Report of LAZYFCA

NINGYU ZHOU

1.Introduction of Dataset

In this analysis, we have employed three diverse datasets: weather_tokyo_data, mushrooms, and iris.

1.1 weather_tokyo_data

Website: https://www.kaggle.com/datasets/risakashiwabara/tokyo-weatherdata/data

The weather_tokyo_data dataset comprises 7 columns:

- year:Year,with366non-nullintegervalues(int64).
- day:Date,with366non-nullobjectvalues.
- temperature:Temperature,with366non-nullobjectvalues.
- humidity:Humidity,with366non-nullfloatvalues(float64).
- atmosphericpressure:AtmosphericPressure,with366non-nullfloat values (float64).
- Date:Date(convertedtodatetimeformat),with366non-nullvalues (datetime64[ns]).
- Temperature:Temperature,with366non-nullfloatvalues(float64).

1.2 mushrooms Dataset

Website: https://www.kaggle.com/datasets/uciml/mushroom-classification?rvi=1

About this File

Attribute Information: (classes: edible=e, poisonous=p)

- cap-shape:bell(b),conical(c),convex(x),flat(f),knobbed(k), sunken(s).
- cap-surface:fibrous(f),grooves(g),scaly(y),smooth(s).
- cap-color:brown(n),buff(b),cinnamon(c),gray(g),green(r),pink(p), purple(u), red(e), white(w), yellow(y).
- ...(otherattributesomitted)

1.3 iris Dataset

Website: https://archive.ics.uci.edu/dataset/53/iris

Variable Names
- sepallength:Continuousfeaturerepresentingthelengthofthesepal,
-

measured in cm.
- sepalwidth:Continuousfeaturerepresentingthewidthofthesepal,

measured in cm.
- petallength:Continuousfeaturerepresentingthelengthofthepetal,

measured in cm.
- petalwidth:Continuousfeaturerepresentingthewidthofthepetal,

measured in cm.
- class:Targetvariable,categorical,indicatingtheclassofirisplant:Iris

Setosa, Iris Versicolour, or Iris Virginica.

2. Binarization

2.1 Binarization of the mushrooms Dataset

In the mushrooms dataset, the binarization strategy is applied based on the values in the class column. If the value is equal to 'e' (edible), it is represented as true; otherwise, it is represented as false.

2.2 Binarization of the iris Dataset

For the iris dataset, the binarization strategy is determined by the values in the sepal_length column. If the value is greater than 4.8, it is assigned the binary value 1; otherwise, it is assigned the binary value 0.

2.3 Binarization of the weather_tokyo_data Dataset

In the weather_tokyo_data dataset, binarization is performed based on the values in the year column. If the value is equal to 2023, it is represented as true; otherwise, it is represented as false.
These binarization strategies allow us to transform the datasets into a format conducive to certain types of analyses, facilitating the exploration of patterns and relationships within the data.

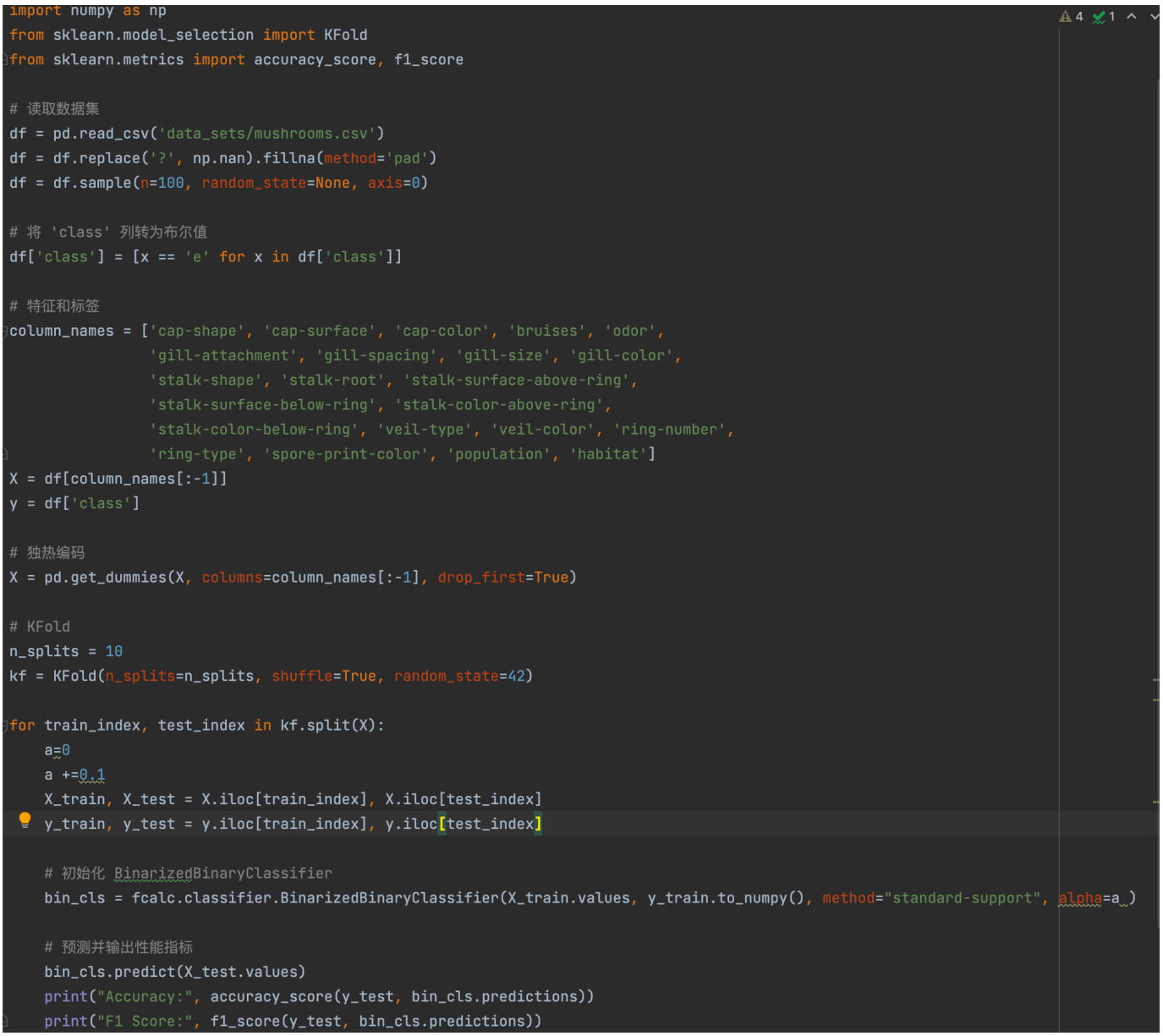
3.Code Examples

mushroomPatternKfold:

This code snippet demonstrates the usage of sklearn.model_selection.KFold for performing k-fold cross-validation. The dataset is split into k consecutive folds (default behavior without shuffling), using k-1 folds for training and the remaining fold for testing. This process is repeated k times, with each fold used exactly once as a validation data set.

Following the dataset split into k consecutive folds, the code then enters a loop where the alpha parameter is adjusted, and the resulting changes in the

performance metrics are observed during each iteration.



```python
import numpy as np
from sklearn.model_selection import KFold
from sklearn.metrics import accuracy_score, f1_score

# 读取数据集
df = pd.read_csv('data_sets/mushrooms.csv')
df = df.replace('?', np.nan).fillna(method='pad')
df = df.sample(n=100, random_state=None, axis=0)

# 将 'class' 列转为布尔值
df['class'] = [x == 'e' for x in df['class']]

# 特征和标签
column_names = ['cap-shape', 'cap-surface', 'cap-color', 'bruises', 'odor',
                'gill-attachment', 'gill-spacing', 'gill-size', 'gill-color',
                'stalk-shape', 'stalk-root', 'stalk-surface-above-ring',
                'stalk-surface-below-ring', 'stalk-color-above-ring',
                'stalk-color-below-ring', 'veil-type', 'veil-color', 'ring-number',
                'ring-type', 'spore-print-color', 'population', 'habitat']
X = df[column_names[:-1]]
y = df['class']

# 独热编码
X = pd.get_dummies(X, columns=column_names[:-1], drop_first=True)

# KFold
n_splits = 10
kf = KFold(n_splits=n_splits, shuffle=True, random_state=42)

for train_index, test_index in kf.split(X):
    a=0
    a +=0.1
    X_train, X_test = X.iloc[train_index], X.iloc[test_index]
    y_train, y_test = y.iloc[train_index], y.iloc[test_index]

    # 初始化 BinarizedBinaryClassifier
    bin_cls = fcalc.classifier.BinarizedBinaryClassifier(X_train.values, y_train.to_numpy(), method="standard-support", alpha=a )

    # 预测并输出性能指标
    bin_cls.predict(X_test.values)
    print("Accuracy:", accuracy_score(y_test, bin_cls.predictions))
    print("F1 Score:", f1_score(y_test, bin_cls.predictions))
```

```
Accuracy: 0.7
F1 Score: 0.7692307692307692
Accuracy: 0.8
F1 Score: 0.8000000000000002
Accuracy: 0.8
F1 Score: 0.8571428571428571
Accuracy: 1.0
F1 Score: 1.0
Accuracy: 0.9
F1 Score: 0.9333333333333333
Accuracy: 0.7
F1 Score: 0.7692307692307692
Accuracy: 1.0
F1 Score: 1.0
Accuracy: 0.9
F1 Score: 0.888888888888889
Accuracy: 1.0
F1 Score: 1.0
Accuracy: 0.9
F1 Score: 0.888888888888889
```

mushroomtest:(Binary and Classifier)

n the mushroomtest code snippet, the alpha parameter is adjusted iteratively, and the resulting differences in the performance metrics, such as accuracy, are observed. This process allows for a comprehensive evaluation of the model's sensitivity to variations in the alpha parameter, aiding in the selection of an optimal value for model training and

testing.

```
print(df.head())

)#
#print(df.columns)
column_names = ['cap-shape', 'cap-surface', 'cap-color', 'bruises', 'odor',
                'gill-attachment', 'gill-spacing', 'gill-size', 'gill-color',
                'stalk-shape', 'stalk-root', 'stalk-surface-above-ring',
                'stalk-surface-below-ring', 'stalk-color-above-ring',
                'stalk-color-below-ring', 'veil-type', 'veil-color', 'ring-number',
                'ring-type', 'spore-print-color', 'population', 'habitat']

X = pd.get_dummies(df[column_names[:-1]], prefix=column_names[:-1]).astype(bool)
#print(X.columns)
y = df['class']
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42)
bin_cls = fcalc.classifier.BinarizedBinaryClassifier(X_train.values, y_train.to_numpy(), method="standard-support", alpha=0.2)
bin_cls.predict(X_test.values)

print(accuracy_score(y_test, bin_cls.predictions))
print(f1_score(y_test, bin_cls.predictions))

bin_cls = fcalc.classifier.BinarizedBinaryClassifier(X_train.values, y_train.to_numpy(), method="standard-support", alpha=0.3)
bin_cls.predict(X_test.values)

print(accuracy_score(y_test, bin_cls.predictions))
print(f1_score(y_test, bin_cls.predictions))

bin_cls = fcalc.classifier.BinarizedBinaryClassifier(X_train.values, y_train.to_numpy(), method="standard-support", alpha=0.4)
bin_cls.predict(X_test.values)

print(accuracy_score(y_test, bin_cls.predictions))
print(f1_score(y_test, bin_cls.predictions))

bin_cls = fcalc.classifier.BinarizedBinaryClassifier(X_train.values, y_train.to_numpy(), method="standard-support", alpha=0.5)
bin_cls.predict(X_test.values)

print(accuracy_score(y_test, bin_cls.predictions))
```

```
[5 rows x 23 columns]
0.6333333333333333
0.6857142857142857
0.6
0.6666666666666667
0.5666666666666667
0.6486486486486486
0.5666666666666667
0.6486486486486486
0.5666666666666667
0.6486486486486486
```

mushroompattren:(Pattern and Classifier)

```
#load dataset
df = pd.read_csv('data_sets/mushrooms.csv')

#print(df.head())
df.replace('?', np.nan).fillna(method='pad')
df = df.sample(n=100, random_state=None, axis=0)

df['class'] = [x == 'e' for x in df['class']]

column_names = ['cap-shape', 'cap-surface', 'cap-color', 'bruises', 'odor',
                'gill-attachment', 'gill-spacing', 'gill-size', 'gill-color',
                'stalk-shape', 'stalk-root', 'stalk-surface-above-ring',
                'stalk-surface-below-ring', 'stalk-color-above-ring',
                'stalk-color-below-ring', 'veil-type', 'veil-color', 'ring-number',
                'ring-type', 'spore-print-color', 'population', 'habitat']


X = df[column_names[:-1]]
y = df['class']
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42)

pat_cls = fcalc.classifier.PatternBinaryClassifier(X_train.values, y_train.to_numpy(),
                                                   categorical=np.arange(X_train.shape[1]))

pat_cls.predict(X_test.values)

from sklearn.metrics import accuracy_score, f1_score

print(accuracy_score(y_test, pat_cls.predictions))
print(f1_score(y_test, pat_cls.predictions))
```

```
      class cap-shape cap-surface  ... spore-print-color population habitat
7807   True         k           s  ...                 b          c       l
4198  False         x           f  ...                 k          v       d
7182   True         f           s  ...                 n          c       l
710    True         f           y  ...                 k          s       p
1293  False         f           s  ...                 n          v       g

[5 rows x 23 columns]
0.9666666666666667
0.9696969696969697
```

iris.py:(Binary and Classifier)

```python
import fcalc
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score, f1_score
from sklearn.preprocessing import Binarizer

#load the dataset
df = pd.read_csv('data_sets/iris.data', names=['sepal_length',  'sepal_width',  'petal_length', 'petal_width','sp
#binarizer = Binarizer(threshold=4.0)
#df['sepal_length'] = Binarizer(threshold=4.0, copy=True).fit_transform(df['sepal_length'])
#df['sepal_length']=Binarizer(threshold=4.0).fit_transform(df['sepal_length'])
def binarize_sepal_length(value):
    return 1 if value > 4.8 else 0
df['sepal_length'] = df['sepal_length'].apply(binarize_sepal_length)
#print(iris_df[['sepal length (cm)', 'sepal_length_binary']])

print(df.head())

X = df.iloc[:,:-1]
y = df['sepal_length']
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=100)

pat_cls = fcalc.classifier.PatternBinaryClassifier(X_train.values, y_train.to_numpy())
pat_cls.predict(X_test.values)

print("accuracy:",round(accuracy_score(y_test, pat_cls.predictions),4))
print("f1 score:",round(f1_score(y_test, pat_cls.predictions),4))
```

```
   sepal_length  sepal_width  petal_length  petal_width      species
0             1          3.5           1.4          0.2  Iris-setosa
1             1          3.0           1.4          0.2  Iris-setosa
2             0          3.2           1.3          0.2  Iris-setosa
3             0          3.1           1.5          0.2  Iris-setosa
4             1          3.6           1.4          0.2  Iris-setosa
accuracy: 1.0
f1 score: 1.0
```

mushroomKNN:(Binary and Classifier)

```
df['class'] = [x == 'e' for x in df['class']]

column_names = ['cap-shape', 'cap-surface', 'cap-color', 'bruises', 'odor',
                'gill-attachment', 'gill-spacing', 'gill-size', 'gill-color',
                'stalk-shape', 'stalk-root', 'stalk-surface-above-ring',
                'stalk-surface-below-ring', 'stalk-color-above-ring',
                'stalk-color-below-ring', 'veil-type', 'veil-color', 'ring-number',
                'ring-type', 'spore-print-color', 'population', 'habitat']


X = df[column_names[:-1]]
y = df['class']
# label_encoder = LabelEncoder()
# y = label_encoder.fit_transform(y)
print(df.head())

# Perform one-hot encoding on the features
X = pd.get_dummies(X)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42)

# Create a KNeighborsClassifier
knn_classifier = KNeighborsClassifier(n_neighbors=3)

# Fit the model
knn_classifier.fit(X_train, y_train)
# 在测试集上进行预测
y_pred = knn_classifier.predict(X_test)

# 输出模型性能
accuracy = accuracy_score(y_test, y_pred)
print("Accuracy:", accuracy)
```

```
      class cap-shape cap-surface  ... spore-print-color population habitat
686    True         f           f  ...                 k          v       u
656    True         f           f  ...                 n          s       g
3322  False         x           f  ...                 n          s       d
3286   True         x           y  ...                 n          v       d
7623   True         k           s  ...                 w          s       g

[5 rows x 23 columns]
Accuracy: 0.8666666666666667
```

Another(Pattern):

```python
#load dataset
df = pd.read_csv('data_sets/weather_tokyo_data.csv')

# print(df.head())
# df = df.sample(n=100, random_state=None, axis=0)

df['year'] = [x == 2023 for x in df['year']]
print(df.values)

print(df.columns)


column_names = ['year', 'day', 'temperature', 'humidity ', 'atmospheric pressure']


X = df[column_names[:-1]]
y = df['year']
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42)

pat_cls = fcalc.classifier.PatternBinaryClassifier(X_train.values, y_train.to_numpy(), method="standard-support", alpha=0.)

pat_cls.predict(X_test.values)

# from sklearn.metrics import accuracy_score, f1_score

print(accuracy_score(y_test, pat_cls.predictions))
print(f1_score(y_test, pat_cls.predictions))
```

```
0.9545454545454546
0.9723756906077348
```

Conclusion

In conclusion, our analysis encompassed three distinct datasets:

weather_tokyo_data, mushrooms, and iris. We began by providing an overview of each dataset, detailing their respective columns, data types, and key features. Subsequently, the binarization strategies were outlined for each dataset. In the case of mushrooms, binarization was based on the 'class' column, designating 'edible' as true and 'poisonous' as false. For iris, binarization relied on the 'sepal_length' values, with those greater than 4.8 assigned 1, and others assigned 0. Meanwhile, weather_tokyo_data underwent binarization based on the 'year' column, marking 2023 as true and other years as false.

We then delved into code examples, showcasing techniques such as k-fold cross-validation (mushroomPatternKfold), parameter adjustment (mushroomtest), and classifier implementations for different datasets (mushroompattren, iris.py, mushroomKNN). These examples highlighted the versatility of the employed models and the impact of parameter tuning on model performance.

In the mushroomtest section, adjusting the alpha parameter revealed noteworthy differences in results, emphasizing the importance of fine-tuning

hyperparameters for optimal model performance.

This comprehensive exploration of datasets and model implementations lays the foundation for further in-depth analyses, allowing us to uncover hidden patterns, relationships, and insights. The iterative nature of parameter adjustment ensures robustness in model selection, contributing to the overall success of our analytical endeavors.