sheet4

November 9, 2020

[5]:
```
import numpy as np
import pandas as pd
import statsmodels.api as sm
import matplotlib.pyplot as plt
import sklearn as sk
from sklearn.model_selection import KFold
from sklearn.linear_model import Ridge, Lasso
```

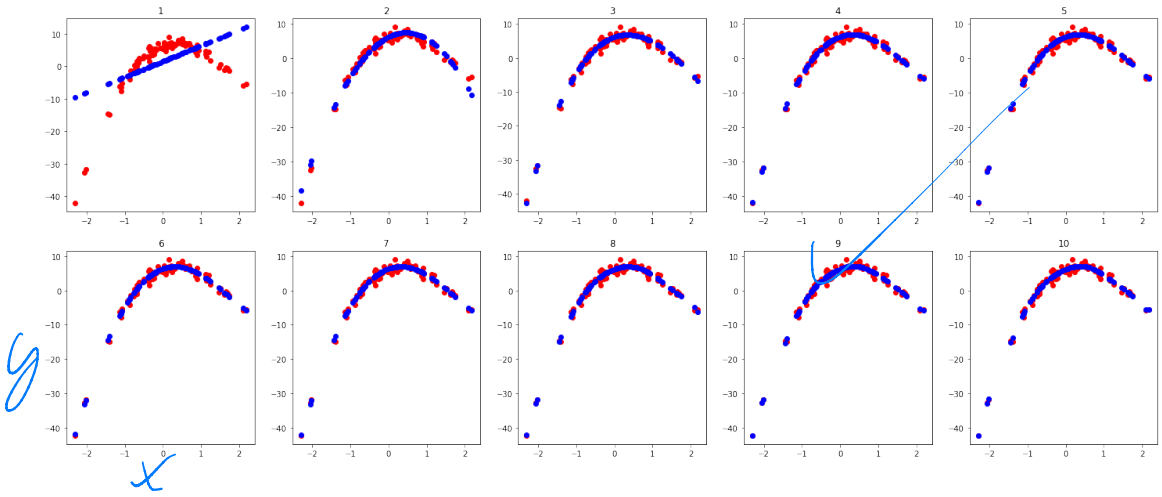
[7]:
```
data = pd.read_csv('Data_Q2',index_col=0)
data
```

[7]:
```
           x         y
0   1.624345 -1.209235
1  -0.611756  2.394638
2  -0.528172  2.528607
3  -1.072969 -5.347027
4   0.865408  4.262015
..       ...       ...
95  0.077340  4.646311
96 -0.343854  4.493099
97  0.043597  6.583315
98 -0.620001  1.881545
99  0.698032  7.117147

[100 rows x 2 columns]
```

[9]:
```
x = data['x']
y = data['y']
```

[10]:
```
bic_vec = np.zeros((10,1))
aic_vec = np.zeros((10,1))
plt.figure(figsize=(25,10))
for i in range(10):
    p = i+1
    x_vec = np.polynomial.polynomial.polyvander(x,p)
    model = sm.OLS(y,x_vec).fit()
    if i==0:
```

```
        bic_min = model.bic
    elif model.bic < bic_min:
        bic_min = model.bic
        best_bic_params = model.params
    if i==0:
        aic_min = model.aic
    elif model.aic < aic_min:
        aic_min = model.aic
        best_aic_params = model.params
    bic_vec[i]=model.bic
    aic_vec[i]=model.aic
    plt.subplot(2,5,p)
    y_predict = model.fittedvalues
    plt.scatter(x,y,c='r')
    plt.scatter(x,y_predict,c='b')
    plt.title(str(p))
```



[11]:
```
bic_aic_df = pd.DataFrame(np.concatenate((bic_vec,aic_vec),axis =1 ),columns =␣
↪['bic value','aic value'], index = np.linspace(1,10,10))
```

