EEE418: Advanced Pattern Recognition **Spring 2020**

Lab 3: Discriminant Function

Student: Zhiyu Xu *ID: 1926906*

3.1 Estimation of Classification Methods

1. Read the dataset and shuffle it

Table 3.1: Attribute Information: (class attribute has been moved to last column)

Attribute	Domain
Sample code number	1 - 10
Clump Thickness	1 - 10
Uniformity of Cell Size	1 - 10
Uniformity of Cell Shape	1 - 10
Marginal Adhesion	1 - 10
Single Epithelial Cell Size	1 - 10
Bare Nuclei	1 - 10
Bland Chromatin	1 - 10
Normal Nucleoli	1 - 10
Mitoses	1 - 10
Class	**2 for benign, 4 for malignant**

In order to make the operation and display beautiful, the Pandas library is used here, the data is read as a Dataframe file, and the column name is added.

Code as shown below:

```
columnNames = [
        'Sample code number',
        'Clump Thickness',
        'Uniformity of Cell Size',
        'Uniformity of Cell Shape',
        'Marginal Adhesion',
        'Single Epithelial Cell Size',
        'Bare Nuclei',
        'Bland Chromatin',
        'Normal Nucleoli',
        'Mitoses',
        'Class'
]
data = pd.read_csv('breast-cancer-wisconsin.data', names = columnNames)
print(data.shape)
data.head(10)
```

Result:

	Sample code number	Clump Thickness	Uniformity of Cell Size	Uniformity of Cell Shape	Marginal Adhesion	Single Epithelial Cell Size	Bare Nuclei	Bland Chromatin	Normal Nucleoli	Mitoses	Class
0	1000025	5	1	1	1	2	1	3	1	1	2
1	1002945	5	4	4	5	7	10	3	2	1	2
2	1015425	3	1	1	1	2	2	3	1	1	2
3	1016277	6	8	8	1	3	4	3	7	1	2
4	1017023	4	1	1	3	2	1	3	1	1	2
5	1017122	8	10	10	8	7	10	9	7	1	4
6	1018099	1	1	1	1	2	10	3	1	1	2
7	1018561	2	1	2	1	2	1	3	1	1	2
8	1033078	2	1	1	1	2	1	1	1	5	2
9	1033078	4	2	1	1	2	1	2	1	1	2

Figure 3.1: Top 10 of breast-cancer-wisconsin.data

2. Remove missing attribute values
   According to the documentation: There are 16 instances in Groups 1 to 6 that contain a single missing (i.e., unavailable) attribute value, now denoted by "?".

   Code as shown below:

```
#Replace ? with NaN
data = data.replace(to_replace='?',value = np.nan)
#Discard data with missing values (discard as long as one dimension is missing)
data = data.dropna(how='any')
data.shape
```

   Result:

```
(683, 11)
```

Figure 3.2: Size of new data

   Because there are 16 missing values, missing values where the row has been deleted, the data from the (699,11) into the (683,11)

3. Suffle dataset
   random.seed = 17

   Code as shown below:

```
data_shuffled = shuffle(data,random_state = 17).reset_index(drop=True)
data_shuffled.head(10)
```

Result:

	Sample code number	Clump Thickness	Uniformity of Cell Size	Uniformity of Cell Shape	Marginal Adhesion	Single Epithelial Cell Size	Bare Nuclei	Bland Chromatin	Normal Nucleoli	Mitoses	Class
0	1231853	4	2	2	1	2	1	2	1	1	2
1	1304595	3	1	1	1	1	1	2	1	1	2
2	1083817	3	1	1	1	2	1	2	1	1	2
3	1175937	5	4	6	7	9	7	8	10	1	4
4	188336	5	3	2	8	5	10	8	1	2	4
5	1080233	7	6	6	3	2	10	7	1	1	4
6	1232225	10	4	5	5	5	10	4	1	1	4
7	1230175	10	10	10	3	10	10	9	10	1	4
8	1212422	3	1	1	1	2	1	3	1	1	2
9	1243256	10	4	3	2	3	10	5	3	2	4

Figure 3.3: Shuffled data

4. Split the dataset as five parts
   Each of 5 subsets was used as test set and the remaining data was used for training. Five subsets were used for testing rotationally to evaluate the classification accuracy

   Code as shown below:

```
1  from sklearn.model_selection import KFold
2  kf = KFold(n_splits=5)
3  i = 0
4  data_train_index,data_test_index = [],[]
5  for train_index, test_index in kf.split(data):
6          i +=1
7          data_train_index.append(train_index)
8          data_test_index.append(test_index)
9          print('No.%i train:%i test:%i'%(i,len(data_train_index[i-1]),len(
               data_test_index[i-1])))
10 print(data_train_index[0])
```

