

```
In [62]: 1 import numpy as np
          2 import matplotlib
          3 import matplotlib.pyplot as plt
          4 import numpy.polynomial.polynomial as poly
          5 %matplotlib inline
```

```
In [63]: 1 import scipy.io
          2 mat_dict = scipy.io.loadmat('StevensonV2.mat')
          3 mat_dict.keys()
```

```
Out[63]: dict_keys(['__header__', '__version__', '__globals__', 'Publication',
                    'timeBase', 'spikes', 'time', 'handVel', 'handPos', 'target', 'startBins', 'targets', 'startBinned'])
```

```
In [64]: 1 X0 = mat_dict['spikes'].T
          2 y0 = mat_dict['handVel'][0,:]
```

```
In [65]: 1 nt = X0.shape[0]
          2 nneuron = X0.shape[1]
          3 print("nt = {0:d}, nneuron = {1:d}".format(nt, nneuron))
```

```
nt = 15536, nneuron = 196
```

```
In [66]: 1 time = mat_dict['time'][0, :]
          2 tsamp = time[1] - time[0]
          3 tttotal = time[nt-1] - time[0]
          4 print("tsamp = {0:f}, tttotal = {1:f}".format(tsamp, tttotal))
          5 print(time)
```

```
tsamp = 0.050000, tttotal = 776.750000
```

```
[ 12.591  12.641  12.691 ..., 789.241 789.291 789.341]
```

```
In [67]: 1 dat = range(nt)
          2 tr_dat = dat[0: nt: 2]
          3 ts_dat = dat[1: nt: 2]
          4 Xtr = X0[tr_dat]
          5 ytr = y0[tr_dat]
          6 Xts = X0[ts_dat]
          7 yts = y0[ts_dat]
```

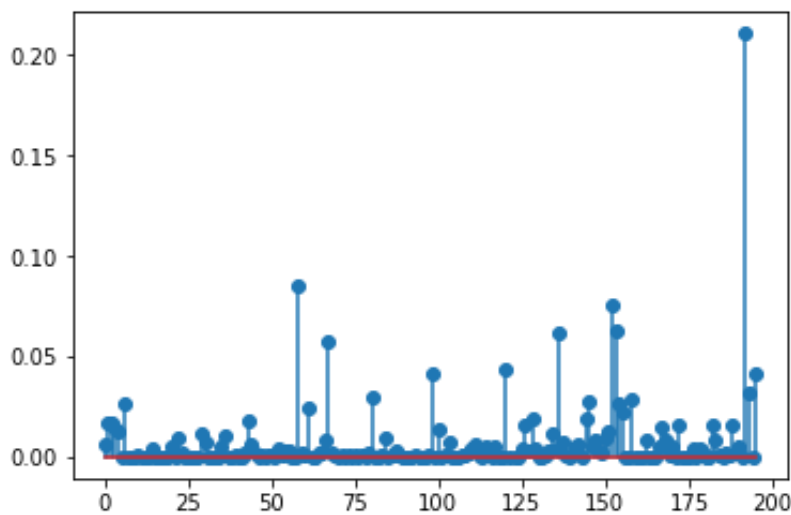
```
In [68]: 1 from sklearn import linear_model
          2 regr = linear_model.LinearRegression()
          3 regr.fit(Xtr, ytr)
          4 y_ts_pred = regr.predict(Xts)
          5 y_tr_pred = regr.predict(Xtr)
          6 RSS_ts = np.mean((y_ts_pred-yts)**2)/(np.std(yts)**2)
          7 print("normalized RSS on the test data is = {0:f}".format(RSS_ts))
```

```
normalized RSS on the test data is = 1323769529480601665536.000000
```

```
In [69]: 1 ym = np.mean(y0)
2          syy = np.mean((y0-ym)**2)
3          Req = np.zeros(nneuron)
4          for k in range(nneuron):
5              Xm = np.mean(X0[:, k])
6              sxy = np.mean((X0[:, k]-Xm)*(y0-ym))
7              sxx = np.mean((X0[:, k]-Xm)**2)
8              Req[k] = (sxy)**2/sxx/syy
9
10         plt.stem(range(nneuron), Req)
```

/Users/jennifer/anaconda/lib/python3.6/site-packages/ipykernel\_launcher.py:8: RuntimeWarning: invalid value encountered in double\_scalars

Out[69]: <Container object of 3 artists>



```
In [70]: 1 d = 100
2
3          Isel = np.argsort(-Req)[:d]
4          Sel = np.argsort(-Req)
5          print("The neurons with the ten highest R^2 values = {0}".format(Isel[0:10]))
```