[12]:
```
bic_aic_df
```

[12]:

	bic value	aic value
1.0	677.598182	672.387842
2.0	357.330237	349.514726
3.0	285.734557	275.313877
4.0	284.276388	271.250537
5.0	288.881554	273.250533
6.0	293.410946	275.174754
7.0	298.012627	277.171265

```
8.0   300.985570  277.539038
9.0   303.509673  277.457971
10.0  307.654182  278.997310
```

[13]:
```
best_bic_params
```

[13]:
```
const    6.313982
x1       3.912735
x2      -6.544477
x3       0.699182
x4       0.139361
dtype: float64
```

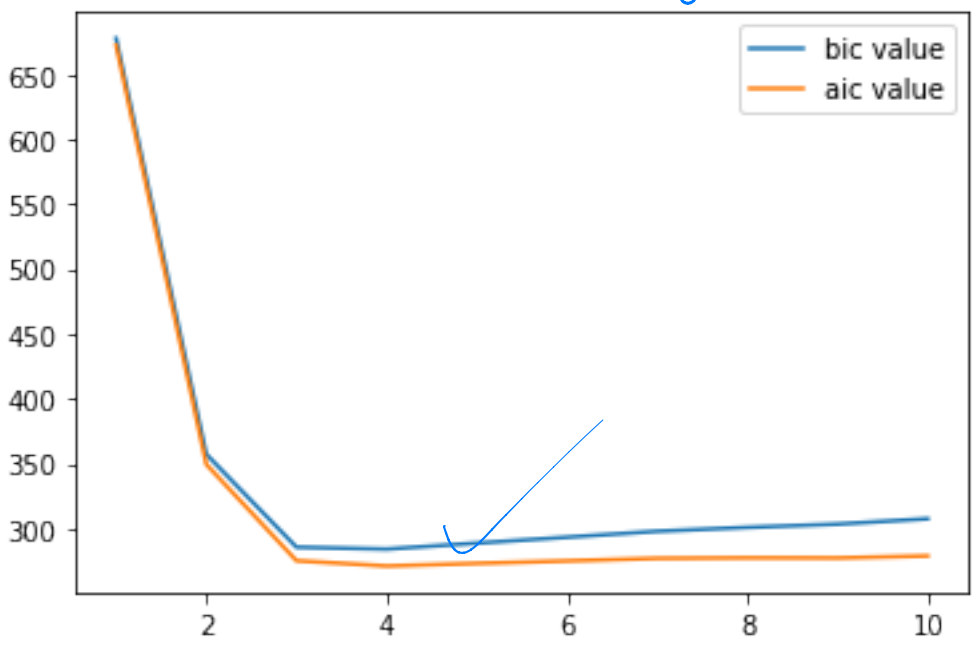
[14]:
```
best_aic_params
```

[14]:
```
const    6.313982
x1       3.912735
x2      -6.544477
x3       0.699182
x4       0.139361
dtype: float64
```

[15]:
```
plt.figure()
bic_aic_df.plot()
plt.show()
```

Which one is best? Write a short answer.

```
<Figure size 432x288 with 0 Axes>
```



0.1 part(e)

[16]:
```
def ridge_lasso_reg2 (lambda_vec,option = 'ridge',params = False):

    kf = KFold(n_splits = 10)
    err = np.zeros(len(lambda_vec))
    X = np.polynomial.polynomial.polyvander(x,10)
    y = data['y']
    coefficients = np.zeros((len(lambda_vec),11))
#     err1_vec = np.zeros(10)
    err2_vec = np.zeros(10)

    #carry out a binary search to find the optimal lambda
    for i in range(len(lambda_vec)):
        fold = 0
        if option == 'ridge':
            reg = Ridge(alpha = lambda_vec[i],fit_intercept = False)
        elif option == 'lasso':
            reg = Lasso(alpha = lambda_vec[i],fit_intercept = False,tol=0.
 ↪01,max_iter = 10000)
        for train_index, test_index in kf.split(X):
            X_train, X_test = X[train_index],X[test_index]
            y_train, y_test = y[train_index],y[test_index]

#             if i==0:
#                 ridreg1 = f(alpha = lambda_vec[0],fit_intercept = False,␣
 ↪normalize=True)
#                 model1 = ridreg1.fit(X_train,y_train)
#                 y_pred_1 = model1.predict(X_test)
#                 err1_vec[fold] = np.mean((y_pred_1-y_test)**2)

          #  reg = f
            model = reg.fit(X_train,y_train)
            y_pred_2 = model.predict(X_test)
            err2_vec[fold] = np.mean((y_pred_2-y_test)**2)
            fold += 1

#         if i==0:
#             err1 = np.mean(err1_vec)
#             err.append(err1)
 #       print(err2_vec)
        err2 = np.mean(err2_vec)
        err[i]=err2
        coefficients[i,:] = model.coef_
```

