

BIG HOMEWORK

The dataset I selected is publicly available, has various data types (**numeric, categorical, graphical**, etc.) and has hundreds of rows.

The datasets are:

The Boston Housing Dataset

Congressional Voting Record

MNIST

1. The Boston Housing Dataset

This dataset type is **numeric**

Data introduction:

- CRIM - per capita crime rate by town
- ZN - proportion of residential land zoned for lots over 25,000 sq.ft.
- INDUS - proportion of non-retail business acres per town
- CHAS - Charles River dummy variable (1 if tract bounds river; 0 otherwise)
- NOX - nitric oxides concentration (parts per 10 million)
- RM - average number of rooms per dwelling
- AGE - proportion of owner-occupied units built prior to 1940
- DIS - weighted distances to five Boston employment centres
- RAD - index of accessibility to radial highways
- TAX - full-value property-tax rate per \$10,000
- PTRATIO - pupil-teacher ratio by town
- B - $1000(B_k - 0.63)^2$ where B_k is the proportion of blacks by town
- LSTAT - % lower status of the population

binarization strategy:

We first calculate the average price of house prices

Then we divide the prices into high and low (1 and 0)

CRIM	ZN	INDUS	CHAS	NOX	RM	AGE	DIS	RAD	TAX	PIRATIO	B	LSTAT	Degree
0.00632	18	2.31	0	0.538	6.575	65.2	4.09	1	296	15.3	396.9	4.98	1
0.02731	0	7.07	0	0.469	6.421	78.9	4.9671	2	242	17.8	396.9	9.14	0
0.02729	0	7.07	0	0.469	7.185	61.1	4.9671	2	242	17.8	392.83	4.03	1
0.03237	0	2.18	0	0.458	6.998	45.8	6.0622	3	222	18.7	394.63	2.94	1
0.17004	12.5	7.87	0	0.524	6.004	85.9	6.5921	5	311	15.2	386.71	17.1	0

Then we use the **lazyfca** method to process the data

Finally we use **Kfold cross-validation** and then adjust and find the **best alpha value**

Unadjusted alpha code and accuracy:

```
In [3]: import fcalc
...: import pandas as pd
...: import numpy as np
...: from sklearn.model_selection import train_test_split

In [4]: df = pd.read_csv('data_sets/boston.csv',
...: names=['CRIM','ZN','INDUS','CHAS','NOX','RM','AGE','DIS','RAD','TAX','PIRATIO','B','LSTAT','Degree'])
...: df['Degree'] = [x == '1' for x in df['Degree']]
...: df.sample(10)

Out[4]:
   CRIM  ZN  INDUS  CHAS  NOX   RM  AGE   DIS  RAD  TAX  PIRATIO  \
160  0.06588  0  2.46  0  0.488  7.765  83.3  2.741  3  193  17.8
157  0.05425  0  4.05  0  0.51  6.315  73.4  3.3175  5  296  16.6
407  9.51363  0  18.1  0  0.713  6.728  94.1  2.4961  24  666  20.2
324  9.2323  0  18.1  0  0.631  6.216  100  1.1691  24  666  20.2
224  0.19073  22  5.86  0  0.431  6.718  17.5  7.8265  7  330  19.1
114  0.97617  0  21.89  0  0.624  5.757  98.4  2.346  4  437  21.2
174  0.01439  60  2.93  0  0.401  6.604  18.8  6.2196  1  265  15.6
389  14.4208  0  18.1  0  0.74  6.461  93.3  2.0026  24  666  20.2
414  3.69311  0  18.1  0  0.713  6.376  88.4  2.5671  24  666  20.2
168  0.12579  45  3.44  0  0.437  6.556  29.1  4.5667  5  398  15.2

   B  LSTAT  Degree
160  395.56  7.56  True
157  395.6  6.29  True
407  6.68  18.71  False
324  366.15  9.53  True
224  393.74  6.56  True
114  262.76  17.31  False
174  376.7  4.38  True
389  27.49  18.05  False
414  391.43  14.65  False
168  382.84  4.56  True

In [5]: X = df.iloc[:, :-1]
...: y = df['Degree']
...: X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42)

In [6]: pat_cls = fcalc.classifier.PatternBinaryClassifier(X_train.values, y_train.to_numpy())

In [7]: pat_cls.predict(X_test.values)

In [8]: from sklearn.metrics import accuracy_score, f1_score
...: print("accuracy:", round(accuracy_score(y_test, pat_cls.predictions), 4))
...: print("f1 score:", round(f1_score(y_test, pat_cls.predictions), 4))
accuracy: 0.6423
```

