxesStack.html#matplotlib.figure)(figsize=(6, 5))
* [plt.title](http://matplotlib.org/api/_as_gen/matplotlib.pyplot.title.html#matplotlib.pyplot.title)("Marginal log-likelihood")
* [plt.plot](http://matplotlib.org/api/_as_gen/matplotlib.pyplot.plot.html#matplotlib.pyplot.plot)(clf.scores\_, color='navy', linewidth=2)
* [plt.ylabel](http://matplotlib.org/api/_as_gen/matplotlib.pyplot.ylabel.html#matplotlib.pyplot.ylabel)("Score")
* [plt.xlabel](http://matplotlib.org/api/_as_gen/matplotlib.pyplot.xlabel.html#matplotlib.pyplot.xlabel)("Iterations")
* *# Plotting some predictions for polynomial regression*
* **def** f(x, noise\_amount):
* y = [np.sqrt](http://docs.scipy.org/doc/numpy-1.8.1/reference/generated/numpy.sqrt.html#numpy.sqrt)(x) \* [np.sin](http://docs.scipy.org/doc/numpy-1.8.1/reference/generated/numpy.sin.html#numpy.sin)(x)
* noise = [np.random.normal](http://docs.scipy.org/doc/numpy-1.8.1/reference/generated/numpy.random.normal.html#numpy.random.normal)(0, 1, len(x))
* **return** y + noise\_amount \* noise
* degree = 10
* X = [np.linspace](http://docs.scipy.org/doc/numpy-1.8.1/reference/generated/numpy.linspace.html#numpy.linspace)(0, 10, 100)
* y = f(X, noise\_amount=1)
* clf\_poly = [ARDRegression](http://scikit-learn.org/stable/modules/generated/sklearn.linear_model.ARDRegression.html#sklearn.linear_model.ARDRegression)(threshold\_lambda=1e5)
* clf\_poly.fit([np.vander](http://docs.scipy.org/doc/numpy-1.8.1/reference/generated/numpy.vander.html#numpy.vander)(X, degree), y)
* X\_plot = [np.linspace](http://docs.scipy.org/doc/numpy-1.8.1/reference/generated/numpy.linspace.html#numpy.linspace)(0, 11, 25)
* y\_plot = f(X\_plot, noise\_amount=0)
* y\_mean, y\_std = clf\_poly.predict([np.vander](http://docs.scipy.org/doc/numpy-1.8.1/reference/generated/numpy.vander.html#numpy.vander)(X\_plot, degree), return\_std=True)
* [plt.figure](http://matplotlib.org/api/_as_gen/matplotlib.figure.AxesStack.html#matplotlib.figure)(figsize=(6, 5))
* [plt.errorbar](http://matplotlib.org/api/_as_gen/matplotlib.pyplot.errorbar.html#matplotlib.pyplot.errorbar)(X\_plot, y\_mean, y\_std, color='navy',
* label="Polynomial ARD", linewidth=2)
* [plt.plot](http://matplotlib.org/api/_as_gen/matplotlib.pyplot.plot.html#matplotlib.pyplot.plot)(X\_plot, y\_plot, color='gold', linewidth=2,
* label="Ground Truth")
* [plt.ylabel](http://matplotlib.org/api/_as_gen/matplotlib.pyplot.ylabel.html#matplotlib.pyplot.ylabel)("Output y")
* [plt.xlabel](http://matplotlib.org/api/_as_gen/matplotlib.pyplot.xlabel.html#matplotlib.pyplot.xlabel)("Feature X")
* [plt.legend](http://matplotlib.org/api/legend_api.html#matplotlib.legend)(loc="lower left")
* [plt.show](http://matplotlib.org/api/_as_gen/matplotlib.pyplot.show.html#matplotlib.pyplot.show)()

**1.1.11. Logistic regression**

Logistic regression, despite its name, is a linear model for classification rather than regression. Logistic regression is also known in the literature as logit regression, maximum-entropy classification (MaxEnt) or the log-linear classifier. In this model, the probabilities describing the possible outcomes of a single trial are modeled using a [logistic function](https://en.wikipedia.org/wiki/Logistic_function).

The implementation of logistic regression in scikit-learn can be accessed from class [**LogisticRegression**](http://scikit-learn.org/stable/modules/generated/sklearn.linear_model.LogisticRegression.html#sklearn.linear_model.LogisticRegression). This implementation can fit binary, One-vs- Rest, or multinomial logistic regression with optional L2 or L1 regularization.

As an optimization problem, binary class L2 penalized logistic regression minimizes the following cost function:

\underset{w, c}{min\,} \frac{1}{2}w^T w + C \sum_{i=1}^n \log(\exp(- y_i (X_i^T w + c)) + 1) .

Similarly, L1 regularized logistic regression solves the following optimization problem

\underset{w, c}{min\,} \|w\|_1 + C \sum_{i=1}^n \log(\exp(- y_i (X_i^T w + c)) + 1) .

The solvers implemented in the class [**LogisticRegression**](http://scikit-learn.org/stable/modules/generated/sklearn.linear_model.LogisticRegression.html#sklearn.linear_model.LogisticRegression) are “liblinear”, “newton-cg”, “lbfgs”, “sag” and “saga”:

The solver “liblinear” uses a coordinate descent (CD) algorithm, and relies on the excellent C++ [LIBLINEAR library](http://www.csie.ntu.edu.tw/~cjlin/liblinear/), which is shipped with scikit-learn. However, the CD algorithm implemented in liblinear cannot learn a true multinomial (multiclass) model; instead, the optimization problem is decomposed in a “one-vs-rest” fashion so separate binary classifiers are trained for all classes. This happens under the hood, so [**LogisticRegression**](http://scikit-learn.org/stable/modules/generated/sklearn.linear_model.LogisticRegression.html#sklearn.linear_model.LogisticRegression) instances using this solver behave as multiclass classifiers. For L1 penalization [**sklearn.svm.l1\_min\_c**](http://scikit-learn.org/stable/modules/generated/sklearn.svm.l1_min_c.html#sklearn.svm.l1_min_c) allows to calculate the lower bound for C in order to get a non “null” (all feature weights to zero) model.

