A1 a) knn, leave one out and leave one out error (ssd)

In [204…

```
def k_nn (x_test, x_train, lrtd_train):
    labels = []
    distances = np.sqrt(np.sum((x_test[:, np.newaxis, :] - x_train)**2, axis=-1))

    for dist in distances:
        min_index = np.argmin(dist)
        # picks the index of the closest object and appends the train data angles j
        labels.append(lrtd_train[min_index])
    return labels
```

In [205…

```
def ssd (pred, test):
    sum_of_squared_diff = np.sum((pred-test)**2)
    # returns the sums between all predicted faces over the true angles
    return sum_of_squared_diff
```

In [220…

```
def result_compare(old_sum, new_sum):
    if old_sum > new_sum:
        return True
    else:
        return False
```

In [206…

```
import numpy as np
from sklearn.neighbors import KNeighborsRegressor


# Read the contents of the file into a np array
data = np.loadtxt("noisy_sculpt_faces.txt")

data.shape
```

Out[206]:

```
(100, 259)
```

In [287…

```
from sklearn.model_selection import train_test_split

X = data[:,0:256]

angles = data[:,256:259]


# Split the data for the model
# The results may vary since the function splits the data randomly and due to very
x_train, x_test = train_test_split(X, test_size=0.2)
angles_train, angles_test =train_test_split(angles, test_size=0.2)


print(angles.shape,X.shape)
```

```
(100, 3) (100, 256)
```

In [288…

```
# prediction as a list of angles
pred = k_nn(x_test, x_train, angles_train)

# sum the squared error over all the faces
ssd(pred, angles_test)
```

Out[288]:

```
165242.77204331846
```

A1 b) forward selection

In [336...

```
def feature_selection(x_train,x_test,angles_train,angles_test):
    all_SSD = []
    filter_indices = []
    current_model = float('inf')


    is_better = True
    while is_better:
        best_index = 0
        current_ssd = float('inf')
        is_better = False
        for i in range(X.shape[1]):
            if i not in filter_indices:

                if not filter_indices:
                    new_indices = [i]
                else:
                    new_indices = np.concatenate((filter_indices, [i]))

                new_x_train = x_train[:,new_indices]
                new_x_test = x_test[:,new_indices]
                # Now the data has been filtered with the chosen indices (selected
                # a new index across all indexes in the data
                new_ssd = ssd(k_nn(new_x_test, new_x_train, angles_train), angles_t

                if new_ssd < current_ssd:
                    current_ssd = new_ssd
                    best_index = i

        # Adds the best index to the filter index list
        filter_indices.append(best_index)
        # If the new selected features propose a better model than current, the lo
        new_model = ssd(k_nn(new_x_test, new_x_train, angles_train), angles_test)


        if new_model < current_model:

            all_SSD.append(new_model)
            current_model = new_model
            is_better = True

    return all_SSD
```

A1 c) add best feature in each iteration until the end

In [466...

```
 def feature_selection_variant(x_train,x_test,angles_train,angles_test):
    all_SSD = []
    filter_indices = []
    current_model = float('inf')

    for j in range(X.shape[1]):
        best_index = 0
        current_ssd = float('inf')
        for i in range(X.shape[1]):
            if i not in filter_indices:

                if not filter_indices:
                    new_indices = [i]
                else:
                    new_indices = np.concatenate((filter_indices, [i]))

                new_x_train = x_train[:,new_indices]
```

