zz2694_hw1

October 1, 2019

0.1 Personalization - Homework#1

Zihui Zhou, zz2694@columbia.edu (uni: zz2694) Answer to each question iswritten in the green box below. The homework is collaborated with Yi-ping Tseng (yt2690).

1 Setup

Download and load the data set for red wine into a data frame: http://archive.ics.uci.edu/ml/datasets/Wine+Quality (Use only the red wine data, not the white wine data)

[1]:
```
import os
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from scipy.optimize import minimize
%matplotlib inline
```

[2]:
```
df = pd.read_csv('winequality-red.csv', sep=';') # Setting the separator to␣
 ↪semi-colon to separate the data
df.head()
```

[2]:
```
   fixed acidity  volatile acidity  citric acid  residual sugar  chlorides  \
0            7.4              0.70         0.00             1.9      0.076
1            7.8              0.88         0.00             2.6      0.098
2            7.8              0.76         0.04             2.3      0.092
3           11.2              0.28         0.56             1.9      0.075
4            7.4              0.70         0.00             1.9      0.076

   free sulfur dioxide  total sulfur dioxide  density    pH  sulphates  \
0                 11.0                  34.0   0.9978  3.51       0.56
1                 25.0                  67.0   0.9968  3.20       0.68
2                 15.0                  54.0   0.9970  3.26       0.65
3                 17.0                  60.0   0.9980  3.16       0.58
4                 11.0                  34.0   0.9978  3.51       0.56

   alcohol  quality
0      9.4        5
1      9.8        5
```

```
2      9.8         5
3      9.8         6
4      9.4         5
```

Split the data by columns into features that you will use for prediction, X, and the feature you will try to predict ('quality'), y

[3]:
```
a = df.shape[1] - 1
dfs = np.split(df, [a], axis=1)
X = dfs[0]
y = dfs[1]
X.head()
```

[3]:
```
   fixed acidity  volatile acidity  citric acid  residual sugar  chlorides  \
0            7.4              0.70         0.00             1.9      0.076
1            7.8              0.88         0.00             2.6      0.098
2            7.8              0.76         0.04             2.3      0.092
3           11.2              0.28         0.56             1.9      0.075
4            7.4              0.70         0.00             1.9      0.076

   free sulfur dioxide  total sulfur dioxide  density    pH  sulphates  \
0                 11.0                  34.0   0.9978  3.51       0.56
1                 25.0                  67.0   0.9968  3.20       0.68
2                 15.0                  54.0   0.9970  3.26       0.65
3                 17.0                  60.0   0.9980  3.16       0.58
4                 11.0                  34.0   0.9978  3.51       0.56

   alcohol
0      9.4
1      9.8
2      9.8
3      9.8
4      9.4
```

Split both X and y by rows into training sets and testing sets: Randomly split the data, keeping 80% of instances for training and 20% for testing – At the end, you should have 4 data sets: X_train, y_train, X_test, and y_test

[4]:
```
X_train = X.sample(frac=0.8, random_state=0)
X_test = X.drop(X_train.index)
X_train.head()
```

[4]:
```
      fixed acidity  volatile acidity  citric acid  residual sugar  chlorides  \
1109           10.8             0.470         0.43            2.10      0.171
1032            8.1             0.820         0.00            4.10      0.095
1002            9.1             0.290         0.33            2.05      0.063
487            10.2             0.645         0.36            1.80      0.053
979            12.2             0.450         0.49            1.40      0.075

      free sulfur dioxide  total sulfur dioxide  density    pH  sulphates  \
```

```
1109                    27.0                     66.0  0.99820  3.17       0.76
1032                     5.0                     14.0  0.99854  3.36       0.53
1002                    13.0                     27.0  0.99516  3.26       0.84
487                      5.0                     14.0  0.99820  3.17       0.42
979                      3.0                      6.0  0.99690  3.13       0.63

      alcohol
1109     10.8
1032      9.6
1002     11.7
487      10.0
979      10.4
```

[5]:
```
y_train = y.iloc[X_train.index]
y_test = y.iloc[X_test.index]
y_train.head()
y_test.head()
```

[5]:
```
    quality
11      5.0
23      5.0
24      6.0
25      5.0
28      5.0
```

[6]:
```
y_train.shape
```

[6]:
```
(1279, 1)
```

2 Regression equations and functions

Write out two equations: (1) the equation for a the linear model that predicts y from X, and (2) the equation for computing the Residual Sum of Squares (RSS) for the linear model, given data, vector x, and parameters, vector .

