IEOR 165 Final Project

May 9, 2021

1 IEOR 165 Final Project

Name: Won Shil Park

SID: 3033452021

The authors of the following research paper:Cortez, A. Cerdeira, F. Almeida, T. Matos, and J. Reis, “Modeling wine preferences by data miningfrom physicochemical properties”,Decision Support Systems, vol. 47, no. 4:547-553, 2009.

considered the problem of modeling wine preferences. Wine can be evaluated by experts whogive a subjective score, and the question the authors of this paper considered was how to builda model that relates objective features of the wine (e.g., pH values) to its rated quality. For this project, we will use the data set available at: http://courses.ieor.berkeley.edu/ieor165/homeworks/winequality-red.csv

Use the following methods to identify the coefficients of a linear model relating wine quality to different features of the wine: (1) ordinary least squares (OLS), (2) ridge regression (RR), (3) lasso regression. Make sure to include a constant (intercept) term in your model, and choose thetuning parameters using cross-validation. You may use any programming language you would liketo. For your solutions, please include (i) plots of tuning parameters versus cross-validation error, (ii) tables of coefficients (labeled by the feature) computed by each method, and (iii) the sourcecode used to generate the plots and coefficients.

[1]:
```
# import helpful functions
import numpy as np
import pandas as pd

import matplotlib.pyplot as plt
%matplotlib inline

import seaborn as sns
sns.set(style = "whitegrid",
        color_codes = True,
        font_scale = 1.5)

from sklearn import linear_model
from sklearn.linear_model import LinearRegression
from sklearn.linear_model import Ridge
from sklearn.linear_model import Lasso
```

```
from sklearn.model_selection import train_test_split
from sklearn.model_selection import cross_val_score
```

2 Load Data

[2]:
```
data = pd.read_csv('winequality-red.csv')
data = data.fillna('')
data
```

[2]:
```
      fixed acidity  volatile acidity  citric acid  residual sugar  chlorides  \
0               7.4             0.700         0.00             1.9      0.076
1               7.8             0.880         0.00             2.6      0.098
2               7.8             0.760         0.04             2.3      0.092
3              11.2             0.280         0.56             1.9      0.075
4               7.4             0.700         0.00             1.9      0.076
…                …                …          …               …          …
1594            6.2             0.600         0.08             2.0      0.090
1595            5.9             0.550         0.10             2.2      0.062
1596            6.3             0.510         0.13             2.3      0.076
1597            5.9             0.645         0.12             2.0      0.075
1598            6.0             0.310         0.47             3.6      0.067

      free sulfur dioxide  total sulfur dioxide  density    pH  sulphates  \
0                    11.0                  34.0  0.99780  3.51       0.56
1                    25.0                  67.0  0.99680  3.20       0.68
2                    15.0                  54.0  0.99700  3.26       0.65
3                    17.0                  60.0  0.99800  3.16       0.58
4                    11.0                  34.0  0.99780  3.51       0.56
…                     …                   …      …     …        …
1594                 32.0                  44.0  0.99490  3.45       0.58
1595                 39.0                  51.0  0.99512  3.52       0.76
1596                 29.0                  40.0  0.99574  3.42       0.75
1597                 32.0                  44.0  0.99547  3.57       0.71
1598                 18.0                  42.0  0.99549  3.39       0.66

      alcohol  quality
0         9.4        5
1         9.8        5
2         9.8        5
3         9.8        6
4         9.4        5
…        …      …
1594     10.5        5
1595     11.2        6
1596     11.0        6
1597     10.2        5
```

```
1598     11.0        6

[1599 rows x 12 columns]
```