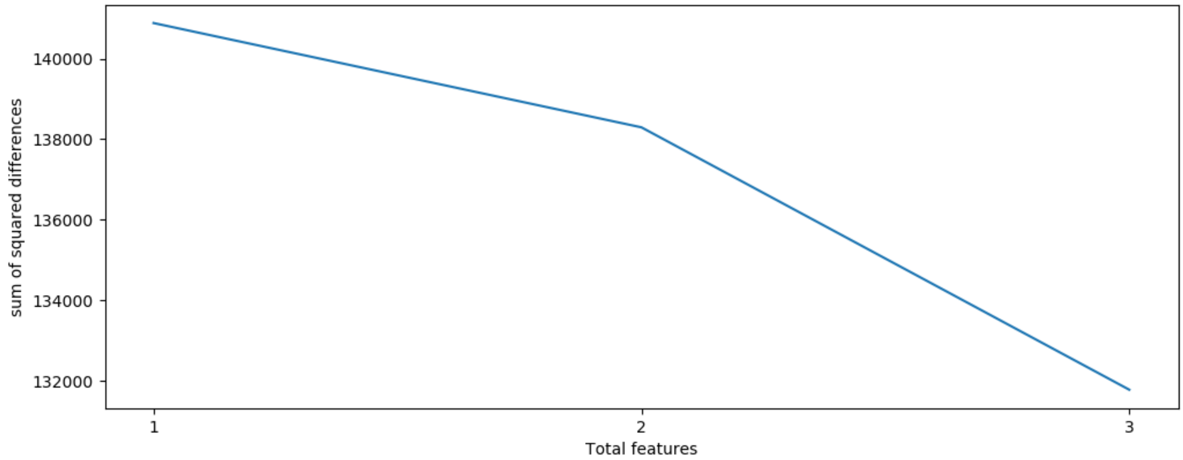
```
                new_x_test = x_test[:,new_indices]
                # Now the data has been filtered with the chosen indices (selected
                # and a new index across all indexes in the data
                new_ssd = ssd(k_nn(new_x_test, new_x_train, angles_train), angles_

                if new_ssd < current_ssd:
                    current_ssd = new_ssd
                    best_index = i

        # Adds the best index to the filter index list
        filter_indices.append(best_index)
        # If the new selected features propose a better model than current, the loc
        new_model = ssd(k_nn(new_x_test, new_x_train, angles_train), angles_test)

        all_SSD.append(new_model)
        if new_model < current_model:
            current_model = new_model

    return all_SSD, filter_indices
```

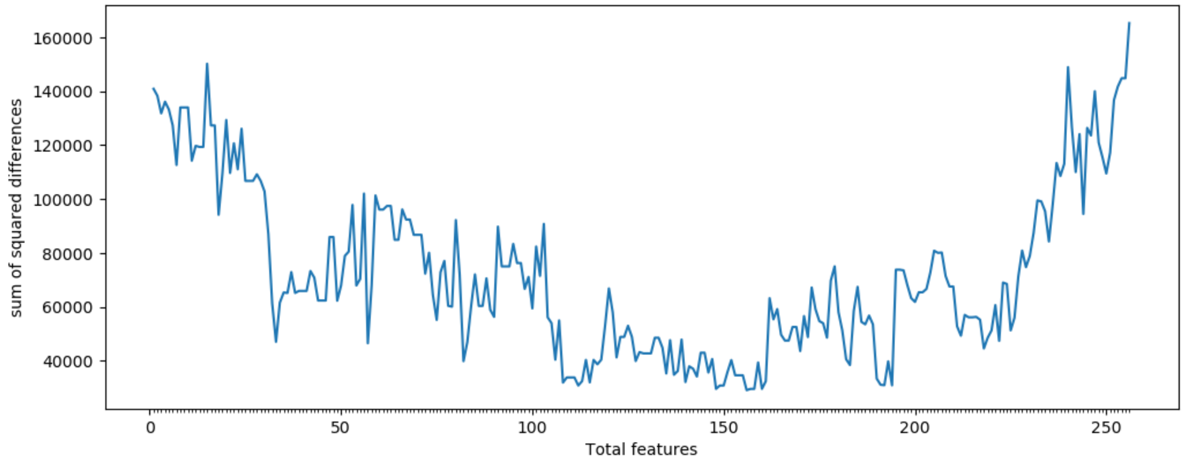
In [467…

```
import matplotlib.pyplot as plt
SSD = feature_selection(x_train,x_test,angles_train,angles_test)
SSD_var, indices = feature_selection_variant(x_train,x_test,angles_train,angles_te

fig, (ax1, ax2) = plt.subplots(2,1, figsize=(12,10))
x = np.arange(1, len(SSD) + 1, 1)
ax1.plot(x,SSD)
ax1.set_xticks(range(1,len(SSD)+1))
ax1.set_ylabel("sum of squared differences")
ax1.set_xlabel("Total features")

x2 = np.arange(1, len(SSD_var) + 1, 1)
ax2.plot(x2,SSD_var)
ax2.set_xticks(range(1,len(SSD_var)+1),10)
ax2.set_ylabel("sum of squared differences")
ax2.set_xlabel("Total features")
```

Out[467]:

```
Text(0.5, 0, 'Total features')
```





1A d)

With this split in training data and test data, the later algorithm gets better results quite early in the iterations due to there being a small change in the beginning. At 4th feature summed ssd score rises which stops the first algorithm. We can see from graph 2 that 2nd algorithm gets quite stable good results at 100-150 features and then starts getting worse again. Keep in mind that the order of the samples in test and train data will vary if the program is run from the start again, hence the results will be different as well.