> See equations 3.1 and 3.2 in the Elements of Statistical Learning book Feel free to ignore the intercept term for this homework (e.g. 0)

Answer:

1) The linear model is

$$y = \sum_j x_j \beta_j + \beta_0$$

2) The residual sum of squares (RSS)

$$RSS = \sum_i (y_i - f(x_i))^2$$

Translate these equations into code in the form of two functions

The first function should compute the estimated value of y, which is y_hat, for particular values of x, and . That is, there should be two arguments, one for the data and one for the linear function parameters.) The second function should compute the RSS for the first function

[7]:
```
def predict(beta, X):
    if isinstance(beta, (pd.DataFrame, pd.Series)): # If beta is DataFrame/
↪Series from Pandas, then transform it into numpy arrays
        beta = beta.values

    if isinstance(X, (pd.DataFrame, pd.Series)):
        index = X.index.values
        X = X.values

    if beta.shape[0] - X.shape[1] == 1:
        X = np.concatenate([np.ones(shape=(X.shape[0], 1)), X], axis=1)

    return pd.DataFrame(np.matmul(X, beta), index=index, columns=["y_hat"])


def RSS(beta, X, y):
    pred_y = predict(beta, X)
    return np.sum((y.values - pred_y.values)**2)
```

3 Optimizing the model

Use Scipy's minimize function to find the value of that minimize the RSS https://docs.scipy.org/doc/scipy/reference/generated/scipy.optimize.minimize.html Your call to minimize method will take three arguments:

(1) fun: the RSS function you defined above that you are trying to minimize
(2) x0: your initial values of
(3) args: pass in all the data a tuple here.

For example: args=(y_train, X_train) For the second argument you will need to initialize to some starting value. Try using a random vector with Numpy random methods numpy.random.normal(0, 1, X_train.shape[1]) Your final set of functions to fit your model should have the form: def RSS(beta, X, y):
return res = minimize(fun=RSS, x0=beta0, args=(X_train,y_train)) beta_hat = res.x

[8]:
```
beta = np.random.normal(0, 1, (X_train.shape[1] + 1, 1))
opt = minimize(fun=RSS, x0=beta, args=(X_train, y_train))
beta_hat = opt.x
```

[9]:
```
print(beta_hat)
RSS(beta_hat, X_train, y_train)
```

```
[ 1.40628163e+01  8.27834973e-03 -1.11265715e+00 -2.37056254e-01
  3.94612173e-03 -1.49863999e+00  3.48265450e-03 -2.94954608e-03
 -9.30296839e+00 -5.96736860e-01  9.03382746e-01  2.88019940e-01]
```

[9]:
```
548.7796105427644
```

[10]:
```python
['intercept'] + X_train.columns.values.tolist() # Intercept before the index:␣
↪Pandas(index -> array -> list)
```

[10]:
```
['intercept',
 'fixed acidity',
 'volatile acidity',
 'citric acid',
 'residual sugar',
 'chlorides',
 'free sulfur dioxide',
 'total sulfur dioxide',
 'density',
 'pH',
 'sulphates',
 'alcohol']
```

4 Questions

What are the qualitative results from your model? Which features seem to be most important? Do you think that the magnitude of the features in X may affect the results (for example, the average total sulfur dioxide across all wines is 46.47, but the average chlorides is only 0.087).