Accuracy: 0.6423

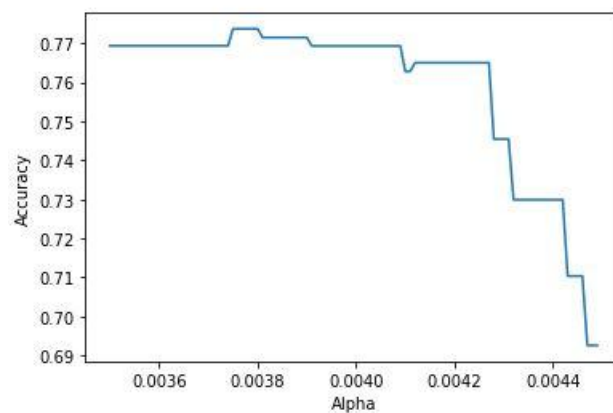
Test optimal alpha parameters:

step1: start, step2: end, step3:length

```
a=[]
b=[]
for j in np.arange(step1, step2, step3):
    sum = 0.0
    for i, (train_index, test_index) in enumerate(kf.split(X,y)):
        print(f"Fold {i}:")
        X_train=X.iloc[train_index]
        y_train=y.iloc[train_index]
        X_test=X.iloc[test_index]
        y_test=y.iloc[test_index]
        pat_cls = fcalc.classifier.PatternBinaryClassifier(X_train.values, y_train.to_numpy(), alpha=j)
        pat_cls.predict(X_test.values)
        acc=accuracy_score(y_test, pat_cls.predictions)
        print(acc)
        sum += acc
    print(f"alpha={j}: average_accuracy={sum/10}")
    num = sum/10
    a.append(j)
    b.append(num)

import matplotlib.pyplot as plt
plt.plot(a, b)
#添加坐标轴标签
plt.xlabel('Alpha')
plt.ylabel('Accuracy')
#显示图形
plt.show()
```

Result graph:



Decision tree:

```
11 # 将房价转化为分类标签
12 data['Price_Category'] = pd.cut(data['PRICE'], bins=[0, data['PRICE'].median(), data['PRICE'].max()])
13
14 # 划分特征和目标变量
15 X = data.drop(['PRICE', 'Price_Category'], axis=1)
16 y = data['Price_Category']
17
18 # 将目标变量编码为数字
19 y = pd.factorize(y)[0]
20
21 # 划分数据集
22 X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
23
24 # 使用 Decision Tree Classifier
25 dt_classifier = DecisionTreeClassifier()
26
27 # 定义 F1 分数作为评估指标
28 f1_scorer = make_scorer(f1_score)
29
30 # 交叉验证调整模型参数并评估 F1 分数
31 cv_scores = cross_val_score(dt_classifier, X_train, y_train, cv=5, scoring=f1_scorer)
32
33 # 输出交叉验证的 F1 分数
34 print("Cross-Validation F1 Scores:", cv_scores)
35
36 # 输出平均 F1 分数
37 print("Average F1 Score:", np.mean(cv_scores))
38
39 # 训练模型
40 dt_classifier.fit(X_train, y_train)
41
42 # 在数据集上进行预测
43 y_pred = dt_classifier.predict(X_test)
44
45 # 输出最准确的分类结果
46 print("Most Accurate Predictions:")
47 print(pd.DataFrame({'Actual': y_test, 'Predicted': y_pred}).head(10))
```