Result:

```
No.1 train:546 test:137
No.2 train:546 test:137
No.3 train:546 test:137
No.4 train:547 test:136
No.5 train:547 test: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
 281 282 283 284 285 286 287 288 289 290 291 292 293 294 295 296 297 298
 299 300 301 302 303 304 305 306 307 308 309 310 311 312 313 314 315 316
 317 318 319 320 321 322 323 324 325 326 327 328 329 330 331 332 333 334
 335 336 337 338 339 340 341 342 343 344 345 346 347 348 349 350 351 352
 353 354 355 356 357 358 359 360 361 362 363 364 365 366 367 368 369 370
 371 372 373 374 375 376 377 378 379 380 381 382 383 384 385 386 387 388
 389 390 391 392 393 394 395 396 397 398 399 400 401 402 403 404 405 406
 407 408 409 410 411 412 413 414 415 416 417 418 419 420 421 422 423 424
 425 426 427 428 429 430 431 432 433 434 435 436 437 438 439 440 441 442
 443 444 445 446 447 448 449 450 451 452 453 454 455 456 457 458 459 460
 461 462 463 464 465 466 467 468 469 470 471 472 473 474 475 476 477 478
 479 480 481 482 483 484 485 486 487 488 489 490 491 492 493 494 495 496
 497 498 499 500 501 502 503 504 505 506 507 508 509 510 511 512 513 514
 515 516 517 518 519 520 521 522 523 524 525 526 527 528 529 530 531 532
 533 534 535 536 537 538 539 540 541 542 543 544 545 546 547 548 549 550
 551 552 553 554 555 556 557 558 559 560 561 562 563 564 565 566 567 568
 569 570 571 572 573 574 575 576 577 578 579 580 581 582 583 584 585 586
 587 588 589 590 591 592 593 594 595 596 597 598 599 600 601 602 603 604
 605 606 607 608 609 610 611 612 613 614 615 616 617 618 619 620 621 622
 623 624 625 626 627 628 629 630 631 632 633 634 635 636 637 638 639 640
 641 642 643 644 645 646 647 648 649 650 651 652 653 654 655 656 657 658
 659 660 661 662 663 664 665 666 667 668 669 670 671 672 673 674 675 676
 677 678 679 680 681 682]
```

Figure 3.4: Splitted result and first train data index

3.2 Least Mean Squared Error

1. Training stage: Augment the feature vector x with an additional constant dimension Code as shown below:

```
data, label =data_shuffled[columnNames[1:10]], data_shuffled[columnNames[10]]
data['Extra features'] = 1
data.head(10)
```

Result:

	Clump Thickness	Uniformity of Cell Size	Uniformity of Cell Shape	Marginal Adhesion	Single Epithelial Cell Size	Bare Nuclei	Bland Chromatin	Normal Nucleoli	Mitoses	Extra features
0	4	2	2	1	2	1	2	1	1	1
1	3	1	1	1	1	1	2	1	1	1
2	3	1	1	1	2	1	2	1	1	1
3	5	4	6	7	9	7	8	10	1	1
4	5	3	2	8	5	10	8	1	2	1
5	7	6	6	3	2	10	7	1	1	1
6	10	4	5	5	5	10	4	1	1	1
7	10	10	10	3	10	10	9	10	1	1
8	3	1	1	1	2	1	3	1	1	1
9	10	4	3	2	3	10	5	3	2	1

Figure 3.5: New train data

In this step, I first extracted 9 original features as the data set, class as the label set, and then added an additional feature constant 1 at the end of the data set.

2. Scale linearly the attribute values x_{ij} into $[-1, 1]$ for each dimensional feature as follows:

$$x_{ij} \leftarrow 2\frac{x_{ij} - min_i x_{ij} + 10^{-6}}{max_i - min_i x_{ij} + 10^{-6}} - 1$$

Function code as shown below:

```
def Scale(data):
        arr =  np.array(data)
        new_data = []
        for i in range(len(arr)):
                tmp = []
                for j in range(len(arr[i])):
                tmp.append(2*((arr[i][j]-np.min(arr[i])*arr[i][j]+1E-6)/(np.max(
                    arr[i])-np.min(arr[i])*arr[i][j]+1E-6))-1)
        new_data.append(tmp)
        return new_data
```

```
arr = Scale(data_list)
print(data_list[0])
print(arr[0])
```

Result:

```
[4, 2, 2, 1, 2, 1, 2, 1, 1, 1]
[1.0, -0.9999990000005, -0.9999990000005, -0.9999993333335555, -0.9999990000005, -0.9999993333335555, -0.9999990000005, -0.99
99993333335555, -0.9999993333335555, -0.9999993333335555]
```

Figure 3.6: Scale result

It can be seen from the above results that the function can change the maximum value in a set of numbers to 1, and the remaining values to values close to -1.

