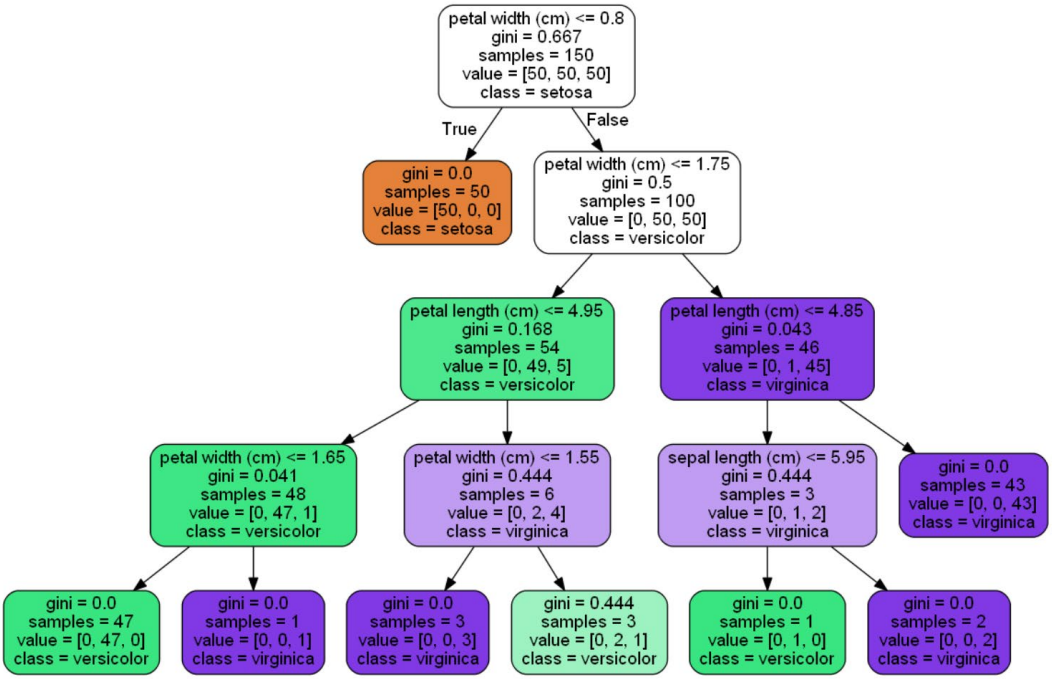
作业 3 Readme

191250111 裴为东

内容：运用 SVM、决策树、BPNN 对 IRIS 鸢尾花数据集进行分类任务

1、 决策树



决策树是一种树形结构，其中每一个内部节点表示一个属性上的判断，每一个分支代表一个判断结果的输出，每一个叶节点代表代表一种分类结果。

在构建决策树模型时，我们需要做的就是将数据集按照一个个节点不断划分，比如在鸢尾花数据集中，根据花瓣宽度是否小于等于 0.8 我们就可以区分出 setosa 类别和非 setosa 类别的数据。那么在一堆单纯的数据参数中，我们是怎么想到应该将花瓣宽度 0.8cm 作为一个节点呢，这就涉及到节点的选择问题，对此常用的三中算法分别是 ID3（使用信息增益进行特征选择）、C4.5（使用信息增益率）、CART（使用基尼系数）。本文讲涉及 ID3

算法的运用。

在了解信息增益之前，我们需要了解纯度和信息熵。决策树的构建过程可以理解为寻找纯净划分的过程，目的是让每一个划分的纯度尽可能高，即让目标变量的分歧最小，比如对于是否吃饭，集合 1 包含 5 次吃饭，集合 2 包含 3 次吃饭 2 次不吃饭，集合 3 包含 4 次吃饭 1 次不吃饭，那么他们的纯度大小关系是集合 1>集合 3>集合 2，即集合 1 中变量的分歧最小纯度最大。信息熵描述了信息的不确定性，当信息熵越高，表示信息的不确定性越大，它的纯度越低。当集合中的所有样本均匀混合时，信息熵最大，纯度最低。

信息增益就是指通过划分实现纯度的提高和信息熵的下降,在决策树构建时通过计算各特征的信息增益选择信息增益最大的特征作为根节点，再对子树进行同样的处理实现一个递归的划分。

调用 api 实现：

引入 iris 鸢尾花数据集，通过 sklearn 直接引入

引入数据库

```
: from sklearn.datasets import load_iris
  import numpy as np
  iris = load_iris()
  iris
```

将数据集分为训练集和测试集，训练集用于构建模型，测试用于验证我们所构建的模型的精度。使用了 sklearn 中的 train_test_split 方法对数据集进行划分，

此方法会对数据集进行随机划分，其中默认 75%的数据将作为训练集，剩下 25%的数据将作为测试集

将数据分为训练集和测试集

```
from sklearn.model_selection import train_test_split
X_train, X_test, Y_train, Y_test = train_test_split(iris['data'],iris['target'],random_state = 0)
```

