Exercises2_answers

February 16, 2021

1 Exercises 2 answers

1.1 Problem A1

1.1.1 Load the data

[1]:
```
import numpy as np
import os
os.chdir('/home/tuomas/Python/DATA.STAT.770/E2/')
data = np.loadtxt('noisy_sculpt_faces.txt')

images = data[:,:-3]
angles_gt = data[:,-3:]
```

1.1.2 a)

[2]:
```
#%% a) Nearest neighbor predictor & errors
from numpy.linalg import norm

def NN1_predictor(images, angles_gt):
    angles_pred = []
    for i in range(images.shape[0]):
        img = images[i]
        distances = np.square(np.sum(images - img, axis=1))
        distances[i] = np.finfo('float').max
        closestidx = np.argmin(distances)

        angles_pred.append(angles_gt[closestidx])

    return np.array(angles_pred)

def leaveoneout_error(angles_gt, angles_pred):
    errors = np.sum(angles_gt - angles_pred, axis=1)
    errors = np.square(errors)

    return errors.sum()

pred = NN1_predictor(images, angles_gt)
```

```
err = leaveoneout_error(angles_gt, pred)
print('Error with all features = {}'.format(err))
```

```
Error with all features = 940954.4692629011
```

1.1.3 b)

[5]:
```
#%% b) Forward selection
def forward_selection(images, angles_gt):
    errors_iter = [np.finfo('float').max]
    n = 1
    best_features = []
    while True:
        errors = []
        # Calculate error terms with specific set of features
        for i in range(images.shape[1]):
            if i in best_features:
                errors.append(np.finfo('float').max)
                continue

            curr_features = best_features + [i]
            imgs = images[:, curr_features]
            angles_pred = NN1_predictor(imgs, angles_gt)
            error = leaveoneout_error(angles_gt, angles_pred)
            errors.append(error)

        errors = np.array(errors)
        bf = np.argmin(errors)
        # If performance was improved
        if errors[bf] < errors_iter[-1]:
            best_features.append(bf)
            errors_iter.append(errors[bf])

        else:
            break

        print('Round {}:'.format(n))
        print('Error = {}'.format(errors_iter[-1]))
        print('Best features = {}'.format(best_features))
        n+=1

    return np.array(best_features), np.array(errors_iter[1:])

bf, errors = forward_selection(images, angles_gt)
print()
print('Error with forward selection = {}'.format(errors[-1]))
print('Used features = {}'.format(bf))
```

```
Round 1:
Error = 296153.8447611084
Best features = [189]

Error with forward selection = 296153.8447611084
Used features = [189]
```

1.1.4 c)

