جواب سوال 8 و 9 و 10 و 11)

می خواهیم کلاسیفایر SVM را با استفاده از پکیج sklearn بر روی سه فیچر خواسته شده با استفاده از داده های آموزشی ترین کنیم :

می توانیم برای این کار از کرنل های مختلف SVM استفاده کنیم. در صورت استفاده نکردن از kernel به صورت پیش فرض 'rbf' انتخاب می شود. (default = 'rbf').

kernel*{'linear', 'poly', 'rbf', 'sigmoid', 'precomputed'} , defult = 'rbf'*

برای اطمینان از این موضوع ما یک بار بدون استفاده از کرنل و هم با استفاده از کرنل rbf جواب ها را مقایسه می کنیم.

خروجی در هر دو یکسان خواهد بود :

```
from sklearn import svm
x1 = ds[['trestbps','chol','thalach']]

# Create a SVM classifier
svclassifier = svm.SVC()
X_train, X_test, y_train, y_test = tts( x1 , y , test_size = 0.2, random_state = 0)
svclassifier.fit(X_train, y_train)
pred = svclassifier.predict(X_test)
print(confusion_matrix(y_test,pred))
print(classification_report(y_test,pred))
```

```
[[18 11]
 [ 4 25]]
              precision    recall  f1-score   support

           0       0.82      0.62      0.71        29
           1       0.69      0.86      0.77        29

    accuracy                           0.74        58
   macro avg       0.76      0.74      0.74        58
weighted avg       0.76      0.74      0.74        58
```

```
# Create a Radial basis function SVM classifier
svclassifier = svm.SVC(kernel='rbf')
X_train, X_test, y_train, y_test = tts( x1 , y , test_size = 0.2, random_state = 0)
svclassifier.fit(X_train, y_train)
pred = svclassifier.predict(X_test)
print(confusion_matrix(y_test,pred))
print(classification_report(y_test,pred))
```

```
[[18 11]
 [ 4 25]]
              precision    recall  f1-score   support

           0       0.82      0.62      0.71        29
           1       0.69      0.86      0.77        29

    accuracy                           0.74        58
   macro avg       0.76      0.74      0.74        58
weighted avg       0.76      0.74      0.74        58
```

اکنون می خواهیم با استفاده از دو کرنل متفاوت دیگر('linear' , 'poly') خروجی را مقایسه کنیم :

```
# Create a polynomial SVM classifier
svclassifier = svm.SVC(kernel='poly')
X_train, X_test, y_train, y_test = tts( x1 , y , test_size = 0.2, random_state = 0)
svclassifier.fit(X_train, y_train)
pred = svclassifier.predict(X_test)
print(confusion_matrix(y_test,pred))
print(classification_report(y_test,pred))
```

```
[[11 18]
 [ 4 25]]
              precision    recall  f1-score   support

           0       0.73      0.38      0.50        29
           1       0.58      0.86      0.69        29

    accuracy                           0.62        58
   macro avg       0.66      0.62      0.60        58
weighted avg       0.66      0.62      0.60        58
```

```
# Create a linear SVM classifier
svclassifier = svm.SVC(kernel='linear')
X_train, X_test, y_train, y_test = tts( x1 , y , test_size = 0.2, random_state = 0)
svclassifier.fit(X_train, y_train)
pred = svclassifier.predict(X_test)
print(confusion_matrix(y_test,pred))
print(classification_report(y_test,pred))
```

```
[[18 11]
 [ 3 26]]
              precision    recall  f1-score   support

           0       0.86      0.62      0.72        29
           1       0.70      0.90      0.79        29

    accuracy                           0.76        58
   macro avg       0.78      0.76      0.75        58
weighted avg       0.78      0.76      0.75        58
```

همان طور که ملاحظه می کنید خروجی های متفاوتی به دست آمد. کرنل poly مقدار خطای کمتری در کلاس 0 نسبت به بقیه کرنل ها دارد. اما میزان صحت سنجی کمتری دارد. کرنل linear بیشترین میزان صحت سنجی دارد اما بیشترین مقدار خطا را در کلاس 1 به خود اختصاص داده است. با مقایسه خروجی کرنل ها می توان به این نتیجه رسید بهترین جواب در این مثال متعلق به کرنل rbf است.

حال می خواهیم کلاسیفایر SVM را روی همه ی فیچرها با استفاده از داده های آموزشی ترین کنیم :

```
# Create a  SVM classifier on all dataset
svclassifier = svm.SVC()
X_train, X_test, y_train, y_test = tts( x , y , test_size = 0.2, random_state = 0)
svclassifier.fit(X_train, y_train)
pred = svclassifier.predict(X_test)
print(confusion_matrix(y_test,pred))
print(classification_report(y_test,pred))
```

```
[[25  4]
 [ 4 25]]
              precision    recall  f1-score   support

           0       0.86      0.86      0.86        29
           1       0.86      0.86      0.86        29

    accuracy                           0.86        58
   macro avg       0.86      0.86      0.86        58
weighted avg       0.86      0.86      0.86        58
```

با مقایسه خروجی کلاسیفایر SVM روی همه فیچر ها نسبت به خروجی کلاسیفایر SVM روی سه فیچر خواسته شده در می یابیم که کلاسیفایر روی همه فیچرها مقدار خطای مدل یکسانی در کلاس 0 و 1 دارد و میزان صحت سنجی بیشتری هم نسبت به سه فیچر دارد.