构建决策树时可以直接调用 sklearn 中的 api 构建决策树模型，

构建决策树模型

```
from sklearn import tree
clf_tree=tree.DecisionTreeClassifier(max_depth=4)
clf_tree = clf_tree.fit(X_train,Y_train)
```

验证决策树模型的精度，如下图，结果一为测试集精度，结果二为训练集精度

决策树精度

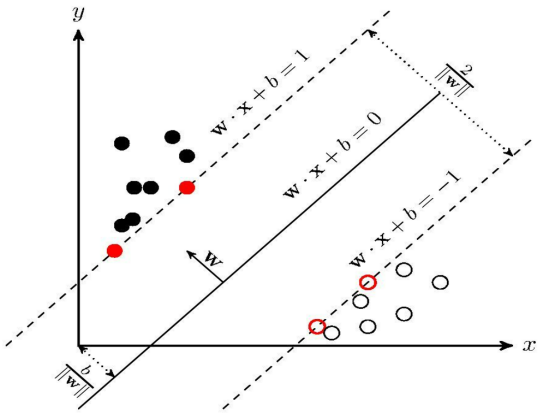
```
np.mean(clf_tree.predict(X_test)==Y_test)
```

```
0.9736842105263158
```

```
clf_tree.predict(X_train)
np.mean(clf_tree.predict(X_train)==Y_train)
```

```
1.0
```

2、 SVM



支持向量机是一种二分类模型，其基本模型是定义在特征空间上的间隔最大的线性分类器。如上图所示，我们需要在白圈和黑圈找到一个分割边界，使得各个白圈和黑圈离这个分割边界都尽量远，以此实现对白圈和黑圈的分类。

调用 api 实现：

引入数据集和划分数据集操作同上

构建 SVM 模型。其中 api 函数调用的参数 C 表示误差项惩罚系数，当 C 越大表示对误分类的惩罚越大，可以提高训练的准确率但是泛化能力会减弱；当 C 越小时，表示对误分类的惩罚越小，容错率增高，泛化能力提高。参数 kernel=linear 表示次模型构建的核函数为线性核，当线性划分无法满足数据划分的需求时可以通过改变核函数使得 SVM 可以实现更高维度的分类，比如使用高斯核。参数 decision_function_shape='ovr'表示将一个类别与其他类别进行划分，若参数为值设置为 ovo，表示将类别两两之间进行划分。

```
from sklearn import svm
clf_svm_linear = svm.SVC(C = 1.0,
                         kernel = 'linear',
                         decision_function_shape = 'ovr')
```

验证 SVM 模型精度，如下图，其中结果一为测试集精度，结果二为训练集精度

SVM精度

```
np.mean(clf_svm_linear.predict(X_test)==Y_test)
```

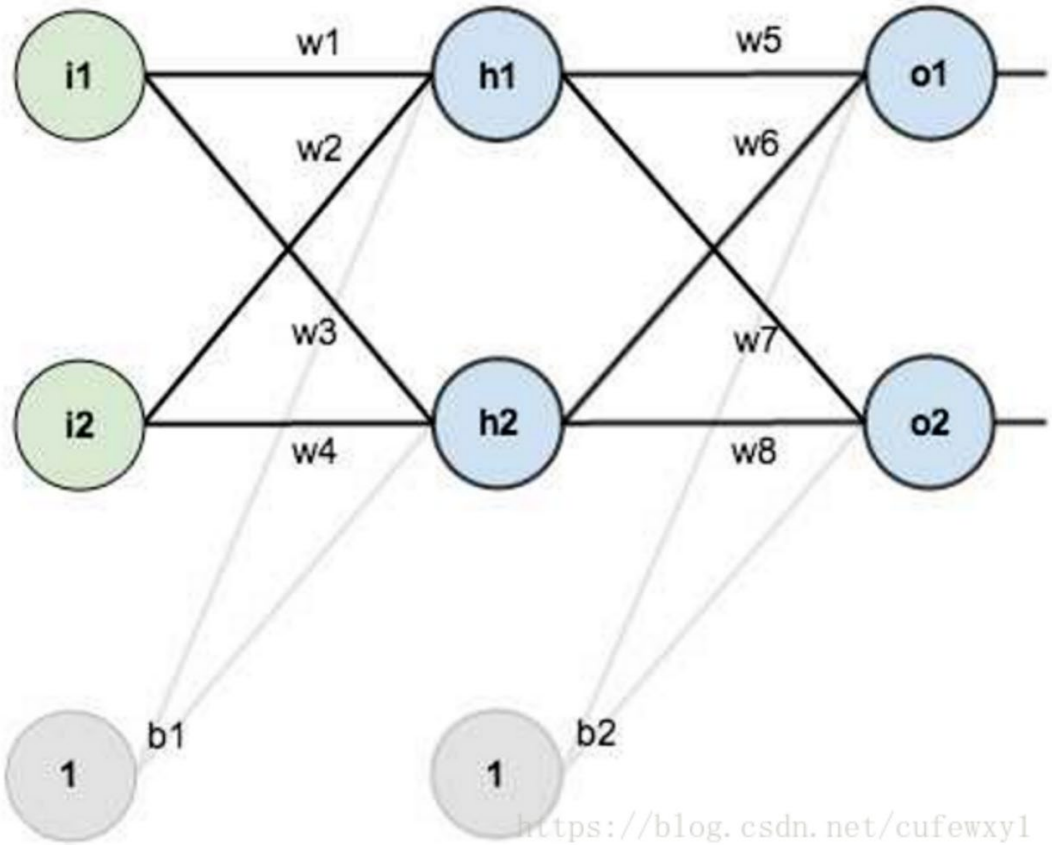
```
0.9736842105263158
```

```
clf_svm_linear.predict(X_train)
np.mean(clf_svm_linear.predict(X_train)==Y_train)
```

```
0.9821428571428571
```