[9]:

```python
#%% c) Variant of forward selection
def forward_selection_v(images, angles_gt):
    errors_iter = []
    n = 1
    best_features = []
    while True:
        errors = []
        # Calculate error terms with specific set of features
        for i in range(images.shape[1]):
            if i in best_features:
                errors.append(np.finfo('float').max)
                continue

            curr_features = best_features + [i]
            imgs = images[:, curr_features]
            angles_pred = NN1_predictor(imgs, angles_gt)
            error = leaveoneout_error(angles_gt, angles_pred)
            errors[i] = error
            errors.append(error)

        errors = np.array(errors)
        bf = np.argmin(errors)
        # If there are unused features
        if len(best_features) < images.shape[1]:
            best_features.append(bf)
            errors_iter.append(errors[bf])

        else:
            break

        print('Round {}/{}:'.format(n, images.shape[1]))
        #print('Error = {}'.format(errors_iter[-1]))
        #print('Best features = {}'.format(best_features))
        n+=1

    return np.array(best_features), np.array(errors_iter)

bf, errors_itr = forward_selection_v(images, angles_gt)
```

```
Round 1/256:
Round 2/256:
Round 3/256:
Round 4/256:
Round 5/256:
Round 6/256:
Round 7/256:
Round 8/256:
Round 9/256:
Round 10/256:
Round 11/256:
Round 12/256:
Round 13/256:
Round 14/256:
Round 15/256:
Round 16/256:
Round 17/256:
Round 18/256:
Round 19/256:
Round 20/256:
Round 21/256:
Round 22/256:
Round 23/256:
Round 24/256:
Round 25/256:
Round 26/256:
Round 27/256:
Round 28/256:
Round 29/256:
Round 30/256:
Round 31/256:
Round 32/256:
Round 33/256:
Round 34/256:
Round 35/256:
Round 36/256:
Round 37/256:
Round 38/256:
Round 39/256:
Round 40/256:
Round 41/256:
Round 42/256:
Round 43/256:
Round 44/256:
Round 45/256:
Round 46/256:
Round 47/256:
Round 48/256:
```

```
Round 49/256:
Round 50/256:
Round 51/256:
Round 52/256:
Round 53/256:
Round 54/256:
Round 55/256:
Round 56/256:
Round 57/256:
Round 58/256:
Round 59/256:
Round 60/256:
Round 61/256:
Round 62/256:
Round 63/256:
Round 64/256:
Round 65/256:
Round 66/256:
Round 67/256:
Round 68/256:
Round 69/256:
Round 70/256:
Round 71/256:
Round 72/256:
Round 73/256:
Round 74/256:
Round 75/256:
Round 76/256:
Round 77/256:
Round 78/256:
Round 79/256:
Round 80/256:
Round 81/256:
Round 82/256:
Round 83/256:
Round 84/256:
Round 85/256:
Round 86/256:
Round 87/256:
Round 88/256:
Round 89/256:
Round 90/256:
Round 91/256:
Round 92/256:
Round 93/256:
Round 94/256:
Round 95/256:
Round 96/256:
```

```
Round 97/256:
Round 98/256:
Round 99/256:
Round 100/256:
Round 101/256:
Round 102/256:
Round 103/256:
Round 104/256:
Round 105/256:
Round 106/256:
Round 107/256:
Round 108/256:
Round 109/256:
Round 110/256:
Round 111/256:
Round 112/256:
Round 113/256:
Round 114/256:
Round 115/256:
Round 116/256:
Round 117/256:
Round 118/256:
Round 119/256:
Round 120/256:
Round 121/256:
Round 122/256:
Round 123/256:
Round 124/256:
Round 125/256:
Round 126/256:
Round 127/256:
Round 128/256:
Round 129/256:
Round 130/256:
Round 131/256:
Round 132/256:
Round 133/256:
Round 134/256:
Round 135/256:
Round 136/256:
Round 137/256:
Round 138/256:
Round 139/256:
Round 140/256:
Round 141/256:
Round 142/256:
Round 143/256:
Round 144/256:
```

```
Round 145/256:
Round 146/256:
Round 147/256:
Round 148/256:
Round 149/256:
Round 150/256:
Round 151/256:
Round 152/256:
Round 153/256:
Round 154/256:
Round 155/256:
Round 156/256:
Round 157/256:
Round 158/256:
Round 159/256:
Round 160/256:
Round 161/256:
Round 162/256:
Round 163/256:
Round 164/256:
Round 165/256:
Round 166/256:
Round 167/256:
Round 168/256:
Round 169/256:
Round 170/256:
Round 171/256:
Round 172/256:
Round 173/256:
Round 174/256:
Round 175/256:
Round 176/256:
Round 177/256:
Round 178/256:
Round 179/256:
Round 180/256:
Round 181/256:
Round 182/256:
Round 183/256:
Round 184/256:
Round 185/256:
Round 186/256:
Round 187/256:
Round 188/256:
Round 189/256:
Round 190/256:
Round 191/256:
Round 192/256:
```

```
Round 193/256:
Round 194/256:
Round 195/256:
Round 196/256:
Round 197/256:
Round 198/256:
Round 199/256:
Round 200/256:
Round 201/256:
Round 202/256:
Round 203/256:
Round 204/256:
Round 205/256:
Round 206/256:
Round 207/256:
Round 208/256:
Round 209/256:
Round 210/256:
Round 211/256:
Round 212/256:
Round 213/256:
Round 214/256:
Round 215/256:
Round 216/256:
Round 217/256:
Round 218/256:
Round 219/256:
Round 220/256:
Round 221/256:
Round 222/256:
Round 223/256:
Round 224/256:
Round 225/256:
Round 226/256:
Round 227/256:
Round 228/256:
Round 229/256:
Round 230/256:
Round 231/256:
Round 232/256:
Round 233/256:
Round 234/256:
Round 235/256:
Round 236/256:
Round 237/256:
Round 238/256:
Round 239/256:
Round 240/256:
```

```
Round 241/256:
Round 242/256:
Round 243/256:
Round 244/256:
Round 245/256:
Round 246/256:
Round 247/256:
Round 248/256:
Round 249/256:
Round 250/256:
Round 251/256:
Round 252/256:
Round 253/256:
Round 254/256:
Round 255/256:
Round 256/256:
```