The neurons with the ten highest R^2 values = [192 58 152 153 136 67 120 195 98 193]

```
In [71]: 1 Xtr2 = Xtr[:,Isel]
          2 Xts2 = Xts[:,Isel]
          3 print(Xtr2)
          4 Xtr2.shape

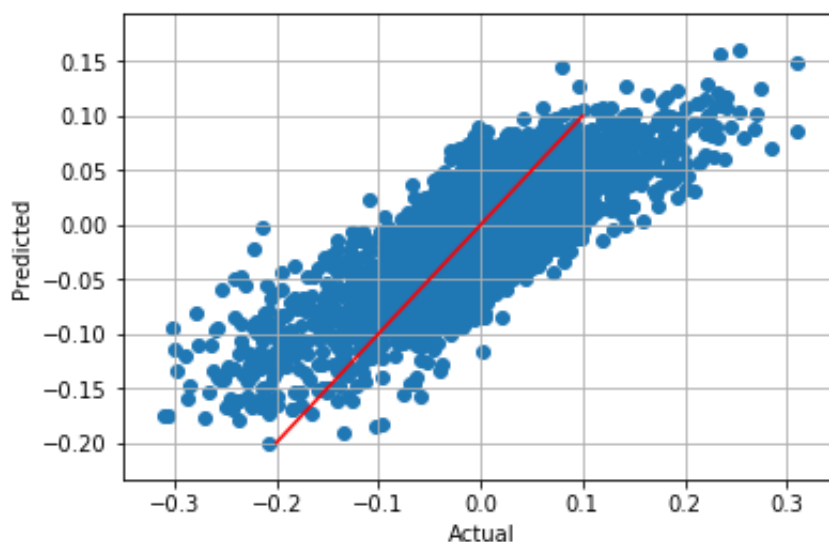
[[ 0  1  1 ...,  1  0  1]
 [ 0  2  0 ..., 10  0  0]
 [ 0  1  2 ...,  4  1  1]
 ...,
 [ 0  2  2 ...,  2  1  0]
 [ 0  1  2 ...,  3  0  0]
 [ 0  3  0 ...,  3  0  0]]
```

Out[71]: (7768, 100)

```
In [72]: 1 regr2 = linear_model.LinearRegression()
          2 regr2.fit(Xtr2, ytr)
          3 y_ts_pred2 = regr2.predict(Xts2)
          4 RSS_ts2 = np.mean((y_ts_pred2-yts)**2)
          5 RSS_ts2_nor = np.mean((y_ts_pred2-yts)**2)/(np.std(yts)**2)
          6 print("test RSS per sample is : {0:f}".format(RSS_ts2))
          7 print("normalized RSS on the test data is : {0:f}".format(RSS_ts2_nor))

test RSS per sample is : 0.001494
normalized RSS on the test data is : 0.483749
```

```
In [73]: 1 plt.scatter(yts, y_ts_pred2)
          2 plt.plot([-0.2,0.1],[-0.2,0.1], 'r')
          3 plt.xlabel('Actual')
          4 plt.ylabel('Predicted')
          5 plt.grid()
```



```
In [74]: 1 yts.shape
```

Out[74]: (7768,)

```

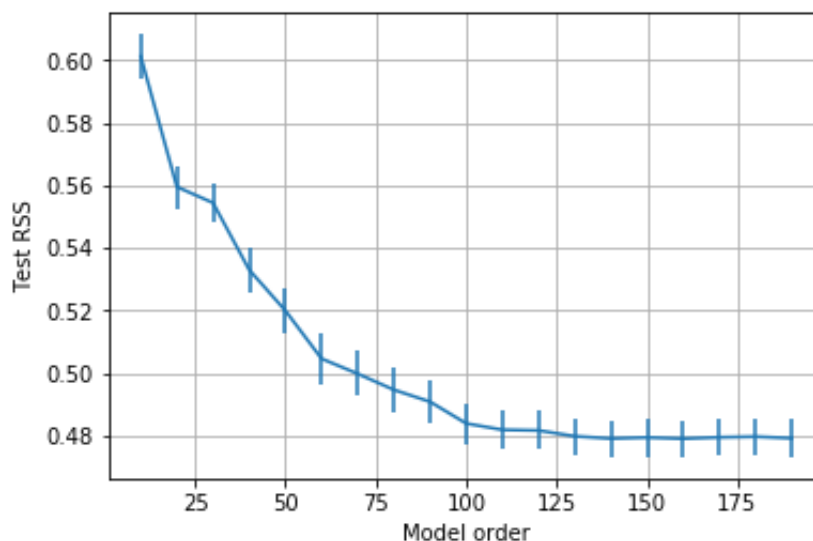
In [75]: 1 import sklearn.model_selection
2
3 nfold = 10
4 kf = sklearn.model_selection.KFold(n_splits=nfold, shuffle=True)
5
6 dtest = np.arange(10, 200, 10)
7 nd = len(dtest)
8
9 RSSts = np.zeros((nd, nfold))
10 for isplit, Ind in enumerate(kf.split(X0)):
11     Itr, Its = Ind
12     xtr_d = X0[Itr]
13     ytr_d = y0[Itr]
14     xts_d = X0[Its]
15     yts_d = y0[Its]
16
17     for it, d in enumerate(dtest):
18         Isel_tr = xtr_d[:,Sel[:d]]
19         Isel_ts = xts_d[:,Sel[:d]]
20         regr_d = linear_model.LinearRegression()
21         regr_d.fit(Isel_tr, ytr_d)
22         y_ts_hat = regr_d.predict(Isel_ts)
23         RSSts[it, isplit] = np.mean((y_ts_hat-yts_d)**2)/(np.std(y