The “lbfgs”, “sag” and “newton-cg” solvers only support L2 penalization and are found to converge faster for some high dimensional data. Setting multi\_class to “multinomial” with these solvers learns a true multinomial logistic regression model [[5]](http://scikit-learn.org/stable/modules/linear_model.html#id23), which means that its probability estimates should be better calibrated than the default “one-vs-rest” setting.

The “sag” solver uses a Stochastic Average Gradient descent [[6]](http://scikit-learn.org/stable/modules/linear_model.html#id24). It is faster than other solvers for large datasets, when both the number of samples and the number of features are large.

The “saga” solver [[7]](http://scikit-learn.org/stable/modules/linear_model.html#id25) is a variant of “sag” that also supports the non-smooth penalty=”l1” option. This is therefore the solver of choice for sparse multinomial logistic regression.

In a nutshell, one may choose the solver with the following rules:

| **Case** | **Solver** |
| --- | --- |
| L1 penalty | “liblinear” or “saga” |
| Multinomial loss | “lbfgs”, “sag”, “saga” or “newton-cg” |
| Very Large dataset (n\_samples) | “sag” or “saga” |

The “saga” solver is often the best choice. The “liblinear” solver is used by default for historical reasons.

For large dataset, you may also consider using [**SGDClassifier**](http://scikit-learn.org/stable/modules/generated/sklearn.linear_model.SGDClassifier.html#sklearn.linear_model.SGDClassifier) with ‘log’ loss.