3. Reset the example vector x according its label y

$$x \leftarrow \begin{cases} x, & y \in \omega_1 \\ -x, & y \in \omega_2 \end{cases}$$

In this case:

$$\omega_1 = 2$$

$$\omega_2 = 4$$

Code as shown below:

```
def reset(data,label,num):
        data_new=[]
        for i in range(len(label)):
                if label[i] == num:
                        tmp = []
                        for j in range(len(data[i])):
                                tmp.append(data[i][j]*-1)
                        data_new.append(tmp)
                else:
                        tmp = []
                        for j in range(len(data[i])):
                                tmp.append(data[i][j])
                        data_new.append(tmp)
        return data_new
```

```
arr_rs  = reset(arr,label_list,4)
for i in range(5):
        print('Label is {}\nOrignal data is {}\nReseted data is {}'.format(
            label_list[i],arr[i],arr_rs[i]))
```

Result:

```
Label is 2
Orignal data is [1.0, -0.9999990000005, -0.9999990000005, -0.9999993333335555, -0.9999990000005, -0.9999993333335555, -0.9999
990000005, -0.9999993333335555, -0.9999993333335555, -0.9999993333335555]
Reseted data is [1.0, -0.9999990000005, -0.9999990000005, -0.9999993333335555, -0.9999990000005, -0.9999993333335555, -0.9999
990000005, -0.9999993333335555, -0.9999993333335555, -0.9999993333335555]
Label is 2
Orignal data is [1.0, -0.9999990000005, -0.9999990000005, -0.9999990000005, -0.9999990000005, -0.9999990000005, -0.9999980000
02, -0.9999990000005, -0.9999990000005, -0.9999990000005]
Reseted data is [1.0, -0.9999990000005, -0.9999990000005, -0.9999990000005, -0.9999990000005, -0.9999990000005, -0.9999980000
02, -0.9999990000005, -0.9999990000005, -0.9999990000005]
Label is 2
Orignal data is [1.0, -0.9999990000005, -0.9999990000005, -0.9999990000005, -0.999998000002, -0.9999990000005, -0.99999800000
2, -0.9999990000005, -0.9999990000005, -0.9999990000005]
Reseted data is [1.0, -0.9999990000005, -0.9999990000005, -0.9999990000005, -0.999998000002, -0.9999990000005, -0.99999800000
2, -0.9999990000005, -0.9999990000005, -0.9999990000005]
Label is 4
Orignal data is [-0.99999960000008, -0.9999996666667222, -0.999999500000125, -0.9999993333335555, -0.999998000002, -0.9999993
333335555, -0.9999990000005, 1.0, -0.9999997777778025, -0.9999997777778025]
Reseted data is [0.99999960000008, 0.9999996666667222, 0.999999500000125, 0.9999993333335555, 0.999998000002, 0.9999993333335
555, 0.9999990000005, -1.0, 0.9999997777778025, 0.9999997777778025]
Label is 4
Orignal data is [-0.99999960000008, -0.9999997142857551, -0.9999997500000313, -0.9999990000005, -0.99999960000008, 1.0, -0.99
999990000005, -0.9999997777778025, -0.9999997500000313, -0.9999997777778025]
Reseted data is [0.99999960000008, 0.9999997142857551, 0.9999997500000313, 0.9999990000005, 0.99999960000008, -1.0, 0.9999990
000005, 0.9999997777778025, 0.9999997500000313, 0.9999997777778025]
```

Figure 3.7: Reset function test result

4. Find the weight vector ω

$$\omega \leftarrow \left(X^T X\right)^{-1} X^T y$$

Here, the least square method is used to obtain the feature parameter ω. In the formula, X is the feature matrix and y is the label matrix. Code as shown below:

```
def find_wv(X,y):
        X_T =np.transpose(X)
        w = np.dot(np.dot(np.linalg.inv(np.dot(X_T,X)),X_T),y)
        return w
```

```
w = find_wv(arr_rs,label_list)
print(w)
```

Result:

```
[ 0.1804811  -0.87740383 -0.00652876 -0.2380186   0.23179818 -1.61842745
  0.50557268 -1.0195706  -0.47094499  2.59043216]
```