[7]:
```
# Report the order in which the features were added, and the performance␣
↪achieved with each number of features.
print('Order of added features:')
print(bf)
print()
print('Achieved performance with respect to the added featuers:')
print(errors_itr)
```

```
Order of added features:
[189 107 224 120  52  92  91 245  53  75   2 167   8 102 226 222 207 128
   9 106 238  33 124 174   0 137 253  24  49 182 206  87 230 244  32 250
 228  59 201 229  17  40  13 100  18  12 205  56 218  37 125 215  26  96
  42 200 216  84 246 231  66 188 147 142 152  78 114 170 211 135 219 194
  73  58  54 113 127 171 115 133 195 138 198 166 144 117  71 153 105 197
  93 112 254 208 151 165 179  85 122  89 101 220  10 104  39 251   1 192
 242 141 199  57  79  80 116   3  77 140 158 169 145 210 191 132 155 146
  98  76 177  41 234  15  68  63 190  62  70 180 162 187 214  51  35  94
 181 202   6  64  50  99 184 126  38 175  67 157 131  16 149  36  44 163
 111 212  45 176  83 240  29 134 217   5  81  46  28 154  95 148 186  60
 252 247  86  21 143  69 119 249   7  48 160  27  20  31  34 239  25 255
 172 136 109  82 209  23 183 236 108  30 185 203  90  65 156  97  22  61
 241 123 232  47 150 118 221 159 129 178 168 161  43   4  19 233 243 164
  72 130 121  74 235 225 248 223 193  11  55  88 204 196 237 139 110 173
  14 213 103 227]

Achieved performance with respect to the added featuers:
[296153.84476111 326704.29844994 345324.84124088 215447.7897567
 292456.97954462 365603.58514977 400546.80482636 322014.87311328
 339270.20581776 280642.10192167 283735.73740546 248454.00213985
 237345.05715992 267439.30965924 234009.0660711  243040.15319177
 214177.83037494 257162.00511836 220430.38674856 231946.06081274
```

220074.89768815	246798.41157715	270797.94266498	264554.96333124
243130.31093583	243992.33005595	227023.47577584	212583.37575089
238156.41870668	182008.22788525	238885.55030774	226557.32238094
174798.12931474	255713.73724416	170830.1969302	184941.38851825
223304.77361561	246335.89219809	220421.57355949	184384.12597099
261161.70625258	272485.99772997	282998.43931926	262065.36030969
230067.76558083	237361.45027176	229161.53776939	185169.77449326
241880.88037582	204164.59681836	160615.53109892	211075.56805372
195729.0660469	215414.31460766	190334.02731215	203340.11570622
240492.08633546	230426.58697892	170702.91514294	264298.44436226
218171.45256125	317615.74000089	204864.68406859	275567.66421948
288508.92674472	205063.06149087	226950.14489796	259812.63659155
302878.5437999	190862.27130385	254346.16825617	259655.53019679
322849.22392655	228686.91846878	276265.84978048	285092.11933373
332182.91315084	305260.53561784	294136.5251053	293394.77968103
259260.03229541	255281.32848474	289582.29785613	274983.27352586
290604.25202224	325572.71958134	241346.08672889	228340.8051909
188032.30151452	249285.06588392	201886.59038014	211780.88367307
202127.02689782	193658.62084647	198557.8771039	180120.98475009
219739.67199437	222280.27019121	244055.45654509	230181.71895284
234030.58504035	281608.0562475	235146.77533838	307697.46113311
310584.41341949	246246.15611587	274789.63613011	251319.28897635
227325.33963361	283063.10329572	189730.63448737	232485.52898576
208929.67593894	194320.35657609	193144.89596199	218300.87932086
303461.79563457	205519.48020537	242613.5623154	242284.28924687
214536.72596804	217861.64226221	225812.04914946	226806.69849381
212810.56409725	228013.55777907	259839.65827875	317862.72699219
238757.20665835	303553.24525294	260813.96369324	212110.72239538
267181.99134881	261349.53351617	217616.95029683	177224.82247636
214071.19209916	198379.05115145	163813.20514143	217468.08722811
272044.9149005	296130.41952048	280625.65652848	227438.79007665
231087.1136025	270762.76178189	211682.42929353	247408.81094519
232270.52540605	209430.65127332	209887.69736361	210170.80924372
210849.01253338	231164.67724688	171852.16498671	212436.29934114
157595.47221999	199080.72995328	183963.46265942	171326.46238909
176905.9587833	208105.72716437	227477.2372911	212667.39638128
220350.25354184	223420.25690335	238733.78505363	199348.79393
253152.96247424	289895.48084233	205588.59142885	177191.24432124
156682.23367429	194361.94189038	222224.69652606	174332.7723067
157658.25168264	192139.31680105	190924.45477234	216136.48351374
227592.08964505	245107.43725735	188447.54252819	224676.99059139
277685.55128773	253540.21335906	220619.06982044	188173.20609352
262641.64639067	224421.94610336	255515.90883821	194103.45126709
199226.98572475	299369.14825737	264503.25002178	235696.36105985
217417.85054694	232183.19097192	160052.26940269	230682.93916774
288514.66833521	339239.95191158	194328.11550632	290545.0556994
194348.55646055	254295.27943614	262706.17495304	282008.24701923
240178.29486177	282781.60862642	275612.08776136	266292.94338242

```
 353928.92896064 314093.58505906 317165.58790032 267236.29732119
 286810.16397208 321540.31326338 279199.56505857 348903.25540301
 362576.03322593 235916.53169148 290602.02800105 337066.37101643
 307208.81540111 308877.5541587  290897.00049064 266499.31390413
 285774.75457488 267876.87467478 400336.32401895 402267.66062895
 372491.60891246 456064.89395476 431100.65244123 442083.05741991
 533554.50345008 538467.45278277 550344.70631614 485414.01916757
 527655.84179959 511987.05463298 461978.73488139 500557.3133428
 522510.46196646 398282.38810457 422932.50817489 512378.32494269
 617968.2449507  495851.99855313 555767.48379054 755286.73425451
 532360.31857418 755377.53301791 794784.06131068 940954.4692629 ]
```

