Big Data Analysis: Assignment 1 Part One: scikit-learn (plain python)

Baochen Hu
Ning Zhang

June 20, 2015

Abstract

This part of the assignment aims to practice algorithms in Python with scikit-learn. We used the white vinho verde wine samples data, from the north of Portugal. We separated the dataset into two parts. The first part is our training dataset. The second part is our test dataset for the results.

1 Development Environment

Python 2.7.9
pip 6.1.1 from /Library/Frameworks/Python.framework/Versions/2.7/lib/python2.7/site-packages (python 2.7)
scikit-learn (http://scikit-learn.org/dev/install.html)

2 Running Introduction

Command:

```
python2 [ ]
```

3 Tryout

We will try out the algorithms in Python using SciPy's capabilities with the following models:

Classification:

```
sklearn.linear_model.SGDClassifier (SVM => Hinge loss)
sklearn.linear_model.SGDClassifier (Logarithmic regression => log loss)
sklearn.linear_model.LogisticRegression (LBFGS version)
```

Regression:

```
sklearn.linear_model.LinearRegression
sklearn.linear_model.SGDRegressor
sklearn.linear_model.Ridge
sklearn.linear_model.Lasso
```

3.1 SGDClassifier(SVM Hinge loss)

3.1.1 Introduction

SVM: Average hinge loss (non-regularized)
The function expects that either all the labels are included in ytrue or an optional labels argument is provided which contains all the labels. The multilabel margin is calculated according to Crammer-Singer's method. As in the binary case, the cumulated hinge loss is an upper bound of the number of mistakes made by the classifier.

3.1.2 Source Code:

```
import numpy as np
from sklearn.linear_model import SGDClassifier
def loadDataSet():
        feature_train = []
        feature_lable = []
        test_feature = []
        test_lable = []
        fr = open ('winequality-white.csv')
        count = 0
        for line in fr.readlines():
                if count == 0:
                        header = line.strip().split(';')
                elif count > 0 and count < 4000:
                        lineAttr = line.strip().split(';')
                        feature_train.append(
                        [float(lineAttr[0]),float(lineAttr[1]),
                        float(lineAttr[2]),float(lineAttr[3]),
                                float(lineAttr[4]),float(lineAttr[5]),
                                float(lineAttr[6]),float(lineAttr[7]),
                                float(lineAttr[8]),float(lineAttr[9]),
                                float(lineAttr[10])])
                        feature_lable.append(int(lineAttr[11]))
                else:
                        lineAttr = line.strip().split(';')
                        test_feature.append(
                        [float(lineAttr[0]),float(lineAttr[1]),
                        float(lineAttr[2]),float(lineAttr[3]),
                                float(lineAttr[4]),float(lineAttr[5]),
                                float(lineAttr[6]),float(lineAttr[7]),
```

```
                                    float(lineAttr[8]),float(lineAttr[9]),
                                    float(lineAttr[10])])
                            test_lable.append(int(lineAttr[11]))
                    count += 1
            return np.array(feature_train), np.array(feature_lable),
            np.array(test_feature), np.array(test_lable)

feature_train, feature_lable, test_feature,test_lable = loadDataSet()

clf = SGDClassifier(loss="hinge", penalty="l2", alpha=0.001, n_iter=100)
clf.fit (feature_train,feature_lable)
pred = clf.predict (test_feature)
from sklearn.metrics import mean_absolute_error
print 'mean_absolute_error:'
print mean_absolute_error(test_lable, pred)

print 'score:'
print clf.score (test_feature, test_lable)

print clf.coef_
print clf.intercept_

print '-------------------------'
from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()
scaler.fit(feature_train)  # Don't cheat - fit only on training data
feature_train = scaler.transform(feature_train)
clf1 = SGDClassifier(loss="hinge", penalty="l2", alpha=0.001, n_iter=100)
clf1.fit (feature_train,feature_lable)
test_feature = scaler.transform(test_feature)
pred = clf1.predict (test_feature)
from sklearn.metrics import mean_absolute_error
print 'mean_absolute_error:'
print mean_absolute_error(test_lable, pred)
```