[3]:
```python
col = data.iloc[:, 0:11]
qual = data.iloc[:, 11]
print(col)
print(qual)
```

```
      fixed acidity  volatile acidity  citric acid  residual sugar  chlorides  \
0               7.4             0.700         0.00             1.9      0.076
1               7.8             0.880         0.00             2.6      0.098
2               7.8             0.760         0.04             2.3      0.092
3              11.2             0.280         0.56             1.9      0.075
4               7.4             0.700         0.00             1.9      0.076
…               …                 …          …               …        …
1594            6.2             0.600         0.08             2.0      0.090
1595            5.9             0.550         0.10             2.2      0.062
1596            6.3             0.510         0.13             2.3      0.076
1597            5.9             0.645         0.12             2.0      0.075
1598            6.0             0.310         0.47             3.6      0.067

      free sulfur dioxide  total sulfur dioxide  density    pH  sulphates  \
0                    11.0                  34.0  0.99780  3.51       0.56
1                    25.0                  67.0  0.99680  3.20       0.68
2                    15.0                  54.0  0.99700  3.26       0.65
3                    17.0                  60.0  0.99800  3.16       0.58
4                    11.0                  34.0  0.99780  3.51       0.56
…                     …                   …      …    …        …
1594                 32.0                  44.0  0.99490  3.45       0.58
1595                 39.0                  51.0  0.99512  3.52       0.76
1596                 29.0                  40.0  0.99574  3.42       0.75
1597                 32.0                  44.0  0.99547  3.57       0.71
1598                 18.0                  42.0  0.99549  3.39       0.66

      alcohol
0         9.4
1         9.8
2         9.8
3         9.8
4         9.4
…       …
1594     10.5
1595     11.2
1596     11.0
1597     10.2
1598     11.0
```

```
[1599 rows x 11 columns]
0       5
1       5
2       5
3       6
4       5
       ..
1594    5
1595    6
1596    6
1597    5
1598    6
Name: quality, Length: 1599, dtype: int64
```

3 Training Validation Split

[4]:
```
X_train, X_test, Y_train, Y_test = train_test_split(col, qual, test_size = 0.1,␣
 ↪random_state = 42)
X_train
X_test
Y_train
Y_test
```

[4]:
```
803     6
124     5
350     6
682     5
1326    6
       ..
1522    5
297     5
405     6
1378    6
1049    6
Name: quality, Length: 160, dtype: int64
```

4 Ordinary Least Squares

No summary () in lr. Refer to: https://stackoverflow.com/questions/64148189/linearregression-object-has-no-attribute-summary

[5]:
```
ols_model = linear_model.LinearRegression(fit_intercept=True)
ols_model.fit(X_train, Y_train)

coef_access = ols_model.coef_
```

```
inter_access = ols_model.intercept_
print("Coefficient is: " + str(coef_access))
print("Intercept is: " + str(inter_access))

# ols coefficient is close to 0, wine quality can't relate. Use only selected␣
↪columns
col1 = data.iloc[:, [0, 3, 5, 6, 8, 10, 11]]
X_train1, X_test1, Y_train1, Y_test1 = train_test_split(col1, qual, test_size =␣
↪0.1, random_state = 42)
ols_model1 = linear_model.LinearRegression(fit_intercept=True)
ols_model1.fit(X_train1, Y_train1)

coef_access1 = ols_model1.coef_
inter_access1 = ols_model1.intercept_
print("\nCoefficient after taking out data is: " + str(coef_access1))
print("Intercept after taking out data is: " + str(inter_access1))
```

```
Coefficient is: [ 2.25927798e-02 -1.04214422e+00 -1.03974475e-01  1.33101561e-02
 -1.76414361e+00  4.62739928e-03 -3.24754611e-03 -1.72396293e+01
 -3.42743209e-01  8.76382340e-01  2.71548817e-01]
Intercept is: 21.13255032686593

Coefficient after taking out data is: [-2.34274376e-17  2.11636264e-16
-4.16333634e-17  1.73472348e-17
 -1.24932616e-15  1.55040911e-16  1.00000000e+00]
Intercept after taking out data is: 8.881784197001252e-16
```

[6]:
```
ols_coef_table = pd.DataFrame(data = col.columns).rename({0: "features"}, axis␣
↪= 1)
ols_coef_table["coefficients"] = coef_access
ols_coef_table
```

[6]:

	features	coefficients
0	fixed acidity	0.022593
1	volatile acidity	-1.042144
2	citric acid	-0.103974
3	residual sugar	0.013310
4	chlorides	-1.764144
5	free sulfur dioxide	0.004627
6	total sulfur dioxide	-0.003248
7	density	-17.239629
8	pH	-0.342743
9	sulphates	0.876382
10	alcohol	0.271549

[7]:
```
ols_coef_table = pd.DataFrame(data = col1.columns).rename({0: "features"}, axis␣
↪= 1)
```

```
ols_coef_table["coefficients"] = coef_access1
ols_coef_table
```

[7]:

	features	coefficients
0	fixed acidity	-2.342744e-17
1	residual sugar	2.116363e-16
2	free sulfur dioxide	-4.163336e-17
3	total sulfur dioxide	1.734723e-17
4	pH	-1.249326e-15
5	alcohol	1.550409e-16
6	quality	1.000000e+00

5 Ridge Regression

Implementation: https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.Ridge.html

Definition: http://courses.ieor.berkeley.edu/ieor165/lecture_notes/ieor165_lec7.pdf

Cross-Validation: http://courses.ieor.berkeley.edu/ieor165/lecture_notes/ieor165_lec9.pdf, https://www.kaggle.com/jnikhilsai/cross-validation-with-linear-regression

6 k=5 kfold cross validation

[8]:

```
# Cross Validation
# lamdas = 0 is just OLS. Use range to figure out what happens to abs value
lambdas = np.arange(-5, 5, 0.1)
est_error = []
for i in lambdas:
    # ridge_model replace lm
    ridge_model = Ridge(alpha = i, fit_intercept = True)
    # cv = 5 default
    scores = np.mean(cross_val_score(ridge_model, X_train, Y_train, scoring =␣
 ↪'neg_mean_squared_error'))
    est_error.append(scores)
est_error = np.abs(est_error)
est_error
```

[8]:

```
array([ 0.45027482,  0.45001618,  0.44980281,  0.44963537,  0.44951498,
        0.4494433 ,  0.44942265,  0.44945605,  0.44954743,  0.44970177,
        0.4499254 ,  0.45022628,  0.45061447,  0.45110272,  0.45170731,
        0.45244921,  0.45335571,  0.45446283,  0.45581889,  0.45748993,
        0.45956834,  0.46218673,  0.46554176,  0.4699364 ,  0.4758602 ,
        0.48415303,  0.49637285,  0.51573364,  0.54996893,  0.6228668 ,
        0.85128607,  3.82088315, 17.15668824,  2.10161036,  7.44985635,
        1.73872669,  0.66744009,  0.52548953,  0.47975272,  0.45954   ,
        0.44900201,  0.44291978,  0.43916034,  0.43672061,  0.4350797 ,
        0.43394637,  0.43314821,  0.43257837,  0.43216819,  0.43187272,
```

```
0.43142357, 0.43150016, 0.43139394, 0.43132132, 0.43127479,
0.43124841, 0.43123768, 0.43123917, 0.43125022, 0.4312688 ,
0.43129332, 0.43132255, 0.4313555 , 0.4313914 , 0.43142965,
0.43146973, 0.43151128, 0.43155397, 0.43159755, 0.43164182,
0.43168662, 0.43173181, 0.43177729, 0.43182297, 0.43186879,
0.43191469, 0.43196063, 0.43200657, 0.43205249, 0.43209837,
0.43214418, 0.43218993, 0.43223559, 0.43228117, 0.43232665,
0.43237204, 0.43241734, 0.43246254, 0.43250765, 0.43255267,
0.4325976 , 0.43264244, 0.4326872 , 0.43273188, 0.43277648,
0.432821  , 0.43286545, 0.43290984, 0.43295416, 0.43299841])
```

[9]:
```
ridge_table = pd.DataFrame(data = lambdas).rename({0: 'lambdas'}, axis = 1)
ridge_table['CV errors'] = est_error
ridge_table
```

[9]:

	lambdas	CV errors
0	-5.0	0.450275
1	-4.9	0.450016
2	-4.8	0.449803
3	-4.7	0.449635
4	-4.6	0.449515
..	…	…
95	4.5	0.432821
96	4.6	0.432865
97	4.7	0.432910
98	4.8	0.432954
99	4.9	0.432998

[100 rows x 2 columns]

[10]:
```
r_argmin = min(ridge_table['CV errors'])
r_tuning = ridge_table[ridge_table['CV errors'] == r_argmin]['lambdas'].values.
 ↪item(0)
r_tuning
```

[10]: 0.5999999999999801

[11]:
```
ridge_model = Ridge(fit_intercept = True, alpha = r_tuning)
ridge_model = ridge_model.fit(X_train, Y_train)
ridge_coef = ridge_model.coef_
ridge_int = ridge_model.intercept_
print("Minimum CV error is: " + str(r_argmin) + ".\nRidge coefficients are: " +␣
 ↪str(ridge_coef) + ".\nRidge Intercept: " + str(ridge_int))
```

```
Minimum CV error is: 0.431237684958319.
Ridge coefficients are: [ 0.01004131 -1.06350615 -0.11943824  0.00530362
-1.41666293  0.00476129
```

```
 -0.00322347 -0.0281848  -0.38769767  0.80370966  0.29122728].
Ridge Intercept: 4.074959516432782
```

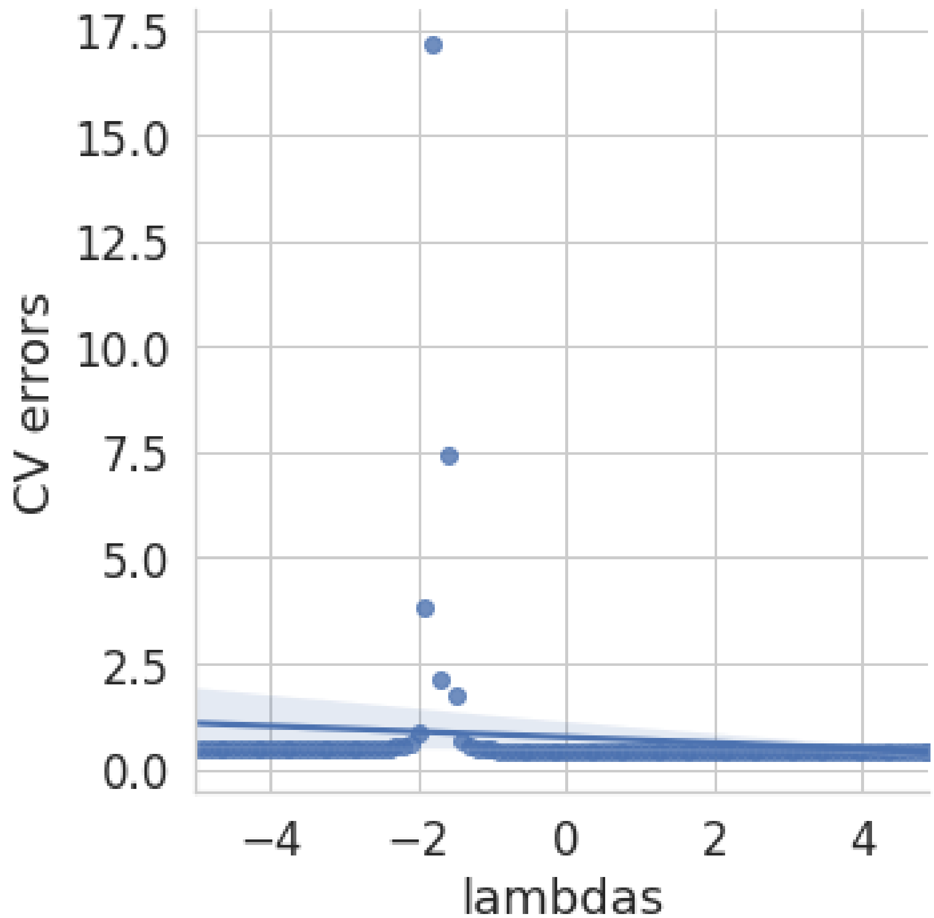
[12]:
```
ridge_coef_table = pd.DataFrame(data = col.columns).rename({0: "features"},␣
 ↪axis = 1)
ridge_coef_table['coefficients'] = ridge_coef
ridge_coef_table
```

[12]:

	features	coefficients
0	fixed acidity	0.010041
1	volatile acidity	-1.063506
2	citric acid	-0.119438
3	residual sugar	0.005304
4	chlorides	-1.416663
5	free sulfur dioxide	0.004761
6	total sulfur dioxide	-0.003223
7	density	-0.028185
8	pH	-0.387698
9	sulphates	0.803710
10	alcohol	0.291227

[13]:
```
sns.lmplot(x = "lambdas", y = "CV errors", data = ridge_table)
```

[13]: <seaborn.axisgrid.FacetGrid at 0x7ff7f9b7ac40>



7 k=10 kfold cross validation

[14]:
```
# Cross Validation
# lamdas = 0 is just OLS. Use range to figure out what happens to abs value
lambdas1 = np.arange(-10, 10, 0.1)
est_error1 = []
for i in lambdas1:
    # ridge_model replace lm
    ridge_model = Ridge(alpha = i, fit_intercept = True)
    # cv = 5 default
    scores1 = np.mean(cross_val_score(ridge_model, X_train, Y_train, scoring = 'neg_mean_squared_error'))
    est_error1.append(scores1)
est_error1 = np.abs(est_error1)

ridge_table1 = pd.DataFrame(data = lambdas1).rename({0: 'lambdas'}, axis = 1)
ridge_table1['CV errors'] = est_error1
```

```
r_argmin1 =  min(ridge_table1['CV errors'])
r_tuning1 = ridge_table1[ridge_table1['CV errors'] == r_argmin1]['lambdas'].
 ↪values.item(0)

ridge_model1 = Ridge(fit_intercept = True, alpha = r_tuning1)
ridge_model1 = ridge_model1.fit(X_train, Y_train)
ridge_coef1 = ridge_model1.coef_
ridge_int1 = ridge_model1.intercept_
print("Minimum CV error is: " + str(r_argmin1) + "\nRidge coefficients are: " +␣
 ↪str(ridge_coef1) + ".\nRidge Intercept: " + str(ridge_int1) + str(r_tuning1))
```

```
Minimum CV error is: 0.431237684958319
Ridge coefficients are: [ 0.01004131 -1.06350615 -0.11943824  0.00530362
-1.41666293  0.00476129
 -0.00322347 -0.0281848  -0.38769767  0.80370966  0.29122728].
Ridge Intercept: 4.0749595164327860.5999999999999623
```

[15]:
```
ridge_table1
```

[15]:

	lambdas	CV errors
0	-10.0	0.778091
1	-9.9	0.738100
2	-9.8	0.704365
3	-9.7	0.675662
4	-9.6	0.651052
..	…	…
195	9.5	0.434998
196	9.6	0.435041
197	9.7	0.435084
198	9.8	0.435127
199	9.9	0.435170

[200 rows x 2 columns]

[16]:
```
ridge_coef_table1 = pd.DataFrame(data = col.columns).rename({0: "features"},␣
 ↪axis = 1)
ridge_coef_table1['coefficients'] = ridge_coef1
ridge_coef_table1
```

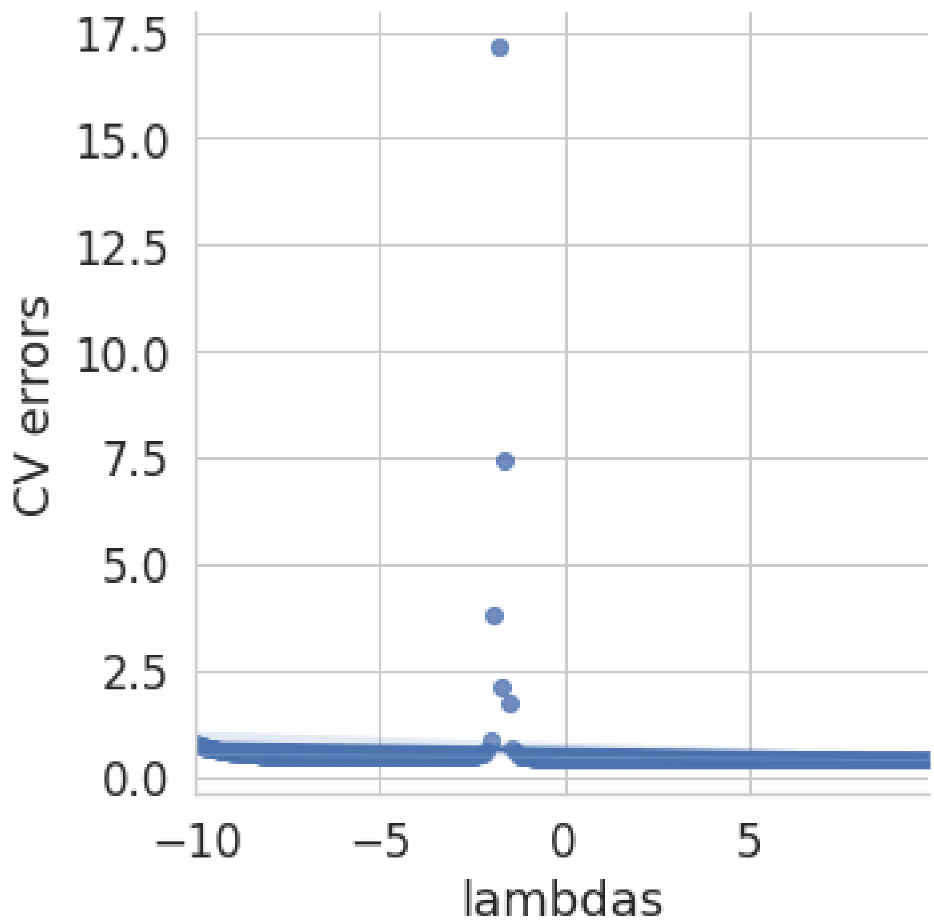
[16]:

	features	coefficients
0	fixed acidity	0.010041
1	volatile acidity	-1.063506
2	citric acid	-0.119438
3	residual sugar	0.005304
4	chlorides	-1.416663
5	free sulfur dioxide	0.004761

```
6   total sulfur dioxide     -0.003223
7                density     -0.028185
8                     pH     -0.387698
9              sulphates      0.803710
10               alcohol      0.291227
```

[17]:
```
sns.lmplot(x = "lambdas", y = "CV errors", data = ridge_table1)
```

[17]: <seaborn.axisgrid.FacetGrid at 0x7ff7f581fd00>



8 Lasso Regression

Definition: http://courses.ieor.berkeley.edu/ieor165/lecture_notes/ieor165_lec8.pdf

Pred Lasso Regression model: https://machinelearningmastery.com/lasso-regression-with-python/

[18]:
```python
# Cross Validation
# lamdas = 0 is just OLS. Use range to figure out what happens to abs value
# default alpha is 1? Ask about lambda
lambdas = np.arange(0.001, 5, 0.001)
est_error = []
for i in lambdas:
    # ridge_model replace lm
    lasso_model = Lasso(alpha = i, fit_intercept = True)
    # cv = 5 default
    scores = np.mean(cross_val_score(lasso_model, X_train, Y_train, scoring =␣
 ↪'neg_mean_squared_error'))
    est_error.append(scores)
est_error = np.abs(est_error)
est_error
```

[18]:
```
array([0.43155873, 0.4335857 , 0.43580411, …, 0.64940826, 0.64941012,
       0.64941198])
```

[19]:
```python
lasso_table = pd.DataFrame(data = lambdas).rename({0: 'lambdas'}, axis = 1)
lasso_table['CV errors'] = est_error
lasso_table
```

[19]:

	lambdas	CV errors
0	0.001	0.431559
1	0.002	0.433586
2	0.003	0.435804
3	0.004	0.435836
4	0.005	0.435911
…	…	…
4994	4.995	0.649405
4995	4.996	0.649406
4996	4.997	0.649408
4997	4.998	0.649410
4998	4.999	0.649412

[4999 rows x 2 columns]

[20]:
```python
l_argmin = min(lasso_table['CV errors'])
l_tuning = lasso_table[lasso_table['CV errors'] == l_argmin]['lambdas'].values.
 ↪item(0)
l_tuning
```

[20]: 0.001

[21]:
```python
lasso_model = Lasso(fit_intercept = True, alpha = l_tuning)
lasso_model = lasso_model.fit(X_train, Y_train)
lasso_coef = lasso_model.coef_
```

```
lasso_int = lasso_model.intercept_
print("Lasso coefficients are: " + str(lasso_coef) + " and the Lasso Intercept:␣
↪" + str(lasso_int))
```

```
Lasso coefficients are: [ 0.01048974 -1.03266187 -0.0367246   0.00339385
-1.16859943  0.00489661
 -0.00324263 -0.         -0.29360346  0.7507817   0.29037469] and the Lasso
Intercept: 3.7182409302962087
```

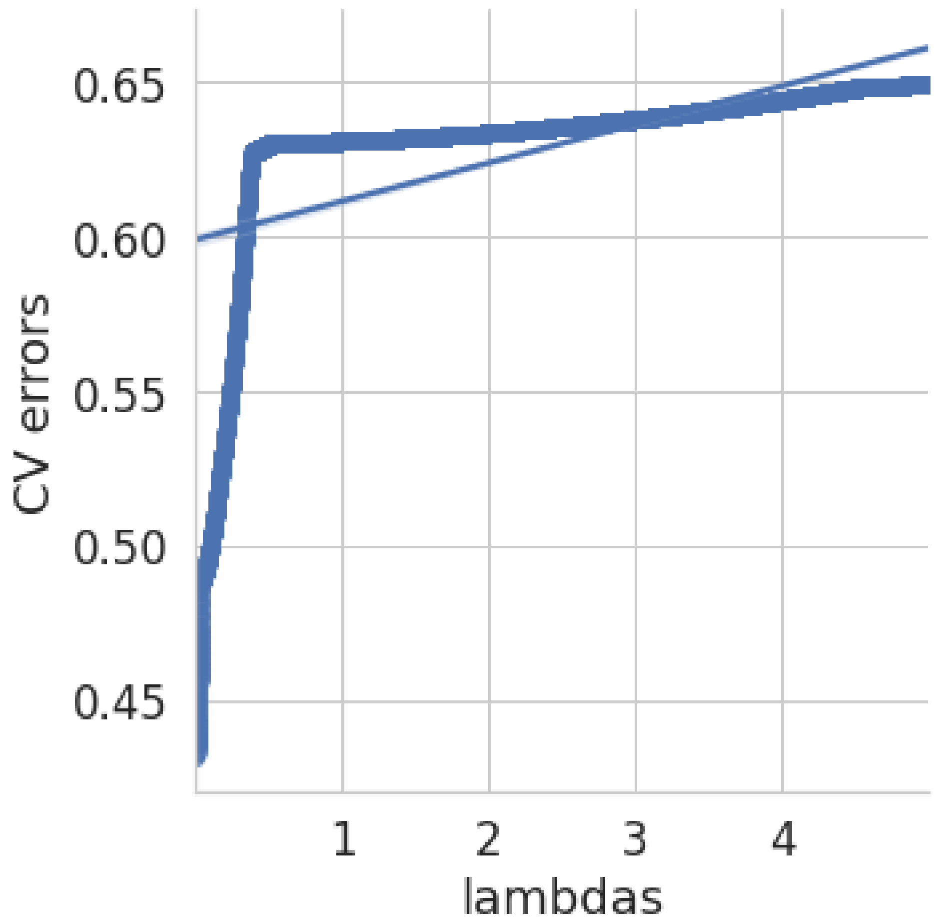
[22]:
```
lasso_coef_table = pd.DataFrame(data = col.columns).rename({0: "features"},␣
↪axis = 1)
lasso_coef_table['coefficients'] = lasso_coef
lasso_coef_table
```

[22]:

	features	coefficients
0	fixed acidity	0.010490
1	volatile acidity	-1.032662
2	citric acid	-0.036725
3	residual sugar	0.003394
4	chlorides	-1.168599
5	free sulfur dioxide	0.004897
6	total sulfur dioxide	-0.003243
7	density	-0.000000
8	pH	-0.293603
9	sulphates	0.750782
10	alcohol	0.290375

[23]:
```
sns.lmplot(x = "lambdas", y = "CV errors", data = lasso_table)
```

[23]: <seaborn.axisgrid.FacetGrid at 0x7ff7f5894a00>


0.65
0.60
0.55
0.50
0.45
CV errors
1
2
3
4
lambdas