Scikit-Learn Notes

Stephen CUI

October, 25, 2022

Contents

Part I

User Guide

Chapter 1

Supervised learning

1.1 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 features. In mathematical notation, if $\hat{y}$ is the predicted value.

$$\hat{y}(w,x) = w_0 + w_1 x_1 + ... + w_p x_p \tag{1.1}$$

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.

1.1.1 Ordinary Least Squares

LinearRegression fits a linear model with coefficients $w = (w_1, ..., w_p)$ to minimize the residual sum of squares between the observed targets in the dataset, and the targets predicted by the linear approximation. Mathematically it solves a problem of the form:

$$\min_w ||Xw - y||_2^2 \tag{1.2}$$

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])
reg.coef_
```

The coefficient estimates for Ordinary Least Squares **rely on the independence of the features**. When features are correlated and the columns of the design matrix X have an approximately 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 target, producing a large variance. This situation of multicollinearity can arise, for example, when data are collected without an experimental design.

Examples:

- Linear Regression Example

Non-Negative Least Squares

It is possible to constrain all the coefficients to be non-negative, which may be useful when they represent some physical or naturally non-negative quantities (e.g., frequency counts or prices of goods). LinearRegression accepts a boolean `positive` parameter: when set to `True` Non-Negative Least Squares are then applied.

Examples:

- Non-negative least squares

Ordinary Least Squares Complexity

The least squares solution is computed using the singular value decomposition of X. If X is a matrix of shape `(n_samples, n_features)` this method has a cost of $O(n_{\text{samples}} n_{\text{features}}^2)$, assuming that $n_{\text{samples}} \geq n_{\text{features}}$.

1.1.2 Ridge regression and classification

Regression

Ridge regression addresses some of the problems of Ordinary Least Squares **by imposing a penalty on the size of the coefficients**. The ridge coefficients minimize a penalized residual sum of squares:

$$\min_w ||Xw - y||_2^2 + \alpha ||w||_2^2 \tag{1.3}$$

The complexity parameter $\alpha \geq 0$ controls the amount of shrinkage: the larger the value of α, the greater the amount of shrinkage and thus the coefficients become more robust to collinearity.

Classification

The Ridge regressor has a classifier variant: RidgeClassifier. This classifier first converts binary targets to `{-1, 1}` and then treats the problem as a regression task, optimizing the same objective as above. The predicted class corresponds to the sign of the regressor's prediction. For multiclass classification, the problem is treated as multi-output regression, and the predicted class corresponds to the output with the highest value.

It might seem questionable to use a (penalized) Least Squares loss to fit a classification model instead of the more traditional logistic or hinge losses. However, in practice, all those models can lead to similar cross-validation scores in terms of accuracy or precision/recall, while the penalized least squares loss used by the RidgeClassifier allows for a very different choice of the numerical solvers with distinct computational performance profiles.

The RidgeClassifier can be significantly faster than e.g. LogisticRegression with a high number of classes because it can compute the projection matrix $(X^T X)^{-1} X^T$ only once.

This classifier is sometimes referred to as a Least Squares Support Vector Machines with a linear kernel.

Example

- Plot Ridge coefficients as a function of the regularization
- Classification of text documents using sparse features
- Hilbert matrix

1.1.3 Lasso

1.1.4 Multi-task Lasso

1.1.5 Logistic regression

1.2 Ensemble methods

1.2.1 Gradient Tree Boosting

Gradient Tree Boosting or Gradient Boosted Decision Trees (GBDT) is a generalization of boosting to arbitrary differentiable loss functions, see the seminal work of [**?**]. GBDT is an accurate and effective off-the-shelf procedure that can be used for both regression and classification problems in a variety of areas including Web search ranking and ecology.

The module `sklearn.ensemble` provides methods for both classification and regression via gradient boosted decision trees.

The 2 most important parameters of these estimators are `n_estimators` and `learning_rate`.

Classification

GradientBoostingClassifier supports both binary and multi-class classification. The following example shows how to fit a gradient boosting classifier with 100 decision stumps as weak learners:

```
>>> from sklearn.datasets import make_hastie_10_2
>>> from sklearn.ensemble import GradientBoostingClassifier