3、 BPNN

BPNN（Back Propagation Neural Network）通过神经网络正向传播输出结果，通过反向传播方式传递误差，并对网络中的参数进行优化，实现一个神经网络的训练。



如图是一个 MLP 示意图，包含 3 个层次，从左往右分别是输入层、隐藏层、输

出层

向前传播:

如图 w 是连接前后两层节点的权重，b 是偏置，后一层节点的输入等于所连接的前一层节点的输出乘以其权重加上偏置，比如 h1 的输入为 i1*w1 + i2*w2 + b1*1；隐藏层的输出为其输入带入激活函数的值，这个步骤被称为激活，本次使用的激活函数是 Sigmoid 函数，

$$y = \frac{1}{1+e^{-x}}$$

反向传播:

我们通过正向传播得到了一个输出，我们使用的参数是一个随机值，接下来就需要通过反向传播调整参数的值，使其成为一个合格的神经网络模型。

$$\text{Cost} = \frac{1}{2}(target_{o_1} - out_{o_1})^2 + \frac{1}{2}(target_{o_2} - out_{o_2})^2$$

如图为损失函数，表示我们输出值和预期值的偏差，通过对权重和偏置求偏导我们可以找到合适的权重和偏置的值使得这个损失函数的值尽可能的小

隐藏层:

在神经网络中，隐藏层层数大于 0 时用于处理需要非线性分离的数据，对于一般简单的数据集，1 到 2 层隐藏层通常就足够了，理论上讲，隐藏层层数越多，拟合函数的能力越强，效果可能更好，但是更可能出现过拟合的问题，训练难度更大，模型更加难以收敛。本次实现根据网络上已有的实现经验决定设置一层隐藏层。

隐藏层中使用的节点太少会导致欠拟合的问题，而使用的节点过多会导致过拟合的问题，所以我们需要选择合适的节点数。

节点数的设置有许多经验之谈

stackoverflow上有大神给出了经验公式以供参考：

$$N_h = \frac{N_s}{(\alpha * (N_i + N_o))}$$

> 其中： N_i 是输入层神经元个数；
> N_o 是输出层神经元个数；
> N_s 是训练集的样本数；
> α 是可以自取的任意值变量，通常范围可取 2-10。

还有另一种方法可供参考，神经元数量通常可以由一下几个原则大致确定：

- **隐藏神经元的数量应在输入层的大小和输出层的大小之间。**
- **隐藏神经元的数量应为输入层大小的2/3加上输出层大小的2/3。**
- **隐藏神经元的数量应小于输入层大小的两倍。**

本次实现采用了第一条准则，最终选取 10 作为节点数，可获得较好的拟合效果

代码实现：

本次实现将鸢尾花数据集的 80%作为训练集，剩下的 20%作为测试集

构建的神经网络包含一层隐藏层，输入层有 4 个节点对应数据的 4 个特征，隐藏层有 10 个节点，输出层有 3 个节点对应数据的 3 个类型。

代码实现见 main.py

输出：拟合精度在 93.33%与 100%之间，出错数经过观察小于等于 1

预测结果： [1. 2. 0. 2. 0. 2. 2. 1. 1. 1. 0. 2. 1. 2. 1. 0. 0. 1. 1. 2. 1. 2. 2. 0.
 2. 0. 0. 1. 0. 1.]
真实结果： [1. 2. 0. 2. 0. 2. 2. 1. 1. 1. 0. 2. 1. 2. 1. 0. 0. 1. 1. 1. 1. 2. 1. 0.
 2. 0. 0. 1. 0. 1.]
错误分类样本序号： 20
错误分类样本序号： 23
BPNN精度：93.33%

预测结果： [2. 0. 0. 2. 1. 2. 0. 2. 0. 2. 0. 0. 1. 0. 0. 1. 2. 2. 0. 2. 0. 1. 2. 2.
 2. 2. 2. 1. 0. 2.]
真实结果： [2. 0. 0. 2. 1. 2. 0. 2. 0. 2. 0. 0. 1. 0. 0. 1. 2. 2. 0. 2. 0. 1. 2. 2.
 2. 2. 2. 1. 0. 2.]
BPNN精度：100.00%