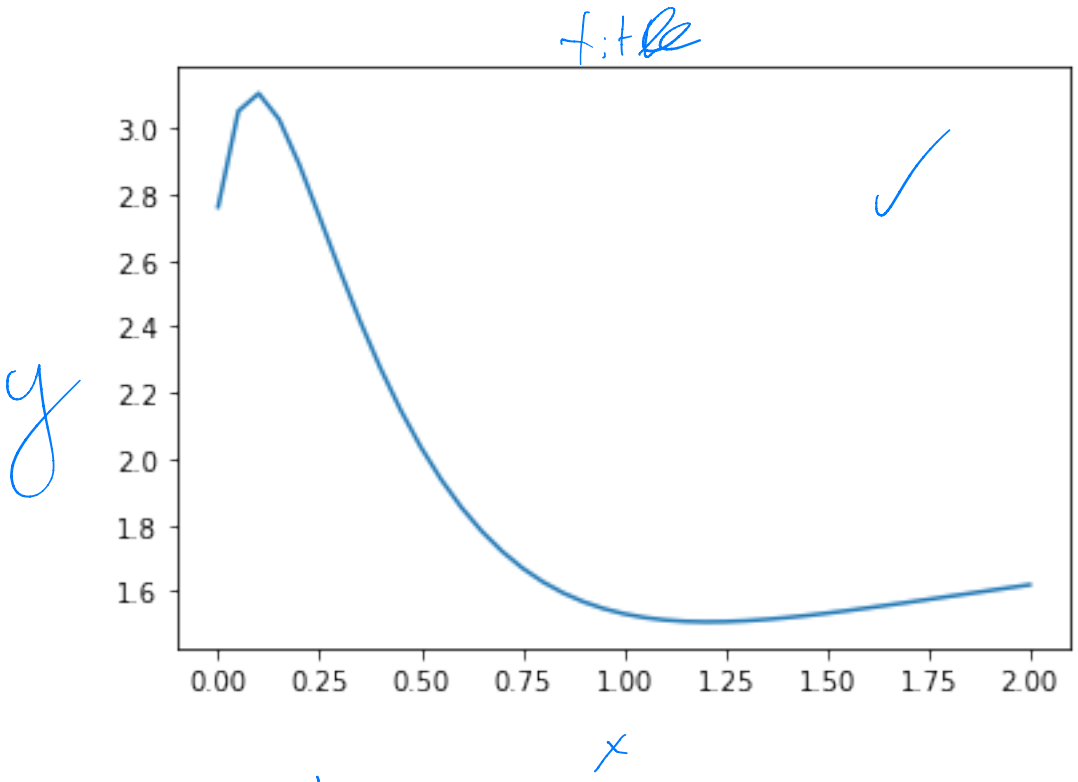
```
#         if i!=(n-1):
#             lambda_storage_vec.append(np.mean(lambda_vec))

#         if err1 > err2:
#             lambda_vec = [lambda_vec[1],np.mean(lambda_vec)]
#             err1 = err2
#         else:
#             lambda_vec = [lambda_vec[0],np.mean(lambda_vec)]

    if params:
        return err, coefficients
    else:
        return err
```

[17]:
```
l1 = np.linspace(0,2,41)
ridge_err_2 = ridge_lasso_reg2(l1)
ridge_err_2
```