[16]:

```python
# Report the set of features that achieved the best performance
min_error = np.argmin(errors_itr)
bbf = bf[:min_error]
print('Best features:')
print(bbf)
print('Number of features = {}'.format(len(bbf)))
print('With error:')
print(errors_itr[min_error])
```

```
Best features:
[189 107 224 120  52  92  91 245  53  75   2 167   8 102 226 222 207 128
   9 106 238  33 124 174   0 137 253  24  49 182 206  87 230 244  32 250
 228  59 201 229  17  40  13 100  18  12 205  56 218  37 125 215  26  96
  42 200 216  84 246 231  66 188 147 142 152  78 114 170 211 135 219 194
  73  58  54 113 127 171 115 133 195 138 198 166 144 117  71 153 105 197
  93 112 254 208 151 165 179  85 122  89 101 220  10 104  39 251   1 192
 242 141 199  57  79  80 116   3  77 140 158 169 145 210 191 132 155 146
  98  76 177  41 234  15  68  63 190  62  70 180 162 187 214  51  35  94
 181 202   6  64  50  99 184 126  38 175  67 157 131  16 149  36  44 163
 111 212  45 176  83 240  29 134 217   5]
Number of features = 172
With error:
156682.2336742861
```

1.1.5 d)

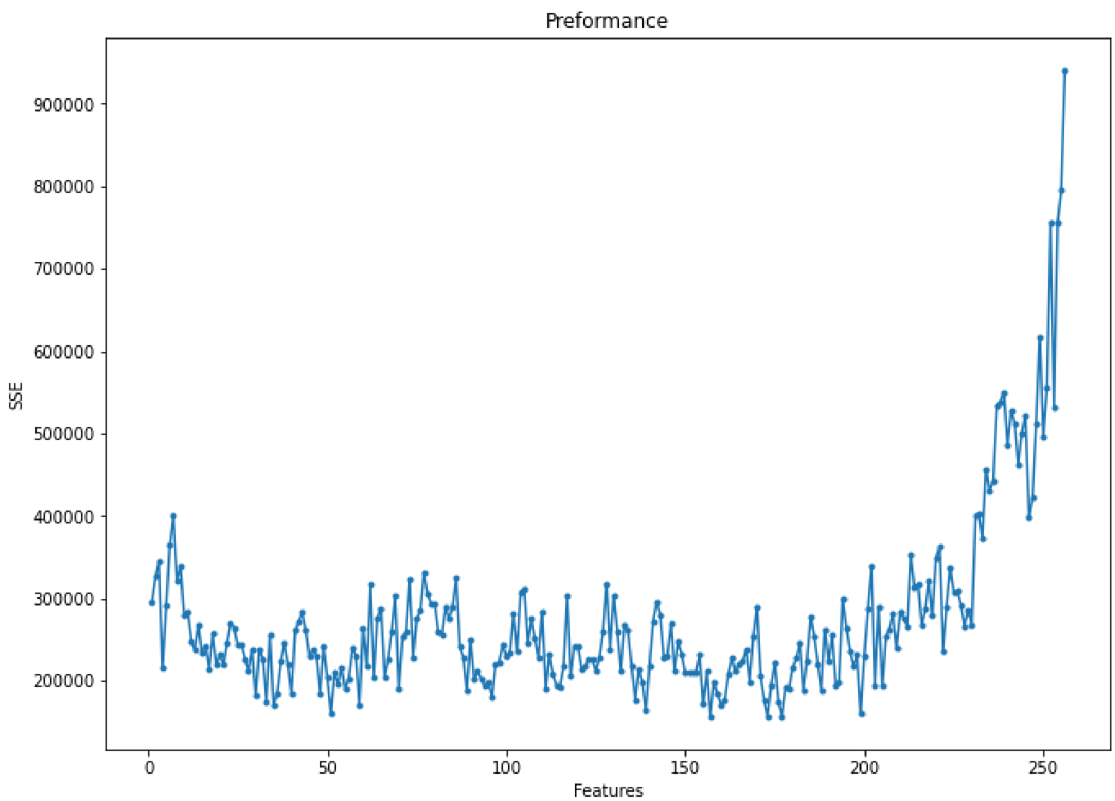
1.1.6 Part 1:

The set of features obtained in part b) ([189]) was not the overall best collection of features (eg. example case above with 172 features obtained best result). This can happen since the forward selection stops immediatly when the performance does not improve. However, there is still a possibility that the joint effect of the set of arbitary features reaches better performance, although the performance seems to first decrease.

1.1.7 Part 2:

[10]:
```python
#%% d), Part 2. Performance plotting
import matplotlib.pyplot as plt
plt.figure(figsize=(11,8))
plt.title('Preformance')
plt.ylabel('SSE')
plt.xlabel('Features')
plt.plot(np.arange(1,bf.shape[0]+1), errors_itr, '.-')
```

[10]:
```
[<matplotlib.lines.Line2D at 0x7f3c938b6550>]
```



1.2 Problem A2

1.2.1 1.

Ranking by Pearson correlation can be heoretically applied since the pixel averages and angle averages can be calculated. Therefore the estimate for Pearson correlaton can be calculated (slide 15), where the R(i) is the correlation of feature i (1<=i<=256) and the angle. However, intuitively this doesn't seem like a appropriate approach since the single pixel value doesn't really consist any spatial information.

Ranking by mutual information can't be used since both of the variables (grayscaled pixel values and angles) are continuous and the estimation of the probability distributions becomes difficult.

1.2.2 2.

This feature method differs from the one presented in 1c such that the each feature is ranked individually (not in the context of the alredy selected features). This method of course lead to decreased performance since joint effect is not considered when selecting the features. For example, this mehod treats redundant features as evenly good in terms of the final performance.