>>> X, y = make_hastie_10_2(random_state=0)
>>> X_train, X_test = X[:2000], X[2000:]
>>> y_train, y_test = y[:2000], y[2000:]

>>> clf = GradientBoostingClassifier(n_estimators=100, learning_rate=1.0,
...     max_depth=1, random_state=0).fit(X_train, y_train)
>>> clf.score(X_test, y_test)
0.913...
```

The number of weak learners (i.e. regression trees) is controlled by the parameter `n_estimators`; The size of each tree can be controlled either by setting the tree depth via `max_depth` or by setting the number of leaf nodes via `max_leaf_nodes`. The `learning_rate` is a hyper-parameter in the range (0.0, 1.0] that controls overfitting via shrinkage.

Note: Classification with more than 2 classes requires the induction of `n_classes` regression trees at each iteration, thus, the total number of induced trees equals `n_classes * n_estimators`.

Chapter 2

Dataset loading utilities

2.1

2.1.1 The 20 newsgroups text dataset

Part II

API

Chapter 3

sklearn.linear_model: Linear Models

The sklearn.linear_model module implements a variety of Linear Models.

User Guide: See the Linear Models section for further details.

Part III

Example

Chapter 4

Generalized Linear Models

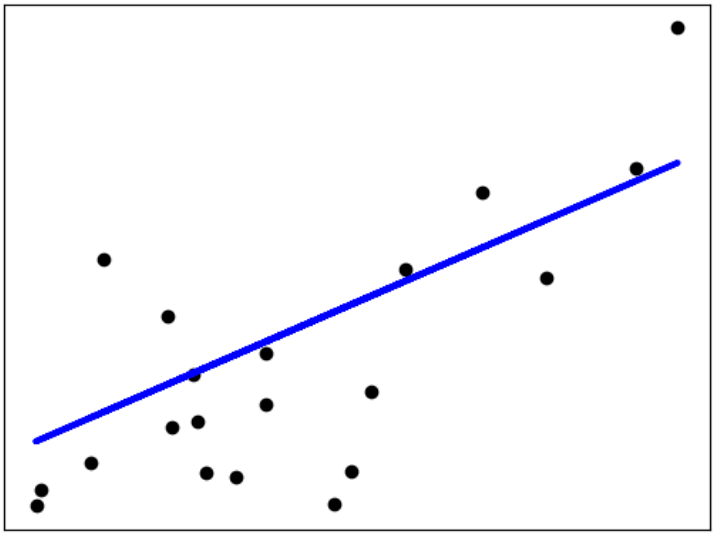
4.1 Linear Regression Example

The example below uses only the first feature of the `diabetes` dataset, in order to illustrate the data points within the two-dimensional plot. The straight line can be seen in the plot, showing how linear regression attempts to draw a straight line that will best minimize the residual sum of squares between the observed responses in the dataset, and the responses predicted by the linear approximation.

The coefficients, residual sum of squares and the coefficient of determination are also calculated.

Example 1 Linear Regression Example

```
import matplotlib.pylab as plt
import numpy as np
from sklearn import datasets, linear_model
from sklearn.metrics import mean_squared_error, r2_score
diabetes_X, diabetes_y = datasets.load_diabetes(return_X_y=True)

diabetes_X = diabetes_X[:, np.newaxis, 2]
diabetes_X_train = diabetes_X[:-20]
diabetes_X_test = diabetes_X[-20:]

diabetes_y_train = diabetes_y[:-20]
diabetes_y_test = diabetes_y[-20:]

reg = linear_model.LinearRegression()
reg.fit(diabetes_X_train, diabetes_y_train)
diabetes_y_pred = reg.predict(diabetes_X_test)

print('Coefficients: \n', reg.coef_)
print('Mean squared error: {:.2f}'.format(mean_squared_error(diabetes_y_test,
 ↪  diabetes_y_pred)))
print('Coefficient of determination: {:.2f}'.format(r2_score(diabetes_y_test,
 ↪  diabetes_y_pred)))