3.1.3 Output:

```
mean_absolute_error:
0.563959955506
score:
0.519466073415
[[  1.64793348e+00   2.27081385e-01  -8.44837732e-02  -3.03941622e-01
    2.52251550e-02   2.62449591e-02  -4.43664785e-01  -2.13657563e-01
   -7.60203988e-01  -2.30530722e-01  -2.02336137e+00]
 [  3.38385006e+00   3.21668217e+00  -7.83499513e-01  -7.30859624e-01
    1.62311325e-01  -7.59854055e-01  -2.47452472e-01   5.11357496e-01
    1.93335366e+00   1.73066759e-01  -2.92093898e+00]
 [  8.17730439e+00   7.68458652e+00   2.62374606e-01   1.93087914e-01
    9.53576847e-01  -5.21149903e-01  -1.56219995e-01   5.96301288e+00
    1.49146853e+01  -1.41820261e+00  -1.01702966e+01]
```

```
 [  7.02615049e-01  -7.66170342e+00   1.50766042e+00   5.87637133e-01
   -1.82639920e-02   2.01211353e-01   4.12420786e-02  -3.44712316e-01
    1.03730077e-01   3.67404433e-01   4.33166802e-01]
 [ -6.52824612e+00  -3.90107572e+00  -8.54635850e-01  -7.63603335e-02
   -8.20264952e-01   3.16189270e-01  -7.09863657e-01  -3.74970142e+00
   -9.37130006e+00   1.62511286e+00   7.10938450e+00]
 [ -6.21643101e+00  -5.43020702e-01  -1.03680086e-01  -1.32474556e-02
   -1.95822388e-01   1.67467834e-01  -1.38723355e-01  -1.39334802e+00
   -3.78469807e+00  -2.76321927e-01   5.02728441e+00]
 [  4.12920690e-01  -6.03759036e-02   7.82599686e-02  -6.41501787e-01
   -1.87364013e-02   1.86214234e-01  -2.14958713e-01  -8.60348003e-02
   -1.47546661e-01  -3.52682260e-02   4.72909168e-01]]
[ -2.70127932   3.85555628 -24.48004475 -11.15358262   9.23020849
   1.43389982  -5.62008057]
------------------------
mean_absolute_error:
0.548387096774
```

3.2 SGDClassifier(Logarithmic regression: log loss)

3.2.1 Introduction

This is the loss function used in (multinomial) logistic regression and extensions of it such as neural networks, defined as the negative log-likelihood of the true labels given a probabilistic classifier?s predictions. For a single sample with true label yt in 0,1 and estimated probability yp that yt = 1, the log loss is

```
-log P(yt|yp) = -(yt log(yp) + (1 - yt) log(1 - yp))
```

3.2.2 Source Code:

```
import numpy as np
from sklearn.linear_model import SGDClassifier
def loadDataSet():
        feature_train = []
        feature_lable = []
        test_feature = []
        test_lable = []
        fr = open ('winequality-white.csv')
        count = 0
        for line in fr.readlines():
                if count == 0:
                        header = line.strip().split(';')
                elif count > 0 and count < 4000:
                        lineAttr = line.strip().split(';')
                        feature_train.append(
                        [float(lineAttr[0]),float(lineAttr[1]),f
                        loat(lineAttr[2]),float(lineAttr[3]),
                                float(lineAttr[4]),float(lineAttr[5]),
                                float(lineAttr[6]),float(lineAttr[7]),
```

```
                                float(lineAttr[8]),float(lineAttr[9]),
                                float(lineAttr[10])])
                        feature_lable.append(int(lineAttr[11]))
                else:
                        lineAttr = line.strip().split(';')
                        test_feature.append(
                        [float(lineAttr[0]),float(lineAttr[1]),
                        float(lineAttr[2]),float(lineAttr[3]),
                                float(lineAttr[4]),float(lineAttr[5]),
                                float(lineAttr[6]),float(lineAttr[7]),
                                float(lineAttr[8]),float(lineAttr[9]),
                                float(lineAttr[10])])
                        test_lable.append(int(lineAttr[11]))
                count += 1
        return np.array(feature_train), np.array(feature_lable),
        np.array(test_feature), np.array(test_lable)

feature_train, feature_lable, test_feature,test_lable = loadDataSet()

clf = SGDClassifier(loss="log", penalty="l2", alpha=0.001, n_iter=100)
clf.fit (feature_train,feature_lable)
pred = clf.predict (test_feature)
from sklearn.metrics import mean_absolute_error
print 'mean_absolute_error:'
print mean_absolute_error(test_lable, pred)

print 'score:'
print clf.score (test_feature, test_lable)

print clf.coef_
print clf.intercept_

print '------------------------'
from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()
scaler.fit(feature_train)  # Don't cheat - fit only on training data
feature_train = scaler.transform(feature_train)
clf1 = SGDClassifier(loss="log", penalty="l2", alpha=0.001, n_iter=100)
clf1.fit (feature_train,feature_lable)
test_feature = scaler.transform(test_feature)
pred = clf1.predict (test_feature)
from sklearn.metrics import mean_absolute_error
print 'mean_absolute_error:'
print mean_absolute_error(test_lable, pred)
```

3.2.3 Output:

```
mean_absolute_error:
1.92658509455
score:
```

```
0.0367074527253
[[  1.77935667e+00   2.25565415e-01  -9.06985060e-02  -5.33743464e-01
    2.65272736e-02   2.88308055e-01  -1.03374749e-01  -1.96923919e-01
   -6.83591362e-01  -2.23678175e-01  -1.99211861e+00]
 [  3.29048169e+00   3.22622402e+00  -7.46197296e-01  -6.22600806e-01
    1.62363322e-01  -4.72627061e-01  -3.55756756e-03   5.37488195e-01
    2.08304195e+00   1.55082024e-01  -2.80176091e+00]
 [  7.13125923e+00   7.65635990e+00   3.82057284e-01   3.05297572e-01
    9.15280122e-01  -1.83137313e-01  -2.87178982e-01   5.61345979e+00
    1.36018582e+01  -1.53481482e+00  -1.03940812e+01]
 [  3.73036841e-01  -7.89659220e+00   1.36437584e+00   2.97551367e-02
   -7.59445397e-02   4.66069508e-01  -4.01649164e-01  -5.82741256e-01
   -7.05250650e-01   1.74905954e-01   1.29227904e-01]
 [ -5.94224082e+00  -3.87115721e+00  -9.69645763e-01  -3.78714616e-01
   -8.14065927e-01  -1.67338120e-01  -4.15508012e-01  -3.51468280e+00
   -8.60094207e+00   1.64601811e+00   7.30111433e+00]
 [ -6.17621330e+00  -5.17153476e-01  -2.07620095e-01   2.44253209e-01
   -1.97197420e-01   4.66461143e-01  -4.63307272e-01  -1.34186465e+00
   -3.56680783e+00  -2.23935161e-01   5.25919743e+00]
 [  3.54315892e-01  -8.51531744e-02   8.16105239e-02  -9.45954144e-01
   -1.84385320e-02   1.28576561e-01  -5.83662288e-01  -9.87062261e-02
   -1.78963632e-01  -3.69761381e-02   2.41992728e-01]]
[-7.6115563   3.28017682 -9.77908678 -2.8225681   0.21098066 -3.65875393
 -2.01886629]
------------------------
mean_absolute_error:
0.44382647386
```

3.3 LogisticRegression (LBFGS version)

3.3.1 Introduction

In the multiclass case, the training algorithm uses the one-vs-rest (OvR) scheme if the 'multi class' option is set to 'ovr' and uses the cross-entropy loss, if the 'multi class' option is set to 'multinomial'. (Currently the ?multinomial? option is supported only by the 'lbfgs' and 'newton-cg' solvers.) This class implements regularized logistic regression using the liblinear library, newton-cg and lbfgs solvers. It can handle both dense and sparse input. Use C-ordered arrays or CSR matrices containing 64-bit floats for optimal performance; any other input format will be converted (and copied). The newton-cg and lbfgs solvers support only L2 regularization with primal formulation. The liblinear solver supports both L1 and L2 regularization, with a dual formulation only for the L2 penalty.

3.3.2 Source Code:

```
import numpy as np
from sklearn.linear_model import LogisticRegression
def loadDataSet():
        feature_train = []
```

```
        feature_lable = []
        test_feature = []
        test_lable = []
        fr = open ('winequality-white.csv')
        count = 0
        for line in fr.readlines():
                if count == 0:
                        header = line.strip().split(';')
                elif count > 0 and count < 4000:
                        lineAttr = line.strip().split(';')
                        feature_train.append(
                        [float(lineAttr[0]),float(lineAttr[1]),
                        float(lineAttr[2]),float(lineAttr[3]),
                                float(lineAttr[4]),float(lineAttr[5]),
                                float(lineAttr[6]),float(lineAttr[7]),
                                float(lineAttr[8]),float(lineAttr[9]),
                                float(lineAttr[10])])
                        feature_lable.append(int(lineAttr[11]))
                else:
                        lineAttr = line.strip().split(';')
                        test_feature.append(
                        [float(lineAttr[0]),float(lineAttr[1]),
                        float(lineAttr[2]),float(lineAttr[3]),
                                float(lineAttr[4]),float(lineAttr[5]),
                                float(lineAttr[6]),float(lineAttr[7]),
                                float(lineAttr[8]),float(lineAttr[9]),
                                float(lineAttr[10])])
                        test_lable.append(int(lineAttr[11]))
                count += 1
        return np.array(feature_train), np.array(feature_lable),
        np.array(test_feature), np.array(test_lable)

feature_train, feature_lable, test_feature,test_lable = loadDataSet()

clf = LogisticRegression()
clf.fit (feature_train,feature_lable)
pred = clf.predict (test_feature)
from sklearn.metrics import mean_absolute_error
print 'mean_absolute_error:'
print mean_absolute_error(test_lable, pred)

print 'score:'
print clf.score (test_feature, test_lable)

print clf.coef_
print clf.intercept_

print '-------------------------'
from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()
```

```
scaler.fit(feature_train)  # Don't cheat - fit only on training data
feature_train = scaler.transform(feature_train)
clf1 = LogisticRegression()
clf1.fit (feature_train,feature_lable)
test_feature = scaler.transform(test_feature)
pred = clf1.predict (test_feature)
from sklearn.metrics import mean_absolute_error
print 'mean_absolute_error:'
print mean_absolute_error(test_lable, pred)
```

3.3.3 Output:

```
mean_absolute_error:
0.461624026696
score:
0.57063403782
[[  4.83653803e-01   8.16030040e-01  -4.74144666e-01  -6.26436421e-02
    7.71481419e-02   1.49312508e-02   1.06609719e-03  -5.92667004e-01
   -1.22317148e+00  -5.82712379e-01  -3.73461792e-01]
 [  3.23160708e-01   4.20210477e+00  -4.08030987e-01  -6.59149386e-02
    1.04912300e-01  -4.71951997e-02  -5.79417345e-04  -1.12370818e-01
    2.69948924e-01   2.06534511e-01  -5.41631848e-01]
 [  1.12136611e-01   3.31816524e+00   2.54806711e-01  -3.32935134e-02
    3.52749062e-01  -9.04255140e-03   2.96746206e-03   3.20226724e+00
    1.53477006e-01  -1.46491619e+00  -8.51824817e-01]
 [  2.78994676e-02  -3.50989486e+00   8.94141268e-02   1.46369573e-02
    3.88201915e-01   8.60413753e-04   1.27609167e-03  -3.68027437e-01
    5.63566780e-02  -1.36442711e-02   6.86365973e-02]
 [ -1.99503677e-01  -2.85814254e+00  -8.37499564e-01   2.54957137e-02
   -1.23815913e+00   8.35880683e-03  -4.00724842e-03  -3.28132142e+00
   -2.44522375e-01   1.34578333e+00   7.20739659e-01]
 [ -5.27017073e-01  -8.43478128e-01   8.64856303e-02   4.25930805e-02
   -2.91030374e-01   2.37898990e-02  -6.74141051e-03  -1.84351696e+00
   -1.28159295e+00   1.85406737e-01   7.34539073e-01]
 [ -1.76868148e-01   9.51101588e-03   1.31151352e-01  -7.07825490e-02
   -8.61901233e-02   2.57455886e-02  -2.43563296e-02  -4.64708377e-01
   -9.02024338e-01  -2.65333055e-01   9.06553952e-02]]
[-0.59786293 -0.14157174  3.2281473  -0.40046758 -3.27155168 -1.84396807
 -0.46458388]
------------------------
mean_absolute_error:
0.48275862069
```

