Personalization Theory - HW1

October 2, 2019

1 Name: Harguna Sood

2 Email: hs3159@columbia.edu

3 Github: harguna.sood@gmail.com

4 UNI: hs3159

[1]:
```
import pandas as pd
from sklearn.model_selection import train_test_split  as tts
import numpy as np
from scipy.optimize import minimize
import random
from sklearn.metrics import mean_squared_error as mse
```

[2]:
```
data = pd.read_csv("/Users/hargunasood/Downloads/winequality-red.csv",␣
 ↪delimiter=";")
data.head()
X = data.drop("quality", axis=1)
y = data.loc[:, "quality"]
```

[3]:
```
X_train, X_test, y_train, y_test = tts(X, y, test_size=0.2, shuffle="False",␣
 ↪random_state=5)
```

Equations:

```
y_hat[i] = b[1]*x1[i] + b[2]*x2[i] + b[3]*x3[i] + b[4]*x4[i] + ... + b[n]*x5[i]

rss = (y[i] - y_hat[i])^2
```

[4]:
```
def eq_calc(x, beta):
    y_hat = [0]*len(x)
    x = x.values

    for i in range(len(x)):
        for j in range(len(beta)):
            y_hat[i] = y_hat[i] + beta[j]*x[i, j]
    return (y_hat)

def rss(beta, x, y):
    y_hat = eq_calc(x, beta)
    rss_value = 0
    y = list(y)

    for i in range(len(y_hat)):
        rss_value = rss_value + (y[i] - y_hat[i])**2

    return (rss_value)
```

[5]:
```
res = minimize(fun = rss, x0 = np.random.normal(0, 1, X_train.shape[1]),
                               args = (X_train, y_train))
print(rss(res.x, X_train, y_train))
print(rss(res.x, X_test, y_test))
print(res.x)
```

```
539.8870842253951
128.94933772839764
[ 7.90333756e-03 -1.15055890e+00 -1.28864706e-01  1.90435786e-02
 -1.99076083e+00  3.07793975e-03 -2.91717215e-03  4.63150956e+00
 -5.01209402e-01  8.68376475e-01  2.77283939e-01]
```

[6]:
```
print(X_train.columns)
```

```
Index(['fixed acidity', 'volatile acidity', 'citric acid', 'residual sugar',
       'chlorides', 'free sulfur dioxide', 'total sulfur dioxide', 'density',
       'pH', 'sulphates', 'alcohol'],
      dtype='object')
```

Q. 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).

R. The results give the parameter values (beta values) for the equation of linear regression written earlier.

Features which seem to be important: 'density', 'chlorides', 'volatile acidity', 'sulphates'. (This represents only the value of beta and NOT the correlation. So, we still cannot say which feature is important or which is not)

The magnitude of the feature should not affect the results (RSS) as in such a case, the parameter value (beta value) assigned to that particular feature can be changed (reduced) to not let magnitude affect the whole calculation for 'y_hat'. However, it will change the "important" features as it will affect the respective beta value.

Q. How well does your model fit?

R. Calculating RMSE and R Squared to check goodness of fit:

Calculating RMSE:

```
[7]: print(np.sqrt(mse(eq_calc(X_test, res.x), y_test)))
```

```
0.6347965661542624
```

Calculating R Squared:

```
[8]: def ss(y):
         y = list(y)
         ss = 0
         y_avg = np.mean(y)

         for i in range(len(y)):
             ss = ss + (y[i] - y_avg)**2

         return (ss)

     def r_sq(rss_, ss_):
         return(1-(rss_/ss_))
```

```
[9]: print(r_sq(rss(res.x, X_test, y_test), ss(y_test)))
```

```
0.3790717316516854
```

Conclusion: Judging by the metrics above, it can be concluded that the model does not fit the data very well.

Q. Does the end result or RSS change if you try different initial values of ? What happens if you change the magnitude of the initial ?

