TASK

Machine Learning III

Visit our website

Introduction

Welcome to The Third Machine Learning Task!

By this point, you should have a clear understanding of machine learning, and be fairly familiar with the uses of and the distinction between machine learning and artificial intelligence. In this task, we will delve one step further by making use of the machine learning algorithms included in scikit-learn. More specifically, we will be doing some regression analysis, which is a statistical process used to estimate the relationship between variables.

Get in touch

Connect for support

Remember that with our courses, you're not alone! You can contact your mentor to get support on any aspect of your course.

The best way to get help is to login to **www.hyperiondev.com/portal** to start a chat with your mentor. You can also schedule a call or get support via email.

Your mentor is happy to offer you support that is tailored to your individual career or education needs. Do not hesitate to ask a question or for additional support!

Reminder on loading the dataset

There are two ways in which you can load a dataset from sklearn. If you are using a slow PC or laptop this might take a few seconds.

1. The first is to use sklearn module to import all datasets and then use the dot (.) notation to specify which dataset to load.

```
from sklearn import datasets
iris = datasets.load_iris()
```

2. Another way is to use sklearn.datasets module to import load_iris. Using this approach will omit the use of the dot (.) notation and just invoke the load_iris() function directly.

```
from sklearn.datasets import load_iris
iris = load_iris()
```

Linear Regression

The objective of LinearRegression is to fit a linear model to the dataset by adjusting a set of parameters in order to make the sum of the squared residuals of the model as small as possible.

A linear model is defined by $y = X\beta + \varepsilon$, where y is the target variable, X is the data, β represents the coefficients and ε is the observation noise.

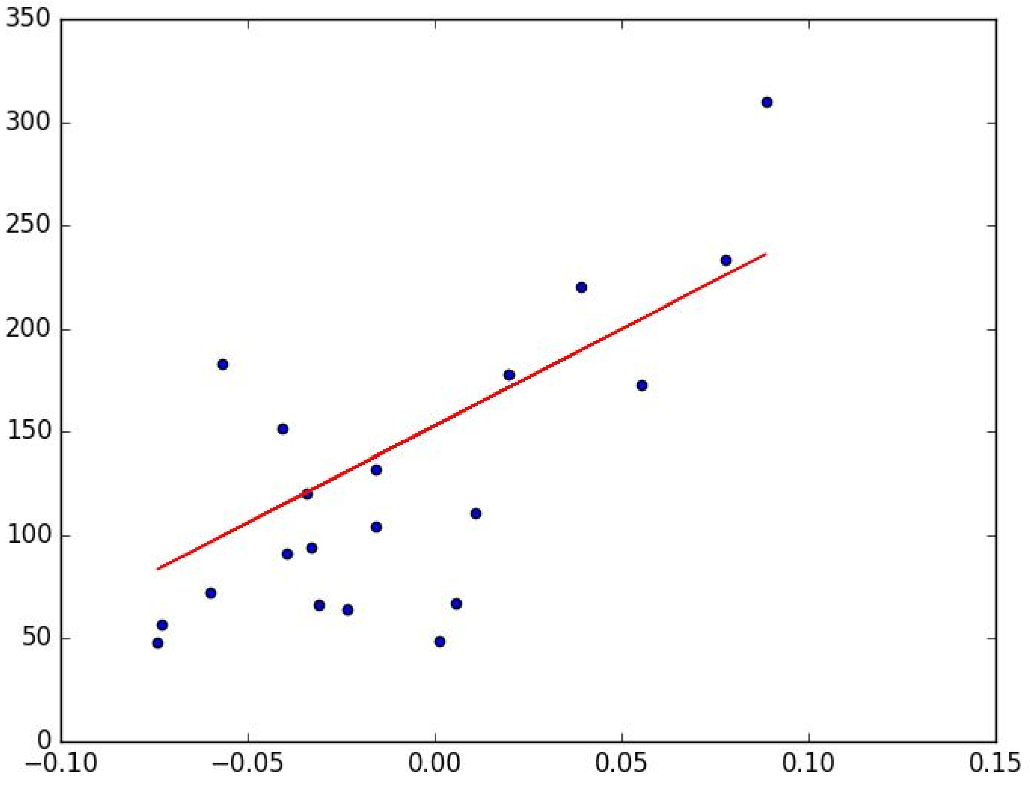
Let's try and predict something using linear regression. The diabetes dataset comes with sklearn, consists of 10 physiological variables (age, sex, weight and blood pressure) measure on 442 patients, and an indication of the disease progression after one year. The goal is to predict disease progression from physiological variables.

