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(Bk - 0.63)^2 where Bk 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
--------------------------------------------------------------------------
```

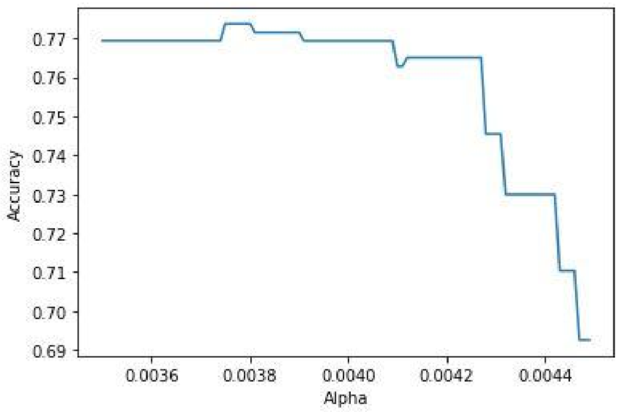
Accuracy: 0.6423

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:

```
# 将房价转化为分类标签
data['Price_Category'] = pd.cut(data['PRICE'], bins=[0, data['PRICE'].median(), data['PRICE'].max()])

# 划分特征和目标变量
X = data.drop(['PRICE', 'Price_Category'], axis=1)
y = data['Price_Category']

# 将目标变量编码为数字
y = pd.factorize(y)[0]

# 划分数据集
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# 使用 Decision Tree Classifier
dt_classifier = DecisionTreeClassifier()

# 定义 F1 分数作为评估指标
f1_scorer = make_scorer(f1_score)

# 交叉验证调整模型参数并评估 F1 分数
cv_scores = cross_val_score(dt_classifier, X_train, y_train, cv=5, scoring=f1_scorer)

# 输出交叉验证的 F1 分数
print("Cross-Validation F1 Scores:", cv_scores)

# 输出平均 F1 分数
print("Average F1 Score:", np.mean(cv_scores))

# 训练模型
dt_classifier.fit(X_train, y_train)

# 在数据集上进行预测
y_pred = dt_classifier.predict(X_test)

# 输出最准确的分类结果
print("Most Accurate Predictions:")
print(pd.DataFrame({'Actual': y_test, 'Predicted': y_pred}).head(10))
```

Accuracy: 0.7362

Random forest:

```python
# 将房价转化为分类标签
data['Price_Category'] = pd.cut(data['PRICE'], bins=[0, data['PRICE'].median(), data['PRICE'].max()])

# 划分特征和目标变量
X = data.drop(['PRICE', 'Price_Category'], axis=1)
y = data['Price_Category']

# 将目标变量编码为数字
y = pd.factorize(y)[0]

# 划分数据集
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# 使用 Random Forest Classifier
rf_classifier = RandomForestClassifier(n_estimators=100, random_state=42)

# 定义 F1 分数作为评估指标
f1_scorer = make_scorer(f1_score)

# 交叉验证调整模型参数并评估 F1 分数
cv_scores = cross_val_score(rf_classifier, X_train, y_train, cv=5, scoring=f1_scorer)

# 输出交叉验证的 F1 分数
print("Cross-Validation F1 Scores:", cv_scores)

# 输出平均 F1 分数
print("Average F1 Score:", np.mean(cv_scores))

# 训练模型
rf_classifier.fit(X_train, y_train)

# 在数据集上进行预测
y_pred = rf_classifier.predict(X_test)

# 输出最准确的分类结果
print("Most Accurate Predictions:")
print(pd.DataFrame({'Actual': y_test, 'Predicted': y_pred}).head(10))
```

Accuracy: 0.8241

xGboost:

```
# 将房价转化为分类标签
data['Price_Category'] = pd.cut(data['PRICE'], bins=[0, data['PRICE'].median(), data['PRICE'].max()

# 划分特征和目标变量
X = data.drop(['PRICE', 'Price_Category'], axis=1)
y = data['Price_Category']

# 将目标变量编码为数字
y = pd.factorize(y)[0]

# 划分数据集
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# 使用 XGBoost Classifier
xgb_classifier = XGBClassifier(random_state=42)

# 定义 F1 分数作为评估指标
f1_scorer = make_scorer(f1_score)

# 交叉验证调整模型参数并评估 F1 分数
cv_scores = cross_val_score(xgb_classifier, X_train, y_train, cv=5, scoring=f1_scorer)

# 输出交叉验证的 F1 分数
print("Cross-Validation F1 Scores:", cv_scores)

# 输出平均 F1 分数
print("Average F1 Score:", np.mean(cv_scores))

# 训练模型
xgb_classifier.fit(X_train, y_train)

# 在数据集上进行预测
y_pred = xgb_classifier.predict(X_test)

# 输出最准确的分类结果
print("Most Accurate Predictions:")
print(pd.DataFrame({'Actual': y_test, 'Predicted': y_pred}).head(10))
```

Accuracy: 0.8333

k-NN:

```python
# 将房价转化为分类标签
data['Price_Category'] = pd.cut(data['PRICE'], bins=[0, data['PRICE'].median(), data['PRICE'].max()])

# 划分特征和目标变量
X = data.drop(['PRICE', 'Price_Category'], axis=1)
y = data['Price_Category']

# 将目标变量编码为数字
y = pd.factorize(y)[0]

# 划分数据集
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# 使用 k-NN 分类器
knn_classifier = KNeighborsClassifier()

# 定义 F1 分数作为评估指标
f1_scorer = make_scorer(f1_score)

# 交叉验证调整模型参数并评估 F1 分数
cv_scores = cross_val_score(knn_classifier, X_train, y_train, cv=5, scoring=f1_scorer)

# 打印交叉验证的 F1 分数
print("Cross-Validation F1 Scores:", cv_scores)

# 打印平均 F1 分数
print("Average F1 Score:", np.mean(cv_scores))

# 训练模型
knn_classifier.fit(X_train, y_train)

# 在数据集上进行预测
y_pred = knn_classifier.predict(X_test)

# 打印最准确的分类结果
print("Most Accurate Predictions:")
print(pd.DataFrame({'Actual': y_test, 'Predicted': y_pred}).head(10))
```

Accuracy: 0.7252

Naive Bayes:

```
# 将房价转化为分类标签
data['Price_Category'] = pd.cut(data['PRICE'], bins=[0, data['PRICE'].median(), data['PRICE'].max()])

# 划分特征和目标变量
X = data.drop(['PRICE', 'Price_Category'], axis=1)
y = data['Price_Category']

# 将目标变量编码为数字
y = pd.factorize(y)[0]

# 划分数据集
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# 使用 Naive Bayes 分类器
nb_classifier = GaussianNB()

# 定义 F1 分数作为评估指标
f1_scorer = make_scorer(f1_score)

# 交叉验证调整模型参数并评估 F1 分数
cv_scores = cross_val_score(nb_classifier, X_train, y_train, cv=5, scoring=f1_scorer)

# 打印交叉验证的 F1 分数
print("Cross-Validation F1 Scores:", cv_scores)

# 打印平均 F1 分数
print("Average F1 Score:", np.mean(cv_scores))

# 训练模型
nb_classifier.fit(X_train, y_train)

# 在数据集上进行预测
y_pred = nb_classifier.predict(X_test)

# 打印最准确的分类结果
print("Most Accurate Predictions:")
print(pd.DataFrame({'Actual': y_test, 'Predicted': y_pred}).head(10))
```

Accuracy: 0.7472

logistic regression:

```
# 将房价转化为分类标签
data['Price_Category'] = pd.cut(data['PRICE'], bins=[0, data['PRICE'].median(), data['PRICE'].max()

# 划分特征和目标变量
X = data.drop(['PRICE', 'Price_Category'], axis=1)
y = data['Price_Category']

# 将目标变量编码为数字
y = pd.factorize(y)[0]

# 划分数据集
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# 使用 Logistic Regression 分类器
lr_classifier = LogisticRegression()

# 定义 F1 分数作为评估指标
f1_scorer = make_scorer(f1_score)

# 交叉验证调整模型参数并评估 F1 分数
cv_scores = cross_val_score(lr_classifier, X_train, y_train, cv=5, scoring=f1_scorer)

# 打印交叉验证的 F1 分数
print("Cross-Validation F1 Scores:", cv_scores)

# 打印平均 F1 分数
print("Average F1 Score:", np.mean(cv_scores))

# 训练模型
lr_classifier.fit(X_train, y_train)

# 在数据集上进行预测
y_pred = lr_classifier.predict(X_test)

# 打印最准确的分类结果
print("Most Accurate Predictions:")
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

1	handicapp	water-prc	adoption-	physiciar	el-salvac	religious	anti-sate	aid-to-ni	mx-missil	immigrati	synfuels-	educatior	superfunc	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                                n                  y
2                                n                  y
3                                y                  ?
4                                y                  n

   el-salvador-aid   religious-groups-in-schools   anti-satellite-test-ban  \
0  el-salvador-aid   religious-groups-in-schools   anti-satellite-test-ban
1                y                             y                         n
2                y                             y                         n
3                y                             y                         n
4                ?                             y                         n

   aid-to-nicaraguan-contras   mx-missile   immigration  \
0  aid-to-nicaraguan-contras   mx-missile   immigration
1                          n            n             y
2                          n            n             n
3                          n            n             n
4                          n            n             n

   synfuels-corporation-cutback   education-spending  \
0  synfuels-corporation-cutback   education-spending
1                             ?                    y
2                             n                    y
3                             y                    n
4                             y                    n

   superfund-right-to-sue   crime   duty-free-exports  \
0  superfund-right-to-sue   crime   duty-free-exports
1                       y       y                   n
2                       y       y                   n
3                       y       y                   n
4                       y       n                   n

   export-administration-act-south-africa  Class
0  export-administration-act-south-africa  False
1                                       y  False
2                                       ?  False
3                                       n   True
4                                       y   True

In [6]: X = pd.get_dummies(df[column_names[:-1]], prefix=column_names[:-1]).astype(bool)
   ...: y = df['Class']
   ...: X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42)

In [7]: bin_cls = fcalc.classifier.BinarizedBinaryClassifier(X_train.values, y_train.to_numpy(), method="standard-support")

In [8]: bin_cls.predict(X_test.values)

In [9]: from sklearn.metrics import accuracy_score, f1_score
   ...:
   ...: print(accuracy_score(y_test, bin_cls.predictions))
   ...: print(f1_score(y_test, bin_cls.predictions))
0.9312977099236641
```

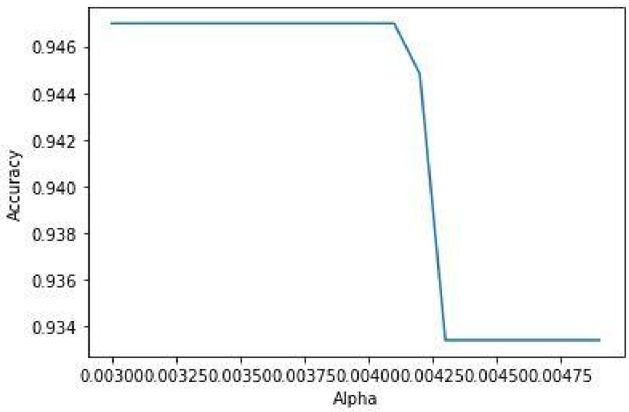
Accuracy: 0.9312

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 = 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:



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

	RR	RS	RT	RU	RV	RW	RX	RY	RZ	SA	SB	SC
1	18x10	18x11	18x12	18x13	18x14	18x15	18x16	18x17	18x18	18x19	18x20	18x21
2	0	0	0	0	0	0	126	254	182	0	0	0
3	253	253	173	12	0	0	0	0	0	0	0	0
4	0	0	0	120	254	159	0	0	0	0	0	0
5	251	0	0	0	0	0	0	144	251	251	251	221
6	245	252	252	252	252	232	231	251	252	252	9	0

	SD	SE	SF	SG	SH	SI	SJ	SK	SL	SM	SN	SO
1	18x22	18x23	18x24	18x25	18x26	18x27	18x28	19x1	19x2	19x3	19x4	19x5
2	0	0	0	0	0	0	0	0	0	0	0	
3	0	0	0	0	0	0	0	0	0	0	0	
4	0	0	0	0	0	0	0	0	0	0	0	
5	61	0	0	0	0	0	0	0	0	0	0	
6	0	0	0	0	0	0	0	0	0	0	0	

Class: (number)

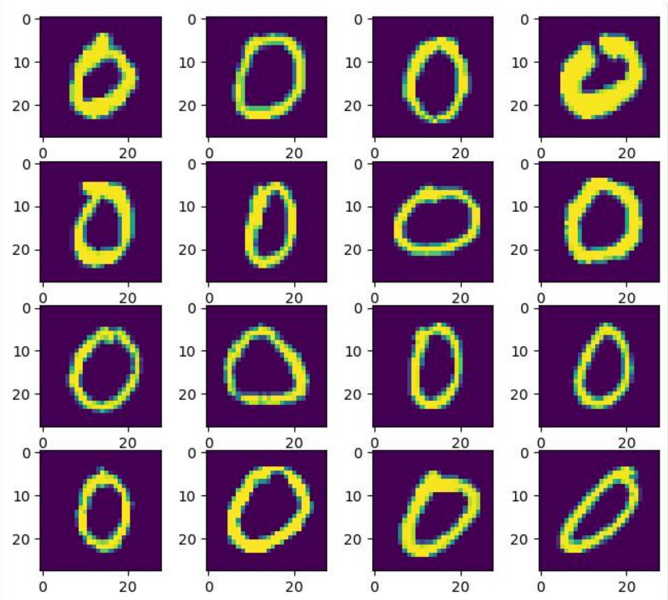
ADE
Class
7
2
1
0
4
1
4
9

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)
```

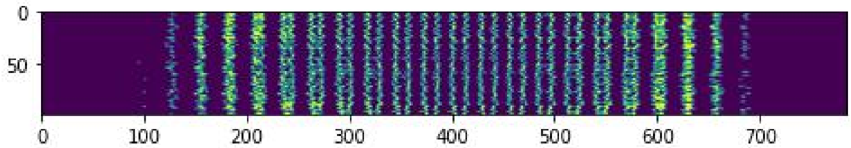
Show all pictures under data:

```
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