Figure 3.8: weight vector ω

5. Predict the example x

$$c =\leftarrow \begin{cases} \omega_1, & \omega^T \begin{bmatrix} x \\ 1 \end{bmatrix} \geq 0 \\ \omega_2, & \text{otherwise} \end{cases}$$

Code as shown below:

```
def predict_label(w,x):
        w_T = np.transpose(w)
        label_pre = []
        for i in range(len(x)):
                if np.dot(w_T,np.transpose(x[i])) >= 0:
                        label_pre.append(2)
                else:
                        label_pre.append(4)
        return label_pre
```

```
label_pre = predict_label(w,data_list)
print(label_pre[:20])
print(label_list[:20])
```

Result:

```
[4, 2, 2, 4, 4, 4, 4, 4, 2, 4, 4, 4, 2, 2, 2, 4, 2, 2, 4, 2]
[2, 2, 2, 4, 4, 4, 4, 4, 2, 4, 2, 2, 2, 2, 2, 4, 2, 2, 4, 2]
```

Figure 3.9: Top20 of ture label and predicted label

After the trained parameter ω is used to predict the label, the accuracy of the comparison result is still relatively high, because the training set and the prediction set are consistent.

6. Test stage :
   Estimate the accuracy on the test set, that is the fraction of the correctly classified examples in the total test set.
   By calculating the same number of predicted labels and actual labels, and then removing the total number to get the accurate prediction value.
   Code as shown below:

```
def acc(predict,original):
        if len(predict)!=len(original):
                print('Error:The length of the predicted value does not match the
                    length of the original value!')
        right = 0
        for i in range(len(predict)):
                if predict[i]==original[i]:
                        right+=1
        return right/len(predict)
```

```
print(acc(label_pre,label_list))
```

   Result: Test accuracy result is 0.6852122986822841

7. Repeat 5-cross-fold validation for 5 times to average all accuracy performance
   Code as shown below:

```
import matplotlib.pyplot as plt
import matplotlib.ticker as ticker
from sklearn.model_selection import KFold
mean_Acc = []
plt.figure(figsize=(16,20))
plt.subplots_adjust(wspace =0.2, hspace =0.2)
x = range(1,6)
for z in x:
        num = np.random.randint(1,1000)
        kf = KFold(n_splits=5,shuffle=True,random_state=num)
        data_train_index,data_test_index = [],[]
        for train_index, test_index in kf.split(data):
                data_train_index.append(train_index)
                data_test_index.append(test_index)
                data_train, label_train = [], []
                data_test,label_test = [],[]
        for i in range(5):
                temp = []
                for j in range(len(data_train_index[i])):
                        temp.append(data_list[data_train_index[i][j]])
                data_train.append(temp)
                temp = []
                for j in range(len(data_train_index[i])):
                        temp.append(label_list[data_train_index[i][j]])
                label_train.append(temp)

                temp = []
                for k in range(len(data_test_index[i])):
                        temp.append(data_list[data_test_index[i][k]])
                data_test.append(temp)
```

```
                temp = []
                for k in range(len(data_test_index[i])):
                        temp.append(label_list[data_test_index[i][k]])
                label_test.append(temp)
        Acc = []
        for j in x:
                X = Scale(data_train[j-1])
                y = label_train[j-1]
                X = reset(X,y,4)
                w = find_wv(X,y)
                y_pre = predict_label(w,data_test[j-1])
                Acc.append(acc(y_pre,label_test[j-1]))
        plt.subplot(3,2,z)
        plt.plot(x,Acc,'-.o')
        def to_percent(temp, position):
                return '%1.0f'%(100*temp) + '%'
        plt.gca().yaxis.set_major_formatter(ticker.FuncFormatter(to_percent))
        plt.xlabel('No.K-Flod')
        plt.ylabel('Accuarcy')
        plt.title('The mean of five accuracies is {:.2%}(random_seed={})'.format(
            np.mean(np.array(Acc)),random_seed))
        mean_Acc.append(np.mean(np.array(Acc)))
plt.subplot(3,2,6)
plt.plot(x,mean_Acc,'-.o')
plt.gca().yaxis.set_major_formatter(ticker.FuncFormatter(to_percent))
plt.ylim([0.6,0.7])
plt.xlabel('K-Flod by different random seed')
plt.ylabel('Accuarcy')
plt.title('The mean of five mean_Acc is {:.2%}'.format(np.mean(mean_Acc)))
```