R. Running the 'random' function again to try different values of initial beta:

```
[10]: res = minimize(fun = rss, x0 = [round(random.uniform(0,10),1) for i in␣
       ↪range(X_train.shape[1])],
                                      args = (X_train, y_train))
      print(rss(res.x, X_train, y_train))
      print(rss(res.x, X_test, y_test))
      print(res.x)
```

```
539.8870842245311
128.94933625144355
[ 7.90365805e-03 -1.15056151e+00 -1.28867804e-01  1.90435614e-02
 -1.99075711e+00  3.07792656e-03 -2.91716399e-03  4.63150178e+00
 -5.01207668e-01  8.68376302e-01  2.77284056e-01]
```

Since, we still get the same values, it proves that changing initial values of beta does not change the result.

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

R. Trying different methods:

[11]:
```
beta0 = [round(random.uniform(0,10),1) for i in range(X_train.shape[1])]

for met in ["Nelder-Mead", "Powell", "BFGS", "Newton-CG", "L-BFGS-B", "TNC",␣
↪"COBYLA", "SLSQP", "trust-constr", "dogleg", "trust-ncg", "trust-exact",␣
↪"trust-krylov"]:
    try:
        res = minimize(fun = rss, x0 = beta0, args = (X_train, y_train),␣
↪method=str(met))
        print(str(met)  + "'s RSS: " + str(rss(res.x, X_test, y_test)))
       ␣
↪print("_________________________________________________________________")
       ␣
↪print("_________________________________________________________________")

    except Exception as e:
        print(str(met), str(e))
```

```
Nelder-Mead's RSS: 336.6936660687132
_________________________________________________________________
_________________________________________________________________
Powell's RSS: 127.60156322640495
_________________________________________________________________
_________________________________________________________________
BFGS's RSS: 128.94933956403943
_________________________________________________________________
_________________________________________________________________
Newton-CG Jacobian is required for Newton-CG method
L-BFGS-B's RSS: 128.98098268741464
_________________________________________________________________
_________________________________________________________________
TNC's RSS: 170482.61629152394
_________________________________________________________________
_________________________________________________________________
COBYLA's RSS: 16548.061161057347
_________________________________________________________________
_________________________________________________________________
SLSQP's RSS: 128.9493652702524
_________________________________________________________________
_________________________________________________________________
trust-constr's RSS: 128.94944201638597
_________________________________________________________________
_________________________________________________________________
dogleg Jacobian is required for dogleg minimization
trust-ncg Jacobian is required for Newton-CG trust-region minimization
```

```
trust-exact Jacobian is required for trust region exact minimization.
trust-krylov ('Jacobian is required for trust region ', 'exact minimization.')
```

As it can be seen, majority of the methods give the same result for RSS, except for Nelar-Mead(gives different result everytime), TNC, COBYLA which under-performed.

Q. Try adding in an L2 (aka Ridge) regularization penalty to your model above to create a new, regularized model

5 Ridge

[12]:
```
def ridge(beta, x, y, l):
    y_hat = eq_calc(x, beta)
    rss_value = 0
    y = list(y)

    for i in range(len(y_hat)):
        rss_value = rss_value + (y[i] - y_hat[i])**2

    for i in range(len(beta)):
        rss_value = rss_value + l*(beta[i]**2)

    return (rss_value)
```

[13]:
```
res = minimize(fun = ridge, x0 = [round(random.uniform(0,10),1) for i in␣
 ↪range(X_train.shape[1])],
                                args = (X_train, y_train, 0.01))
print("RSS (train):", rss(res.x, X_train, y_train))
print("RSS (test):", rss(res.x, X_test, y_test))
```

```
RSS (train): 539.8901530427785
RSS (test): 128.9353943372993
```

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

R. Compared to unregularized, train set RSS and test set RSS are almost the same.
RSS:
For train set, from 539.89 (unregularized) to 539.89 (regularized)
For test set, from 128.95 (unregularized) to 128.94 (regularized)

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

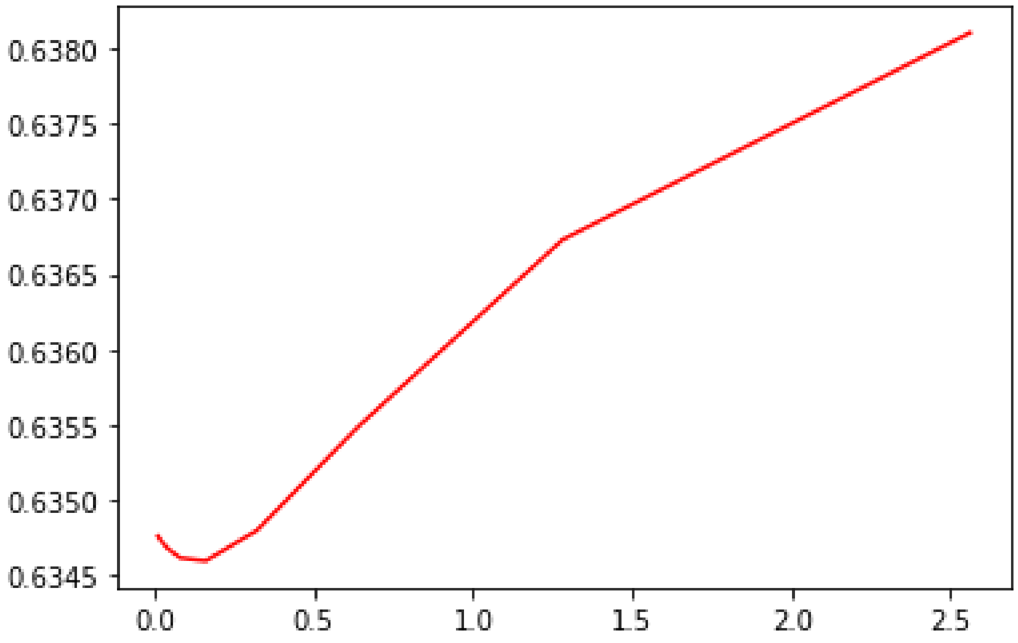
R. Plotting a graph showing change of test and train set R Sqaured and RMSE , with lambda, to tune lambda:

[14]:
```
l_values = []
test_rmse = []
```

```
test_r_sq = []

for l in [0.01*(2**i) for i in range(9)]:
    l_values.append(l)
    res = minimize(fun = ridge, x0 = [round(random.uniform(0,10),1) for j in␣
 ↪range(X_train.shape[1])],
                                    args = (X_train, y_train, l))
    test_rmse.append(np.sqrt(mse(eq_calc(X_test, res.x), y_test)))
    test_r_sq.append(r_sq(rss(res.x, X_test, y_test), ss(y_test)))
```

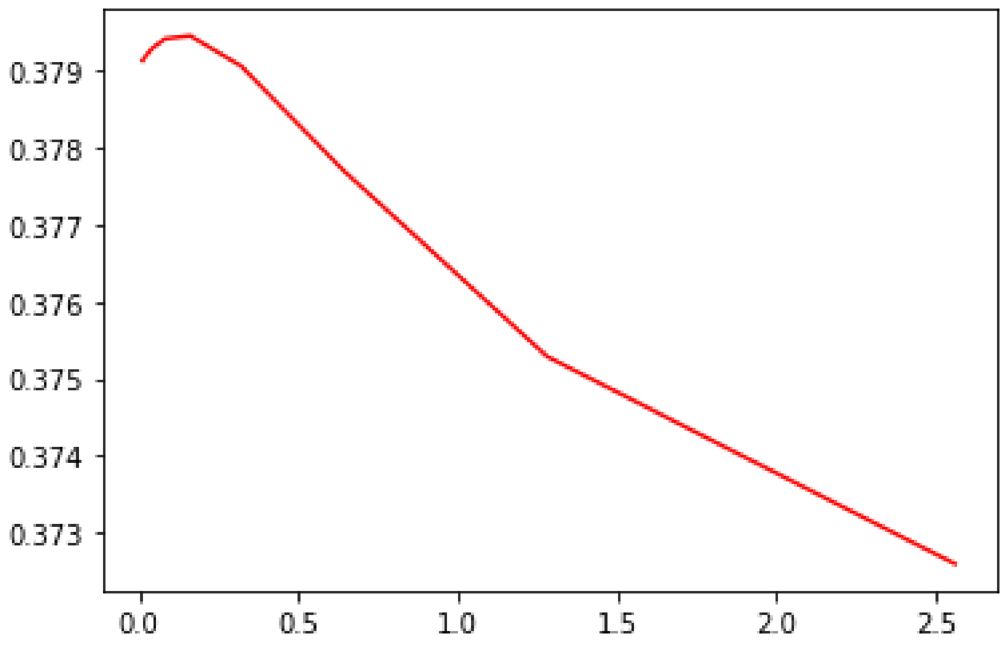
RMSE graph:

[16]:
```
from matplotlib import pyplot as plt

plt.plot(l_values, test_rmse, color="red")
plt.show()
```



R Squared graph:

[17]:
```
from matplotlib import pyplot as plt

plt.plot(l_values, test_r_sq, color="red")
plt.show()
```



We can see, closer to 0.2, the model's performance is relatively better.

```
[18]: res = minimize(fun = ridge, x0 = [round(random.uniform(0,10),1) for i in␣
       ↪range(X_train.shape[1])],
                                        args = (X_train, y_train, 0.2))
      print("RSS (test):", rss(res.x, X_test, y_test))
```

```
RSS (test): 128.88011635323952
```

RSS (test) value has decreased from 128.95 to 128.88

6 Lasso

```
[19]: def lasso(beta, x, y, l):
          y_hat = eq_calc(x, beta)
          rss_value = 0
          y = list(y)

          for i in range(len(y_hat)):
              rss_value = rss_value + (y[i] - y_hat[i])**2

          for i in range(len(beta)):
              rss_value = rss_value + l*abs(beta[i])

          return (rss_value)
```

[20]:
```
res = minimize(fun = lasso, x0 = [round(random.uniform(0,10),1) for i in␣
↪range(X_train.shape[1])],
                                args = (X_train, y_train, 0.01))
print("RSS (train):", rss(res.x, X_train, y_train))
print("RSS (test):", rss(res.x, X_test, y_test))
```

```
RSS (train): 539.8871520805058
RSS (test): 128.94804802229768
```

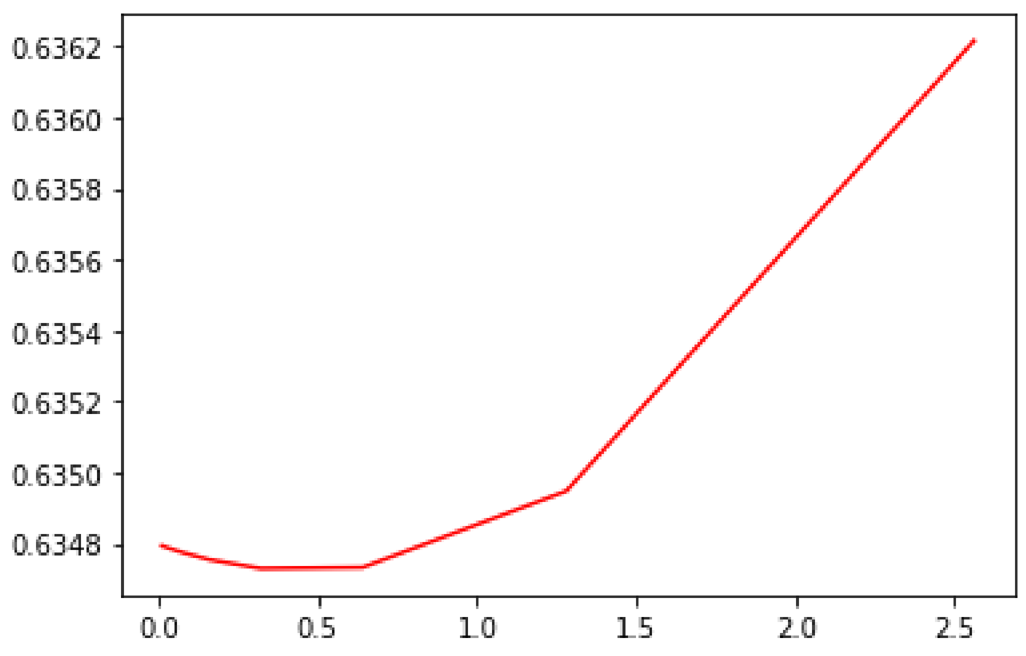
Compared to unregularized, train set RSS and test set RSS are almost the same.
RSS:
For train set, from 539.89 (unregularized) to 539.89 (regularized)
For test set, from 128.95 (unregularized) to 128.95 (regularized)

Plotting a graph showing change of test and train set R Sqaured and RMSE, with lambda, to tune lambda:

[21]:
```
l_values = []
test_rmse = []
test_r_sq = []

for l in [0.01*(2**i) for i in range(9)]:
    l_values.append(l)
    res = minimize(fun = lasso, x0 = [round(random.uniform(0,10),1) for j in␣
↪range(X_train.shape[1])],
                                    args = (X_train, y_train, l))
    test_rmse.append(np.sqrt(mse(eq_calc(X_test, res.x), y_test)))
    test_r_sq.append(r_sq(rss(res.x, X_test, y_test), ss(y_test)))
```

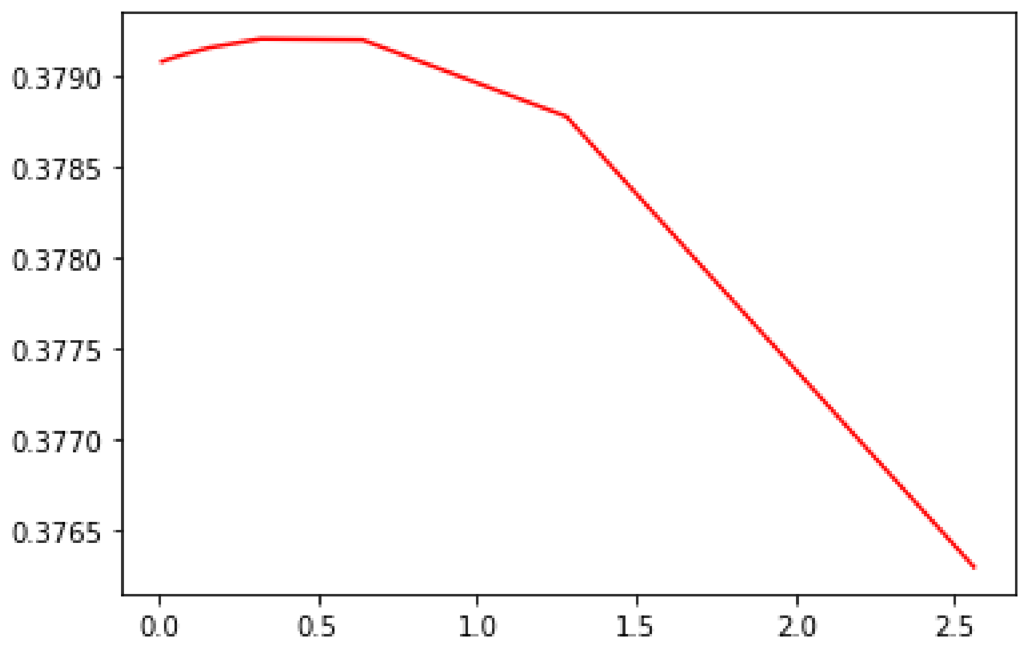
RMSE graph:

[22]:
```
from matplotlib import pyplot as plt

plt.plot(l_values, test_rmse, color="red")
plt.show()
```



R Squared graph:

[23]:
```
from matplotlib import pyplot as plt

plt.plot(l_values, test_r_sq, color="red")
plt.show()
```



We can see that model performs relatively better when lambda is closer to 0.6.

[24]:
```
res = minimize(fun = lasso, x0 = [round(random.uniform(0,10),1) for i in␣
 ↪range(X_train.shape[1])],
                                args = (X_train, y_train, 0.6))
print("RSS (test):", rss(res.x, X_test, y_test))
```

```
RSS (test): 128.92159977985997
```

RSS (test) has reduced from 128.95 to 128.92

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

R. Yes, the magnitude will affect the results (RSS) as in this case, if the value of feature is high, it's parameter value will be low and thus, it will affect the RSS calculation for ridge during optimization, thus the beta values and evetually the final RSS value. Vice versa is true for low magnitude of feature.