```

```

In [76]: 1 RSS_mean = np.mean(RSSts, axis=1)
2 RSS_std = np.std(RSSts,axis=1)/np.sqrt(nfold-1)
3 plt.errorbar(dtest, RSS_mean, yerr = RSS_std, fmt='-')
4 plt.xlabel('Model order')
5 plt.ylabel('Test RSS')
6 plt.grid()

```

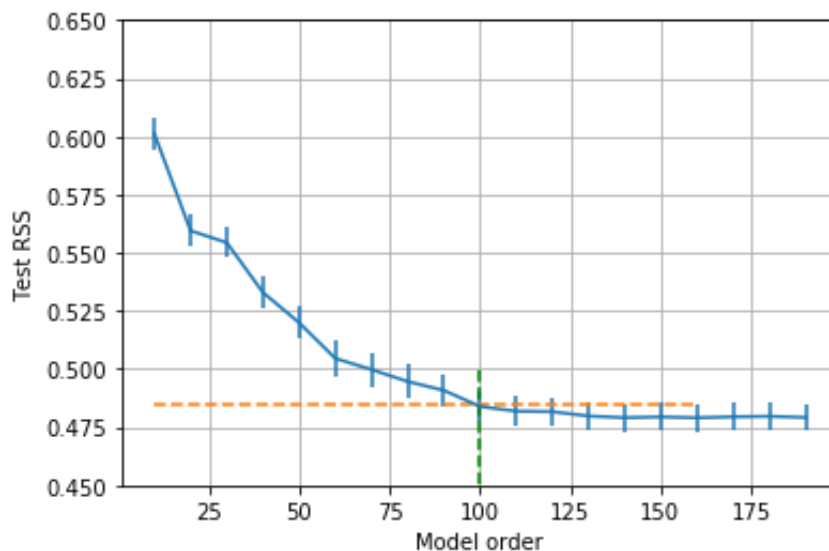


```
In [106]: 1 imin = np.argmin(RSS_mean)
2 print("Find minimize RSS, "
3       "opetimal d is: {0:d} and it's RSS_mean is {1:f}"
4       .format((imin+1)*nfold,RSS_mean[imin]))
```

Find minimize RSS, opetimal d is: 160 and it's RSS\_mean is 0.479032

```
In [107]: 1 RSS_tgt = RSS_mean[imin] + RSS_std[imin]
2
3 # Find the lowest model order below the target
4 I = np.where(RSS_mean <= RSS_tgt)[0]
5 iopt = I[0]
6 dopt = dtest[iopt]
7
8 plt.errorbar(dtest, RSS_mean, yerr=RSS_std, fmt='-')
9
10 # Plot the line at the RSS target
11 plt.plot([dtest[0],dtest[imin]], [RSS_tgt, RSS_tgt], '--')
12
13 # Plot the line at the optimal model order
14 plt.plot([dopt,dopt], [0,0.5], 'g--')
15
16 plt.ylim(0.45,0.65)
17 plt.xlabel('Model order')
18 plt.ylabel('Test RSS')
19 plt.grid()
20
21 # Print results
22 print("By using one standard deviation rule, ")
23 print("The estimated model order is %d" % dopt)
24 print("The opetimal RSS_mean is %f" % RSS_mean[iopt])
```