plt.scatter(diabetes_X_test, diabetes_y_test, color='black')
plt.plot(diabetes_X_test, diabetes_y_pred, color='blue', linewidth=3)
plt.xticks(())
plt.yticks(())
plt.show()
```

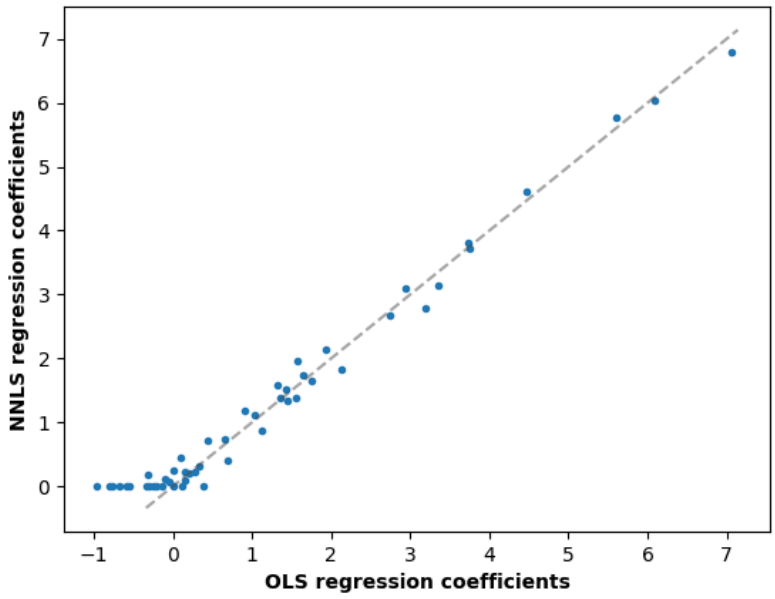


4.2 Non-negative least squares

In this example, we fit a linear model with positive constraints on the regression coefficients and compare the estimated coefficients to a classic linear regression.

Example 2 Non-negative least squares

```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.metrics import r2_score
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
np.random.seed(39)

n_samples, n_features = 200, 50
X = np.random.randn(n_samples, n_features)
true_coef = 3 * np.random.randn(n_features)

true_coef[true_coef < 0] = 0
y = np.dot(X, true_coef)
y += 5 * np.random.normal(size=(n_samples, ))

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=.5)

reg_nnls = LinearRegression(positive=True)
y_pred_nnls = reg_nnls.fit(X_train, y_train).predict(X_test)
r2_score_nnls = r2_score(y_test, y_pred_nnls)
print('NNLS R2 score: {}'.format(r2_score_nnls))

reg_ols = LinearRegression()
y_pred_ols = reg_ols.fit(X_train, y_train).predict(X_test)
```




```
r2_score_ols = r2_score(y_test, y_pred_ols)
print('OLS R2 score: {}'.format(r2_score_ols))

fig, ax = plt.subplots()
ax.plot(reg_ols.coef_, reg_nnls.coef_, linewidth=0, marker='.')
low_x, high_x = ax.get_xlim()
low_y, high_y = ax.get_ylim()
low = max(low_x, low_y)
high = min(high_x, high_y)
ax.plot([low, high], [low, high], ls='--', c='.3', alpha=.5)
ax.set_xlabel('OLS regression coefficients', fontweight='bold')
ax.set_ylabel('NNLS regression coefficients', fontweight='bold')
plt.show()
```

Comparing the regression coefficients between OLS and NNLS, we can observe they are highly correlated (the dashed line is the identity relation), but the non-negative constraint shrinks some to 0. The Non-Negative Least squares inherently yield sparse results.

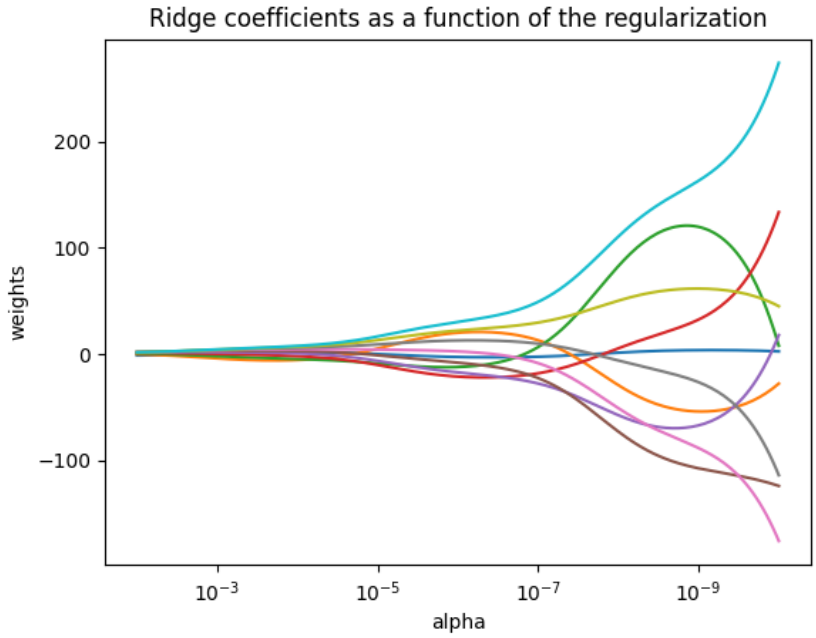
4.3 Plot Ridge coefficients as a function of the regularization

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

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.

Example 3 Ridge coefficients as a function of the regularization