3.4 Linear Regression

3.4.1 Source Code:

```
def loadDataSet():
        feature_train = []
        feature_lable = []
        test_feature = []
```

```
        test_lable = []
        fr = open ('winequality-white.csv')
        count = 0
        for line in fr.readlines():
                if count == 0:
                        header = line.strip().split(';')
                elif count > 0 and count < 4000:
                        lineAttr = line.strip().split(';')
                        feature_train.append([float(lineAttr[0]),
                        float(lineAttr[1]),
                        float(lineAttr[2]),
                        float(lineAttr[3]),
                                float(lineAttr[4]),
                                float(lineAttr[5]),
                                float(lineAttr[6]),
                                float(lineAttr[7]),
                                float(lineAttr[8]),
                                float(lineAttr[9]),
                                float(lineAttr[10])])
                        feature_lable.append(int(lineAttr[11]))
                else:
                        lineAttr = line.strip().split(';')
                        test_feature.append([float(lineAttr[0]),
                        float(lineAttr[1]),
                        float(lineAttr[2]),
                        float(lineAttr[3]),
                                float(lineAttr[4]),
                                float(lineAttr[5]),
                                float(lineAttr[6]),
                                float(lineAttr[7]),
                                float(lineAttr[8]),
                                float(lineAttr[9]),
                                float(lineAttr[10])])
                        test_lable.append(int(lineAttr[11]))
                count += 1
        return feature_train, feature_lable, test_feature, test_lable

feature_train, feature_lable, test_feature,test_lable = loadDataSet()

#sklearn.linear_model.LinearRegression
from sklearn import linear_model
import numpy as np
from sklearn.metrics import mean_absolute_error
clf = linear_model.LinearRegression()
clf.fit(feature_train, feature_lable)
y_pred = clf.predict (test_feature)
y_true =  np.array(test_lable)
print 'The accuracy for linear_model is: '
accuracy = mean_absolute_error(y_true, y_pred)
print accuracy
```

3.4.2 Output:

```
The accuracy for linear_model is:
0.554217799423
```

3.5 SGD Regressor

3.5.1 Introduction

GD stands for Stochastic Gradient Descent: the gradient of the loss is estimated each sample at a time and the model is updated along the way with a decreasing strength schedule (aka learning rate). The regularizer is a penalty added to the loss function that shrinks model parameters towards the zero vector using either the squared euclidean norm L2 or the absolute norm L1 or a combination of both (Elastic Net). If the parameter update crosses the 0.0 value because of the regularizer, the update is truncated to 0.0 to allow for learning sparse models and achieve online feature selection.

3.5.2 Source Code:

```
import numpy as np
def loadDataSet():
        feature_train = []
        feature_lable = []
        test_feature = []
        test_lable = []
        fr = open ('winequality-white.csv')
        count = 0
        for line in fr.readlines():
                if count == 0:
                        header = line.strip().split(';')
                elif count > 0 and count < 4000:
                        lineAttr = line.strip().split(';')
                        feature_train.append(
                        [float(lineAttr[0]),float(lineAttr[1]),
                        float(lineAttr[2]),float(lineAttr[3]),
                                float(lineAttr[4]),float(lineAttr[5]),
                                float(lineAttr[6]),float(lineAttr[7]),
                                float(lineAttr[8]),float(lineAttr[9]),
                                float(lineAttr[10])])
                        feature_lable.append(int(lineAttr[11]))
                else:
                        lineAttr = line.strip().split(';')
                        test_feature.append(
                        [float(lineAttr[0]),float(lineAttr[1]),
                        float(lineAttr[2]),float(lineAttr[3]),
                                float(lineAttr[4]),float(lineAttr[5]),
                                float(lineAttr[6]),float(lineAttr[7]),
                                float(lineAttr[8]),float(lineAttr[9]),
```

```
                                float(lineAttr[10])])
                        test_lable.append(int(lineAttr[11]))
                count += 1
        return np.array(feature_train), np.array(feature_lable),
        np.array(test_feature), np.array(test_lable)

feature_train, feature_lable, test_feature,test_lable = loadDataSet()

from sklearn import linear_model
clf = linear_model.SGDRegressor()
clf.fit (feature_train,feature_lable)
pred = clf.predict (test_feature)
from sklearn.metrics import mean_absolute_error
print 'mean_absolute_error:'
print mean_absolute_error(test_lable, pred)
```

3.5.3 Output:

```
mean_absolute_error:
2.49420064939e+12
```

3.6 Ridge

3.6.1 Introduction

Linear least squares with l2 regularization. This model solves a regression model where the loss function is the linear least squares function and regularization is given by the l2-norm. Also known as Ridge Regression or Tikhonov regularization. This estimator has built-in support for multi-variate regression (i.e., when y is a 2d-array of shape [n-samples, n-targets]).

3.6.2 Source Code:

```
import numpy as np
def loadDataSet():
        feature_train = []
        feature_lable = []
        test_feature = []
        test_lable = []
        fr = open ('winequality-white.csv')
        count = 0
        for line in fr.readlines():
                if count == 0:
                        header = line.strip().split(';')
                elif count > 0 and count < 4000:
                        lineAttr = line.strip().split(';')
                        feature_train.append(
                        [float(lineAttr[0]),float(lineAttr[1]),
                        float(lineAttr[2]),float(lineAttr[3]),
                                float(lineAttr[4]),float(lineAttr[5]),
```

```
                                float(lineAttr[6]),float(lineAttr[7]),
                                float(lineAttr[8]),float(lineAttr[9]),
                                float(lineAttr[10])])
                        feature_lable.append(int(lineAttr[11]))
                else:
                        lineAttr = line.strip().split(';')
                        test_feature.append(
                        [float(lineAttr[0]),float(lineAttr[1]),
                        float(lineAttr[2]),float(lineAttr[3]),
                                float(lineAttr[4]),float(lineAttr[5]),
                                float(lineAttr[6]),float(lineAttr[7]),
                                float(lineAttr[8]),float(lineAttr[9]),
                                float(lineAttr[10])])
                        test_lable.append(int(lineAttr[11]))
                count += 1
        return np.array(feature_train), np.array(feature_lable),
        np.array(test_feature), np.array(test_lable)

feature_train, feature_lable, test_feature,test_lable = loadDataSet()

from sklearn import linear_model
clf = linear_model.Ridge()
clf.fit (feature_train,feature_lable)
pred = clf.predict (test_feature)
from sklearn.metrics import mean_absolute_error
print 'mean_absolute_error:'
print mean_absolute_error(test_lable, pred)

print 'score:'
print clf.score (test_feature, test_lable)
```

3.6.3 Output:

```
mean_absolute_error:
0.552040601218
score:
0.122108569077
```

3.7 Lasso

3.7.1 Source Code:

```
import numpy as np
def loadDataSet():
        feature_train = []
        feature_lable = []
        test_feature = []
        test_lable = []
        fr = open ('winequality-white.csv')
        count = 0
        for line in fr.readlines():
```

```
                if count == 0:
                        header = line.strip().split(';')
                elif count > 0 and count < 4000:
                        lineAttr = line.strip().split(';')
                        feature_train.append(
                        [float(lineAttr[0]),float(lineAttr[1]),
                        float(lineAttr[2]),float(lineAttr[3]),
                                float(lineAttr[4]),float(lineAttr[5]),
                                float(lineAttr[6]),float(lineAttr[7]),
                                float(lineAttr[8]),float(lineAttr[9]),
                                float(lineAttr[10])])
                        feature_lable.append(int(lineAttr[11]))
                else:
                        lineAttr = line.strip().split(';')
                        test_feature.append(
                        [float(lineAttr[0]),float(lineAttr[1]),
                        float(lineAttr[2]),float(lineAttr[3]),
                                float(lineAttr[4]),float(lineAttr[5]),
                                float(lineAttr[6]),float(lineAttr[7]),
                                float(lineAttr[8]),float(lineAttr[9]),
                                float(lineAttr[10])])
                        test_lable.append(int(lineAttr[11]))
                count += 1
        return np.array(feature_train), np.array(feature_lable),
        np.array(test_feature), np.array(test_lable)

feature_train, feature_lable, test_feature,test_lable = loadDataSet()

from sklearn import linear_model
clf = linear_model.Lasso(alpha=0.1)
clf.fit (feature_train,feature_lable)
pred = clf.predict (test_feature)
from sklearn.metrics import mean_absolute_error
print 'mean_absolute_error:'
print mean_absolute_error(test_lable, pred)

print 'score:'
print clf.score (test_feature, test_lable)

print clf.coef_
print clf.intercept_
```

3.7.2 Output:

```
mean_absolute_error:
0.54749967331
score:
0.104882100588
[-0.          -0.          -0.           0.0035966   -0.
0.00943337
```

```
 -0.00247112 -0.          0.          0.          0.27319437]
3.01045514403
```