华东师范大学数据科学与工程学院实验报告

课程名称：数据科学与工程算法基础	**年级：**2020	**上机实践成绩：**
指导教师：董启文	**姓名：**李昕原	**学号：**10205501425
上机实践名称：Project1		**上机实践日期：**11.28
上机实践编号：	**组号：**	**上机实践时间：**

一、 实验目的

- UCI 数据集：http://archive.ics uci.edu/ml/index.php
- 任选一个数据集
- 任选一种 ML 算法:逻辑回归、决策树、神经网络、SVM 等
- 源码+实验报告

二、 实验任务

本次实验选取 UCI 数据集中的 wine 数据集进行实验(https://archive.ics.uci.edu/ml/datasets/wine)，该数据集是一个三分类问题（同样可以作为回归问题），依据酒精、碱度等特征判断葡萄酒所处的类。本次实验将通过 SVM、softmax 回归、决策树、多层感知机、随机森林、集成学习等多种算法解决该三分类问题，并基于简单验证与五折交叉验证得到的结果对各个模型进行结果比较。

三、 实验过程

首先载入数据及特征标签，查看数据的基本信息：

```
Data = pd.read_csv("wine.csv")
#增加列名
Data.columns = ["class", "alcohol", "malic acid", "ash", "alcalinity", "magnesi
um", "phenols", "flavanoids", "noflavanoid", "proanthocyanins", "color intensit
y", "hue", "diluted wines", "proline"]
print(Data.info())
print(Data.head(10))
n = Data.shape[0]
p = Data.shape[1] - 1
print("数据集样本数为",n)
print("数据集特征数为",p)
```

```
RangeIndex: 177 entries, 0 to 176
Data columns (total 14 columns):
 #   Column           Non-Null Count  Dtype
---  ------           --------------  -----
 0   class            177 non-null    int64
 1   alcohol          177 non-null    float64
 2   malic acid       177 non-null    float64
 3   ash              177 non-null    float64
 4   alcalinity       177 non-null    float64
 5   magnesium        177 non-null    int64
 6   phenols          177 non-null    float64
 7   flavanoids       177 non-null    float64
 8   noflavanoid      177 non-null    float64
 9   proanthocyanins  177 non-null    float64
 10  color intensity  177 non-null    float64
 11  hue              177 non-null    float64
 12  diluted wines    177 non-null    float64
 13  proline          177 non-null    int64
dtypes: float64(11), int64(3)
memory usage: 19.5 KB
None
   class  alcohol  malic acid   ash  alcalinity  magnesium  phenols  \
0      1    13.20        1.78  2.14        11.2        100     2.65
1      1    13.16        2.36  2.67        18.6        101     2.80
...
3           2.93      735
4           2.85     1450
数据集样本数为 177
数据集特征数为 13
```

可知该数据集共有 177 个数据，有 13 个特征，查看葡萄酒的类别及数量：

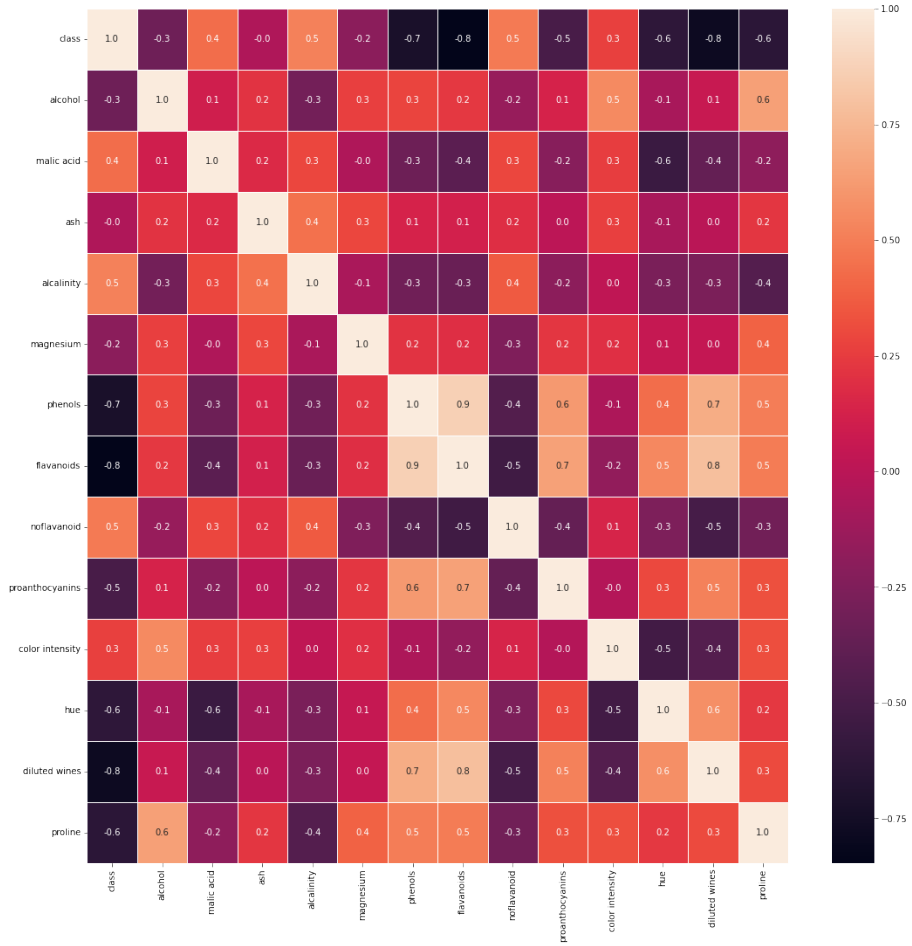
```
count = dict(Data['class'].value_counts())
print(count)
```

```
{2: 71, 1: 58, 3: 48}
```