Answer:

1. The qualitative results are shown as above
2. Since 'density' feature has the most negative value of -9.302968388095524, it seems to be the most important
3. Magnitude of features does not influence the wine quality: reading from the above average value, we could not see a relationship between the magnitude and the co-efficient of a feature

[11]:
```python
np.amin(beta_hat)
```

[11]:
```
-9.302968388095524
```

[12]:
```python
pd.Series(beta_hat, index=["intercept"] + X_train.columns.values.tolist()).
↪sort_values()
```

[12]:
```
density                -9.302968
chlorides              -1.498640
volatile acidity       -1.112657
pH                     -0.596737
citric acid            -0.237056
total sulfur dioxide   -0.002950
free sulfur dioxide     0.003483
```

```
residual sugar           0.003946
fixed acidity            0.008278
alcohol                  0.288020
sulphates                0.903383
intercept               14.062816
dtype: float64
```

[13]:
```
X_train.describe().loc['mean'].sort_values()
```

[13]:
```
chloridess               0.087347
citric acid              0.271618
volatile acidity         0.525571
sulphates                0.655012
density                  0.996739
residual sugar           2.516341
pH                       3.312588
fixed acidity            8.310164
alcohol                 10.436317
free sulfur dioxide     15.868647
total sulfur dioxide    46.488663
Name: mean, dtype: float64
```

How well does your model fit? You should be able to measure the goodness of fit – RSS, on both the training data and the test data, but only report the results on the test data. In Machine Learning we almost always only care about how well the model fits on data that has not been used to fit the model, because we need to use the model in the future, not the past. Therefore, we only report performance with holdout data, or test data.

[14]:
```
# RSS on the test data
RSS_test = RSS(beta_hat, X_test, y_test)
print('RSS on the test data:', RSS_test)
```

```
RSS on the test data: 119.3432725445164
```

Does the end result or RSS change if you try different initial values of ?

Answer: RSS with new doesn't differentiate a lot from the original RSS, as the following results show.

[15]:
```
# Try different initial values of
beta_new0 = np.random.normal(0, 1, (X_train.shape[1] + 1, 1))
opt = minimize(fun=RSS, x0=beta_new0, args=(X_train, y_train))
beta_hat_new0 = opt.x
```

[16]:
```
# Apply new beta to RSS
RSS_new0_train = RSS(beta_hat_new0, X_train, y_train)
RSS_new0_test = RSS(beta_hat_new0, X_test, y_test)
print('RSS on training data with new initial value of beta:', RSS_new0_train)
print('RSS on test data with new initial value of beta:', RSS_new0_test)
print('Comparison of RSS:', RSS_new0_test - RSS_test)
```

```
RSS on training data with new initial value of beta: 548.7796105432501
RSS on test data with new initial value of beta: 119.34328533389389
Comparison of RSS: 1.278937749304987e-05
```

What happens if you change the magnitude of the initial ?

Answer: Changing the magnitude of the initial doesn't have much influence on RSS.

[17]:
```
# Try different initial values of
beta_new = np.random.normal(10, 10, (X_train.shape[1] + 1, 1))
opt = minimize(fun=RSS, x0=beta_new, args=(X_train, y_train))
beta_hat_new = opt.x
```

[18]:
```
# Apply new beta to RSS
RSS_new_train = RSS(beta_hat_new, X_train, y_train)
RSS_new_test = RSS(beta_hat_new, X_test, y_test)
print('RSS on training data with new beta:', RSS_new_train)
print('RSS on test data with new beta:', RSS_new_test)
```

```
RSS on training data with new beta: 548.7796105408585
RSS on test data with new beta: 119.34326306103756
```

[19]:
```
# RSS with original beta
RSS_train = RSS(beta_hat, X_train, y_train)
RSS_test = RSS(beta_hat, X_test, y_test)
print('RSS on training data:', RSS_train)
print('RSS on test data:', RSS_test)
```

```
RSS on training data: 548.7796105427644
RSS on test data: 119.3432725445164
```

Does the choice of solver method change the end result or RSS?

Answer: Trying different methods in the optimize.minimize library, we can see that RSS are different for each method, but do not vary a lot.