[17]:
```
#%% 2)
def forward_selection_new(images, angles_gt):
    errors_iter = []
    n = 1
    best_features = []
    while True:
        errors = []
        # Calculate error terms with specific set of features
        for i in range(images.shape[1]):
            if i in best_features:
                errors.append(np.finfo('float').max)
                continue

            curr_features = [i]
            imgs = images[:, curr_features]
            angles_pred = NN1_predictor(imgs, angles_gt)
            error = leaveoneout_error(angles_gt, angles_pred)
            errors.append(error)

        errors = np.array(errors)
        bf = np.argmin(errors)
        # If there are unused features
        if len(best_features) < images.shape[1]:
            best_features.append(bf)

            imgs = images[:, best_features]
            angles_pred = NN1_predictor(imgs, angles_gt)
            curr_error_tot = leaveoneout_error(angles_gt, angles_pred)

            errors_iter.append(curr_error_tot)

        else:
            break

        print('Round {}/{}:'.format(n, images.shape[1]))
        #print('Error = {}'.format(errors_iter[-1]))
        #print('Best features = {}'.format(best_features))
        n+=1
```

```python
    return np.array(best_features), np.array(errors_iter[:])

bf, errors_itr = forward_selection_new(images, angles_gt)
```

```
Round 1/256:
Round 2/256:
Round 3/256:
Round 4/256:
Round 5/256:
Round 6/256:
Round 7/256:
Round 8/256:
Round 9/256:
Round 10/256:
Round 11/256:
Round 12/256:
Round 13/256:
Round 14/256:
Round 15/256:
Round 16/256:
Round 17/256:
Round 18/256:
Round 19/256:
Round 20/256:
Round 21/256:
Round 22/256:
Round 23/256:
Round 24/256:
Round 25/256:
Round 26/256:
Round 27/256:
Round 28/256:
Round 29/256:
Round 30/256:
Round 31/256:
Round 32/256:
Round 33/256:
Round 34/256:
Round 35/256:
Round 36/256:
Round 37/256:
Round 38/256:
Round 39/256:
Round 40/256:
Round 41/256:
Round 42/256:
Round 43/256:
```

```
Round 44/256:
Round 45/256:
Round 46/256:
Round 47/256:
Round 48/256:
Round 49/256:
Round 50/256:
Round 51/256:
Round 52/256:
Round 53/256:
Round 54/256:
Round 55/256:
Round 56/256:
Round 57/256:
Round 58/256:
Round 59/256:
Round 60/256:
Round 61/256:
Round 62/256:
Round 63/256:
Round 64/256:
Round 65/256:
Round 66/256:
Round 67/256:
Round 68/256:
Round 69/256:
Round 70/256:
Round 71/256:
Round 72/256:
Round 73/256:
Round 74/256:
Round 75/256:
Round 76/256:
Round 77/256:
Round 78/256:
Round 79/256:
Round 80/256:
Round 81/256:
Round 82/256:
Round 83/256:
Round 84/256:
Round 85/256:
Round 86/256:
Round 87/256:
Round 88/256:
Round 89/256:
Round 90/256:
Round 91/256:
```

```
Round 92/256:
Round 93/256:
Round 94/256:
Round 95/256:
Round 96/256:
Round 97/256:
Round 98/256:
Round 99/256:
Round 100/256:
Round 101/256:
Round 102/256:
Round 103/256:
Round 104/256:
Round 105/256:
Round 106/256:
Round 107/256:
Round 108/256:
Round 109/256:
Round 110/256:
Round 111/256:
Round 112/256:
Round 113/256:
Round 114/256:
Round 115/256:
Round 116/256:
Round 117/256:
Round 118/256:
Round 119/256:
Round 120/256:
Round 121/256:
Round 122/256:
Round 123/256:
Round 124/256:
Round 125/256:
Round 126/256:
Round 127/256:
Round 128/256:
Round 129/256:
Round 130/256:
Round 131/256:
Round 132/256:
Round 133/256:
Round 134/256:
Round 135/256:
Round 136/256:
Round 137/256:
Round 138/256:
Round 139/256:
```

```
Round 140/256:
Round 141/256:
Round 142/256:
Round 143/256:
Round 144/256:
Round 145/256:
Round 146/256:
Round 147/256:
Round 148/256:
Round 149/256:
Round 150/256:
Round 151/256:
Round 152/256:
Round 153/256:
Round 154/256:
Round 155/256:
Round 156/256:
Round 157/256:
Round 158/256:
Round 159/256:
Round 160/256:
Round 161/256:
Round 162/256:
Round 163/256:
Round 164/256:
Round 165/256:
Round 166/256:
Round 167/256:
Round 168/256:
Round 169/256:
Round 170/256:
Round 171/256:
Round 172/256:
Round 173/256:
Round 174/256:
Round 175/256:
Round 176/256:
Round 177/256:
Round 178/256:
Round 179/256:
Round 180/256:
Round 181/256:
Round 182/256:
Round 183/256:
Round 184/256:
Round 185/256:
Round 186/256:
Round 187/256:
```

```
Round 188/256:
Round 189/256:
Round 190/256:
Round 191/256:
Round 192/256:
Round 193/256:
Round 194/256:
Round 195/256:
Round 196/256:
Round 197/256:
Round 198/256:
Round 199/256:
Round 200/256:
Round 201/256:
Round 202/256:
Round 203/256:
Round 204/256:
Round 205/256:
Round 206/256:
Round 207/256:
Round 208/256:
Round 209/256:
Round 210/256:
Round 211/256:
Round 212/256:
Round 213/256:
Round 214/256:
Round 215/256:
Round 216/256:
Round 217/256:
Round 218/256:
Round 219/256:
Round 220/256:
Round 221/256:
Round 222/256:
Round 223/256:
Round 224/256:
Round 225/256:
Round 226/256:
Round 227/256:
Round 228/256:
Round 229/256:
Round 230/256:
Round 231/256:
Round 232/256:
Round 233/256:
Round 234/256:
Round 235/256:
```