**Examples:**

* [L1 Penalty and Sparsity in Logistic Regression](http://scikit-learn.org/stable/auto_examples/linear_model/plot_logistic_l1_l2_sparsity.html#sphx-glr-auto-examples-linear-model-plot-logistic-l1-l2-sparsity-py)
* [Path with L1- Logistic Regression](http://scikit-learn.org/stable/auto_examples/linear_model/plot_logistic_path.html#sphx-glr-auto-examples-linear-model-plot-logistic-path-py)
* *# Author: Alexandre Gramfort <alexandre.gramfort@inria.fr>*
* *# License: BSD 3 clause*
* **from** **datetime** **import** datetime
* **import** **numpy** **as** **np**
* **import** **matplotlib.pyplot** **as** **plt**
* **from** **sklearn** **import** linear\_model
* **from** **sklearn** **import** datasets
* **from** **sklearn.svm** **import** [l1\_min\_c](http://scikit-learn.org/stable/modules/generated/sklearn.svm.l1_min_c.html#sklearn.svm.l1_min_c)
* iris = [datasets.load\_iris](http://scikit-learn.org/stable/modules/generated/sklearn.datasets.load_iris.html#sklearn.datasets.load_iris)()
* X = iris.data
* y = iris.target
* X = X[y != 2]
* y = y[y != 2]
* X -= [np.mean](http://docs.scipy.org/doc/numpy-1.8.1/reference/generated/numpy.mean.html#numpy.mean)(X, 0)
* *# #############################################################################*
* *# Demo path functions*
* cs = [l1\_min\_c](http://scikit-learn.org/stable/modules/generated/sklearn.svm.l1_min_c.html#sklearn.svm.l1_min_c)(X, y, loss='log') \* [np.logspace](http://docs.scipy.org/doc/numpy-1.8.1/reference/generated/numpy.logspace.html#numpy.logspace)(0, 3)
* **print**("Computing regularization path ...")
* start = datetime.now()
* clf = [linear\_model.LogisticRegression](http://scikit-learn.org/stable/modules/generated/sklearn.linear_model.LogisticRegression.html#sklearn.linear_model.LogisticRegression)(C=1.0, penalty='l1', tol=1e-6)
* coefs\_ = []
* **for** c **in** cs:
* clf.set\_params(C=c)
* clf.fit(X, y)
* coefs\_.append(clf.coef\_.ravel().copy())
* **print**("This took ", datetime.now() - start)
* coefs\_ = [np.array](http://docs.scipy.org/doc/numpy-1.8.1/reference/generated/numpy.array.html#numpy.array)(coefs\_)
* [plt.plot](http://matplotlib.org/api/_as_gen/matplotlib.pyplot.plot.html#matplotlib.pyplot.plot)([np.log10](http://docs.scipy.org/doc/numpy-1.8.1/reference/generated/numpy.log10.html#numpy.log10)(cs), coefs\_)
* ymin, ymax = [plt.ylim](http://matplotlib.org/api/_as_gen/matplotlib.pyplot.ylim.html#matplotlib.pyplot.ylim)()
* [plt.xlabel](http://matplotlib.org/api/_as_gen/matplotlib.pyplot.xlabel.html#matplotlib.pyplot.xlabel)('log(C)')
* [plt.ylabel](http://matplotlib.org/api/_as_gen/matplotlib.pyplot.ylabel.html#matplotlib.pyplot.ylabel)('Coefficients')
* [plt.title](http://matplotlib.org/api/_as_gen/matplotlib.pyplot.title.html#matplotlib.pyplot.title)('Logistic Regression Path')
* [plt.axis](http://matplotlib.org/api/axis_api.html#matplotlib.axis)('tight')
* [plt.show](http://matplotlib.org/api/_as_gen/matplotlib.pyplot.show.html#matplotlib.pyplot.show)()
* [Plot multinomial and One-vs-Rest Logistic Regression](http://scikit-learn.org/stable/auto_examples/linear_model/plot_logistic_multinomial.html#sphx-glr-auto-examples-linear-model-plot-logistic-multinomial-py)
* **import** **numpy** **as** **np**
* **import** **matplotlib.pyplot** **as** **plt**
* **from** **sklearn.datasets** **import** [make\_blobs](http://scikit-learn.org/stable/modules/generated/sklearn.datasets.make_blobs.html#sklearn.datasets.make_blobs)
* **from** **sklearn.linear\_model** **import** [LogisticRegression](http://scikit-learn.org/stable/modules/generated/sklearn.linear_model.LogisticRegression.html#sklearn.linear_model.LogisticRegression)
* *# make 3-class dataset for classification*
* centers = [[-5, 0], [0, 1.5], [5, -1]]
* X, y = [make\_blobs](http://scikit-learn.org/stable/modules/generated/sklearn.datasets.make_blobs.html#sklearn.datasets.make_blobs)(n\_samples=1000, centers=centers, random\_state=40)
* transformation = [[0.4, 0.2], [-0.4, 1.2]]
* X = [np.dot](http://docs.scipy.org/doc/numpy-1.8.1/reference/generated/numpy.dot.html#numpy.dot)(X, transformation)
* **for** multi\_class **in** ('multinomial', 'ovr'):
* clf = [LogisticRegression](http://scikit-learn.org/stable/modules/generated/sklearn.linear_model.LogisticRegression.html#sklearn.linear_model.LogisticRegression)(solver='sag', max\_iter=100, random\_state=42,
* multi\_class=multi\_class).fit(X, y)
* *# print the training scores*
* **print**("training score : *%.3f* (*%s*)" % (clf.score(X, y), multi\_class))
* *# create a mesh to plot in*
* h = .02 *# step size in the mesh*
* x\_min, x\_max = X[:, 0].min() - 1, X[:, 0].max() + 1
* y\_min, y\_max = X[:, 1].min() - 1, X[:, 1].max() + 1
* xx, yy = [np.meshgrid](http://docs.scipy.org/doc/numpy-1.8.1/reference/generated/numpy.meshgrid.html#numpy.meshgrid)([np.arange](http://docs.scipy.org/doc/numpy-1.8.1/reference/generated/numpy.arange.html#numpy.arange)(x\_min, x\_max, h),
* [np.arange](http://docs.scipy.org/doc/numpy-1.8.1/reference/generated/numpy.arange.html#numpy.arange)(y\_min, y\_max, h))
* *# Plot the decision boundary. For that, we will assign a color to each*
* *# point in the mesh [x\_min, x\_max]x[y\_min, y\_max].*
* Z = clf.predict([np.c\_](http://docs.scipy.org/doc/numpy-1.8.1/reference/generated/numpy.c_.html#numpy.c_)[xx.ravel(), yy.ravel()])
* *# Put the result into a color plot*
* Z = Z.reshape(xx.shape)
* [plt.figure](http://matplotlib.org/api/_as_gen/matplotlib.figure.AxesStack.html#matplotlib.figure)()
* [plt.contourf](http://matplotlib.org/api/_as_gen/matplotlib.pyplot.contourf.html#matplotlib.pyplot.contourf)(xx, yy, Z, cmap=plt.cm.Paired)
* [plt.title](http://matplotlib.org/api/_as_gen/matplotlib.pyplot.title.html#matplotlib.pyplot.title)("Decision surface of LogisticRegression (*%s*)" % multi\_class)
* [plt.axis](http://matplotlib.org/api/axis_api.html#matplotlib.axis)('tight')
* *# Plot also the training points*
* colors = "bry"
* **for** i, color **in** zip(clf.classes\_, colors):
* idx = [np.where](http://docs.scipy.org/doc/numpy-1.8.1/reference/generated/numpy.where.html#numpy.where)(y == i)
* [plt.scatter](http://matplotlib.org/api/_as_gen/matplotlib.pyplot.scatter.html#matplotlib.pyplot.scatter)(X[idx, 0], X[idx, 1], c=color, cmap=plt.cm.Paired,
* edgecolor='black', s=20)
* *# Plot the three one-against-all classifiers*
* xmin, xmax = [plt.xlim](http://matplotlib.org/api/_as_gen/matplotlib.pyplot.xlim.html#matplotlib.pyplot.xlim)()
* ymin, ymax = [plt.ylim](http://matplotlib.org/api/_as_gen/matplotlib.pyplot.ylim.html#matplotlib.pyplot.ylim)()
* coef = clf.coef\_
* intercept = clf.intercept\_
* **def** plot\_hyperplane(c, color):
* **def** line(x0):
* **return** (-(x0 \* coef[c, 0]) - intercept[c]) / coef[c, 1]
* [plt.plot](http://matplotlib.org/api/_as_gen/matplotlib.pyplot.plot.html#matplotlib.pyplot.plot)([xmin, xmax], [line(xmin), line(xmax)],
* ls="--", color=color)
* **for** i, color **in** zip(clf.classes\_, colors):
* plot\_hyperplane(i, color)
* [plt.show](http://matplotlib.org/api/_as_gen/matplotlib.pyplot.show.html#matplotlib.pyplot.show)()
* [Multiclass sparse logisitic regression on newgroups20](http://scikit-learn.org/stable/auto_examples/linear_model/plot_sparse_logistic_regression_20newsgroups.html#sphx-glr-auto-examples-linear-model-plot-sparse-logistic-regression-20newsgroups-py)
* [MNIST classfification using multinomial logistic + L1](http://scikit-learn.org/stable/auto_examples/linear_model/plot_sparse_logistic_regression_mnist.html#sphx-glr-auto-examples-linear-model-plot-sparse-logistic-regression-mnist-py)
* **import** **time**
* **import** **io**
* **import** **matplotlib.pyplot** **as** **plt**
* **import** **numpy** **as** **np**
* **from** **scipy.io.arff** **import** loadarff
* **from** **sklearn.datasets** **import** [get\_data\_home](http://scikit-learn.org/stable/modules/generated/sklearn.datasets.get_data_home.html#sklearn.datasets.get_data_home)
* **from** **sklearn.externals.joblib** **import** Memory
* **from** **sklearn.linear\_model** **import** [LogisticRegression](http://scikit-learn.org/stable/modules/generated/sklearn.linear_model.LogisticRegression.html#sklearn.linear_model.LogisticRegression)
* **from** **sklearn.model\_selection** **import** [train\_test\_split](http://scikit-learn.org/stable/modules/generated/sklearn.model_selection.train_test_split.html#sklearn.model_selection.train_test_split)
* **from** **sklearn.preprocessing** **import** [StandardScaler](http://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.StandardScaler.html#sklearn.preprocessing.StandardScaler)
* **from** **sklearn.utils** **import** [check\_random\_state](http://scikit-learn.org/stable/modules/generated/sklearn.utils.check_random_state.html#sklearn.utils.check_random_state)
* **try**:
* **from** **urllib.request** **import** urlopen
* **except** ImportError:
* *# Python 2*
* **from** **urllib2** **import** urlopen
* **print**(\_\_doc\_\_)
* *# Author: Arthur Mensch <arthur.mensch@m4x.org>*
* *# License: BSD 3 clause*
* *# Turn down for faster convergence*
* t0 = time.time()
* train\_samples = 5000
* memory = Memory([get\_data\_home](http://scikit-learn.org/stable/modules/generated/sklearn.datasets.get_data_home.html#sklearn.datasets.get_data_home)())
* **@memory.cache**()
* **def** fetch\_mnist():
* content = urlopen(
* 'https://www.openml.org/data/download/52667/mnist\_784.arff').read()
* data, meta = loadarff(io.StringIO(content.decode('utf8')))
* data = data.view([('pixels', '<f8', 784), ('class', '|S1')])
* **return** data['pixels'], data['class']
* X, y = fetch\_mnist()
* random\_state = [check\_random\_state](http://scikit-learn.org/stable/modules/generated/sklearn.utils.check_random_state.html#sklearn.utils.check_random_state)(0)
* permutation = random\_state.permutation(X.shape[0])
* X = X[permutation]
* y = y[permutation]
* X = X.reshape((X.shape[0], -1))
* X\_train, X\_test, y\_train, y\_test = [train\_test\_split](http://scikit-learn.org/stable/modules/generated/sklearn.model_selection.train_test_split.html#sklearn.model_selection.train_test_split)(
* X, y, train\_size=train\_samples, test\_size=10000)
* scaler = [StandardScaler](http://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.StandardScaler.html#sklearn.preprocessing.StandardScaler)()
* X\_train = scaler.fit\_transform(X\_train)
* X\_test = scaler.transform(X\_test)
* *# Turn up tolerance for faster convergence*
* clf = [LogisticRegression](http://scikit-learn.org/stable/modules/generated/sklearn.linear_model.LogisticRegression.html#sklearn.linear_model.LogisticRegression)(C=50. / train\_samples,
* multi\_class='multinomial',
* penalty='l1', solver='saga', tol=0.1)
* clf.fit(X\_train, y\_train)
* sparsity = [np.mean](http://docs.scipy.org/doc/numpy-1.8.1/reference/generated/numpy.mean.html#numpy.mean)(clf.coef\_ == 0) \* 100
* score = clf.score(X\_test, y\_test)
* *# print('Best C % .4f' % clf.C\_)*
* **print**("Sparsity with L1 penalty: *%.2f%%*" % sparsity)
* **print**("Test score with L1 penalty: *%.4f*" % score)
* coef = clf.coef\_.copy()
* [plt.figure](http://matplotlib.org/api/_as_gen/matplotlib.figure.AxesStack.html#matplotlib.figure)(figsize=(10, 5))
* scale = np.abs(coef).max()
* **for** i **in** range(10):
* l1\_plot = [plt.subplot](http://matplotlib.org/api/_as_gen/matplotlib.pyplot.subplot.html#matplotlib.pyplot.subplot)(2, 5, i + 1)
* l1\_plot.imshow(coef[i].reshape(28, 28), interpolation='nearest',
* cmap=plt.cm.RdBu, vmin=-scale, vmax=scale)
* l1\_plot.set\_xticks(())
* l1\_plot.set\_yticks(())
* l1\_plot.set\_xlabel('Class *%i*' % i)
* [plt.suptitle](http://matplotlib.org/api/_as_gen/matplotlib.pyplot.suptitle.html#matplotlib.pyplot.suptitle)('Classification vector for...')
* run\_time = time.time() - t0
* **print**('Example run in *%.3f* s' % run\_time)
* [plt.show](http://matplotlib.org/api/_as_gen/matplotlib.pyplot.show.html#matplotlib.pyplot.show)()