[17]:
```
array([2.76115474, 3.05096039, 3.1041114 , 3.02655863, 2.88962884,
       2.73078843, 2.56947497, 2.41565579, 2.2741894 , 2.14711332,
       2.03489071, 1.93711259, 1.85290372, 1.78116113, 1.72069534,
       1.67031355, 1.6288677 , 1.59528087, 1.56856026, 1.54780208,
       1.53219122, 1.5209981 , 1.51357368, 1.50934374, 1.50780274,
       1.50850767, 1.51107219, 1.51516098, 1.52048452, 1.5267943 ,
       1.53387835, 1.54155725, 1.5496805 , 1.55812321, 1.56678314,
       1.57557807, 1.58444339, 1.59332995, 1.60220221, 1.61103646,
       1.61981935])
```

[18]:
```
plt.plot(l1,ridge_err_2)
```

[18]:
```
[<matplotlib.lines.Line2D at 0x2673baad848>]
```



we pick lambda = 1.2

[22]:
```
l2 = np.linspace(0,0.2,21) #no matter how I change this interval, it always␣
 ↪suggests that lambda = 0 gives the samllest value, I have also altered the tol␣
 ↪and max_iter a lot to make it converge, but no effect
lasso_err_2,lasso_params_2 = ridge_lasso_reg2(l2,option='lasso',params=True)
lasso_err_2
```

```
c:\users\xuais\anaconda3\envs\snakes\lib\site-packages\ipykernel_launcher.py:29:
UserWarning: With alpha=0, this algorithm does not converge well. You are
advised to use the LinearRegression estimator
c:\users\xuais\anaconda3\envs\snakes\lib\site-
packages\sklearn\linear_model\_coordinate_descent.py:531: UserWarning:
Coordinate descent with no regularization may lead to unexpected results and is
discouraged.
  positive)
c:\users\xuais\anaconda3\envs\snakes\lib\site-packages\ipykernel_launcher.py:29:
UserWarning: With alpha=0, this algorithm does not converge well. You are
advised to use the LinearRegression estimator
c:\users\xuais\anaconda3\envs\snakes\lib\site-
packages\sklearn\linear_model\_coordinate_descent.py:531: UserWarning:
Coordinate descent with no regularization may lead to unexpected results and is
discouraged.
  positive)
c:\users\xuais\anaconda3\envs\snakes\lib\site-packages\ipykernel_launcher.py:29:
UserWarning: With alpha=0, this algorithm does not converge well. You are
```

```
advised to use the LinearRegression estimator
c:\users\xuais\anaconda3\envs\snakes\lib\site-
packages\sklearn\linear_model\_coordinate_descent.py:531: UserWarning:
Coordinate descent with no regularization may lead to unexpected results and is
discouraged.
  positive)
c:\users\xuais\anaconda3\envs\snakes\lib\site-packages\ipykernel_launcher.py:29:
UserWarning: With alpha=0, this algorithm does not converge well. You are
advised to use the LinearRegression estimator
c:\users\xuais\anaconda3\envs\snakes\lib\site-
packages\sklearn\linear_model\_coordinate_descent.py:531: UserWarning:
Coordinate descent with no regularization may lead to unexpected results and is
discouraged.
  positive)
c:\users\xuais\anaconda3\envs\snakes\lib\site-packages\ipykernel_launcher.py:29:
UserWarning: With alpha=0, this algorithm does not converge well. You are
advised to use the LinearRegression estimator
c:\users\xuais\anaconda3\envs\snakes\lib\site-
packages\sklearn\linear_model\_coordinate_descent.py:531: UserWarning:
Coordinate descent with no regularization may lead to unexpected results and is
discouraged.
  positive)
c:\users\xuais\anaconda3\envs\snakes\lib\site-packages\ipykernel_launcher.py:29:
UserWarning: With alpha=0, this algorithm does not converge well. You are
advised to use the LinearRegression estimator
c:\users\xuais\anaconda3\envs\snakes\lib\site-
packages\sklearn\linear_model\_coordinate_descent.py:531: UserWarning:
Coordinate descent with no regularization may lead to unexpected results and is
discouraged.
  positive)
c:\users\xuais\anaconda3\envs\snakes\lib\site-packages\ipykernel_launcher.py:29:
UserWarning: With alpha=0, this algorithm does not converge well. You are
advised to use the LinearRegression estimator
c:\users\xuais\anaconda3\envs\snakes\lib\site-
packages\sklearn\linear_model\_coordinate_descent.py:531: UserWarning:
Coordinate descent with no regularization may lead to unexpected results and is
discouraged.
  positive)
c:\users\xuais\anaconda3\envs\snakes\lib\site-packages\ipykernel_launcher.py:29:
UserWarning: With alpha=0, this algorithm does not converge well. You are
advised to use the LinearRegression estimator
c:\users\xuais\anaconda3\envs\snakes\lib\site-
packages\sklearn\linear_model\_coordinate_descent.py:531: UserWarning:
Coordinate descent with no regularization may lead to unexpected results and is
discouraged.
  positive)
c:\users\xuais\anaconda3\envs\snakes\lib\site-packages\ipykernel_launcher.py:29:
UserWarning: With alpha=0, this algorithm does not converge well. You are
```

```
advised to use the LinearRegression estimator
c:\users\xuais\anaconda3\envs\snakes\lib\site-
packages\sklearn\linear_model\_coordinate_descent.py:531: UserWarning:
Coordinate descent with no regularization may lead to unexpected results and is
discouraged.
  positive)
c:\users\xuais\anaconda3\envs\snakes\lib\site-packages\ipykernel_launcher.py:29:
UserWarning: With alpha=0, this algorithm does not converge well. You are
advised to use the LinearRegression estimator
c:\users\xuais\anaconda3\envs\snakes\lib\site-
packages\sklearn\linear_model\_coordinate_descent.py:531: UserWarning:
Coordinate descent with no regularization may lead to unexpected results and is
discouraged.
  positive)
```