[20]:
```
opt = minimize(fun=RSS, x0=beta, args=(X_train, y_train), method = 'TNC')
beta_hat_method = opt.x
RSS_train_method = RSS(beta_hat_method, X_train, y_train)
RSS_test_method = RSS(beta_hat_method, X_test, y_test)
print('RSS on training data with new method:', RSS_train_method)
print('RSS on test data with new method:', RSS_test_method)
```

```
RSS on training data with new method: 635.590110784124
RSS on test data with new method: 134.9063220417022
```

[21]:
```
opt = minimize(fun=RSS, x0=beta, args=(X_train, y_train), method = 'Powell')
beta_hat_method = opt.x
```

```
RSS_train_method = RSS(beta_hat_method, X_train, y_train)
RSS_test_method = RSS(beta_hat_method, X_test, y_test)
print('RSS on training data with new method:', RSS_train_method)
print('RSS on test data with new method:', RSS_test_method)
```

```
RSS on training data with new method: 548.8676614320452
RSS on test data with new method: 119.59571084225763
```

[22]:
```
opt = minimize(fun=RSS, x0=beta, args=(X_train, y_train), method = 'BFGS')
beta_hat_method = opt.x
RSS_train_method = RSS(beta_hat_method, X_train, y_train)
RSS_test_method = RSS(beta_hat_method, X_test, y_test)
print('RSS on training data with new method:', RSS_train_method)
print('RSS on test data with new method:', RSS_test_method)
```

```
RSS on training data with new method: 548.7796105427644
RSS on test data with new method: 119.3432725445164
```

5 Regularizing the model

Regularization seeks to simplify a model by decreasing the model's complexity and degrees of freedom. While lowering the degrees of freedom also decreases the flexibility of the model, and therefore the performance of the model on training data, it increases generalizability, and thus it often increases performance on test data. One common method of regularization is called shrinkage, and is defined in section 3.4 of Elements of Statistical Learning.

Try adding in an L2 (aka Ridge) regularization penalty to your model above to create a new, regularized model. See equation 3.41 for guidance. You will need to choose a value of lambda, so start with something small, like 0.01.

[23]:
```
# Ridge Regression

def RSS_L2(beta, X, y, lam):
    pred_y = predict(beta, X)
    penalty = lam * np.sum(beta[1:]**2)
    return np.sum((y.values - pred_y.values)**2) + penalty
```

How does RSS on the training data change? How does RSS on the test data change?

Answer: Both RSS on the training data and test data get slightly larger.

[24]:
```
beta = np.random.normal(0, 1, (X_train.shape[1] + 1, 1))
lam = 0.01
opt = minimize(fun=RSS_L2, x0=beta, args=(X_train, y_train, lam))
beta_hat_L2 = opt.x
```

[25]:
```
```

```
print('RSS with Ridge Regression on the training data:', RSS_L2(beta_hat_L2,
    X_train, y_train, lam))
print('RSS with Ridge Regression on the test data:', RSS_L2(beta_hat_L2, X_test,
    y_test, lam))
# RSS with original beta
RSS_train = RSS(beta_hat, X_train, y_train)
RSS_test = RSS(beta_hat, X_test, y_test)
print('RSS on training data:', RSS_train)
print('RSS on test data:', RSS_test)
```

```
RSS with Ridge Regression on the training data: 548.8874194420018
RSS with Ridge Regression on the test data: 119.58927364152647
RSS on training data: 548.7796105427644
RSS on test data: 119.3432725445164
```

What happens if you try different values of lambda? Can you tune lambda to get the best results on the test data?

Answer: We can notice that RSS increases with larger lambda, the best result was achieved when lambda is smaller, such as $\lambda = 0.001$

[26]:
```
lam = 0.001
opt = minimize(fun=RSS_L2, x0=beta, args=(X_train, y_train, lam))
beta_hat_L2 = opt.x
print('RSS with Ridge Regression on the training data:', RSS_L2(beta_hat_L2,
    X_train, y_train, lam))
print('RSS with Ridge Regression on the test data:', RSS_L2(beta_hat_L2, X_test,
    y_test, lam))
```

```
RSS with Ridge Regression on the training data: 548.8210774717589
RSS with Ridge Regression on the test data: 119.47404581491647
```