**Differences from liblinear:**

There might be a difference in the scores obtained between [**LogisticRegression**](http://scikit-learn.org/stable/modules/generated/sklearn.linear_model.LogisticRegression.html#sklearn.linear_model.LogisticRegression) with solver=liblinear or **LinearSVC**and the external liblinear library directly, when fit\_intercept=False and the fit coef\_ (or) the data to be predicted are zeroes. This is because for the sample(s) with decision\_function zero, [**LogisticRegression**](http://scikit-learn.org/stable/modules/generated/sklearn.linear_model.LogisticRegression.html#sklearn.linear_model.LogisticRegression) and **LinearSVC** predict the negative class, while liblinear predicts the positive class. Note that a model with fit\_intercept=False and having many samples with decision\_function zero, is likely to be a underfit, bad model and you are advised to set fit\_intercept=True and increase the intercept\_scaling.

**Note**

**Feature selection with sparse logistic regression**

A logistic regression with L1 penalty yields sparse models, and can thus be used to perform feature selection, as detailed in [L1-based feature selection](http://scikit-learn.org/stable/modules/feature_selection.html#l1-feature-selection).

[**LogisticRegressionCV**](http://scikit-learn.org/stable/modules/generated/sklearn.linear_model.LogisticRegressionCV.html#sklearn.linear_model.LogisticRegressionCV) implements Logistic Regression with builtin cross-validation to find out the optimal C parameter. “newton-cg”, “sag”, “saga” and “lbfgs” solvers are found to be faster for high-dimensional dense data, due to warm-starting. For the multiclass case, if multi\_class option is set to “ovr”, an optimal C is obtained for each class and if the multi\_classoption is set to “multinomial”, an optimal C is obtained by minimizing the cross-entropy loss.

**1.1.12. Stochastic Gradient Descent - SGD**

Stochastic gradient descent is a simple yet very efficient approach to fit linear models. It is particularly useful when the number of samples (and the number of features) is very large. The partial\_fit method allows only/out-of-core learning.