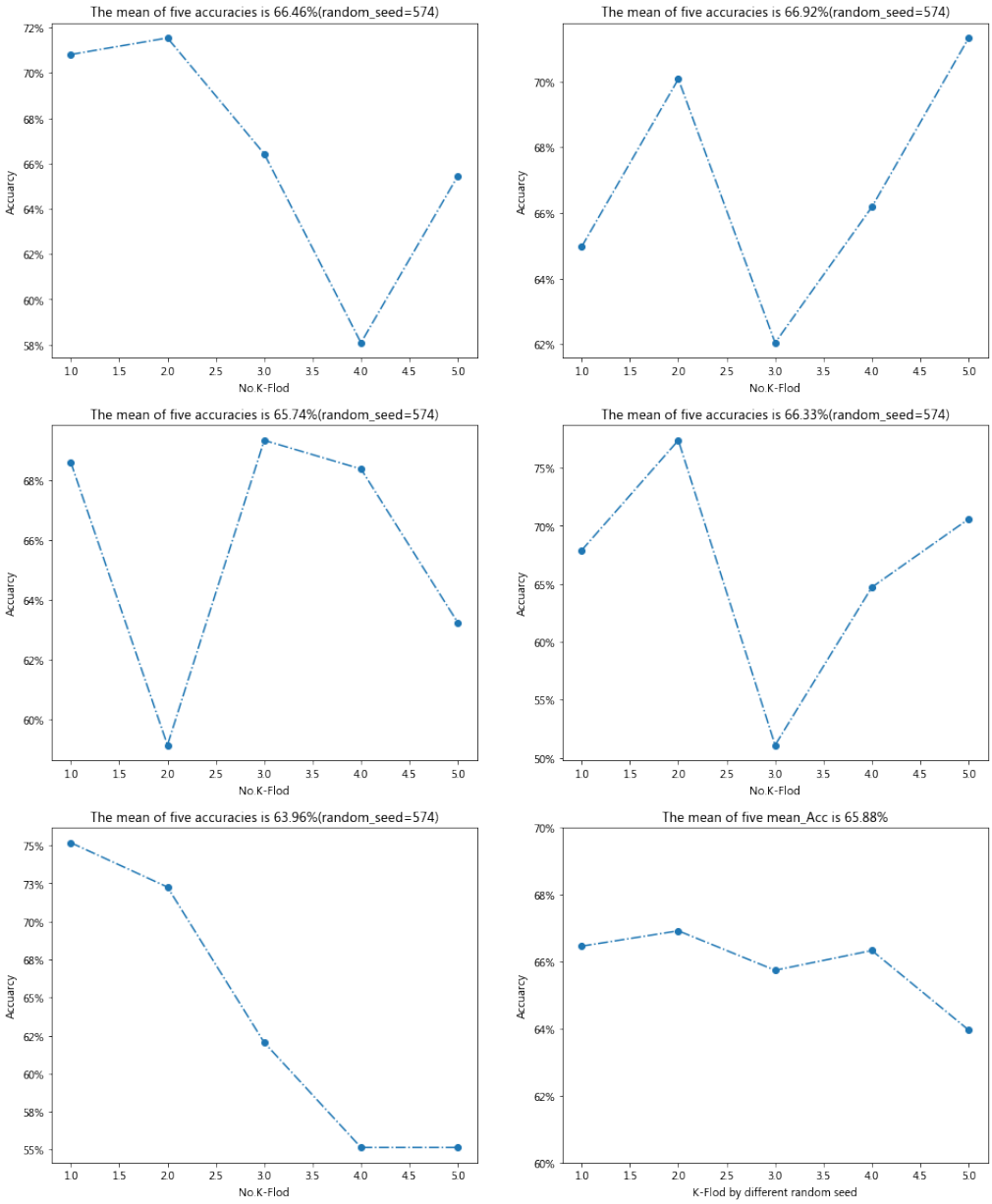

Figure 3.10: 5 times 5-cross-fold validation result

As can be seen from the above figure, subgraphs 1 to 5 are the results of five accuracy rates obtained by using different random seeds for K-Flod splitting, and subgraph 6 is the distribution of the above five average accuracy rates. The final average accuracy rate is 65.88%. Observing the above image, you can find that the accurate floats are all in [55%, 80%], there is no excessive difference, and it is relatively stable.

3.3 Conclusion

- Through the cross-fold validation method, you can avoid the interference of the special data set with the result, and you can get the true and accurate result output through multiple verification.

- Through the LMSE algorithm, the final prediction result is only 65.88%. This result is actually not good, but because the algorithm is a linear problem, the classification effect will be very limited for the adjacent data sets. For data preprocessing, only the largest number in each set of data features is set to 1, and the remaining decimals are set to -1. Therefore, fewer features are considered, which also leads to poor experimental results.