[22]:
```
array([ 0.93712915,  0.98974645,  1.87466238,  3.79763414,  4.31165054,
        4.72633177,  5.15139012,  5.59376147,  6.05532773,  6.48937656,
        6.99306678,  7.53412523,  8.14033929,  8.78599176,  9.47866918,
       10.23401447, 11.08399508, 12.06880469, 13.26868876, 14.27880193,
       15.19249394])
```

[23]:
```
lasso_params_2
```

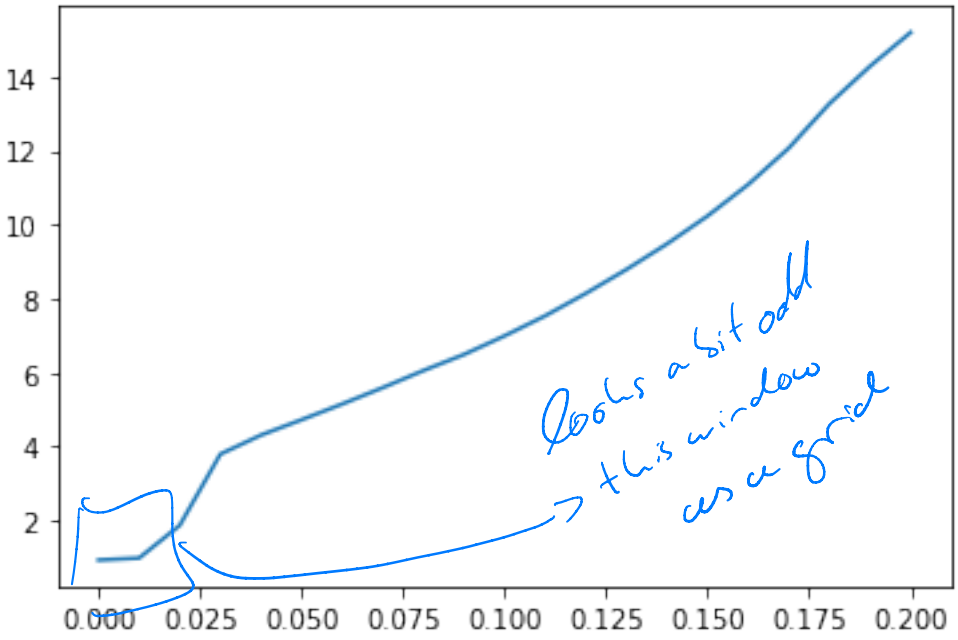
[23]:
```
array([[ 6.02578958e+00,  3.99587237e+00, -5.94340822e+00,
         5.81034046e-01, -5.75622912e-02,  1.16659762e-02,
         8.37006692e-03,  1.30737347e-03,  1.22853896e-03,
         1.93059879e-04,  1.57053426e-04],
       [ 5.99479394e+00,  3.96747478e+00, -5.89318952e+00,
         5.85838642e-01, -6.68623683e-02,  1.21527270e-02,
         7.96396237e-03,  1.25206237e-03,  1.28194942e-03,
         1.90276926e-04,  1.64508548e-04],
       [ 5.96379830e+00,  3.93907719e+00, -5.84297081e+00,
         5.90643237e-01, -7.61624455e-02,  1.26394778e-02,
         7.55785782e-03,  1.19675127e-03,  1.33535988e-03,
         1.87493973e-04,  1.71963670e-04],
       [ 5.93280266e+00,  3.91067959e+00, -5.79275211e+00,
         5.95447833e-01, -8.54625226e-02,  1.31262287e-02,
         7.15175327e-03,  1.14144016e-03,  1.38877034e-03,
         1.84711020e-04,  1.79418792e-04],
       [ 5.92556842e+00,  3.88727699e+00, -5.77964731e+00,
         6.01223608e-01, -9.15185384e-02,  1.36135924e-02,
         7.13649811e-03,  1.05178623e-03,  1.50013443e-03,
         1.82532937e-04,  1.90009738e-04],
       [ 5.92662026e+00,  3.86444359e+00, -5.77747272e+00,
         6.08293061e-01, -9.77621508e-02,  1.42279069e-02,
         7.22112087e-03,  9.01969604e-04,  1.66985026e-03,
         1.80100490e-04,  2.03059105e-04],
```

```
[ 5.91304094e+00,  3.83739111e+00, -5.74952244e+00,
  6.15476815e-01, -1.07835136e-01,  1.49997711e-02,
  7.01738814e-03,  7.26569573e-04,  1.85257647e-03,
  1.78037526e-04,  2.16829102e-04],
[ 5.89666494e+00,  3.80670088e+00, -5.71099809e+00,
  6.24385351e-01, -1.22219866e-01,  1.62409373e-02,
  6.63927566e-03,  4.16986955e-04,  2.14914702e-03,
  1.77877331e-04,  2.33656076e-04],
[ 5.86413674e+00,  3.77334247e+00, -5.64173574e+00,
  6.31817684e-01, -1.42100295e-01,  1.77821282e-02,
  5.89583157e-03,  8.62691216e-05,  2.49249667e-03,
  1.83730066e-04,  2.54079588e-04],
