

合肥工业大学

数据挖掘实验报告

实验名称： **数据集的分类/关联/聚类

姓名： ***

班级： 计科16-2班

学号： 20*****

目录

1	基于 Molecular Biology DataSet 的分类任务	1
1.1	实验目的	1
1.2	实验任务	1
1.3	实验环境	1
1.4	实验内容	1
1.4.1	数据清洗	1
1.4.2	扩增数据集	1
1.4.3	特征映射	2
1.4.4	决策树方法	2
1.4.5	随机森林方法	3
1.4.6	KNN 方法	4
1.4.7	SVC 方法	5
1.5	实验分析	6
1.5.1	原数据集	6
1.5.2	扩增数据集	7
1.5.3	特征映射数据集	8
1.5.4	扩增 + 特征映射数据集	9
1.5.5	总述	10
1.6	实验总结	10

实验 1: 基于 Molecular Biology DataSet 的分类任务

1.1 实验目的

- 熟练掌握基本的数据预处理技术;
- 学会运用决策树、随机森林等方法解决分类任务。

1.2 实验任务

基于 Molecular Biology DataSet 完成分类任务, 决策树, random forest, bagging, boosting 方法任选或组合, 且不限于上述方法和策略, 允许有预处理步骤。

1.3 实验环境

- OS: Window10
- 开发环境: PyCharm、Python3.5、numpy、sklearn

1.4 实验内容

1.4.1 数据清洗

数据集中有 instance name 无用属性, 我们首先将该属性值删除, 只保留 57 个基因序列和标签值。具体操作是利用 split 函数将 instance name 分割出来, 然后将有用的特征值保留即可。实现代码如下所示:

```
1 file=open('dataset/promoters.data')
2 x=[]
3 y=[]
4 for i,line in enumerate(file.readlines()):
5     line=line.strip('\n')
6     line=line.split('\t')
7     x.append(line[-1])
8     y.append(line[0].split(',')[0])
```

1.4.2 扩增数据集

已知 Molecular Biology 数据集有两类标签: + 和-, 属性值有 57 个, 取值分别为 a、c、g、t, 共有 106 条记录。为了合理扩充数据集来增强模型泛化能力, 我采用同类型数据拼接的方法, 将每条记录的属性值增加到 $57*2=114$ 个。

具体操作为将每条记录的属性值与它下一条记录的属性值拼接起来，形成属性值为 114 的记录，从而保证在标签不变的情况下，扩充每条记录的维度，实现代码如下所示：

```
1 for i in range(52):
2     x[i]=x[i]+x[i+1]
3 x[52]=x[52]+x[0]
4 for i in range(53, 105):
5     x[i]=x[i] + x[i + 1]
6 x[105]=x[105] + x[53]
```

1.4.3 特征映射

已知 Molecular Biology 数据集是一种生物基因序列数据集，每条记录的基因序列存在某种序列关系，若将每个基因点作为一个属性，势必会忽略每个基因点与前后基因点的逻辑关系。为了便于挖掘基因序列中的序列关系，我在实验中对每条记录进行了特征映射。

首先算出 a、g、c、t 四种基因所有的搭配组合（共 256 种），之后遍历扩增后的数据集，计算每条记录中计算每种组合出现的次数，从而将每条记录的属性维度由扩增后的 114 维转换为特征映射后的 256 维。实现代码如下所示：

```
1 one=['a','g','c','t']
2 four=[]
3 new_x=[]
4 for x1 in one:
5     for x2 in one:
6         for x3 in one:
7             for x4 in one:
8                 four.append(x1+x2+x3+x4)
9 for xx in x:
10     count=''
11     for i in range(len(four)):
12         count+=str(xx.count(four[i]))+' '
13     new_x.append(count)
```

1.4.4 决策树方法

实验过程中我主要使用 sklearn 函数包来实现决策树方法。sklearn 决策树算法类库内部实现是使用了调优过的 CART 树算法，既可以做分类，又可以做回归。分类决策树的类对应的是 DecisionTreeClassifier。实现代码如下所示：