By using one standard deviation rule,  
The estimated model order is 100  
The opetimal RSS\_mean is 0.483881



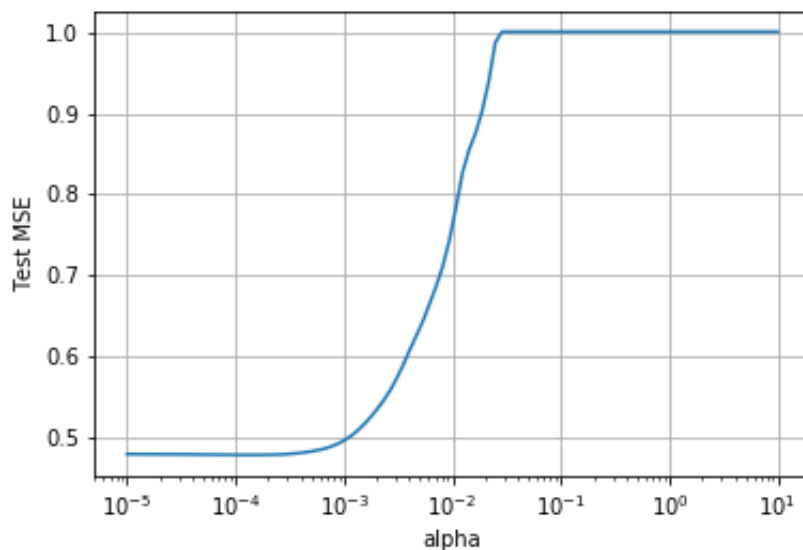
```
In [78]: 1 import sklearn.preprocessing
2 import sklearn.model_selection
3 Xs = sklearn.preprocessing.scale(X0)
```

```
/Users/jennifer/anaconda/lib/python3.6/site-packages/sklearn/utils/validation.py:429: DataConversionWarning: Data with input dtype uint8
was converted to float64 by the scale function.
warnings.warn(msg, _DataConversionWarning)
```

```
In [79]: 1 nfold = 10
2 kf = sklearn.model_selection.KFold(n_splits=nfold,shuffle=True)
3
4 model = linear_model.Lasso(warm_start=True)
5
6 nalpha = 100
7 alphas = np.logspace(-5, 1, nalpha)
8
9 mse = np.zeros((nalpha,nfold))
10 RSS_LA = mse = np.zeros((nalpha,nfold))
11 for ifold, ind in enumerate(kf.split(Xs)):
12
13
14     # Get the training data in the split
15     Itr,Its = ind
16     X_tr = Xs[Itr,:]
17     y_tr = y0[Itr]
18     X_ts = Xs[Its,:]
19     y_ts = y0[Its]
20
21     # Compute the lasso path for the split
22     for ia, a in enumerate(alphas):
23
24         # Fit the model on the training data
25         model.alpha = a
26         model.fit(X_tr,y_tr)
27
28         # Compute the prediction error on the test data
29         y_ts_pred = model.predict(X_ts)
30         mse[ia,ifold] = np.mean((y_ts_pred-y_ts)**2)
31         RSS_LA[ia,ifold] = np.mean((y_ts_pred-y_ts)**2)/(np.std(y_
```

In [80]:

```
1 #mse_mean = np.mean(mse, axis=1)
2 #mse_std = np.std(mse,axis=1)/np.sqrt(nfold-1)
3 RSS_LA_mean = np.mean(RSS_LA, axis=1)
4 RSS_LA_std = np.std(RSS_LA,axis=1)/np.sqrt(nfold-1)
5 plt.semilogx(alphas, RSS_LA_mean)
6 plt.xlabel('alpha')
7 plt.ylabel('Test MSE')
8 plt.grid()
9 plt.show()
```