جواب ۱۱)

اعتبارسنجی متقابل k-fold (k-fold cross validation) :

در این نوع اعتبارسنجی متقابل، نمونه اصلی به‌طور تصادفی به زیرنمونه‌های فرعی با اندازه k تقسیم می‌شود. از زیرنمونه‌های فرعیk ، یک زیرنمونه منفرد به‌عنوان داده‌های اعتبارسنجی برای آزمایش مدل حفظ می‌شود و زیرنمونه‌های k - 1 به‌عنوان داده‌های آموزشی استفاده می‌شوند. سپس فرایند اعتبارسنج متقابل، که k بار تکرار می‌شود، با هر یک از نمونه‌های k به‌طور دقیق یک بار به‌عنوان داده‌های اعتبارسنجی مورد استفاده قرار می‌گیرد. پس از آن نتایج k می‌تواند برای تولید یک برآورد واحد به‌طور میانگین قرار بگیرد. در واقع این است که همه مشاهدات برای هر دو آموزش و اعتبار مورد استفاده قرار می‌گیرند، و هر مشاهده برای اعتبارسنجی به‌طور دقیق استفاده می‌شود.

برای مثال، تعیین k = 2 ، در اعتبارسنجی متقابل k-fold برابر ۲ است. در اعتبارسنجی متقابل 2-fold ما به‌طور تصادفی مجموعه‌داده‌ها را به دو دسته d0 و d1 جابه‌جا می‌کنیم. به‌طوری که هر دو مجموعه با اندازه مساوی باشند. این روش یک اعتبارسنجی متقابل خارج از صفحه است. این دسته‌ها به‌گونه‌ای انتخاب شده‌اند که مقدار پاسخ متوسط تقریباً در همه دسته‌ها برابر است. در طبقه‌بندی دوتایی، این به این معنی است که هر دسته تقریباً شامل نسبت‌های مشابه دو نوع از برچسب‌های کلاس می‌باشد.

```
# 5-fold Cross Validation with SVM classifier on all dataset
kf = KFold(n_splits=5)
for train_index, test_index in kf.split(x):
  print("train_index : ",train_index ,'\n',"test_index : ",test_index,'\n')
  x_train, x_test, y_train, y_test = x.iloc[train_index], x.iloc[test_index], y.iloc[train_index], y.iloc[test_index]
  svclassifier = svm.SVC().fit(x_train, y_train)
  score = svclassifier.score(x_test, y_test)
  print(score,'\n')
```

train_index : [58 59 60 61 62 63 64 65 66 67 68 69 70 71 72 73 74 75
76 77 78 79 80 81 82 83 84 85 86 87 88 89 90 91 92 93
94 95 96 97 98 99 100 101 102 103 104 105 106 107 108 109 110 111
112 113 114 115 116 117 118 119 120 121 122 123 124 125 126 127 128 129
130 131 132 133 134 135 136 137 138 139 140 141 142 143 144 145 146 147
148 149 150 151 152 153 154 155 156 157 158 159 160 161 162 163 164 165
166 167 168 169 170 171 172 173 174 175 176 177 178 179 180 181 182 183
184 185 186 187 188 189 190 191 192 193 194 195 196 197 198 199 200 201
202 203 204 205 206 207 208 209 210 211 212 213 214 215 216 217 218 219
220 221 222 223 224 225 226 227 228 229 230 231 232 233 234 235 236 237
238 239 240 241 242 243 244 245 246 247 248 249 250 251 252 253 254 255
256 257 258 259 260 261 262 263 264 265 266 267 268 269 270 271 272 273
274 275 276 277 278 279 280 281 282 283 284 285 286]
test_index : [0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23
24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40 41 42 43 44 45 46 47
48 49 50 51 52 53 54 55 56 57]

0.7931034482758621

train_index : [0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17
18 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35
36 37 38 39 40 41 42 43 44 45 46 47 48 49 50 51 52 53
54 55 56 57 116 117 118 119 120 121 122 123 124 125 126 127 128 129
130 131 132 133 134 135 136 137 138 139 140 141 142 143 144 145 146 147
148 149 150 151 152 153 154 155 156 157 158 159 160 161 162 163 164 165
166 167 168 169 170 171 172 173 174 175 176 177 178 179 180 181 182 183
184 185 186 187 188 189 190 191 192 193 194 195 196 197 198 199 200 201
202 203 204 205 206 207 208 209 210 211 212 213 214 215 216 217 218 219
220 221 222 223 224 225 226 227 228 229 230 231 232 233 234 235 236 237
238 239 240 241 242 243 244 245 246 247 248 249 250 251 252 253 254 255
256 257 258 259 260 261 262 263 264 265 266 267 268 269 270 271 272 273
274 275 276 277 278 279 280 281 282 283 284 285 286]
test_index : [58 59 60 61 62 63 64 65 66 67 68 69 70 71 72 73 74 75
76 77 78 79 80 81 82 83 84 85 86 87 88 89 90 91 92 93
94 95 96 97 98 99 100 101 102 103 104 105 106 107 108 109 110 111
112 113 114 115]