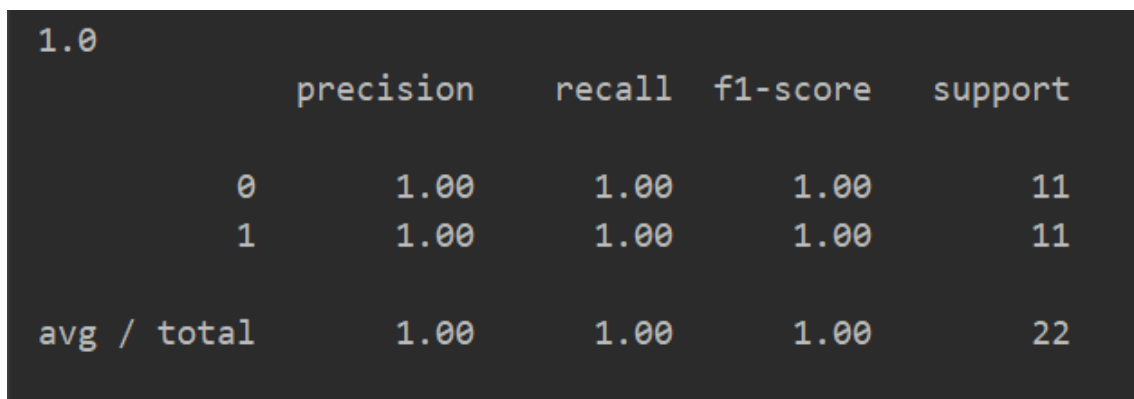
```
1 x_train,x_test,y_train,y_test=train_test_split(x,y,test_size=0.2)
```

```

2 clf=tree.DecisionTreeClassifier(criterion="entropy")
3 clf.fit(x_train,y_train)
4 y_predict=clf.predict(x_test)
5 print(clf.score(x_test,y_test))
6 print(classification_report(y_test,y_predict))

```

首先将数据集按照 8:2 的比例划分为训练集和测试集，分别赋给 x_{train} , x_{test} , y_{train} , y_{test} 。之后直接调用 DecisionTreeClassifier 函数建立决策树，其中参数 criterion= “entropy” 表示决策树使用的是 ID3 算法。再将分割后的训练集传入训练，针对 x_{test} 进行预测，计算输出预测准确率。程序输出结果如1所示。



	precision	recall	f1-score	support
0	1.00	1.00	1.00	11
1	1.00	1.00	1.00	11
avg / total	1.00	1.00	1.00	22

图 1: 决策树预测结果

1.4.5 随机森林方法

实验过程中我主要使用 sklearn 函数包来实现决策树方法。在机器学习中，随机森林是一个包含多个决策树的分类器，并且其输出的类别是由个别树输出的类别的众数而定。利用相同的训练数搭建多个独立的分类模型，然后通过投票的方式，以少数服从多数的原则作出最终的分类决策。

已知一个标准的决策树会根据每维特征对预测结果的影响程度进行排序，进而决定不同的特征从上至下构建分裂节点的顺序，如此以来，所有在随机森林中的决策树都会受这一策略影响而构建的完全一致，从而丧失的多样性。所以在随机森林分类器的构建过程中，每一棵决策树都会放弃这一固定的排序算法，转而随机选取特征。具体的实现代码如下所示：

```

1 rfc=RandomForestClassifier()
2 rfc.fit(x_train,y_train)
3 y_predict = rfc.predict(x_test)
4 print(rfc.score(x_test, y_test))
5 print(classification_report(y_test, y_predict))

```

首先将数据集按照 8:2 的比例划分为训练集和测试集，分别赋给 x_{train} , x_{test} , y_{train} , y_{test} 。然后直接调用 RandomForestClassifier() 函数建立随机森林，森林里的决策树数量取默

取值 10。再将分割后的训练集传入训练，之后基于测试机进行测试，输出预测结果。程序输出结果如2所示。

1.0				
	precision	recall	f1-score	support
0	1.00	1.00	1.00	9
1	1.00	1.00	1.00	13
avg / total	1.00	1.00	1.00	22

图 2: 随机森林预测结果

1.4.6 KNN 方法

实验过程中我主要使用 sklearn 函数包来实现 KNN 方法。KNN 算法的核心思想是如果一个样本在特征空间中的 k 个最相邻的样本中的大多数属于某一个类别，则该样本也属于这个类别，并具有这个类别上样本的特性。该方法在确定分类决策上只依据最邻近的一个或者几个样本的类别来决定待分样本所属的类别。