[ 5.82514889e+00,  3.74243652e+00, -5.56530793e+00,
  6.36581416e-01, -1.60890757e-01,  1.88505224e-02,
  5.14841808e-03, -0.00000000e+00,  2.74029142e-03,
  1.75061355e-04,  2.74360341e-04],
[ 5.78321450e+00,  3.70942131e+00, -5.47616045e+00,
  6.40836869e-01, -1.85158091e-01,  2.04962738e-02,
  4.16591309e-03, -0.00000000e+00,  3.16074087e-03,
  1.34985513e-04,  2.94303986e-04],
[ 5.73836944e+00,  3.67811602e+00, -5.37873913e+00,
  6.43488606e-01, -2.12019763e-01,  2.23084953e-02,
  3.10056097e-03, -0.00000000e+00,  3.62265310e-03,
  1.00242917e-04,  3.18281037e-04],
[ 5.69137158e+00,  3.64815906e+00, -5.27465113e+00,
  6.44739379e-01, -2.41294458e-01,  2.42792279e-02,
  2.06849324e-03, -0.00000000e+00,  4.10684493e-03,
  7.00922439e-05,  3.46390254e-04],
[ 5.63594809e+00,  3.62071922e+00, -5.14046681e+00,
  6.41395015e-01, -2.81801346e-01,  2.74334729e-02,
  4.44365173e-04, -0.00000000e+00,  4.91063477e-03,
  3.37947113e-05,  3.81110817e-04],
[ 5.57602496e+00,  3.59700220e+00, -4.99397947e+00,
  6.34023899e-01, -3.27738448e-01,  3.11883637e-02,
  0.00000000e+00,  0.00000000e+00,  5.57984524e-03,
  1.51687511e-08,  4.20736913e-04],
[ 5.53783225e+00,  3.56632811e+00, -4.92352277e+00,
  6.39682194e-01, -3.44206961e-01,  3.14072096e-02,
  0.00000000e+00,  0.00000000e+00,  5.57591155e-03,
 -0.00000000e+00,  4.51873478e-04],
[ 5.49371500e+00,  3.53801132e+00, -4.83083865e+00,
  6.40591591e-01, -3.69883308e-01,  3.26195187e-02,
  0.00000000e+00,  0.00000000e+00,  5.75538811e-03,
 -0.00000000e+00,  4.88360935e-04],
[ 5.44132390e+00,  3.51535417e+00, -4.70732108e+00,
  6.35851281e-01, -4.08111375e-01,  3.49352588e-02,
  0.00000000e+00,  0.00000000e+00,  6.16287903e-03,
```

```
 -0.00000000e+00,  5.35507539e-04],
[ 5.38769857e+00,  3.49288305e+00, -4.57955445e+00,
  6.30950301e-01, -4.48007774e-01,  3.72941334e-02,
  0.00000000e+00,  0.00000000e+00,  6.59457336e-03,
 -0.00000000e+00,  5.84996692e-04],
[ 5.33300588e+00,  3.47050018e+00, -4.44795755e+00,
  6.25818321e-01, -4.89440203e-01,  3.97175644e-02,
  0.00000000e+00,  0.00000000e+00,  7.05008433e-03,
 -0.00000000e+00,  6.36496175e-04],
[ 5.27738887e+00,  3.44838614e+00, -4.31298693e+00,
  6.20278047e-01, -5.32237320e-01,  4.22339945e-02,
  0.00000000e+00,  0.00000000e+00,  7.52847730e-03,
 -0.00000000e+00,  6.89506876e-04]])
```