0.6379310344827587

train_index : [0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17
18 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35
36 37 38 39 40 41 42 43 44 45 46 47 48 49 50 51 52 53
54 55 56 57 58 59 60 61 62 63 64 65 66 67 68 69 70 71
72 73 74 75 76 77 78 79 80 81 82 83 84 85 86 87 88 89
90 91 92 93 94 95 96 97 98 99 100 101 102 103 104 105 106 107
108 109 110 111 112 113 114 115 173 174 175 176 177 178 179 180 181 182
183 184 185 186 187 188 189 190 191 192 193 194 195 196 197 198 199 200
201 202 203 204 205 206 207 208 209 210 211 212 213 214 215 216 217 218
219 220 221 222 223 224 225 226 227 228 229 230 231 232 233 234 235 236
237 238 239 240 241 242 243 244 245 246 247 248 249 250 251 252 253 254
255 256 257 258 259 260 261 262 263 264 265 266 267 268 269 270 271 272
273 274 275 276 277 278 279 280 281 282 283 284 285 286]
test_index : [116 117 118 119 120 121 122 123 124 125 126 127 128 129 130 131 132 133
134 135 136 137 138 139 140 141 142 143 144 145 146 147 148 149 150 151
152 153 154 155 156 157 158 159 160 161 162 163 164 165 166 167 168 169
170 171 172]

0.8070175438596491

train_index : [0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17
18 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35
36 37 38 39 40 41 42 43 44 45 46 47 48 49 50 51 52 53
54 55 56 57 58 59 60 61 62 63 64 65 66 67 68 69 70 71

```
 72  73  74  75  76  77  78  79  80  81  82  83  84  85  86  87  88  89
 90  91  92  93  94  95  96  97  98  99 100 101 102 103 104 105 106 107
108 109 110 111 112 113 114 115 116 117 118 119 120 121 122 123 124 125
126 127 128 129 130 131 132 133 134 135 136 137 138 139 140 141 142 143
144 145 146 147 148 149 150 151 152 153 154 155 156 157 158 159 160 161
162 163 164 165 166 167 168 169 170 171 172 230 231 232 233 234 235 236
237 238 239 240 241 242 243 244 245 246 247 248 249 250 251 252 253 254
255 256 257 258 259 260 261 262 263 264 265 266 267 268 269 270 271 272
273 274 275 276 277 278 279 280 281 282 283 284 285 286]
test_index :  [173 174 175 176 177 178 179 180 181 182 183 184 185 186 187 188 189 190
191 192 193 194 195 196 197 198 199 200 201 202 203 204 205 206 207 208
209 210 211 212 213 214 215 216 217 218 219 220 221 222 223 224 225 226
227 228 229]

0.7017543859649122

train_index :  [  0   1   2   3   4   5   6   7   8   9  10  11  12  13  14  15  16  17
 18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35
 36  37  38  39  40  41  42  43  44  45  46  47  48  49  50  51  52  53
 54  55  56  57  58  59  60  61  62  63  64  65  66  67  68  69  70  71
 72  73  74  75  76  77  78  79  80  81  82  83  84  85  86  87  88  89
 90  91  92  93  94  95  96  97  98  99 100 101 102 103 104 105 106 107
108 109 110 111 112 113 114 115 116 117 118 119 120 121 122 123 124 125
126 127 128 129 130 131 132 133 134 135 136 137 138 139 140 141 142 143
144 145 146 147 148 149 150 151 152 153 154 155 156 157 158 159 160 161
162 163 164 165 166 167 168 169 170 171 172 173 174 175 176 177 178 179
180 181 182 183 184 185 186 187 188 189 190 191 192 193 194 195 196 197
198 199 200 201 202 203 204 205 206 207 208 209 210 211 212 213 214 215
216 217 218 219 220 221 222 223 224 225 226 227 228 229]
test_index :  [230 231 232 233 234 235 236 237 238 239 240 241 242 243 244 245 246 247
248 249 250 251 252 253 254 255 256 257 258 259 260 261 262 263 264 265
266 267 268 269 270 271 272 273 274 275 276 277 278 279 280 281 282 283
284 285 286]

0.5087719298245614
```

برای تقسیم کردن مجموعه داده از KFold() استفاده کردیم.

با استفاده از KFold(n_splits=5) مجموعه داده را به ۵ قسمت تقسیم کردیم. که داده های تست ۲۰ درصد داده های تست هستند.

و این تقسیم بندی به ۵ شکل انجام می شود. یعنی هر بار از ۲۰ درصد بعدی به عنوان داده تست استفاده می شود.