KNN 的伪代码思想：

对未知类别属性的数据集中的每个点依次执行以下操作：

- 计算已知类别数据集中的点与当前点之间的操作；
- 按照距离递增次序排序；
- 选取与当前点距离最小的 k 个点；
- 确定前 k 个点所在类别出现的概率；
- 返回前 k 个点出现概率最高的类别作为当前点的预测分类；

Python 实现代码如下所示：

```
1 x_train,x_test,y_train,y_test=train_test_split(x,y,test_size=0.2)
2 knn=KNeighborsClassifier()
3 knn.fit(x_train,y_train)
4 y_predict = clf.predict(x_test)
5 print(clf.score(x_test, y_test))
6 print(classification_report(y_test, y_predict))
```

首先将数据集按照 8:2 的比例划分为训练集和测试集，分别赋给 x_{train} , x_{test} , y_{train} , y_{test} 。然后直接调用 `KNeighborsClassifier()` 函数建立 KNN 模型，这里超参数 k 取默认值 5。之后传入训练集，再基于测试集进行测试，计算输出预测结果。程序输出结果如3所示。

1.0				
	precision	recall	f1-score	support
0	1.00	1.00	1.00	10
1	1.00	1.00	1.00	12
avg / total	1.00	1.00	1.00	22

图 3: KNN 预测结果

1.4.7 SVC 方法

实验过程中我主要使用 `sklearn` 函数包来实现 SVC 方法。SVC 全称为 C-Support Vector Classification，即支持向量分类，是一种基于 `libsvm` 实现的，时间复杂度为样本数量平方的算法。Python 实现代码如下所示：

```

1 x_train,x_test,y_train,y_test=train_test_split(x,y,test_size=0.2)
2 clf=SVC()
3 clf.fit(x_train,y_train)
4 y_predict = clf.predict(x_test)
5 print(clf.score(x_test, y_test))
6 print(classification_report(y_test, y_predict))

```

首先将数据集按照 8:2 的比例划分为训练集和测试集，分别赋给 x_{train} , x_{test} , y_{train} , y_{test} 。之后调用 `SVC()` 函数建立 SVC 模型，再将分割后的训练集传入训练，之后基于测试集进行测试，输出预测结果。程序输出结果如4所示。

	precision	recall	f1-score	support
0	1.00	1.00	1.00	10
1	1.00	1.00	1.00	12
avg / total	1.00	1.00	1.00	22

图 4: SVC 输出结果

1.5 实验分析

1.5.1 原数据集

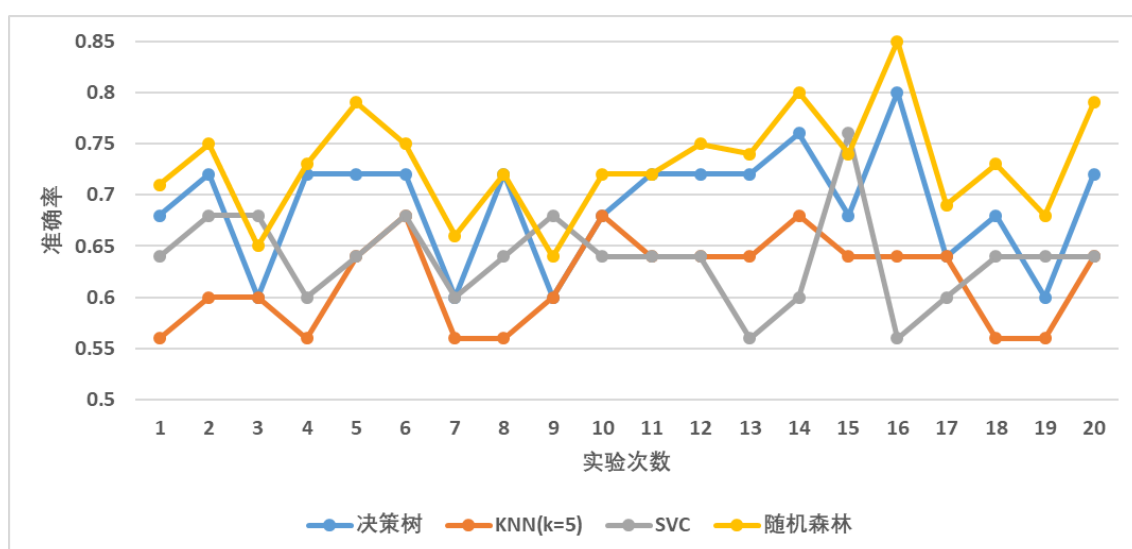


图 5: 原数据集预测结果

为避免实验的偶然性，我们在原数据集（未扩增、未映射）上对每种方法进行了 20 次实验，实验预测准确率的变波动情况如5所示。

由图 5 可以看出，每种方法的准确率均在 55% 以上，但绝大部分都没有超过 80%，维持在中等水平。其中随机森林方法的性能较为突出，始终优于其他方法。这说明相比于其他单一结构的模型，在未处理的原数据集下，随机森林方法凭借“多数服从少数”这种思想取得了一定的优势，增强了模型的鲁棒性。