Accuracy: 0.7362

Random forest:

```
11 # 将房价转化为分类标签
12 data['Price_Category'] = pd.cut(data['PRICE'], bins=[0, data['PRICE'].median(), data['PRICE'].max()])
13
14 # 划分特征和目标变量
15 X = data.drop(['PRICE', 'Price_Category'], axis=1)
16 y = data['Price_Category']
17
18 # 将目标变量编码为数字
19 y = pd.factorize(y)[0]
20
21 # 划分数据集
22 X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
23
24 # 使用 Random Forest Classifier
25 rf_classifier = RandomForestClassifier(n_estimators=100, random_state=42)
26
27 # 定义 F1 分数作为评估指标
28 f1_scorer = make_scorer(f1_score)
29
30 # 交叉验证调整模型参数并评估 F1 分数
31 cv_scores = cross_val_score(rf_classifier, X_train, y_train, cv=5, scoring=f1_scorer)
32
33 # 输出交叉验证的 F1 分数
34 print("Cross-Validation F1 Scores:", cv_scores)
35
36 # 输出平均 F1 分数
37 print("Average F1 Score:", np.mean(cv_scores))
38
39 # 训练模型
40 rf_classifier.fit(X_train, y_train)
41
42 # 在数据集上进行预测
43 y_pred = rf_classifier.predict(X_test)
44
45 # 输出最准确的分类结果
46 print("Most Accurate Predictions:")
47 print(pd.DataFrame({'Actual': y_test, 'Predicted': y_pred}).head(10))
```

Accuracy: 0.8241

xGboost:

```
11 # 将房价转化为分类标签
12 data['Price_Category'] = pd.cut(data['PRICE'], bins=[0, data['PRICE'].median(), data['PRICE'].max()
13
14 # 划分特征和目标变量
15 X = data.drop(['PRICE', 'Price_Category'], axis=1)
16 y = data['Price_Category']
17
18 # 将目标变量编码为数字
19 y = pd.factorize(y)[0]
20
21 # 划分数据集
22 X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
23
24 # 使用 XGBoost Classifier
25 xgb_classifier = XGBClassifier(random_state=42)
26
27 # 定义 F1 分数作为评估指标
28 f1_scorer = make_scorer(f1_score)
29
30 # 交叉验证调整模型参数并评估 F1 分数
31 cv_scores = cross_val_score(xgb_classifier, X_train, y_train, cv=5, scoring=f1_scorer)
32
33 # 输出交叉验证的 F1 分数
34 print("Cross-Validation F1 Scores:", cv_scores)
35
36 # 输出平均 F1 分数
37 print("Average F1 Score:", np.mean(cv_scores))
38
39 # 训练模型
40 xgb_classifier.fit(X_train, y_train)
41
42 # 在数据集上进行预测
43 y_pred = xgb_classifier.predict(X_test)
44
45 # 输出最准确的分类结果
46 print("Most Accurate Predictions:")
47 print(pd.DataFrame({'Actual': y_test, 'Predicted': y_pred}).head(10))
```

Accuracy: 0.8333

k-NN:

```
11 # 将房价转化为分类标签
12 data['Price_Category'] = pd.cut(data['PRICE'], bins=[0, data['PRICE'].median(), data['PRICE'].max()])
13
14 # 划分特征和目标变量
15 X = data.drop(['PRICE', 'Price_Category'], axis=1)
16 y = data['Price_Category']
17
18 # 将目标变量编码为数字
19 y = pd.factorize(y)[0]
20
21 # 划分数据集
22 X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
23
24 # 使用 k-NN 分类器
25 knn_classifier = KNeighborsClassifier()
26
27 # 定义 F1 分数作为评估指标
28 f1_scorer = make_scorer(f1_score)
29
30 # 交叉验证调整模型参数并评估 F1 分数
31 cv_scores = cross_val_score(knn_classifier, X_train, y_train, cv=5, scoring=f1_scorer)
32
33 # 打印交叉验证的 F1 分数
34 print("Cross-Validation F1 Scores:", cv_scores)
35
36 # 打印平均 F1 分数
37 print("Average F1 Score:", np.mean(cv_scores))
38
39 # 训练模型
40 knn_classifier.fit(X_train, y_train)
41
42 # 在数据集上进行预测
43 y_pred = knn_classifier.predict(X_test)
44
45 # 打印最准确的分类结果
46 print("Most Accurate Predictions:")
47 print(pd.DataFrame({'Actual': y_test, 'Predicted': y_pred}).head(10))
```

Accuracy: 0.7252

Naive Bayes:

```
11 # 将房价转化为分类标签
12 data['Price_Category'] = pd.cut(data['PRICE'], bins=[0, data['PRICE'].median(), data['PRICE'].max()])
13
14 # 划分特征和目标变量
15 X = data.drop(['PRICE', 'Price_Category'], axis=1)
16 y = data['Price_Category']
17
18 # 将目标变量编码为数字
19 y = pd.factorize(y)[0]
20
21 # 划分数据集
22 X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
23
24 # 使用 Naive Bayes 分类器
25 nb_classifier = GaussianNB()
26
27 # 定义 F1 分数作为评估指标
28 f1_scorer = make_scorer(f1_score)
29
30 # 交叉验证调整模型参数并评估 F1 分数
31 cv_scores = cross_val_score(nb_classifier, X_train, y_train, cv=5, scoring=f1_scorer)
32
33 # 打印交叉验证的 F1 分数
34 print("Cross-Validation F1 Scores:", cv_scores)
35
36 # 打印平均 F1 分数
37 print("Average F1 Score:", np.mean(cv_scores))
38
39 # 训练模型
40 nb_classifier.fit(X_train, y_train)
41
42 # 在数据集上进行预测
43 y_pred = nb_classifier.predict(X_test)
44
45 # 打印最准确的分类结果
46 print("Most Accurate Predictions:")
47 print(pd.DataFrame({'Actual': y_test, 'Predicted': y_pred}).head(10))
```

Accuracy: 0.7472

logistic regression:

```
11 # 将房价转化为分类标签
12 data['Price_Category'] = pd.cut(data['PRICE'], bins=[0, data['PRICE'].median(), data['PRICE'].max()
13
14 # 划分特征和目标变量
15 X = data.drop(['PRICE', 'Price_Category'], axis=1)
16 y = data['Price_Category']
17
18 # 将目标变量编码为数字
19 y = pd.factorize(y)[0]
20
21 # 划分数据集
22 X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
23
24 # 使用 Logistic Regression 分类器
25 lr_classifier = LogisticRegression()
26
27 # 定义 F1 分数作为评估指标
28 f1_scorer = make_scorer(f1_score)
29
30 # 交叉验证调整模型参数并评估 F1 分数
31 cv_scores = cross_val_score(lr_classifier, X_train, y_train, cv=5, scoring=f1_scorer)
32
33 # 打印交叉验证的 F1 分数
34 print("Cross-Validation F1 Scores:", cv_scores)
35
36 # 打印平均 F1 分数
37 print("Average F1 Score:", np.mean(cv_scores))
38
39 # 训练模型
40 lr_classifier.fit(X_train, y_train)
41
42 # 在数据集上进行预测
43 y_pred = lr_classifier.predict(X_test)
44
45 # 打印最准确的分类结果
46 print("Most Accurate Predictions:")
47 print(pd.DataFrame({'Actual': y_test, 'Predicted': y_pred}).head(10))
```

Accuracy: 0.8461

2. Congressional Voting Record

This dataset type is **categorical**

Data introduction:

- Class Name: 2 (democrat, republican)
- handicapped-infants: 2 (y,n)
- water-project-cost-sharing: 2 (y,n)
- adoption-of-the-budget-resolution: 2 (y,n)
- physician-fee-freeze: 2 (y,n)
- el-salvador-aid: 2 (y,n)
- religious-groups-in-schools: 2 (y,n)
- anti-satellite-test-ban: 2 (y,n)
- aid-to-nicaraguan-contras: 2 (y,n)
- mx-missile: 2 (y,n)
- immigration: 2 (y,n)
- synfuels-corporation-cutback: 2 (y,n)
- education-spending: 2 (y,n)
- superfund-right-to-sue: 2 (y,n)
- crime: 2 (y,n)
- duty-free-exports: 2 (y,n)
- export-administration-act-south-africa: 2 (y,n)

binarization strategy:

From data observation we can know that voting is divided into two categories

We first deal with the one-hot method

Then we can get the relationship between the two voting

	handicap	water-prc	adoption	physician	el-salvac	religious	anti-sat	aid-to-n	mx-missile	immigrati	synfuels	education	superfund	crime	duty-free	export-ac	Class
2	n	y	n	y	y	y	n	n	n	y	?	y	y	y	n	y	republican
3	n	y	n	y	y	y	n	n	n	n	n	y	y	y	n	?	republican
4	?	y	y	?	y	y	n	n	n	n	y	n	y	y	n	n	democrat
5	n	y	y	n	?	y	n	n	n	n	y	n	y	n	n	y	democrat

Then we use the **lazyfca** method to process the data

Finally we use **Kfold cross-validation** and then adjust and find the **best alpha value**

Unadjusted alpha code and accuracy:

```
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
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
683
684
685
686
687
688
689
690
691
692
693
694
695
696
697
698
699
700
701
702
703
704
705
706
707
708
709
710
711
712
713
714
715
716
717
718
719
720
721
722
723
724
725
726
727
728
729
730
731
732
733
734
735
736
737
738
739
740
741
742
743
744
745
746
747
748
749
750
751
752
753
754
755
756
757
758
759
760
761
762
763
764
765
766
767
768
769
770
771
772
773
774
775
776
777
778
779
780
781
782
783
784
785
786
787
788
789
790
791
792
793
794
795
796
797
798
799
800
801
802
803
804
805
806
807
808
809
810
811
812
813
814
815
816
817
818
819
820
821
822
823
824
825
826
827
828
829
830
831
832
833
834
835
836
837
838
839
840
841
842
843
844
845
846
847
848
849
850
851
852
853
854
855
856
857
858
859
860
861
862
863
864
865
866
867
868
869
870
871
872
873
874
875
876
877
878
879
880
881
882
883
884
885
886
887
888
889
890
891
892
893
894
895
896
897
898
899
900
901
902
903
904
905
906
907
908
909
910
911
912
913
914
915
916
917
918
919
920
921
922
923
924
925
926
927
928
929
930
931
932
933
934
935
936
937
938
939
940
941
942
943
944
945
946
947
948
949
950
951
952
953
954
955
956
957
958
959
960
961
962
963
964
965
966
967
968
969
970
971
972
973
974
975
976
977
978
979
980
981
982
983
984
985
986
987
988
989
990
991
992
993
994
995
996
997
998
999
1000
```

Accuracy: 0.9312

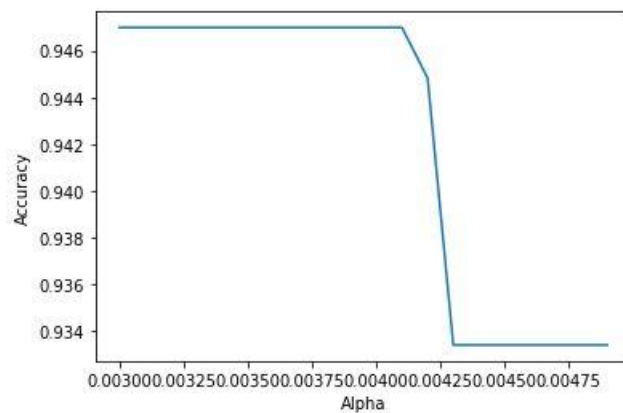
Test optimal alpha parameters:

step1: start, step2: end, step3:length

```
a=[]
b=[]
for j in np.arange(step1, step2, step3):
    sum = 0.0
    for i, (train_index, test_index) in enumerate(kf.split(X,y)):
        print(f"Fold {i}:")
        X_train=X.iloc[train_index]
        y_train=y.iloc[train_index]
        X_test=X.iloc[test_index]
        y_test=y.iloc[test_index]
        pat_cls = fcalcclassifier.PatternBinaryClassifier(X_train.values, y_train.to_numpy(), alpha=j)
        pat_cls.predict(X_test.values)
        acc=accuracy_score(y_test, pat_cls.predictions)
        print(acc)
        sum += acc
    print(f"alpha={j}: average_accuracy={sum/10}")
    num = sum/10
    a.append(j)
    b.append(num)

import matplotlib.pyplot as plt
plt.plot(a, b)
#添加坐标轴标签
plt.xlabel('Alpha')
plt.ylabel('Accuracy')
#显示图形
plt.show()
```

Result graph:



Accuracy:

Decision tree: 0.8870

Random forest: 0.9357

xGboost: 0.9265

k-NN: 0.9325

Naive Bayes: 0.9080

logistic regression: 0.9271

3. MNIST

This dataset type is **graphical**

Data introduction:

The MNIST database is a large database of handwritten digits commonly used to train various image processing systems. The database is also widely used for training and testing in the field of machine learning.



Data preprocessing:

Compared to the previous two data sets, we first need to process these images

We need to change the data into csv format

#	R2	R3	R4	R5	R6	R7	R8	R9	R10	R11	R12	R13	R14	R15	R16	R17	R18	R19	R20	R21	R22	R23	R24	R25	R26	R27	R28	R29	R30	R31	R32	R33	R34	R35	R36	R37	R38	R39	R40	R41	R42	R43	R44	R45	R46	R47	R48	R49	R50																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																
1	18x10	18x11	18x12	18x13	18x14	18x15	18x16	18x17	18x18	18x19	18x20	18x21	18x22	18x23	18x24	18x25	18x26	18x27	18x28	19x1	19x2	19x3	19x4	19x5																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																									

Class: (number)

ADE
Class
7
2
1
0
4
1
4
9

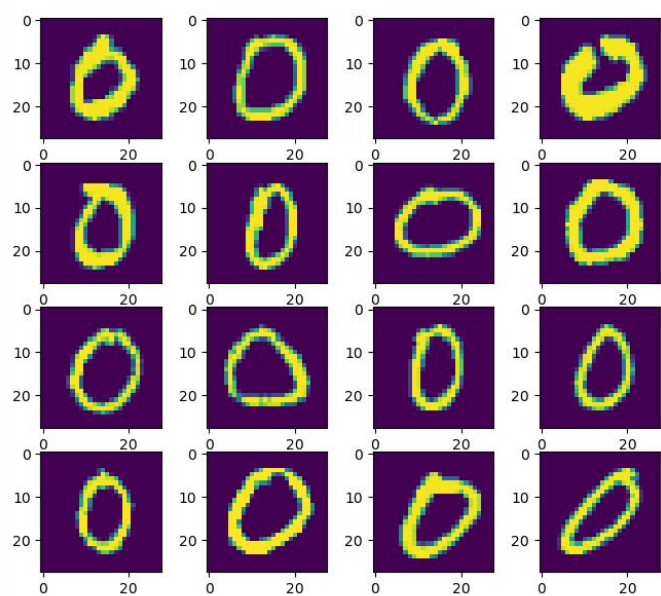
Split the dataset: (2000 rows)

```
In [7]: file = open("data_sets/mnist.csv")
In [8]: data_train = pd.read_csv(file)
In [9]: data_train = data_train.head(2000)
```

Show all pictures under data:

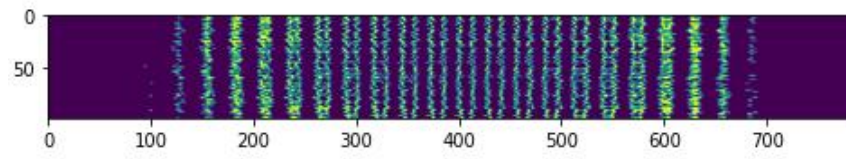
```
In [16]: def show_img(x):
...:     plt.figure(figsize=(8,7))
...:     if x.shape[0] > 100:
...:         print(x.shape[0])
...:         n_imgs = 16
...:         n_samples = x.shape[0]
...:         x = x.reshape(n_samples, size_img, size_img)
...:         for i in range(16):
...:             plt.subplot(4, 4, i+1) #devide figure into 4x4 and choose i+1 to draw
...:             plt.imshow(x[i])
...:             plt.show()
...:     else:
...:         plt.imshow(x)
...:         plt.show()
...:

In [17]: show_img(x_train)
2000
```



Show x_test pictures under data:

```
In [21]: show_img(x_test)
```



Finally, we turned the image into matrix information through preprocessing and saved it in csv

binarization strategy:

We perform two classifications based on the image labels of the csv

We classify based on numbers, for example: 0 and non-0

Then we use the **lazyfca method** to process the data

Finally we use **Kfold cross-validation** and then adjust and find **the best alpha value**

```
In [12]: df = df.head(2000)
```

```
In [13]: print(df)
```

	1x1	1x2	1x3	1x4	1x5	1x6	1x7	1x8	1x9	1x10	...	28x20	28x21	\
0	0	0	0	0	0	0	0	0	0	0	...	0	0	
1	0	0	0	0	0	0	0	0	0	0	...	0	0	
2	0	0	0	0	0	0	0	0	0	0	...	0	0	
3	0	0	0	0	0	0	0	0	0	0	...	0	0	
4	0	0	0	0	0	0	0	0	0	0	...	0	0	
...	
1995	0	0	0	0	0	0	0	0	0	0	...	0	0	
1996	0	0	0	0	0	0	0	0	0	0	...	0	0	
1997	0	0	0	0	0	0	0	0	0	0	...	0	0	
1998	0	0	0	0	0	0	0	0	0	0	...	0	0	
1999	0	0	0	0	0	0	0	0	0	0	...	0	0	

	28x22	28x23	28x24	28x25	28x26	28x27	28x28	Class
0	0	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0	0
2	0	0	0	0	0	0	0	0
3	0	0	0	0	0	0	0	0
4	0	0	0	0	0	0	0	0
...
1995	0	0	0	0	0	0	0	1
1996	0	0	0	0	0	0	0	1
1997	0	0	0	0	0	0	0	1
1998	0	0	0	0	0	0	0	1
1999	0	0	0	0	0	0	0	1

[2000 rows x 785 columns]

Unadjusted alpha code and accuracy:

```
In [15]: X = df.iloc[:, :-1]
```

```
In [16]: y = df['Class']
```

```
In [17]: X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42)
```

```
In [18]: pat_cls = fcalc.classifier.PatternBinaryClassifier(X_train.values, y_train.to_numpy())
```

```
In [19]: pat_cls.predict(X_test.values)
```

```
In [20]: from sklearn.metrics import accuracy_score, f1_score
...: print("accuracy:", round(accuracy_score(y_test, pat_cls.predictions), 4))
...: print("f1 score:", round(f1_score(y_test, pat_cls.predictions), 4))
```

Accuracy: 0.8127

Test optimal alpha parameters:

step1: start, step2: end, step3:length

```
a=[]
b=[]
for j in np.arange(step1, step2, step3):
    sum = 0.0
    for i, (train_index, test_index) in enumerate(kf.split(X,y)):
        print(f"Fold {i}:")
        X_train=X.iloc[train_index]
        y_train=y.iloc[train_index]
        X_test=X.iloc[test_index]
        y_test=y.iloc[test_index]
        pat_cls = fcalc.classifier.PatternBinaryClassifier(X_train.values, y_train.to_numpy(), alpha=j)
        pat_cls.predict(X_test.values)
        acc=accuracy_score(y_test, pat_cls.predictions)
        print(acc)
        sum += acc
    print(f"alpha={j}: average_accuracy={sum/10}")
    num = sum/10
    a.append(j)
    b.append(num)

import matplotlib.pyplot as plt
plt.plot(a, b)
#添加坐标轴标签
plt.xlabel('Alpha')
plt.ylabel('Accuracy')
#显示图形
plt.show()
```

Accuracy:

Decision tree: 0.8697

Random forest: 0.8823

xGboost: 0.8883

k-NN: 0.9721

Naive Bayes: 0.8386

logistic regression: 0.9175

Summarize:

Dataset	Lazy	Lazy (adjusted)	Decision tree	Random Forest	XGBoost	k-NN	Naive Bayes	Logistic
Boston	0.64	0.77	0.73	0.82	0.83	0.72	0.74	0.84
Vote	0.93	0.95	0.88	0.94	0.92	0.93	0.90	0.92
Mnist	0.81	0.85	0.87	0.88	0.89	0.96	0.83	0.91

Link:

https://github.com/LeKo888/osda_hm.git