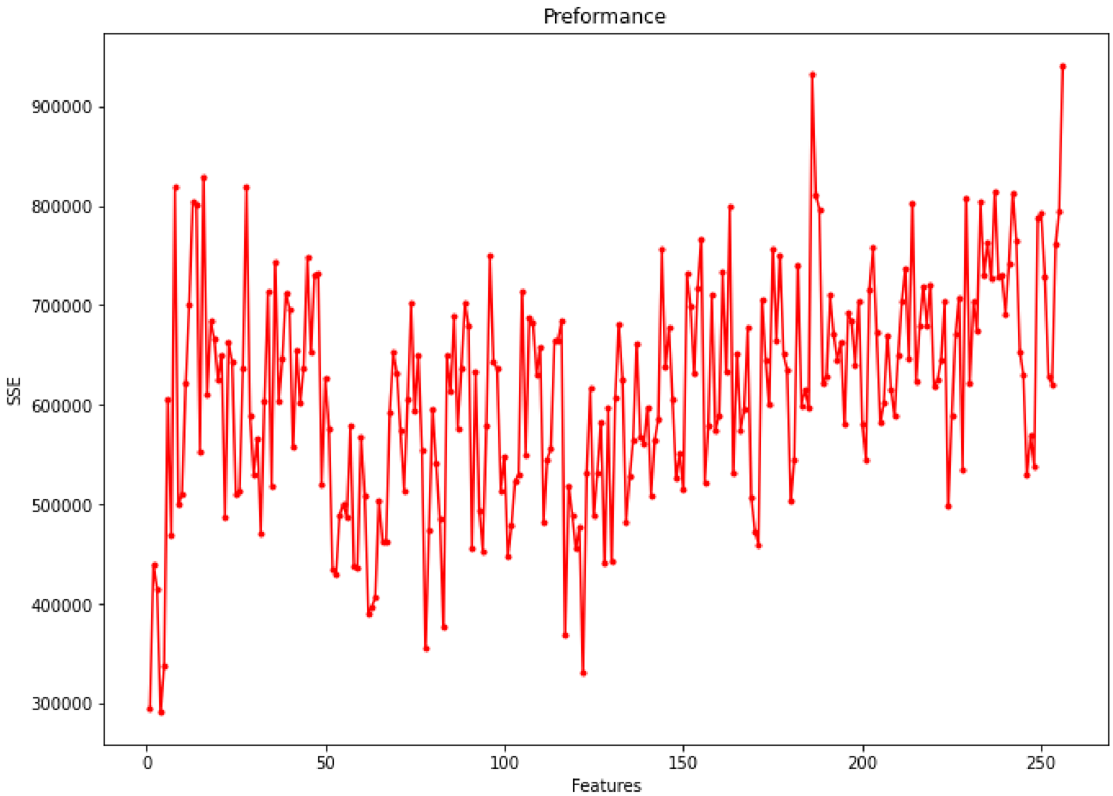
```
Round 236/256:
Round 237/256:
Round 238/256:
Round 239/256:
Round 240/256:
Round 241/256:
Round 242/256:
Round 243/256:
Round 244/256:
Round 245/256:
Round 246/256:
Round 247/256:
Round 248/256:
Round 249/256:
Round 250/256:
Round 251/256:
Round 252/256:
Round 253/256:
Round 254/256:
Round 255/256:
Round 256/256:
```

[18]:

```
#%% Some plotting
import matplotlib.pyplot as plt
plt.figure(figsize=(11,8))
plt.title('Preformance')
plt.ylabel('SSE')
plt.xlabel('Features')
plt.plot(np.arange(1,bf.shape[0]+1), errors_itr, '.-', color='red')
```

[18]:

```
[<matplotlib.lines.Line2D at 0x7f3c93877100>]
```


Preformance
900000
800000
700000
600000
500000
400000
300000
SSE
0
50
100
150
200
250
Features