The classes [**SGDClassifier**](http://scikit-learn.org/stable/modules/generated/sklearn.linear_model.SGDClassifier.html#sklearn.linear_model.SGDClassifier) and [**SGDRegressor**](http://scikit-learn.org/stable/modules/generated/sklearn.linear_model.SGDRegressor.html#sklearn.linear_model.SGDRegressor) provide functionality to fit linear models for classification and regression using different (convex) loss functions and different penalties. E.g., with loss="log", [**SGDClassifier**](http://scikit-learn.org/stable/modules/generated/sklearn.linear_model.SGDClassifier.html#sklearn.linear_model.SGDClassifier) fits a logistic regression model, while with loss="hinge" it fits a linear support vector machine (SVM).

**1.1.13. Perceptron**

The [**Perceptron**](http://scikit-learn.org/stable/modules/generated/sklearn.linear_model.Perceptron.html#sklearn.linear_model.Perceptron) is another simple algorithm suitable for large scale learning. By default:

* It does not require a learning rate.
* It is not regularized (penalized).
* It updates its model only on mistakes.

The last characteristic implies that the Perceptron is slightly faster to train than SGD with the hinge loss and that the resulting models are sparser.

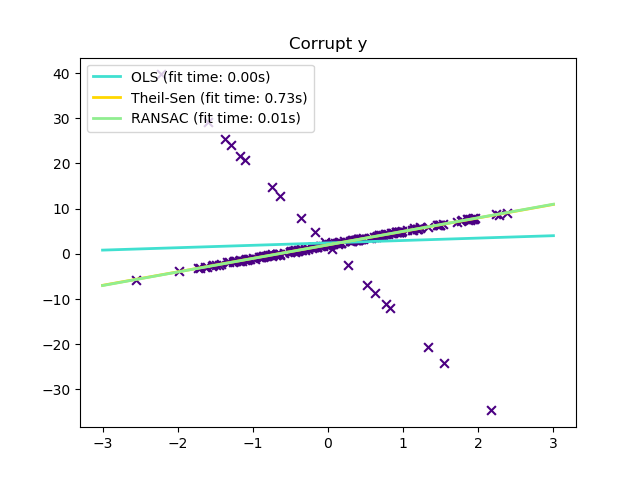
**1.1.14. Passive Aggressive Algorithms**

The passive-aggressive algorithms are a family of algorithms for large-scale learning. They are similar to the Perceptron in that they do not require a learning rate. However, contrary to the Perceptron, they include a regularization parameter C.

For classification, [**PassiveAggressiveClassifier**](http://scikit-learn.org/stable/modules/generated/sklearn.linear_model.PassiveAggressiveClassifier.html#sklearn.linear_model.PassiveAggressiveClassifier) can be used with loss='hinge' (PA-I) or loss='squared\_hinge' (PA-II). For regression, [**PassiveAggressiveRegressor**](http://scikit-learn.org/stable/modules/generated/sklearn.linear_model.PassiveAggressiveRegressor.html#sklearn.linear_model.PassiveAggressiveRegressor) can be used with loss='epsilon\_insensitive' (PA-I) orloss='squared\_epsilon\_insensitive' (PA-II).

**1.1.15. Robustness regression: outliers and modeling errors**

Robust regression is interested in fitting a regression model in the presence of corrupt data: either outliers, or error in the model.

[](http://scikit-learn.org/stable/auto_examples/linear_model/plot_theilsen.html)

**1.1.15.1. Different scenario and useful concepts**

There are different things to keep in mind when dealing with data corrupted by outliers:

* **Outliers in X or in y**?

| **Outliers in the y direction** | **Outliers in the X direction** |
| --- | --- |
| [y_outliers](http://scikit-learn.org/stable/auto_examples/linear_model/plot_robust_fit.html) | [X_outliers](http://scikit-learn.org/stable/auto_examples/linear_model/plot_robust_fit.html) |

* **Fraction of outliers versus amplitude of error**

The number of outlying points matters, but also how much they are outliers.

| **Small outliers** | **Large outliers** |
| --- | --- |
| [y_outliers](http://scikit-learn.org/stable/auto_examples/linear_model/plot_robust_fit.html) | [large_y_outliers](http://scikit-learn.org/stable/auto_examples/linear_model/plot_robust_fit.html) |

An important notion of robust fitting is that of breakdown point: the fraction of data that can be outlying for the fit to start missing the inlying data.

Note that in general, robust fitting in high-dimensional setting (large n\_features) is very hard. The robust models here will probably not work in these settings.

**Trade-offs: which estimator?**

Scikit-learn provides 3 robust regression estimators: [RANSAC](http://scikit-learn.org/stable/modules/linear_model.html#ransac-regression), [Theil Sen](http://scikit-learn.org/stable/modules/linear_model.html#theil-sen-regression) and [HuberRegressor](http://scikit-learn.org/stable/modules/linear_model.html#huber-regression)

* [HuberRegressor](http://scikit-learn.org/stable/modules/linear_model.html#huber-regression) should be faster than [RANSAC](http://scikit-learn.org/stable/modules/linear_model.html#ransac-regression) and [Theil Sen](http://scikit-learn.org/stable/modules/linear_model.html#theil-sen-regression) unless the number of samples are very large, i.e n\_samples >> n\_features. This is because [RANSAC](http://scikit-learn.org/stable/modules/linear_model.html#ransac-regression) and [Theil Sen](http://scikit-learn.org/stable/modules/linear_model.html#theil-sen-regression) fit on smaller subsets of the data. However, both [Theil Sen](http://scikit-learn.org/stable/modules/linear_model.html#theil-sen-regression) and [RANSAC](http://scikit-learn.org/stable/modules/linear_model.html#ransac-regression) are unlikely to be as robust as [HuberRegressor](http://scikit-learn.org/stable/modules/linear_model.html#huber-regression) for the default parameters.
* [RANSAC](http://scikit-learn.org/stable/modules/linear_model.html#ransac-regression) is faster than [Theil Sen](http://scikit-learn.org/stable/modules/linear_model.html#theil-sen-regression) and scales much better with the number of samples
* [RANSAC](http://scikit-learn.org/stable/modules/linear_model.html#ransac-regression) will deal better with large outliers in the y direction (most common situation)
* [Theil Sen](http://scikit-learn.org/stable/modules/linear_model.html#theil-sen-regression) will cope better with medium-size outliers in the X direction, but this property will disappear in large dimensional settings.

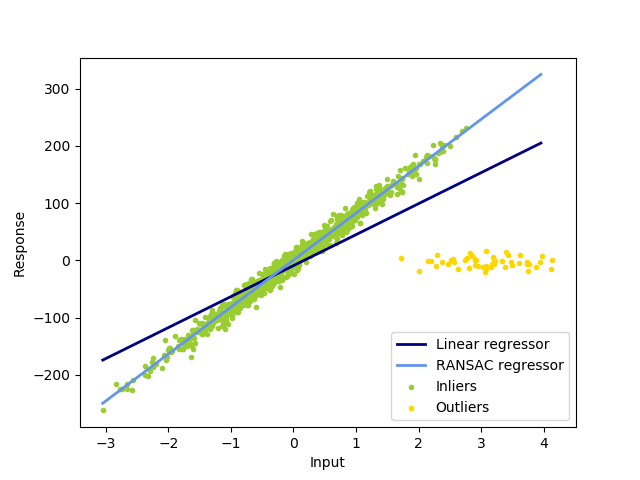
When in doubt, use [RANSAC](http://scikit-learn.org/stable/modules/linear_model.html#ransac-regression)

**1.1.15.2. RANSAC: RANdom SAmple Consensus**

RANSAC (RANdom SAmple Consensus) fits a model from random subsets of inliers from the complete data set.

RANSAC is a non-deterministic algorithm producing only a reasonable result with a certain probability, which is dependent on the number of iterations (see max\_trials parameter). It is typically used for linear and non-linear regression problems and is especially popular in the fields of photogrammetric computer vision.

The algorithm splits the complete input sample data into a set of inliers, which may be subject to noise, and outliers, which are e.g. caused by erroneous measurements or invalid hypotheses about the data. The resulting model is then estimated only from the determined inliers.

[](http://scikit-learn.org/stable/auto_examples/linear_model/plot_ransac.html)

1.1.15.2.1. Details of the algorithm

Each iteration performs the following steps:

1. Select min\_samples random samples from the original data and check whether the set of data is valid (see is\_data\_valid).
2. Fit a model to the random subset (base\_estimator.fit) and check whether the estimated model is valid (see is\_model\_valid).
3. Classify all data as inliers or outliers by calculating the residuals to the estimated model (base\_estimator.predict(X) - y) - all data samples with absolute residuals smaller than the residual\_threshold are considered as inliers.
4. Save fitted model as best model if number of inlier samples is maximal. In case the current estimated model has the same number of inliers, it is only considered as the best model if it has better score.

These steps are performed either a maximum number of times (max\_trials) or until one of the special stop criteria are met (see stop\_n\_inliers and stop\_score). The final model is estimated using all inlier samples (consensus set) of the previously determined best model.

The is\_data\_valid and is\_model\_valid functions allow to identify and reject degenerate combinations of random sub-samples. If the estimated model is not needed for identifying degenerate cases, is\_data\_valid should be used as it is called prior to fitting the model and thus leading to better computational performance.

**Examples:**

* [Robust linear model estimation using RANSAC](http://scikit-learn.org/stable/auto_examples/linear_model/plot_ransac.html#sphx-glr-auto-examples-linear-model-plot-ransac-py)
* [Robust linear estimator fitting](http://scikit-learn.org/stable/auto_examples/linear_model/plot_robust_fit.html#sphx-glr-auto-examples-linear-model-plot-robust-fit-py)

**1.1.15.3. Theil-Sen estimator: generalized-median-based estimator**

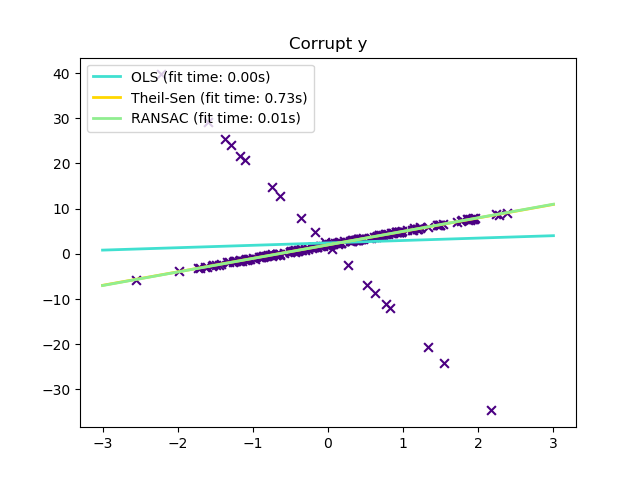
The [**TheilSenRegressor**](http://scikit-learn.org/stable/modules/generated/sklearn.linear_model.TheilSenRegressor.html#sklearn.linear_model.TheilSenRegressor) estimator uses a generalization of the median in multiple dimensions. It is thus robust to multivariate outliers. Note however that the robustness of the estimator decreases quickly with the dimensionality of the problem. It looses its robustness properties and becomes no better than an ordinary least squares in high dimension.

**Examples:**

* [Theil-Sen Regression](http://scikit-learn.org/stable/auto_examples/linear_model/plot_theilsen.html#sphx-glr-auto-examples-linear-model-plot-theilsen-py)
* [Robust linear estimator fitting](http://scikit-learn.org/stable/auto_examples/linear_model/plot_robust_fit.html#sphx-glr-auto-examples-linear-model-plot-robust-fit-py)
* <https://en.wikipedia.org/wiki/Theil%E2%80%93Sen_estimator>

1.1.15.3.1. Theoretical considerations

[**TheilSenRegressor**](http://scikit-learn.org/stable/modules/generated/sklearn.linear_model.TheilSenRegressor.html#sklearn.linear_model.TheilSenRegressor) is comparable to the [Ordinary Least Squares (OLS)](http://scikit-learn.org/stable/modules/linear_model.html#ordinary-least-squares) in terms of asymptotic efficiency and as an unbiased estimator. In contrast to OLS, Theil-Sen is a non-parametric method which means it makes no assumption about the underlying distribution of the data. Since Theil-Sen is a median-based estimator, it is more robust against corrupted data aka outliers. In univariate setting, Theil-Sen has a breakdown point of about 29.3% in case of a simple linear regression which means that it can tolerate arbitrary corrupted data of up to 29.3%.

[](http://scikit-learn.org/stable/auto_examples/linear_model/plot_theilsen.html)

The implementation of [**TheilSenRegressor**](http://scikit-learn.org/stable/modules/generated/sklearn.linear_model.TheilSenRegressor.html#sklearn.linear_model.TheilSenRegressor) in scikit-learn follows a generalization to a multivariate linear regression model [[8]](http://scikit-learn.org/stable/modules/linear_model.html#f1) using the spatial median which is a generalization of the median to multiple dimensions [[9]](http://scikit-learn.org/stable/modules/linear_model.html#f2).

In terms of time and space complexity, Theil-Sen scales according to

\binom{n_{samples}}{n_{subsamples}}

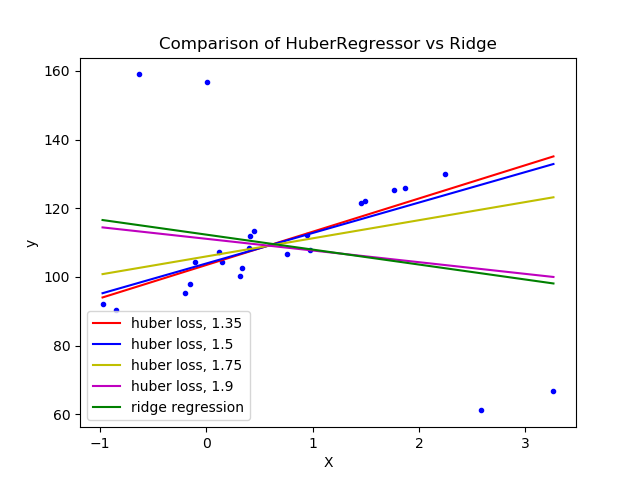
which makes it infeasible to be applied exhaustively to problems with a large number of samples and features. Therefore, the magnitude of a subpopulation can be chosen to limit the time and space complexity by considering only a random subset of all possible combinations.

**Examples:**

* [Theil-Sen Regression](http://scikit-learn.org/stable/auto_examples/linear_model/plot_theilsen.html#sphx-glr-auto-examples-linear-model-plot-theilsen-py)

**1.1.15.4. Huber Regression**

The [**HuberRegressor**](http://scikit-learn.org/stable/modules/generated/sklearn.linear_model.HuberRegressor.html#sklearn.linear_model.HuberRegressor) is different to [**Ridge**](http://scikit-learn.org/stable/modules/generated/sklearn.linear_model.Ridge.html#sklearn.linear_model.Ridge) because it applies a linear loss to samples that are classified as outliers. A sample is classified as an inlier if the absolute error of that sample is lesser than a certain threshold. It differs from [**TheilSenRegressor**](http://scikit-learn.org/stable/modules/generated/sklearn.linear_model.TheilSenRegressor.html#sklearn.linear_model.TheilSenRegressor) and [**RANSACRegressor**](http://scikit-learn.org/stable/modules/generated/sklearn.linear_model.RANSACRegressor.html#sklearn.linear_model.RANSACRegressor) because it does not ignore the effect of the outliers but gives a lesser weight to them.

[](http://scikit-learn.org/stable/auto_examples/linear_model/plot_huber_vs_ridge.html)

The loss function that [**HuberRegressor**](http://scikit-learn.org/stable/modules/generated/sklearn.linear_model.HuberRegressor.html#sklearn.linear_model.HuberRegressor) minimizes is given by

\underset{w, \sigma}{min\,} {\sum_{i=1}^n\left(\sigma + H_m\left(\frac{X_{i}w - y_{i}}{\sigma}\right)\sigma\right) + \alpha {||w||_2}^2}

where

H_m(z) = \begin{cases}
       z^2, & \text {if } |z| < \epsilon, \\
       2\epsilon|z| - \epsilon^2, & \text{otherwise}
\end{cases}

It is advised to set the parameter epsilon to 1.35 to achieve 95% statistical efficiency.

**1.1.15.5. Notes**

The [**HuberRegressor**](http://scikit-learn.org/stable/modules/generated/sklearn.linear_model.HuberRegressor.html#sklearn.linear_model.HuberRegressor) differs from using [**SGDRegressor**](http://scikit-learn.org/stable/modules/generated/sklearn.linear_model.SGDRegressor.html#sklearn.linear_model.SGDRegressor) with loss set to huber in the following ways.

* [**HuberRegressor**](http://scikit-learn.org/stable/modules/generated/sklearn.linear_model.HuberRegressor.html#sklearn.linear_model.HuberRegressor) is scaling invariant. Once epsilon is set, scaling X and y down or up by different values would produce the same robustness to outliers as before. as compared to [**SGDRegressor**](http://scikit-learn.org/stable/modules/generated/sklearn.linear_model.SGDRegressor.html#sklearn.linear_model.SGDRegressor) where epsilon has to be set again when X and y are scaled.
* [**HuberRegressor**](http://scikit-learn.org/stable/modules/generated/sklearn.linear_model.HuberRegressor.html#sklearn.linear_model.HuberRegressor) should be more efficient to use on data with small number of samples while [**SGDRegressor**](http://scikit-learn.org/stable/modules/generated/sklearn.linear_model.SGDRegressor.html#sklearn.linear_model.SGDRegressor) needs a number of passes on the training data to produce the same robustness.

**Examples:**

* [HuberRegressor vs Ridge on dataset with strong outliers](http://scikit-learn.org/stable/auto_examples/linear_model/plot_huber_vs_ridge.html#sphx-glr-auto-examples-linear-model-plot-huber-vs-ridge-py)

Also, this estimator is different from the R implementation of Robust Regression (<http://www.ats.ucla.edu/stat/r/dae/rreg.htm>) because the R implementation does a weighted least squares implementation with weights given to each sample on the basis of how much the residual is greater than a certain threshold.