[24]:
```
plt.plot(l2,lasso_err_2)
```

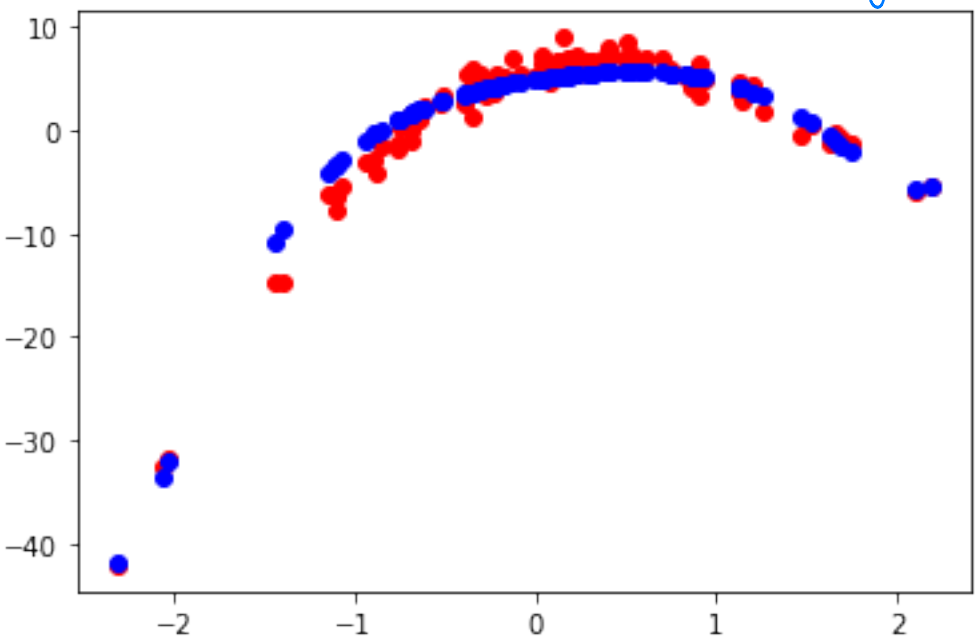
[24]:
```
[<matplotlib.lines.Line2D at 0x2673ba8a508>]
```