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

d = load_diabetes()
d_X = d.data[:, np.newaxis, 2]
dx_train = d_X[:-20]
dy_train = d.target[:-20]
dx_test = d_X[-20:]
dy_test = d.target[-20:]
```

```
12.
13. lr = linear_model.LinearRegression()
14. lr.fit(dx_train, dy_train)
15.
16. mse = np.mean((lr.predict(dx_test) - dy_test) **2)
17. lr_score = lr.score(dx_test, dy_test)
18.
19. print(lr.coef_)
20. print(mse)
21. print(lr_score)
22.
23. plt.scatter(dx_test, dy_test)
24. plt.plot(dx_test, lr.predict(dx_test), c='r')
25. plt.show()
```

First, imports modules to be used. The linear model comes with sklearn so in **line 4** the linear model is imported to allow the use of the load_diabetes() function. Once the dataset is loaded, select the features to use. In this example, only one feature is selected. See **line 7**. The data is then splitted up into training and testing data. The last 20 samples are being used as testing data in this example. You can play around with how much data to use for testing if you want. The next step is to create a linear regression object and then train it with the training data by using the `fit()` function. The mean square error can be calculated easily with the numpy `mean()` function. To obtain the variance score of the model, the `score()` function is used; passing the test data as parameters. Lastly, the scatter plot of the test data and the line plot is then graphed and presented to the screen. The above example will produce the following figure:



Support Vector Machine (SVM)

Support vector machines fall under supervised machine learning and have set of methods that can be used for regression.Support vector machines are effective in high dimensional spaces, uses support vectors as a subset of training points in the decision function, versatile and they are also effective in cases where the number of dimensions is greater than the number of samples. However, they can give poor performances in cases where the number of features is more than the number of samples. So you have to be careful how you use them. Without any waste of time, let's jump to regression with SVMs. This is given a special term "Support Vector Regression" (SVR). It makes sense if they are called SVR as they are specifically for regression.

Implementations of SVR include: SVR, NuSVR and LinearSVR. NuSVR implements a slightly different formulation than SVR and LinearSVR. NuSVR uses a parameter nu to control the number of support vectors. LinearSVR is similar to SVR but uses linear kernels only and provides a faster implementation than SVR. LinearSVR is known to scale better for a large number of samples. Let's have a look at examples of SVR and NuSVR.

Support Vector Regression

The class for SVR is:

```
class sklearn.svm.SVR(kernel='rbf', degree=3, gamma='auto', coef0=0.0, tol=0.001,
C=1.0,epsilon=0.1, shrinking=True, cache_size=200, verbose=False, max_iter=-1)
```

Parameter	**Data type**	**Description**
C	float	(optional - default:1.0) Penalty parameter C of the error term.
epsilon	float	(optional - default:0.1) Epsilon in the epsilon-SVR model. It specifies the epsilon-tube within which no penalty is associated with the training loss function with points predicted within a distance epsilon from the actual value.
kernel	string	(optional - default:‘rbf’) Specifies the kernel type to be used in the algorithm. It must be one of ‘linear’, ‘poly’, ‘rbf’, ‘sigmoid’, ‘precomputed’ or a callable. If none is given, ‘rbf’ will be used. If a callable is given it is used to precompute the kernel matrix.
degree	int	(optional - default:3) Degree of the polynomial kernel function (‘poly’). Ignored by all other kernels.
gamma	float	(optional - default:‘auto’) Kernel coefficient for ‘rbf’, ‘poly’ and ‘sigmoid’. If gamma is ‘auto’ then 1/n_features will be used instead.
coef0	float	(optional - default:0.0) Independent term in kernel function. It is only significant in ‘poly’ and ‘sigmoid’.
shrinking	bool	(optional - default:True) whether to use the shrinking heuristic or not.
tol	float	(optional - default:1e-3) Tolerance for stopping criterion.
cache_size	float	(optional) The kernel cache size in MB
verbose	bool	default:False Enable verbose output. If

		enabled, it may not work properly in multithreaded context.
max_iter	int	(optional - default:-1) Hard limit on iterations within solver, -1 for no limit.

SVR Code Example

```
1.  from sklearn.svm import SVR
2.  import numpy as np
3.  n_samples, n_features = 10, 5
4.  np.random.seed(0)
5.  Y = np.random.randn(n_samples)
6.  x = np.random.randn(n_samples, n_features)
7.  clf = SVR(epsilon=0.2)
8.  clf.fit(X, y)
```

In the above SVR example, 10 samples with 5 features are randomly generated. In **line 7**, an object of SVR class is created and the `epsilon` parameter is set to `0.2`. The `fit()` function is then used to fit the SVR model according to the training data provided. Just like it was done with Linear Regression, you can make use of the functions `predict(X)` and `score(X, y[,sample_weight])` to perform regression on samples in `X` and getting the coefficient of determination R^2 of the prediction, respectively. Other functions you can make use of are: `get_params([deep])` and `set_params(**params)` getting and setting the parameters of the estimator, respectively.

Nu Support Vector Regression

The class for NuSVR is:

```
class sklearn.svm.NuSVR(nu=0.5, C=1.0, kernel='rbf', degree=3, gamma='auto',
coef0=0.0,shrinking=True, tol=0.001, cache_size=200, verbose=False, max_iter=-1)
```

The parameters for the NuSVR are the same as those of SVR except for the following change:

Parameter	Data type	Description

nu	float	(optional - default:0.5) An upper bound on the fraction of training errors and a lower bound of the fraction of support vectors. Should be in the interval (0, 1].

NuSVR Code Example

```
1.   from sklearn.svm import NuSVR
2.   import numpy as np
3.   n_samples, n_features = 10, 5
4.   np.random.seed(0)
5.   Y = np.random.randn(n_samples)
6.   x = np.random.randn(n_samples, n_features)
7.   clf = NuSVR(nu=0.1)
8.   clf.fit(X, y)
```

In the above NuSVR example, 10 samples with 5 features are randomly generated. In **line 7**, an object of NuSVR class is created and the `nu` parameter is set to `0.1`. The `fit()` function is then used to fit the NuSVR model according to the training data provided. Just like it was done with Linear Regression and SVR, you can make use of the functions `predict(X)` and `score(X, y[,sample_weight])` to perform regression on samples in `X` and getting the coefficient of determination R^2 of the prediction, respectively. Other functions you can make use of are: `get_params([deep])` and `set_params(**params)` getting and setting the parameters of the estimator, respectively.

Short exercise: have a look at LinearSVR. Perform fit(), predict() and score using the model.

Hint:

```
class sklearn.svm.LinearSVR(epsilon=0.0, tol=0.0001, C=1.0,
loss='epsilon_insensitive',fit_intercept=True, intercept_scaling=1.0, dual=True, verbose=0,
random_state=None, max_iter=1000)
```

Logistic Regression

Logistic regression, also called logit model, is mostly used for classification rather than regression. The Logistic Regression uses the logistic function to model the probabilities of describing the possible outcomes of a single trial. For example, suppose a researcher is interested in how variables, such as Graduate Record

Exam scores, Grade point Average and prestige of the undergraduate institution affect admission into graduate school. The response variable is whether admit or don't. Note that the response variable is binary.

Logistic regression is applicable if:

- The aim is to model the probabilities of a response variable as a function of some explanatory variables.
- The aim is to perform descriptive discriminative analysis such as describing the difference between individuals in separate groups as a function of explanatory variables.
- The aim is to predict probabilities that individuals fall into two categories of the binary response as a function of some explanatory variables.
- The aim is to classify individuals into two categories based on explanatory variables.

Compulsory Task

Follow these steps:

- Create a python file called "theory.py".

- You have learnt about when and how Logistic Regression is applicable. Using the problem description below, make use of the 4 applications of Logistic Regression to provide examples of how logistic regression is applicable for each application.

A data of students, 50 males and 50 females, containing 400 observations. There are 3 explanatory variables, namely; GRE, GPA and RANK. Treat the variables GRE and GPA as continuous and the RANK variable only takes values [1, 4]. Institutions with a rank of 1 have the highest prestige, while those with a rank of 4 have the lowest.

First few rows for visualization:

```
##   admit gre  gpa rank
## 1     0 380 3.61    3
## 2     1 660 3.67    3
## 3     1 800 4.00    1
## 4     1 640 3.19    4
## 5     0 520 2.93    4
## 6     1 760 3.00    2
```

These are the 4 applications:

- The aim is to model the probabilities of a response variable as a function of some explanatory variables.
- The aim is to perform descriptive discriminative analysis such as describing the difference between individuals in separate groups as a function of explanatory variables.
- The aim is to predict probabilities that individuals fall into two categories of the binary response as a function of some explanatory variables.
- The aim is to classify individuals into two categories based on explanatory variables.

Answer to application 1: "success" of admission as a function of gender.

Compulsory Task 2

Follow these steps:

- Create a python file called "linearRegression.py".
- Use the diabetes dataset to perform linear regression.

- Instead of using linear_model.LinearRegression() from sklearn, create your own function to return the gradient and the best-fit line y-intercept.
  - $m = (\mu(x) * \mu(y) - \mu(x * y))/((\mu(x))^2 - \mu(x^2))$
  - $b = \mu(y) - m * \mu(x)$
  - $Where\ \mu\ is\ a\ mean\ function$

- Reserve the last 20 observations for testing and use the rest for training your model.

- Produce a figure with the following:
  - Scatter plot of training data colored red.
  - Scatter plot of testing data colored green.
  - Line graph for the best-fit line colored blue.
  - Legend

Rate us

Share your thoughts

Hyperion strives to provide internationally-excellent course content that helps you achieve your learning outcomes.

Think the content of this task, or this course as a whole, can be improved or think we've done a good job?

Click here to share your thoughts anonymously.