**1.1.16. Polynomial regression: extending linear models with basis functions**

One common pattern within machine learning is to use linear models trained on nonlinear functions of the data. This approach maintains the generally fast performance of linear methods, while allowing them to fit a much wider range of data.

For example, a simple linear regression can be extended by constructing **polynomial features** from the coefficients. In the standard linear regression case, you might have a model that looks like this for two-dimensional data:

\hat{y}(w, x) = w_0 + w_1 x_1 + w_2 x_2

If we want to fit a paraboloid to the data instead of a plane, we can combine the features in second-order polynomials, so that the model looks like this:

\hat{y}(w, x) = w_0 + w_1 x_1 + w_2 x_2 + w_3 x_1 x_2 + w_4 x_1^2 + w_5 x_2^2

The (sometimes surprising) observation is that this is *still a linear model*: to see this, imagine creating a new variable

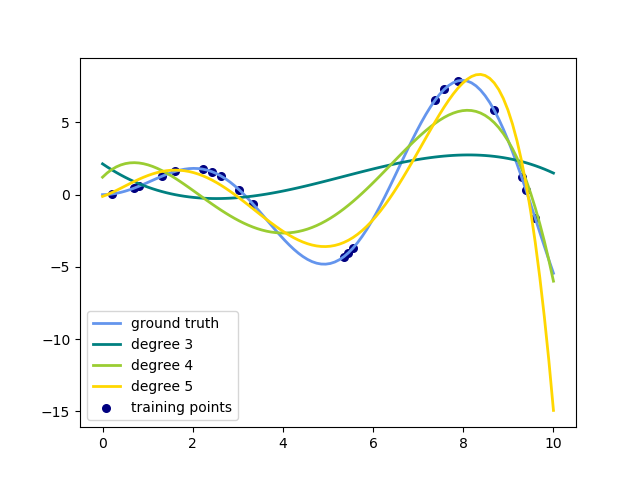
z = [x_1, x_2, x_1 x_2, x_1^2, x_2^2]

With this re-labeling of the data, our problem can be written

\hat{y}(w, x) = w_0 + w_1 z_1 + w_2 z_2 + w_3 z_3 + w_4 z_4 + w_5 z_5

We see that the resulting *polynomial regression* is in the same class of linear models we’d considered above (i.e. the model is linear in w) and can be solved by the same techniques. By considering linear fits within a higher-dimensional space built with these basis functions, the model has the flexibility to fit a much broader range of data.

Here is an example of applying this idea to one-dimensional data, using polynomial features of varying degrees:

[](http://scikit-learn.org/stable/auto_examples/linear_model/plot_polynomial_interpolation.html)

This figure is created using the [**PolynomialFeatures**](http://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.PolynomialFeatures.html#sklearn.preprocessing.PolynomialFeatures) preprocessor. This preprocessor transforms an input data matrix into a new data matrix of a given degree. It can be used as follows:

>>>

**>>> from** **sklearn.preprocessing** **import** PolynomialFeatures

**>>> import** **numpy** **as** **np**

**>>>** X = np.arange(6).reshape(3, 2)

**>>>** X

array([[0, 1],

[2, 3],

[4, 5]])

**>>>** poly = PolynomialFeatures(degree=2)

**>>>** poly.fit\_transform(X)

array([[ 1., 0., 1., 0., 0., 1.],

[ 1., 2., 3., 4., 6., 9.],

[ 1., 4., 5., 16., 20., 25.]])

The features of X have been transformed from [x_1, x_2] to [1, x_1, x_2, x_1^2, x_1 x_2, x_2^2], and can now be used within any linear model.

This sort of preprocessing can be streamlined with the [Pipeline](http://scikit-learn.org/stable/modules/pipeline.html#pipeline) tools. A single object representing a simple polynomial regression can be created and used as follows:

>>>

**>>> from** **sklearn.preprocessing** **import** PolynomialFeatures

**>>> from** **sklearn.linear\_model** **import** LinearRegression

**>>> from** **sklearn.pipeline** **import** Pipeline

**>>> import** **numpy** **as** **np**

**>>>** model = Pipeline([('poly', PolynomialFeatures(degree=3)),

**...**  ('linear', LinearRegression(fit\_intercept=**False**))])

**>>>** *# fit to an order-3 polynomial data*

**>>>** x = np.arange(5)

**>>>** y = 3 - 2 \* x + x \*\* 2 - x \*\* 3

**>>>** model = model.fit(x[:, np.newaxis], y)

**>>>** model.named\_steps['linear'].coef\_

array([ 3., -2., 1., -1.])

The linear model trained on polynomial features is able to exactly recover the input polynomial coefficients.

In some cases it’s not necessary to include higher powers of any single feature, but only the so-called *interaction features*that multiply together at most d distinct features. These can be gotten from [**PolynomialFeatures**](http://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.PolynomialFeatures.html#sklearn.preprocessing.PolynomialFeatures) with the settinginteraction\_only=True.

For example, when dealing with boolean features, x_i^n = x_i for all n and is therefore useless; but x_i x_j represents the conjunction of two booleans. This way, we can solve the XOR problem with a linear classifier:

>>>

**>>> from** **sklearn.linear\_model** **import** Perceptron

**>>> from** **sklearn.preprocessing** **import** PolynomialFeatures

**>>> import** **numpy** **as** **np**

**>>>** X = np.array([[0, 0], [0, 1], [1, 0], [1, 1]])

**>>>** y = X[:, 0] ^ X[:, 1]

**>>>** y

array([0, 1, 1, 0])

**>>>** X = PolynomialFeatures(interaction\_only=**True**).fit\_transform(X).astype(int)

**>>>** X

array([[1, 0, 0, 0],

[1, 0, 1, 0],

[1, 1, 0, 0],

[1, 1, 1, 1]])

**>>>** clf = Perceptron(fit\_intercept=**False**, max\_iter=10, tol=**None**,

**...**  shuffle=**False**).fit(X, y)

And the classifier “predictions” are perfect:

>>>

**>>>** clf.predict(X)

array([0, 1, 1, 0])

**>>>** clf.score(X, y)

1.0