但直接使用原数据集进行训练会导致对基因序列信息的忽略，这是为什么所有模型的准确率都低于 85% 的原因。

1.5.2 扩增数据集

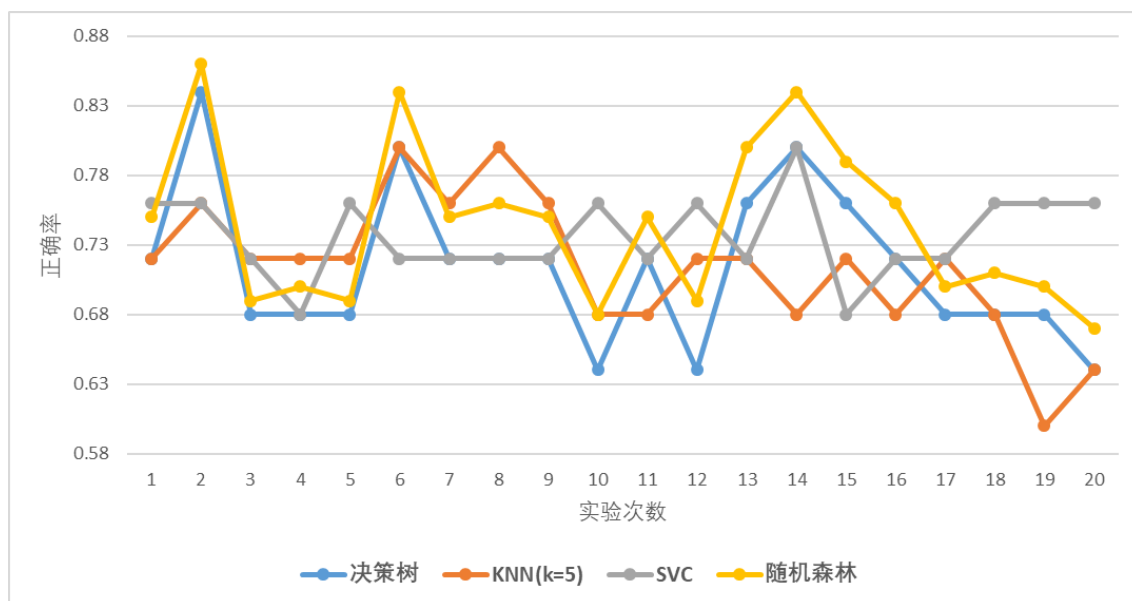


图 6: 扩增数据集预测结果

同样在扩增数据集上进行了 20 次反复性实验，实验结果如6所示。

对比图 5 和图 6 可以发现：扩增数据集后，所有模型的准确率均得到了一定的提升，绝大部分模型的准确率均高于 63%，这说明预处理时采取的扩增数据操作有助于提升模型的泛化能力，使模型更好地学习如何精确分类。

其中随机森林模型的性能仍然处于领先地位，始终优于其他模型。这说明对于属性种类较多、较杂的数据，随机森林方法能够挖掘特征之间的关系，找出最相关的属性，从而准确地进行分类，具有更高的准确率。