```
import numpy as np
import matplotlib.pyplot as plt
from sklearn import linear_model

# X is the 10x10 Hilbert matrix
X = 1.0 / (np.arange(1, 11) + np.arange(0, 10)[:, np.newaxis])
y = np.ones(10)

n_alphas = 200
alphas = np.logspace(-10, -2, n_alphas)
coefs = []
for a in alphas:
    ridge = linear_model.Ridge(alpha=a, fit_intercept=False)
    ridge.fit(X, y)
    coefs.append(ridge.coef_)

ax = plt.gca()
ax.plot(alphas, coefs)
ax.set_xscale("log")
ax.set_xlim(ax.get_xlim()[::-1])
plt.xlabel("alpha")
plt.ylabel("weights")
plt.title("Ridge coefficients as a function of the regularization")
plt.axis("tight")
plt.show()
```

Chapter 5

Working with text documents

5.1 Classification of text documents using sparse features

This is an example showing how scikit-learn can be used to classify documents by topics using a Bag of Words approach. This example uses a Tf-idf-weighted document-term sparse matrix to encode the features and demonstrates various classifiers that can efficiently handle sparse matrices.

For document analysis via an unsupervised learning approach, see the example script Clustering text documents using k-means.

5.1.1 Loading and vectorizing the 20 newsgroups text dataset

We define a function to load data from The 20 newsgroups text dataset, which comprises around 18,000 newsgroups posts on 20 topics split in two subsets: one for training (or development) and the other one for testing (or for performance evaluation). Note that, by default, the text samples contain some message metadata such as 'headers', 'footers' (signatures) and 'quotes' to other posts. The `fetch_20newsgroups` function therefore accepts a parameter named remove to attempt stripping such information that can make the classification problem "too easy". This is achieved using simple heuristics that are neither perfect nor standard, hence disabled by default.

Example 4

```
1
```

5.1.2 Analysis of a bag-of-words document classifier

We will now train a classifier twice, once on the text samples including metadata and once after stripping the metadata. For both cases we will analyze the classification errors on a test set using a confusion matrix and inspect the coefficients that define the classification function of the trained models.

Model without metadata stripping

We start by using the custom function `load_dataset` to load the data without metadata stripping.

```
X_train, X_test, y_train, y_test, feature_names, target_names = load_dataset(verbose=True)
```

Our first model is an instance of the RidgeClassifier class. This is a linear classification model that uses the mean squared error on `{-1, 1}` encoded targets, one for each possible class. Contrary to LogisticRegression, **RidgeClassifier does not provide probabilistic predictions** (no `predict_proba` method), **but it is often faster to train**.

We plot the confusion matrix of this classifier to find if there is a pattern in the classification errors.

```
fig, ax = plt.subplots(figsize=(10, 5))
ConfusionMatrixDisplay.from_predictions(y_test, pred, ax=ax)
ax.xaxis.set_ticklabels(target_names)
ax.yaxis.set_ticklabels(target_names)
ax.set_title('Confusion Matrix for {}\n on thr original documents'.format(clf.__class__.__name__))
plt.show()
```

5.2 Clustering text documents using k-means

5.3 FeatureHasher and DictVectorizer Comparison