2A Variable ranking

1.

Pearsons correlation could be used as a simple ranking method in feature selection by showing what features may have correlation to the target variables. The features with high values could be chosen as the variables to be used in a model. However in this case, the correlation would assume that the variables are independent of each other, which might cause problems. Correlation can also only detect linear dependencies between vaariable and the target.

2.

In [475…
```
def variable_ranking(x_train,x_test,angles_train,angles_test):
    import operator

    all_ssd = {}

    for i in range(X.shape[1]):
```

```
        indences = [i]
        new_x_train = x_train[:,indences]
        new_x_test = x_test[:,indences]

        new_ssd = ssd(k_nn(new_x_test, new_x_train, angles_train), angles_test)

        # adds a pair, index and the squared error sum to a list
        all_ssd[i]=new_ssd
    sorted_ssd = sorted(all_ssd.items(), key=operator.itemgetter(1))
    return sorted_ssd
```

In [476…

```
ranking = variable_ranking(x_train,x_test,angles_train,angles_test)
for i in range(len(indices)):
    print(indices[i], ranking[i][0])
```

```
8 8
168 79
39 196
224 38
20 82
12 244
122 139
26 24
13 124
207 146
244 70
166 103
245 241
98 114
102 207
189 66
201 181
24 90
219 222
59 137
192 229
205 219
138 58
32 205
142 0
114 240
53 147
87 251
118 26
112 164
128 143
0 87
80 194
58 19
133 105
9 104
137 100
54 159
226 254
162 15
89 33
179 77
99 224
246 48
250 72
120 99
84 175
251 145
107 25
125 31
18 253
56 61
104 168
46 183
206 56
212 180
1 195
60 120
101 30
165 119
180 74
222 158
135 243
116 96
```

```
253 184
115 6
229 217
147 156
124 34
218 169
228 80
215 128
15 84
71 91
106 213
63 172
2 140
186 237
57 53
81 211
96 43
105 1
113 122
174 218
188 115
17 163
200 93
153 215
72 234
230 5
190 98
62 209
146 210
198 123
49 188
170 227
97 101
167 117
194 108
41 54
214 7
240 88
35 75
61 89
93 134
31 4
148 176
176 81
73 35
52 238
132 37
155 12
159 160
6 127
38 21
141 23
22 113
68 17
182 223
79 255
199 39
100 52
197 85
158 193
185 47
191 20
111 239
37 32
```

```
67 202
171 249
225 92
187 73
184 41
210 161
27 216
40 57
51 245
163 167
126 102
169 118
238 138
145 11
45 44
208 129
64 182
252 63
131 230
21 212
172 133
151 28
29 67
140 190
150 60
7 199
92 29
195 97
220 86
233 178
254 78
255 152
5 221
28 16
88 242
3 226
175 177
110 233
11 151
70 200
47 203
181 228
33 232
69 144
152 153
108 107
177 142
94 18
144 192
136 131
209 14
34 110
173 208
44 83
42 171
119 141
19 116
48 71
77 174
4 155
247 94
85 231
130 135
127 162
```

```
248 69
249 64
204 166
65 59
161 130
178 132
36 22
16 9
134 40
213 27
241 225
139 157
123 248
82 246
83 45
203 126
25 62
74 150
237 247
149 149
30 252
121 136
14 121
50 125
91 46
43 250
75 109
154 220
55 173
66 13
236 10
103 179
227 214
76 55
239 185
183 186
221 154
78 51
202 2
234 112
231 106
217 206
23 235
129 187
196 197
117 204
242 42
235 236
164 201
232 148
193 165
90 65
10 3
109 170
86 111
216 76
157 49
243 50
223 95
160 191
95 198
143 68
211 189
156 36
```

Basically the first number is the same, but after that, the list is completely different