1.5.3 特征映射数据集

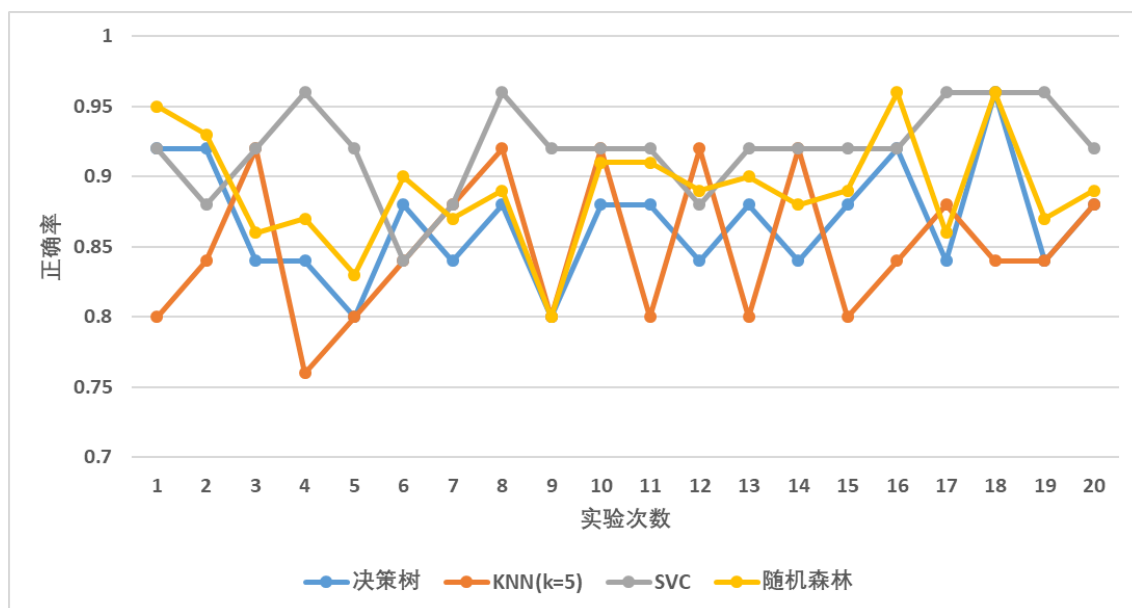


图 7: 特征映射数据集

我们在特征映射数据集上进行了 20 次反复性实验，实验结果如7所示。

从图 7 中可以看出，所有模型的准确率均得到了提升，准确率平均保持在 75% 以上，最高达到了 95%。与原数据集和扩增数据集不同的是，在特征映射数据集上，SVC 模型的准确率较高，甚至达到了 95%+，而随机森林的准确率较以往的上升幅度并不凸显。这说明特征映射使得数据更加关注基因的序列信息，具有一定现实意义的属性值更有助于 SVC 这类算法的学习，随机森林这类算法的优势则被缩小了。

1.5.4 扩增 + 特征映射数据集

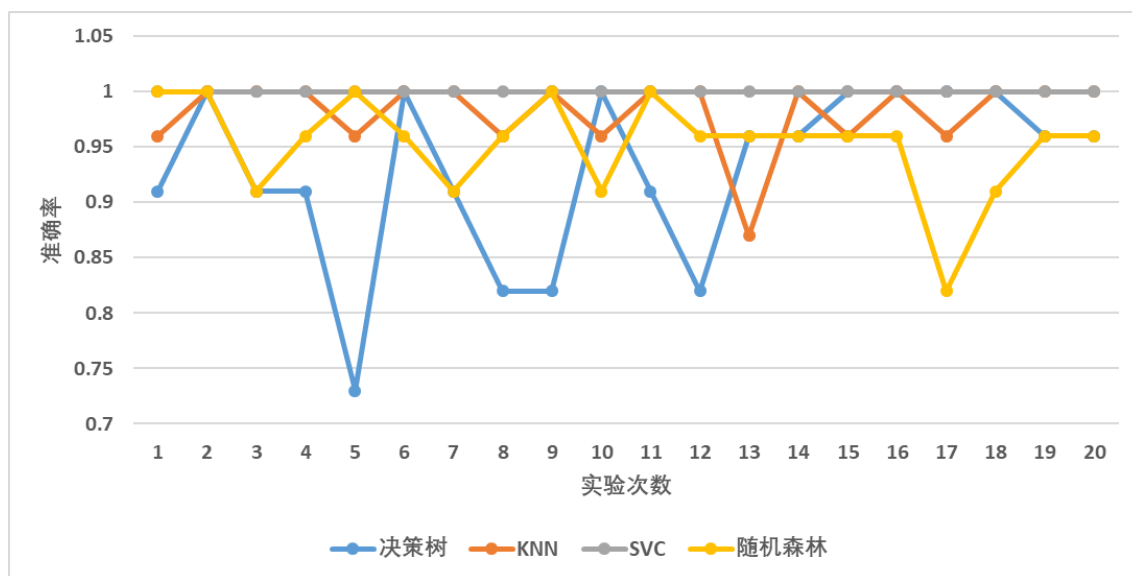


图 8: 扩增特征映射数据集预测结果

我们在特征映射数据集上进行了 20 次反复性实验，实验结果如8所示。

最终在扩增特征映射数据集上进行了 20 次反复性实验，实验结果如图 8 所示。

从图 8 中可以看出，SVC 和 KNN 算法的准确率得到了很大的提升，尤其是 SVC 算法，20 次实验的准确率均为 100%，KNN 算法的准确率也保持在 95% 以上。但决策树和随机森林的准确率却较低，且波动较大，单棵决策树的准确率波动较强烈，模型性能不稳定。

实验结果表明：决策树和随机森林算法适用于数据维度较大、数据属性不明显的数
据，通过模型的数量来提升整体性能；SVC 和 KNN 算法更适用于数据属性有现实意义的
数据，这更有助于帮助他们学习特征与标签之间的内在关系，更准确地进行分类。

1.5.5 总述

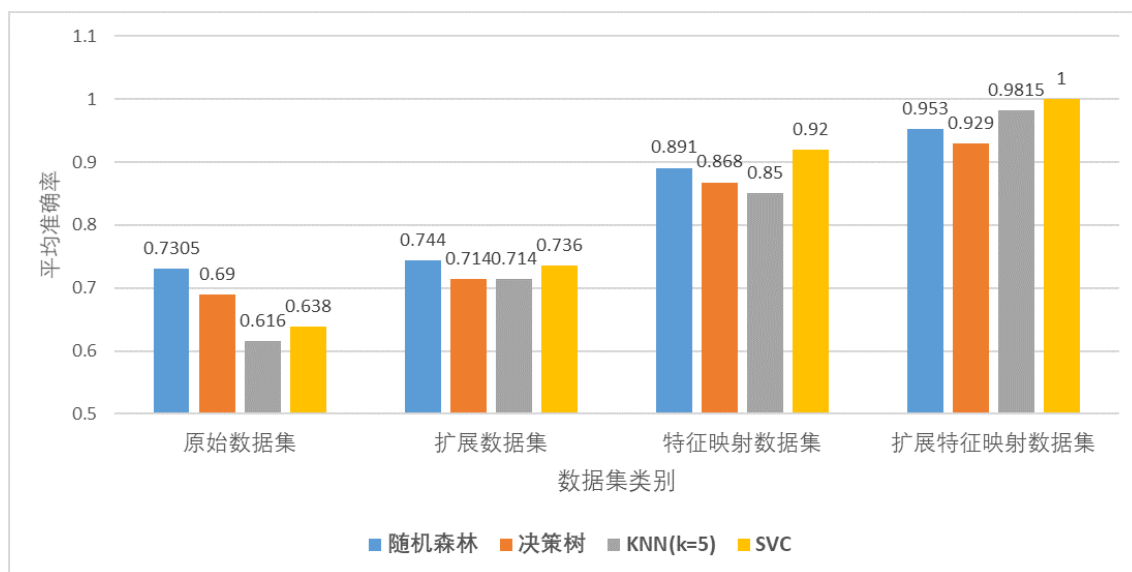


图 9: 不同数据集下的预测结果

我们计算了不同数据集上各模型的平均准确率，实验结果如9所示。

从图 9 中可以看出，每一步数据预处理都会使模型的准确率得到进一步提升。在原始数据集和扩展数据集上，随机森林和决策树算法的性能较好；在特征映射数据集和扩展特征映射数据集上，SVC 和 KNN 算法的性能较为突出，直至最终平均准确率达到了 100

1.6 实验总结

- 通过本次实验，我熟悉了 Molecular Biology 数据集这类序列数据集，这是一个主流的分类数据集，在多篇论文中被反复引用；
- 我还学会了数据预处理的相关技术，包括数据补缺、特征映射等等，尤其是对于这种序列数据的扩增和特征映射，扩展了我对数据预处理的认知；
- 本次实验最大的收获在于加深了对决策树、随机森林、KNN 和 SVC 算法的理解。学会通过 sklearn 包实现各算法的训练和预测。通过对比各个算法在数据集上的准确率，从而直观地感受到不同算法的差异所在，学会观察数据的特点和分布，从而选择最合适的算法。