[27]:
```
lam = 0.1
opt = minimize(fun=RSS_L2, x0=beta, args=(X_train, y_train, lam))
beta_hat_L2 = opt.x
print('RSS with Ridge Regression on the training data:', RSS_L2(beta_hat_L2,
    X_train, y_train, lam))
print('RSS with Ridge Regression on the test data:', RSS_L2(beta_hat_L2, X_test,
    y_test, lam))
```

```
RSS with Ridge Regression on the training data: 549.3177967429126
RSS with Ridge Regression on the test data: 120.11899653645405
```

[28]:
```
lam = 1
opt = minimize(fun=RSS_L2, x0=beta, args=(X_train, y_train, lam))
beta_hat_L2 = opt.x
```

```
print('RSS with Ridge Regression on the training data:', RSS_L2(beta_hat_L2,␣
 ↪X_train, y_train, lam))
print('RSS with Ridge Regression on the test data:', RSS_L2(beta_hat_L2, X_test,␣
 ↪y_test, lam))
```

```
RSS with Ridge Regression on the training data: 552.7758931505845
RSS with Ridge Regression on the test data: 123.74730733493575
```

Now try using an L1 (aka Lasso) regularization penalty instead. See equation 3.51 for example. Report your findings on how RSS changes, and if you can roughly tune lambda.

Answer: Again, we can notice that RSS increases with larger lambda, the best result was achieved when lambda is smaller, such as $\lambda = 0.001$

[29]:
```
# Lasso Regression

def RSS_L1(beta, X, y, lam):
    pred_y = predict(beta, X)
    penalty = lam * np.sum(np.absolute(beta[1:]))
    return np.sum((y.values - pred_y.values)**2) + penalty
```

[30]:
```
lam = 0.001
opt = minimize(fun=RSS_L1, x0=beta, args=(X_train, y_train, lam))
beta_hat_L1 = opt.x
print('RSS with Lasso Regression on the training data:', RSS_L1(beta_hat_L1,␣
 ↪X_train, y_train, lam))
print('RSS with Lasso Regression on the test data:', RSS_L1(beta_hat_L1, X_test,␣
 ↪y_test, lam))
```

```
RSS with Lasso Regression on the training data: 548.793233939927
RSS with Lasso Regression on the test data: 119.37005336726712
```

[31]:
```
lam = 0.01
opt = minimize(fun=RSS_L1, x0=beta, args=(X_train, y_train, lam))
beta_hat_L1 = opt.x
print('RSS with Lasso Regression on the training data:', RSS_L1(beta_hat_L1,␣
 ↪X_train, y_train, lam))
print('RSS with Lasso Regression on the test data:', RSS_L1(beta_hat_L1, X_test,␣
 ↪y_test, lam))
```

```
RSS with Lasso Regression on the training data: 548.8854577078587
RSS with Lasso Regression on the test data: 119.55767969188275
```

[32]:
```
lam = 0.1
opt = minimize(fun=RSS_L1, x0=beta, args=(X_train, y_train, lam))
beta_hat_L1 = opt.x
```

```
print('RSS with Lasso Regression on the training data:', RSS_L1(beta_hat_L1,␣
↪X_train, y_train, lam))
print('RSS with Lasso Regression on the test data:', RSS_L1(beta_hat_L1, X_test,␣
↪y_test, lam))
```

```
RSS with Lasso Regression on the training data: 549.3117676532406
RSS with Lasso Regression on the test data: 120.02740133532379
```

[33]:
```
lam = 1
opt = minimize(fun=RSS_L1, x0=beta, args=(X_train, y_train, lam))
beta_hat_L1 = opt.x
print('RSS with Lasso Regression on the training data:', RSS_L1(beta_hat_L1,␣
↪X_train, y_train, lam))
print('RSS with Lasso Regression on the test data:', RSS_L1(beta_hat_L1, X_test,␣
↪y_test, lam))
```

```
RSS with Lasso Regression on the training data: 553.3432492386936
RSS with Lasso Regression on the test data: 124.14037483347833
```

Again, do you think that the magnitude of the features in X may affect the results with regularization?

Answer: No, since the results with regularization do not depend on the magnitude of the features. Instead, it depends on the importance of the features. Therefore, the magnitude of the features in X do not affect the results with regularization.