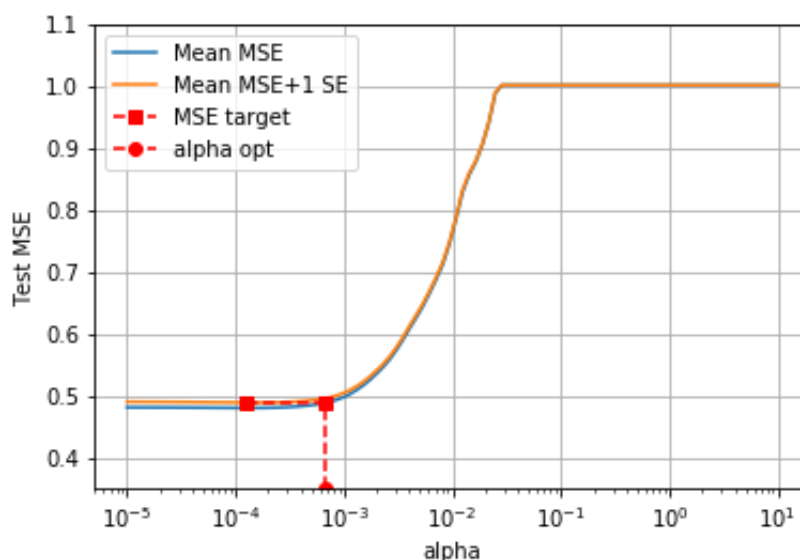
```

In [81]: 1 imin = np.argmin(mse_mean)
2 mse_tgt = mse_mean[imin] + mse_std[imin]
3 alpha_min = alphas[imin]
4
5 # Find the least complex model with mse_mean < mse_tgt
6 I = np.where(mse_mean < mse_tgt)[0]
7 iopt = I[-1]
8 alpha_opt = alphas[iopt]
9 print("Optimal alpha = %f" % alpha_opt)
10 print("mean test RSS using the one standard error rule: {0:f}"
11       .format(mse_mean[iopt]))
12
13 # Plot the mean MSE and the mean MSE + 1 std dev
14 plt.semilogx(alphas, mse_mean)
15 plt.semilogx(alphas, mse_mean+mse_std)
16
17 # Plot the MSE target
18 plt.semilogx([alpha_min,alpha_opt], [mse_tgt,mse_tgt], 'rs--')
19
20 # Plot the optimal alpha line
21 plt.semilogx([alpha_opt,alpha_opt], [0.35,mse_mean[iopt]], 'ro--')
22
23 plt.legend(['Mean MSE', 'Mean MSE+1 SE', 'MSE target','alpha opt'])
24 plt.xlabel('alpha')
25 plt.ylabel('Test MSE')
26 plt.ylim([0.35,1.1])
27 plt.grid()
28 plt.show()

```

Optimal alpha = 0.000658

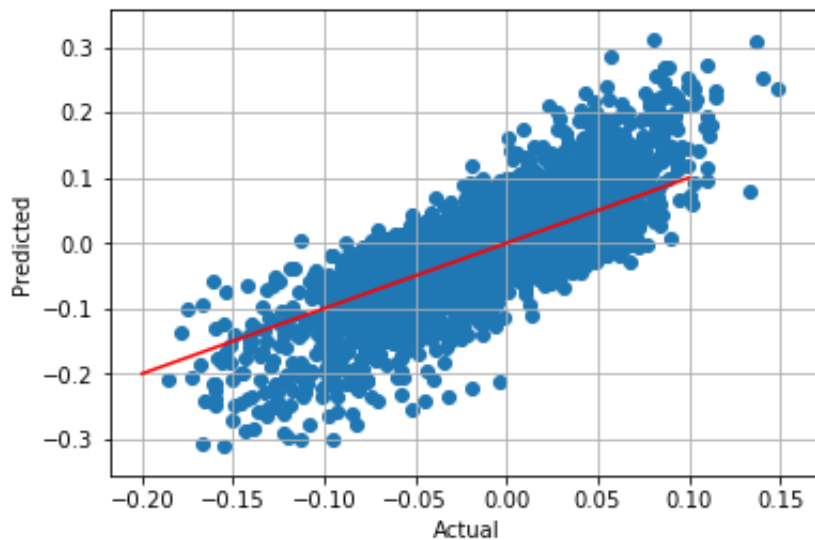
mean test RSS using the one standard error rule: 0.487399





```
In [90]: 1 Xtr_op = Xs[tr_dat]
2 ytr_op = y0[tr_dat]
3 Xts_op = Xs[ts_dat]
4 yts_op = y0[ts_dat]
5
6 model.alpha = alpha_opt
7 model.fit(Xtr_op, ytr_op)
8 y_op_pred = model.predict(Xts_op)
9 RSS_op = np.mean((y_op_pred-yts_op)**2)/(np.std(yts_op))
10 print("Using the optimal alpha, RSS is: {0:f}".format(RSS_op))
11 plt.scatter(y_op_pred, yts_op)
12 plt.plot([-0.2,0.1],[-0.2,0.1], 'r')
13 plt.xlabel('Actual')
14 plt.ylabel('Predicted')
15 plt.grid()
```

Using the optimal alpha, RSS is: 0.026956



```
In [ ]: 1
```