[28]:
```
X = np.polynomial.polynomial.polyvander(x,10)
```

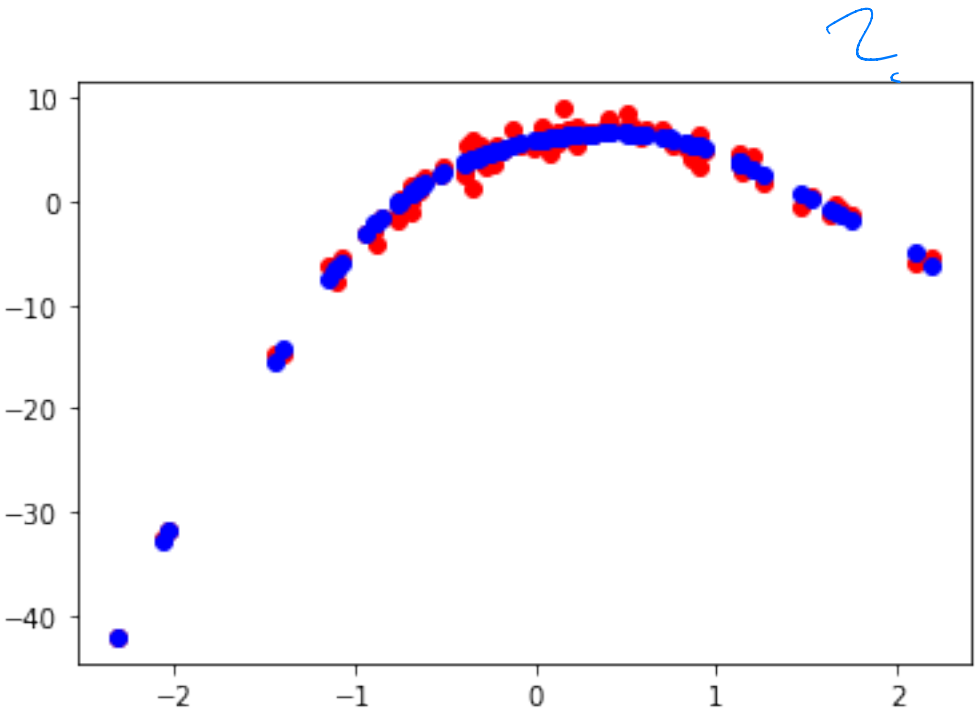
[31]:
```
reg = Lasso(alpha =0.5,tol=0.01)
model_lasso = reg.fit(X,y)
y_pred_lasso = model_lasso.predict(X)
plt.scatter(x,y,c='r')
plt.scatter(x,y_pred_lasso,c='b')
```

[31]: <matplotlib.collections.PathCollection at 0x2673b6c2f88>



[32]:
```
X = np.polynomial.polynomial.polyvander(x,10)
reg = Ridge(alpha =1.20)
model_ridge = reg.fit(X,y)
y_pred_ridge = model_ridge.predict(X)
plt.scatter(x,y,c='r')
plt.scatter(x,y_pred_ridge,c='b')
```

[32]: <matplotlib.collections.PathCollection at 0x2673b5d2d08>


10
0
−10
−20
−30
−40
−2
−1
0
1
2

[ ]: