实验三 分类任务

姓名：张辰菁 班级：计算机2103 学号：2215015048

一、实验目的

1.学习使用sklearn的传统分类算法，包括决策树、随机森林、SVM、KNN、贝叶斯等；

2.应用降维和传统分类算法对数据进行分类。

二、实验过程

1.sklearn降维方法

sklearn库中有主成分分析（PCA）和t-分布随机邻域嵌入（t-SNE）两种降维方法。

1.1 PCA

PCA是一种线性降维技术，它通过将数据投影到主成分空间中，使得投影后的数据在新空间中的方差最大。主成分是原始数据特征的线性组合，且每个主成分相互正交。

优点：

- 简单高效，计算速度快。
- 保留了尽可能多的原始数据的方差信息。
- 结果可以解释为数据的主要方向。

缺点：

- 仅适用于线性关系的数据。
- 对噪声敏感。

1.2 T-SNE

t-SNE是一种非线性降维技术，特别适用于高维数据的可视化。t-SNE通过最小化高维空间中数据点间的相似性和低维空间中数据点间的相似性的差异来保持数据的局部结构。

优点：

- 非线性降维，能有效捕捉复杂的非线性结构。
- 非常适合高维数据的可视化（常用于将数据降到2维或3维）。

缺点：

- 计算复杂度高，适用于小数据集。
- 结果难以解释，主要用于可视化而非建模。

对比两种降维方法而言，对于大型数据集的降维和特征提取，PCA更为高效；对于数据可视化，特别是高维数据的探索性分析，t-SNE则更为强大。

2.sklearn分类算法

常用的聚类算法有decision-tree,svm,random forest,MLP,KNN,等算法。本次实验中使用了SVM,decision-tree和random forest算法。

2.1 支持向量机（SVM）

SVM是一种用于分类和回归分析的监督学习算法。它通过找到最佳的决策边界（超平面）来将数据分成不同的类别。SVM能够处理线性不可分的数据，通过核技巧（Kernel Trick）将数据映射到高维空间，从而使数据线性可分。

```
from sklearn import svm

clf = svm.SVC(kernel='linear')  # 线性核函数
clf.fit(X_train, y_train)
y_pred = clf.predict(X_test)
```

主要参数为kernel,为核函数的类型。

2.2 决策树（Decision Tree）

决策树是一种树形结构的监督学习算法，适用于分类和回归任务。它通过对特征进行条件判断，将数据逐步划分，直到每个子集基本上属于同一类别或满足某个停止条件。

```
from sklearn.tree import DecisionTreeClassifier
clf = DecisionTreeClassifier()
clf.fit(X_train, y_train)
y_pred = clf.predict(X_test)
```

决策树的主要参数较多，将在实验过程中探究。

2.3 随机森林（Random Forest）

随机森林是一种集成学习方法，通过构建多个决策树来提高模型的预测性能。每棵树都在训练集的不同子集上训练，并且在预测时，每棵树的结果都会被综合考虑（基于投票机制）。

```
from sklearn.ensemble import RandomForestClassifier
clf = RandomForestClassifier(n_estimators=100)
clf.fit(X_train, y_train)
y_pred = clf.predict(X_test)
```

参数n_estimators，为树木的数量。与决策树相同，随机森林的主要参数也较多，将在实验过程中探究。

3.分类结果评价

使用精度、召回率和F1-socre来对分类结果进行评价。使用混淆矩阵来对分类结果进行直观观察。

1.精度（Precision）：

- **定义**: 精度是指模型预测为正类的样本中，真正为正类的比例。
- **计算公式**: Precision = TP/(TP+FP)

2.召回率（Recall）：

- **定义**: 召回率是指所有真正为正类的样本中被正确预测为正类的比例。
- **计算公式**: Recall = TP/(TP+FN)

3.F1-score:

- **定义**: F1-score 是精度和召回率的调和平均数，用于综合评估模型的性能。
- **计算公式**:F1-score = 2*precision*recall/(precision+recall)

4.混淆矩阵

- 混淆矩阵（Confusion Matrix）是用于评估分类模型性能的一个工具，它通过矩阵的形式展示了模型预测结果的详细情况，帮助我们更好地理解模型的分类效果。混淆矩阵特别适用于二分类问题，但也可以扩展到多分类问题。

5.多分类问题下的评价指标

在多分类问题中，我们通常使用宏平均（macro-average）和微平均（micro-average）来计算精度、召回率和 F1-score。

- **宏平均（Macro-average Precision）**： 先计算每个类别的精度，然后取这些精度的平均值。
- **微平均（Micro-average Precision）**： 先计算总的 TP 和 FP，然后计算整体精度。

本次实验中使用宏平均方法计算精度、召回率和 F1-score。

```
#输出精度，召回率，F1-score，打印混淆矩阵
y_pred = clf.predict(X_test)
print("decision tree trained by non-pca data")
print(f'Accuracy: { accuracy_score(y_test, y_pred) }')
#Precision = TP/TP+FP 宏平均
print(f'Precision: { precision_score(y_test, y_pred, average="macro") }')
#Recall = TP/TP+FN
print(f'Recall: { recall_score(y_test, y_pred, average="macro") }')
#F1-score = 2*precision*recall/(precision+recall)
print(f'F1-score { f1_score(y_test, y_pred, average="macro") }')
cm2_1 = confusion_matrix(y_test, y_pred)
print(cm2_1.shape)
print('cm2 is:\n', cm2_1)
```

三、实验内容及实验结果

1.读取数据集

本次实验读取了Mnist数据集作为分类的数据集。

```
def get_dataset(split, num):
    data_path = './datasets/MNIST_Data/' + split
    dataset = ds.MnistDataset(data_path)

    #归一化
    image_transforms = [
        C.Rescale(1.0 / 255.0, 0),
        C.Normalize(mean=(0.1307,), std=(0.3081,)),
        C.HWC2CHW()
    ]
    dataset = dataset.map(image_transforms, 'image')

    #获取迭代器
    dataset = dataset.batch(num)
    iterator_show = dataset.create_dict_iterator()
    dict_data = next(iterator_show)

    x = dict_data["image"].reshape(num, -1).asnumpy()
    y = dict_data["label"].asnumpy()
    X_train, X_test, y_train, y_test = train_test_split(x, y, test_size=0.2,
random_state=42)
```

```
    return X_train, X_test, y_train, y_test
```

使用了mindspore自带的mindspore.dataset(as ds)中的Cifar10Dataset(data_path)来对CIFAR10数据集进行读取，对数据进行归一化，然后设置数据加载器的获取批次，使用dataset.create_dict_iterator()来创建一个迭代器来循环读取该数据集里的数据。然后，将x和y转化为numpy数组。最后，使用sklearn库中的train_test_split函数将x和y分为测试集和训练集，返回测试集和训练集。

读取结果

```
322 ms
X_train, X_test, y_train, y_test = get_dataset("train", 1000)
print(f'X_train.shape:{X_train.shape}')
print(f'y_train.shape:{y_train.shape}')
print(f'X_test.shape:{X_test.shape}')
print(f'y_test.shape:{y_test.shape}')
```

```
X_train.shape:(800, 784)
y_train.shape:(800,)
X_test.shape:(200, 784)
y_test.shape:(200,)
```

```
48 ms [17]
print(X_train)
```

```
[[-0.42421296 -0.42421296 -0.42421296 ... -0.42421296 -0.42421296
  -0.42421296]
 [-0.42421296 -0.42421296 -0.42421296 ... -0.42421296 -0.42421296
  -0.42421296]
 [-0.42421296 -0.42421296 -0.42421296 ... -0.42421296 -0.42421296
  -0.42421296]
 ...
 [-0.42421296 -0.42421296 -0.42421296 ... -0.42421296 -0.42421296
  -0.42421296]
 [-0.42421296 -0.42421296 -0.42421296 ... -0.42421296 -0.42421296
  -0.42421296]
 [-0.42421296 -0.42421296 -0.42421296 ... -0.42421296 -0.42421296
  -0.42421296]]
```

2.降维

由于Mnist为图像数据集，均为高维数据，在对其进行可视化前，需要对数据进行降维；另外，使用降维后的数据进行训练，可以提升训练速度和效率，因为有许多可能有相互联系的特征被合并了，剩余的特征之间都是相互独立的。但是，由于数据降维后会损失部分信息，可能会导致训练效果变差。

```
#降维数据
pca = PCA(n_components=2)
X_train_lower = pca.fit_transform(X_train)
X_test_lower = pca.fit_transform(X_test)
```

降维后的训练集

```
#降维数据
pca = PCA(n_components=2)
X_train_lower = pca.fit_transform(X_train)
X_test_lower = pca.fit_transform(X_test)
```

```
print(f'X_train_lower.shape:{X_train_lower.shape}')
print(f'X_test_lower.shape:{X_test_lower.shape}')
print(X_train_lower[1:10])
print(X_test_lower[1:10])
```

```
X_train_lower.shape:(800, 2)
X_test_lower.shape:(200, 2)
[[-9.65491    -2.5064108 ]
 [ 6.499065   10.219176  ]
 [ 3.0091343   5.399989  ]
 [ 1.9668894  -1.9281023 ]
 [ 0.22741629 -5.6329017 ]
 [ 1.0383185   2.3475945 ]
 [-1.7165016   3.9295266 ]
 [-0.479247    2.5563045 ]
 [ 5.9227185  10.58128   ]]
[[ 17.488417    -5.4272532 ]
 [  4.571991     6.2660933 ]
 [ -5.043976     0.2159701 ]
 [  0.62662697   6.7290916 ]
 [-10.024604    -1.1173084 ]
 [ -8.22393      8.149832  ]
 [ -5.2930045   -5.1416607 ]
```

3. 应用分类算法

3.1 决策树

使用sk-learn库中的sklearn.tree.DecisionTreeClassifier进行分类。

```
# 决策树
params = {
    #特征选择标准
    'criterion': 'entropy',
    #特征划分点选择标准
    'splitter': 'best',
    #决策树最大深
    'max_depth': 15,
    #节点最小样本数
    'min_samples_split': 2,
    #叶子节点最小样本数
    'min_samples_leaf': 1,
    #叶子节点最小权重
    'min_weight_fraction_leaf': 0.0,
    #最大特征数
    'max_features': 0.5,
    #随机种子
    'random_state': None,
    #最大叶子节点数
    'max_leaf_nodes': None,
    #最小不纯度下降值
    'min_impurity_decrease': 0.0,
    #类别权重：防止训练集某些类别的样本过多，导致训练的决策树过于偏向这些类别
    'class_weight': None,
```

```
}
clf=DecisionTreeClassifier(**params)
clf2=DecisionTreeClassifier(**params)
```

可见，决策树有许多参数，其中最主要的为criterion(特征选择标准)、max_depth(决策树最大深)、splitter(特征划分点选择标准)、min_samples_split(节点最小样本数)和min_samples_leaf(叶子节点最小样本数)等。

1. **criterion**:
   - 选择特征划分标准，常见的有`'gini'`（基尼不纯度）和`'entropy'`（信息增益）。在这里，使用的是`'entropy'`，即信息增益标准。
2. **splitter**:
   - 特征划分点的选择标准，`'best'`表示选择最优的划分点，`'random'`表示随机选择划分点。
3. **max_depth**:
   - 决策树的最大深度。限制树的最大深度可以防止过拟合。这里设置为15，即树的最大深度为15。
4. **min_samples_split**:
   - 内部节点再划分所需的最小样本数。设置为2表示每个节点至少有2个样本才会继续划分。
5. **min_samples_leaf**:
   - 叶子节点所需的最小样本数。设置为1表示每个叶子节点至少包含1个样本。

```
clf.fit(X_train, y_train)
clf2.fit(X_train_lower,y_train)
```

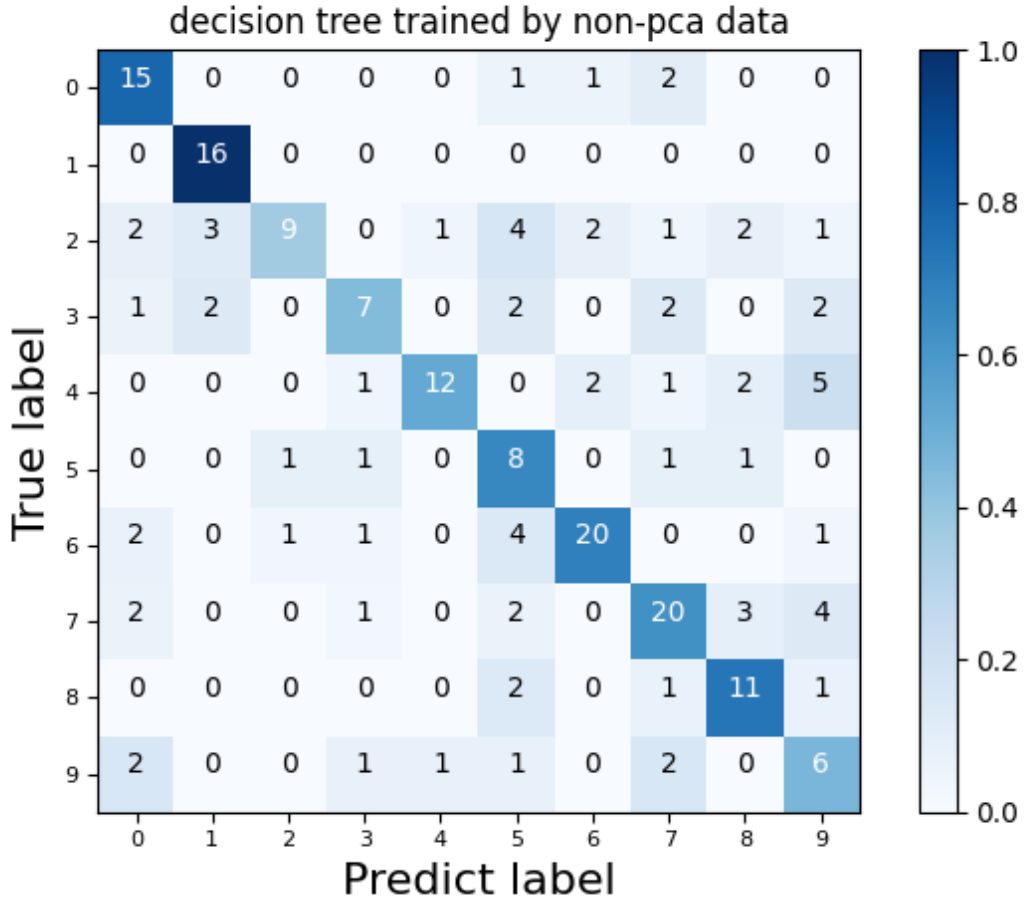
分类结果

```
y_pred = clf.predict(X_test)
print("decision tree trained by non-pca data")
print(f'Accuracy: { accuracy_score(y_test, y_pred) }')
#Precision = TP/TP+FP 宏平均
print(f'Precision: { precision_score(y_test, y_pred, average="macro") }')
#Recall = TP/TP+FN
print(f'Recall: { recall_score(y_test, y_pred, average="macro") }')
#F1-score = 2*precision*recall/(precision+recall)
print(f'F1-score { f1_score(y_test, y_pred, average="macro") }')
cm2_1 = confusion_matrix(y_test, y_pred)
print(cm2_1.shape)
print('cm2 is:\n', cm2_1)
```

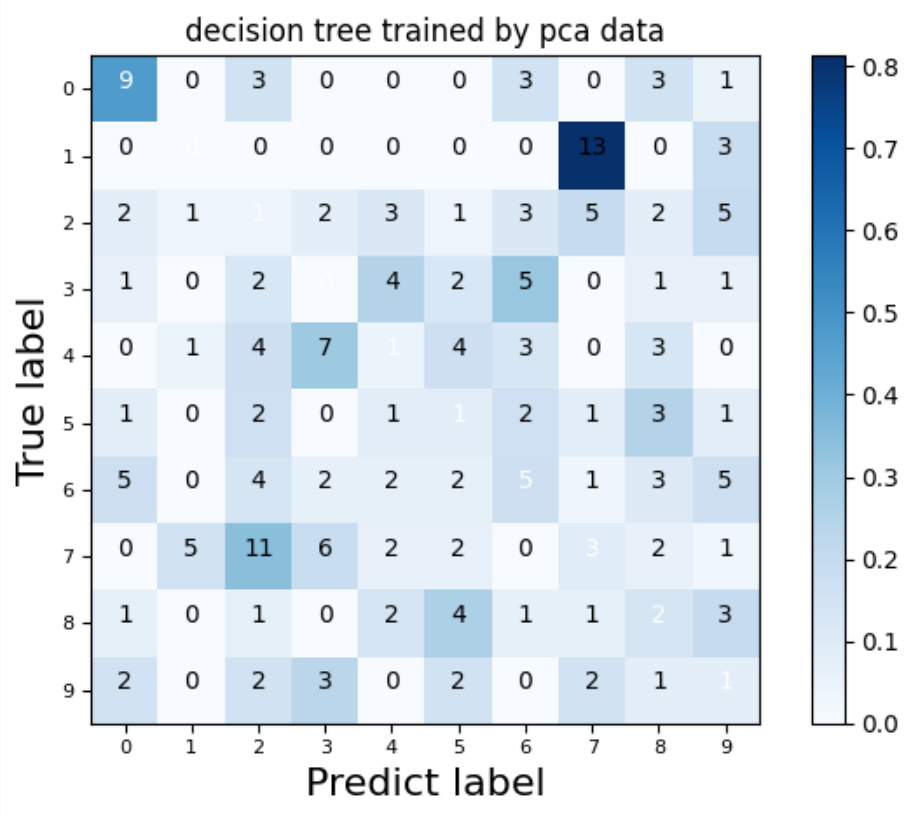
```
decision tree trained by non-pca data
Accuracy: 0.62
Precision: 0.6324510138983823
Recall: 0.6284906448597564
F1-score 0.6052229594791705
(10, 10)
cm2 is:
 [[15  0  0  0  0  1  1  2  0  0]
 [ 0 16  0  0  0  0  0  0  0  0]
 [ 2  3  9  0  1  4  2  1  2  1]
 [ 1  2  0  7  0  2  0  2  0  2]
 [ 0  0  0  1 12  0  2  1  2  5]
 [ 0  0  1  1  0  8  0  1  1  0]
 [ 2  0  1  1  0  4 20  0  0  1]
 [ 2  0  0  1  0  2  0 20  3  4]
 [ 0  0  0  0  0  2  0  1 11  1]
 [ 2  0  0  1  1  1  0  2  0  6]]
```

```
visualize_cm(cm2_1,title='decision tree trained by non-pca data')
```

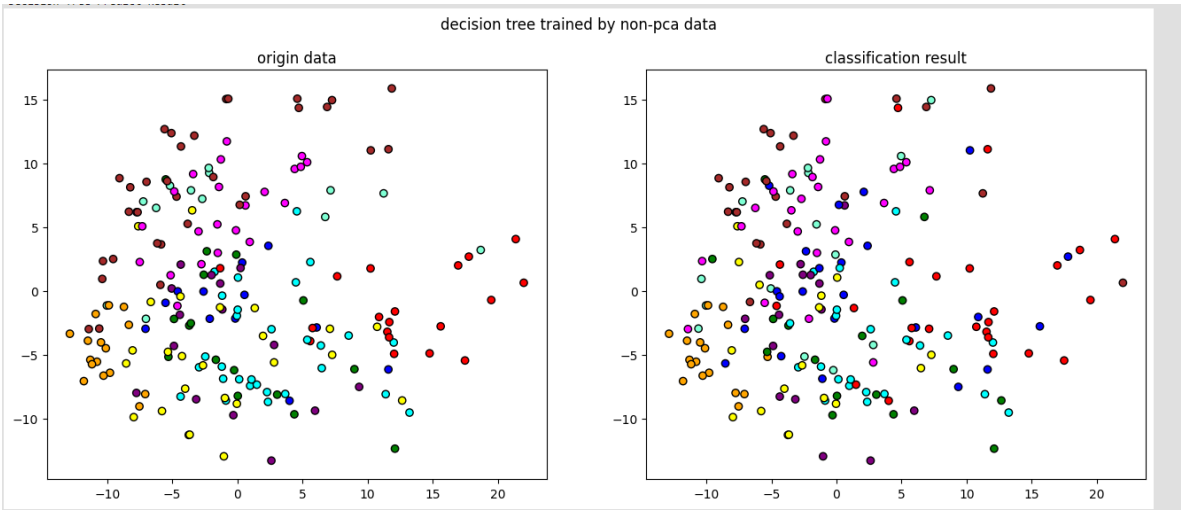
```
10
```

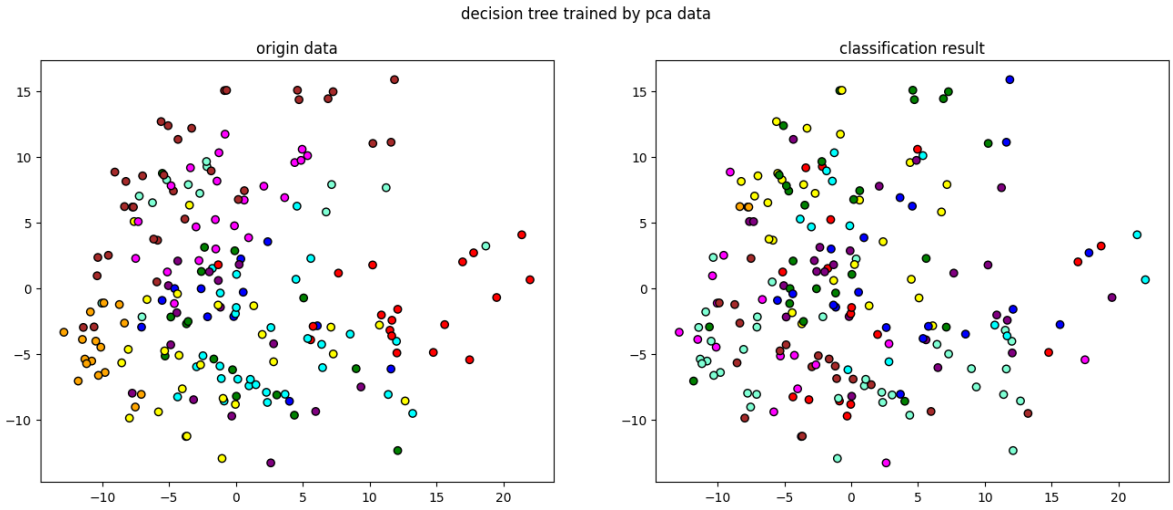


```
desicion tree
Accuracy: 0.115
Precision: 0.10744033744033746
Recall: 0.11169160080890728
F1-score 0.10782978329097685
(10, 10)
cm2 is:
 [[ 9  0  3  0  0  0  3  0  3  1]
 [ 0  0  0  0  0  0  0 13  0  3]
 [ 2  1  1  2  3  1  3  5  2  5]
 [ 1  0  2  0  4  2  5  0  1  1]
 [ 0  1  4  7  1  4  3  0  3  0]
 [ 1  0  2  0  1  1  2  1  3  1]
 [ 5  0  4  2  2  2  5  1  3  5]
 [ 0  5 11  6  2  2  0  3  2  1]
 [ 1  0  1  0  2  4  1  1  2  3]
 [ 2  0  2  3  0  2  0  2  1  1]]
10
```



可以看到决策树分类器对于降维后的数据训练效果很差，准确度只有10%左右(约等于乱猜),而对于只归一化后的训练数据准确度大概有60%。这可能是因为降维力度太大(原先将数据将为2维)，让训练数据损失了大部分信息。





3.2 random forest

使用sk-learn库中的sklearn.ensemble.RandomForestClassifier函数进行分类。

```
# 随机森林
params2={
    #森林中树木的数量
    'n_estimators': 100,
    'criterion': 'entropy',

    #树的最大深度，超过最大深度的树枝都会被剪掉
    'max_depth': 10,

    #一个节点在分支后的每个子节点都必须包含至少min_samples_leaf个训练样本，否则分支就不会发生
    'min_samples_split': 2,

    #一个节点必须要包含至少min_samples_split个训练样本，这个节点才被允许分支，否则分支不会发生。
    'min_samples_leaf': 1,

    'min_weight_fraction_leaf': 0.0,
    #max_features限制分支时考虑的特振奋个数，超过限制个数的特征都会被舍弃，默认值为总特征个数开平方取整。
    'max_features': 0.5,
    'max_leaf_nodes': None,
    #限制信息增益的大小，信息增益小于设定数值的分支不会发生
    'min_impurity_decrease': 0.0,

    'bootstrap': True, #是否有放回的采样训练样本
    'oob_score': False, #是否采用袋外样本来评估模型的好坏
    'n_jobs': None, #并行数

    'random_state': None, #随机种子
    'verbose': 0, #是否显示任务进程
    'warm_start': False,

    'class_weight': None,
}
rfc = RandomForestClassifier(**params2)
```

与决策树相同，随机森林也有许多参数可供设置。其中，max_depth，min_samples_split，min_samples_leaf等与决策树同名的参数意义与在决策树中相同，不同的是随机森林分类器多了n_estimators(森林中树木的数量),bootstrap(是否有放回的采样训练样本),oob_score(是否采用袋外样本来评估模型的好坏)等与训练森林有关的参数。

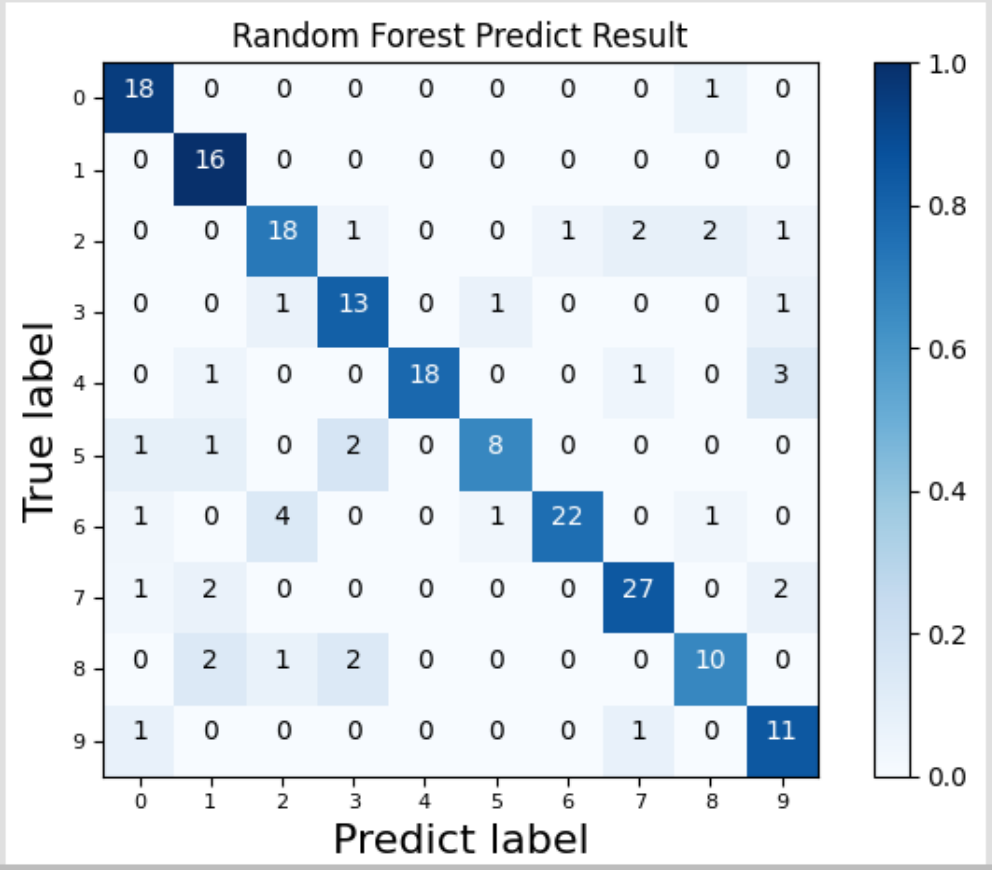
分类结果

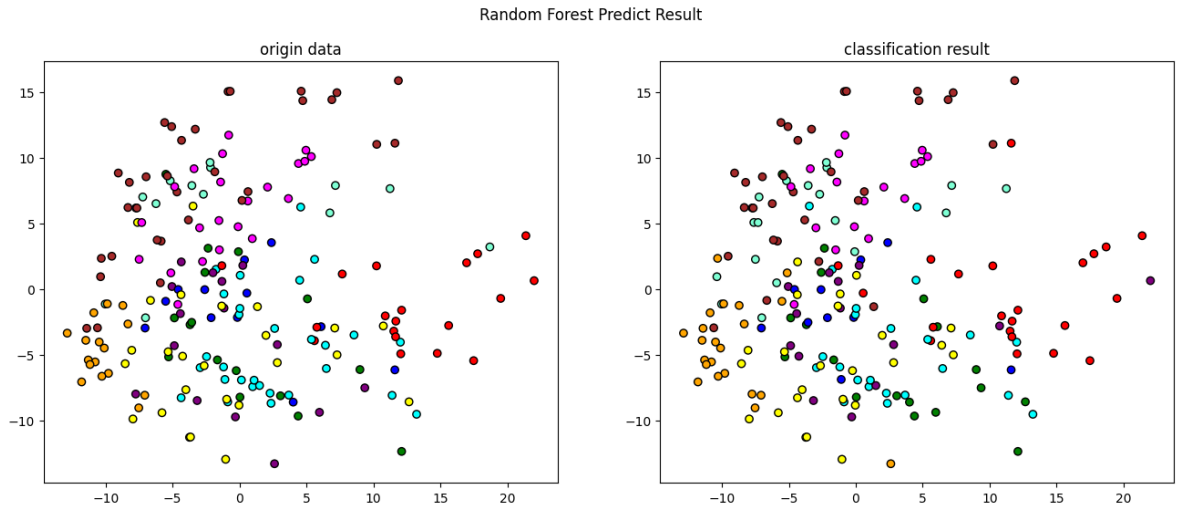
```
y_pred2 = rfc.predict(X_test)
print("Random Forest")
print(f'Accuracy: { accuracy_score(y_test, y_pred2) }')
#Precision = TP/TP+FP 宏平均
print(f'Precision: { precision_score(y_test, y_pred2, average="macro") }')
#Recall = TP/TP+FN
print(f'Recall: { recall_score(y_test, y_pred2, average="macro") }')
#F1-score = 2*precision*recall/(precision+recall)
print(f'F1-score { f1_score(y_test, y_pred2, average="macro") }')
```

```
Random Forest
Accuracy: 0.805
Precision: 0.7970563074139513
Recall: 0.8044334985847158
F1-score 0.7927504606375528
```

```
[123] cm2_2 = confusion_matrix(y_test, y_pred2)
visualize_cm(cm2_2,title='Random Forest Predict Result')
```

```
10
```





可以看到随机森林的分类效果比决策树更好，基本能够正确识别输入的类别。

3.3 SVM

使用sk-learn库函数sklearn.svm进行分类。

```
#svm
params3={
    "C": 1.0, #惩罚系数
    "kernel": "rbf", #核函数
    "degree": 3, #如果核函数是poly,多项式的维度
    "gamma": "scale", #核函数的系数('Poly', 'RBF' and 'Sigmoid')
    "coef0": 0.0, #核函数的常数项
    "shrinking": True, #是否使用启发式
    "probability": False, #是否启用概率估计
    "tol": 0.001, #停止训练的误差
    "cache_size": 200, #核函数的缓存大小
    "class_weight": None, #类别权重
    "verbose": False, #是否启用详细输出
    "max_iter": -1, #最大迭代次数
    "decision_function_shape": "ovr", #决策函数的形状
    "break_ties": False, #是否启用决策函数的平局处理
    "random_state": None, #随机种子
}
svmc = svm.SVC(**params3)
```

SVM也有许多参数，他们的作用如上。

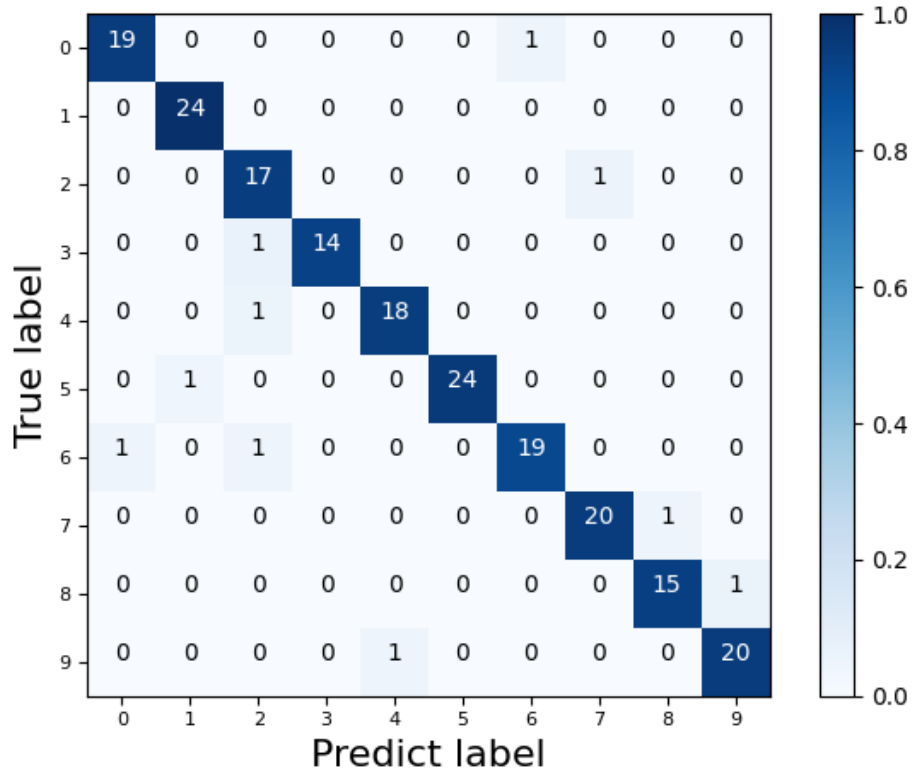
分类结果

```
145 ms
y_pred3 = svmc.predict(X_test)
print("SVM")
print(f'Accuracy: { accuracy_score(y_test, y_pred3) }')
#Precision = TP/TP+FP 宏平均
print(f'Precision: { precision_score(y_test, y_pred3, average="macro") }')
#Recall = TP/TP+FN
print(f'Recall: { recall_score(y_test, y_pred3, average="macro") }')
#F1-score = 2*precision*recall/(precision+recall)
print(f'F1-score { f1_score(y_test, y_pred3, average="macro") }')
```

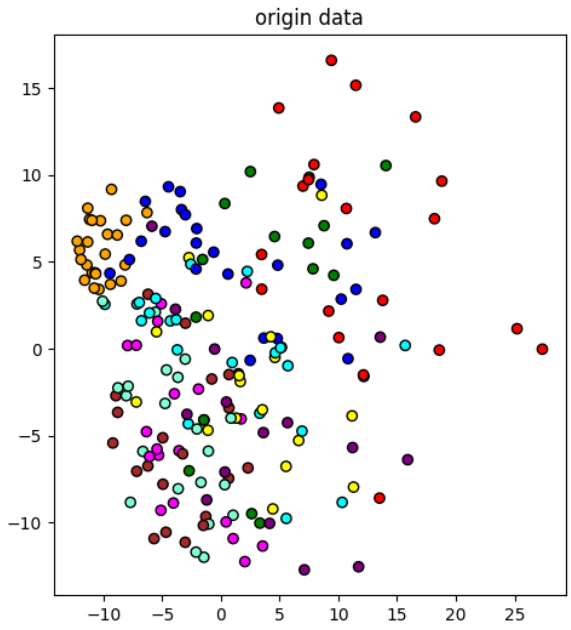
```
SVM
Accuracy: 0.95
Precision: 0.9499630325814536
Recall: 0.9482170008354218
F1-score 0.948589735106118
```

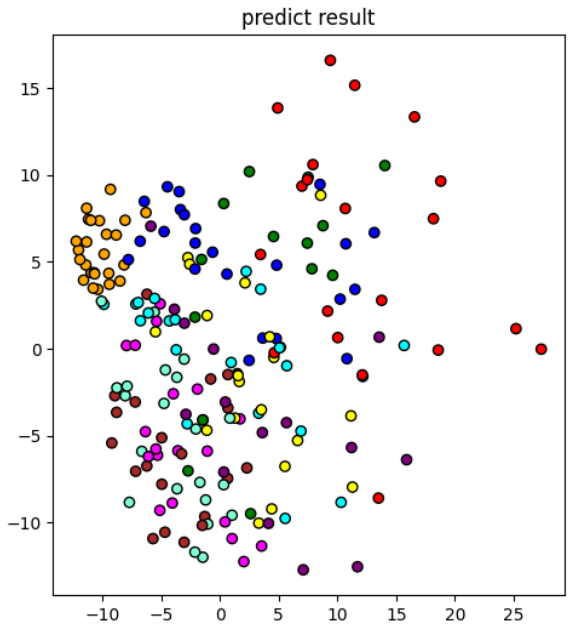
```
837 ms [11]
cm2_3 = confusion_matrix(y_test, y_pred3)
visualize_cm(cm2_3)
```

```
10
```



可以看到，SVM分类器的分类效果很好，准确率达到了95%,是三个分类器中分类效果最好的。





四、总结

1.本次实验通过使用sk-learn库中的分类算法Mnist数据集中的数据进行了分类。

2.sklearn库中的各种分类器都有许许多多的参数，在实际使用这些分类器进行训练时，要注意各种参数之间的作用，根据参数原理不断尝试调整参数，在不断迭代中选择最好的参数训练模型。

五、完整源代码

```
from sklearn.decomposition import PCA
from sklearn.manifold import TSNE
from sklearn.tree import DecisionTreeClassifier
import sklearn.svm as svm
from sklearn.ensemble import RandomForestClassifier

from sklearn.model_selection import train_test_split
import mindspore.dataset as ds
import mindspore.dataset.vision.c_transforms as C
import matplotlib.pyplot as plt

from sklearn.metrics import f1_score
from sklearn.metrics import accuracy_score
from sklearn.metrics import precision_score
from sklearn.metrics import recall_score
from sklearn.metrics import confusion_matrix

import numpy as np
import matplotlib.pyplot as plt
from matplotlib import rcParams
```

```
def visualize(X, y, y_pred, title=None):
    # 可视化,pca降维或者tsne

    pca = PCA(n_components=2)
    X_dim_reduction = pca.fit_transform(X)

    # print(X_dim_reduction.shape)
    plt.figure(figsize=(10, 12))
    def get_color(y):
        color_dict = {-1: 'white', 0: 'red', 1: 'orange', 2: 'yellow', 3:
'green', 4: 'magenta',
                      5: 'blue', 6: 'cyan', 7: 'brown', 8: 'purple', 9:
'aquamarine'}
        color = []
        for i in range(y.shape[0]):
            color.append(color_dict.get(y[i].item(), 'olive'))
        return color
    fig, axs = plt.subplots(1, 2, figsize=(12, 6))

    axs[0].set_title('origin data')
    axs[0].scatter(X_dim_reduction[:, 0], X_dim_reduction[:, 1], c=get_color(y),
edgecolors='black')

    axs[1].set_title('predict result')
```

```
    axs[1].scatter(X_dim_reduction[:, 0], X_dim_reduction[:, 1],
c=get_color(y_pred), edgecolors='black')

    plt.show()
```

```
def get_dataset(split, num):
    data_path = './datasets/MNIST_Data/' + split
    dataset = ds.MnistDataset(data_path)
    #归一化
    image_transforms = [
        C.Rescale(1.0 / 255.0, 0),
        C.Normalize(mean=(0.1307,), std=(0.3081,)),
        C.HWC2CHW()
    ]
    dataset = dataset.map(image_transforms, 'image')

    dataset = dataset.batch(num)
    #获取迭代器
    iterator_show = dataset.create_dict_iterator()
    dict_data = next(iterator_show)
    x = dict_data["image"].reshape(num, -1).asnumpy()
    y = dict_data["label"].asnumpy()
    X_train, X_test, y_train, y_test = train_test_split(x, y, test_size=0.2,
random_state=42)

    # pca = PCA(n_components=8)
    # X_train = pca.fit_transform(X_train)
    # X_test = pca.fit_transform(X_test)

    return X_train, X_test, y_train, y_test
```

```
X_train, X_test, y_train, y_test = get_dataset("train", 1000)
print(f'X_train.shape:{X_train.shape}')
print(f'y_train.shape:{y_train.shape}')
print(f'X_test.shape:{X_test.shape}')
print(f'y_test.shape:{y_test.shape}')
```

```
#降维数据
pca = PCA(n_components=2)
X_train_lower = pca.fit_transform(X_train)
X_test_lower = pca.fit_transform(X_test)
```

```
print(f'X_train_lower.shape:{X_train_lower.shape}')
print(f'X_test_lower.shape:{X_test_lower.shape}')
print(X_train_lower[1:10])
print(X_test_lower[1:10])
```

```
#pca 降维
pca = PCA(n_components=2)
x2_lower_pca = pca.fit_transform(x2)
#tsne 降维
t_sne = TSNE(n_components=2)
x1_lower_tsne = t_sne.fit_transform(x1)
```

```
print('x2_lower.pca.shape')
print(x2_lower_pca.shape)
print('x1_lower.tsne.shape')
print(x1_lower_tsne.shape)
```

```
#混淆矩阵可视化
def visualize_cm(cm2_1):
    classes = ['0','1', '2', '3', '4', '5', '6', '7', '8','9']
    proportion = []
    length = len(cm2_1)
    print(length)
    for i in cm2_1:
        for j in i:
            temp = j / (np.sum(i))
            proportion.append(temp)
    # print(np.sum(confusion_matrix[0]))
    # print(proportion)
    pshow = []
    for i in proportion:
        pt = "%.2f%%" % (i * 100)
        pshow.append(pt)
    proportion = np.array(proportion).reshape(length, length)  # reshape(列的长度，行的长度)
    pshow = np.array(pshow).reshape(length, length)

    # print(pshow)
    config = {
        "font.family": 'DejaVu Sans',  # 设置字体类型
    }
    rcParams.update(config)
    plt.imshow(proportion, interpolation='nearest', cmap=plt.cm.Blues)  # 按照像素显示出矩阵
    # (改变颜色：'Greys', 'Purples', 'Blues', 'Greens', 'Oranges', 'Reds','YlOrBr', 'YlOrRd',
    # 'OrRd', 'PuRd', 'RdPu', 'BuPu','GnBu', 'PuBu', 'YlGnBu', 'PuBuGn', 'BuGn', 'YlGn')
    # plt.title('confusion_matrix')
    plt.colorbar()
    tick_marks = np.arange(len(classes))
    plt.xticks(tick_marks, classes, fontsize=8)
    plt.yticks(tick_marks, classes, fontsize=8)

    iters = np.reshape([[[i, j] for j in range(length)] for i in range(length)], (cm2_1.size, 2))
    for i, j in iters:
        if (i == j):
            plt.text(j, i - 0.12, format(cm2_1[i, j]), va='center', ha='center', fontsize=10, color='white',
                     weight=5)  # 显示对应的数字

        else:
            plt.text(j, i - 0.12, format(cm2_1[i, j]), va='center', ha='center', fontsize=10)  # 显示对应的数字


    plt.ylabel('True label', fontsize=16)
```

```
    plt.xlabel('Predict label', fontsize=16)
    plt.tight_layout()
    plt.show()
```

```
# 决策树
params = {
    #特征选择标准
    'criterion': 'entropy',
    #特征划分点选择标准
    'splitter': 'best',
    #决策树最大深
    'max_depth': 15,
    #节点最小样本数
    'min_samples_split': 2,
    #叶子节点最小样本数
    'min_samples_leaf': 1,
    #叶子节点最小权重
    'min_weight_fraction_leaf': 0.0,
    #最大特征数
    'max_features': 0.5,
    #随机种子
    'random_state': None,
    #最大叶子节点数
    'max_leaf_nodes': None,
    #最小不纯度下降值
    'min_impurity_decrease': 0.0,
    #类别权重：防止训练集某些类别的样本过多，导致训练的决策树过于偏向这些类别
    'class_weight': None,
}
clf=DecisionTreeClassifier(**params)
clf2=DecisionTreeClassifier(**params)
```

```
n_clusters = 5
ypreds_Agglomerative_pca_lower =
AgglomerativeClustering(n_clusters=n_clusters).fit_predict(x1_lower_tsne)
Agglomerative_preds =
AgglomerativeClustering(n_clusters=n_clusters).fit_predict(x1)
visualize(x1,y1,Agglomerative_preds,title=f'Non-pca Agg n_clusters =
{n_clusters}')
visualize(x1_lower_tsne,y1,ypreds_Agglomerative_pca_lower,title=f'pca Agg
n_clusters = {n_clusters}')
```

```
clf.fit(X_train, y_train)
clf2.fit(X_train_lower,y_train)
```

```
y_pred = clf.predict(X_test)
print("desicion tree")
print(f'Accuracy: { accuracy_score(y_test, y_pred) }')
#Precision = TP/TP+FP 宏平均
print(f'Precision: { precision_score(y_test, y_pred, average="macro") }')
#Recall = TP/TP+FN
print(f'Recall: { recall_score(y_test, y_pred, average="macro") }')
#F1-score = 2*precision*recall/(precision+recall)
print(f'F1-score { f1_score(y_test, y_pred, average="macro") }')
cm2_1 = confusion_matrix(y_test, y_pred)
print(cm2_1.shape)
print('cm2 is:\n', cm2_1)
visualize_cm(cm2_1)
```

```
y_pred_lower = clf2.predict(X_test_lower)
print("desicion tree")
print(f'Accuracy: { accuracy_score(y_test, y_pred_lower) }')
#Precision = TP/TP+FP 宏平均
print(f'Precision: { precision_score(y_test, y_pred_lower, average="macro") }')
#Recall = TP/TP+FN
print(f'Recall: { recall_score(y_test, y_pred_lower, average="macro") }')
#F1-score = 2*precision*recall/(precision+recall)
print(f'F1-score { f1_score(y_test, y_pred_lower, average="macro") }')
cm2_1_lower = confusion_matrix(y_test, y_pred_lower)
print(cm2_1_lower.shape)
print('cm2 is:\n', cm2_1_lower)
visualize_cm(cm2_1_lower)
```

```
print('Decision Tree Predict Result')
visualize(X_test,y_test,y_pred,title='Decision Tree Predict Result')
```

```
print('Desicion Tree Predict Result(using Tsne)')
visualize(X_test_lower,y_test,y_pred_lower,title='Desicion Tree Predict
Result(using PCA)')
```

```
# 随机森林

params2={
    #森林中树木的数量
    'n_estimators': 100,
    'criterion': 'entropy',
    #树的最大深度，超过最大深度的树枝都会被剪掉
    'max_depth': 10,

    #一个节点在分支后的每个子节点都必须包含至少min_samples_leaf个训练样本，否则分支就不会发
生
    'min_samples_split': 2,

    #一个节点必须要包含至少min_samples_split个训练样本，这个节点才被允许分支，否则分支不会发
生。
    'min_samples_leaf': 1,

    'min_weight_fraction_leaf': 0.0,
```

```
    #max_features限制分支时考虑的特振奋个数，超过限制个数的特征都会被舍弃，默认值为总特征个数开平方取整。
    'max_features': 0.5,
    'max_leaf_nodes': None,
    #限制信息增益的大小，信息增益小于设定数值的分支不会发生
    'min_impurity_decrease': 0.0,

    'bootstrap': True, #是否有放回的采样训练样本
    'oob_score': False, #是否采用袋外样本来评估模型的好坏
    'n_jobs': None, #并行数

    'random_state': None, #随机种子
    'verbose': 0, #是否显示任务进程
    'warm_start': False,

    'class_weight': None,
}
rfc = RandomForestClassifier(**params2)
```

```
rfc.fit(X_train, y_train)
y_pred2 = rfc.predict(X_test)
print("desicion tree")
print(f'Accuracy: { accuracy_score(y_test, y_pred2) }')
#Precision = TP/TP+FP 宏平均
print(f'Precision: { precision_score(y_test, y_pred2, average="macro") }')
#Recall = TP/TP+FN
print(f'Recall: { recall_score(y_test, y_pred2, average="macro") }')
#F1-score = 2*precision*recall/(precision+recall)
print(f'F1-score { f1_score(y_test, y_pred2, average="macro") }')
```

```
cm2_2 = confusion_matrix(y_test, y_pred2)
print(cm2_2.shape)
print('cm2 is:\n', cm2_2)py
visualize_cm(cm2_2)
```

```
print('Random Forest Predict Result')
visualize(X_test,y_test,y_pred2,title='Random Forest Predict Result')
```

```
#svm
params3={
    "C": 1.0, #惩罚系数
    "kernel": "rbf", #核函数
    "degree": 3, #如果核函数是poly,多项式的维度
    "gamma": "scale", #核函数的系数('Poly', 'RBF' and 'Sigmoid')
    "coef0": 0.0, #核函数的常数项
    "shrinking": True, #是否使用启发式
    "probability": False, #是否启用概率估计
    "tol": 0.001, #停止训练的误差
    "cache_size": 200, #核函数的缓存大小
    "class_weight": None, #类别权重
    "verbose": False, #是否启用详细输出
    "max_iter": -1, #最大迭代次数
    "decision_function_shape": "ovr", #决策函数的形状
    "break_ties": False, #是否启用决策函数的平局处理
```

```
    "random_state": None, #随机种子
}
svmc = svm.SVC(**params3)
```

```
svmc.fit(X_train,y_train)
```

```
y_pred3 = svmc.predict(X_test)
print("SVM")
print(f'Accuracy: { accuracy_score(y_test, y_pred3) }')
#Precision = TP/TP+FP 宏平均
print(f'Precision: { precision_score(y_test, y_pred3, average="macro") }')
#Recall = TP/TP+FN
print(f'Recall: { recall_score(y_test, y_pred3, average="macro") }')
#F1-score = 2*precision*recall/(precision+recall)
print(f'F1-score { f1_score(y_test, y_pred3, average="macro") }')
```

```
cm2_3 = confusion_matrix(y_test, y_pred3)
visualize_cm(cm2_3)
```

```
visualize(X_test,y_test,y_pred3,title='SVM Predict Result')
```