可见这是一个相对均衡的三分类问题，给定 13 个特征对葡萄酒进行分类，由于数据集无缺失数据且考虑非线性分类，通过 SVM、逻辑回归、决策树、多层感知机、随机森林、集成学习、梯度数上升等多种算法解决解决这一分类问题。

绘制相关系数图：

```
f,ax = plt.subplots(figsize=(18, 18))
sns.heatmap(Data.corr(), annot=True, linewidths=.5, fmt= '.1f',ax=ax)
```



由图可知部分特征的相关系数较大，相关性较强，考虑通过 PCA 进行降维处理，要求降维后的数据至少保留 90%的信息：

```
#PCA 降维
X = Data.iloc[:,1:].values
X = StandardScaler().fit_transform(X)
pca = PCA(n_components=0.9) #保证降维后的数据保留90%的信息
pca.fit(X)
X = pca.transform(X)
Y = Data['class']
print(X.shape)
#(177, 8)
```

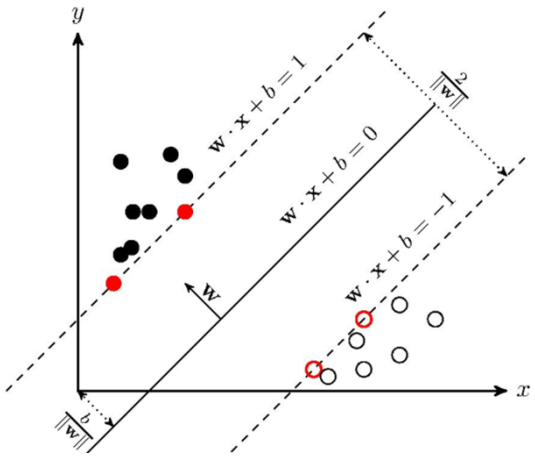
则降维后特征维数降至 8，再对数据进行预处理，通过留出法划分训练集与测试集(比例为 3:1)，并对数据进行标准化用于建模预测：

```
#留出法划分数据集
data = np.column_stack((Data['class'], X))
Data = pd.DataFrame(data = data[0:,0:], columns=['Y','X1','X2','X3','X4','X5','X6','X7','X8'])
n_train = int(n * 0.75) #75%的数据用于训练
n_test = n - n_train
seed = 1425 #学号尾号作为随机种子
random.seed(seed)
index = range(0,n)
index_selected = random.sample(index,n_train)
index_selected.sort()
```

```
Data_train = Data.loc[index_selected]
Data_test = Data.drop(index = index_selected)
print(Data_train.head())
print(Data_test.head())

#数据预处理
X_train = Data_train.drop(columns = ['Y'],axis = 1)
Y_train = Data_train.Y
X_test = Data_test.drop(columns = ["Y"],axis=1)
Y_test = Data_test.Y
X_train_standardized = preprocessing.scale(X_train, with_mean = True, with_std=
True) / np.sqrt(n_train)
X_test_standardized = preprocessing.scale(X_test, with_mean = True, with_std=Tr
ue) / np.sqrt(n_test)
Y_train = Y_train.astype('int')
Y_test = Y_test.astype('int')
```

对数据进行预处理后，使用 7 个算法分别进行建模训练并预测测试集数据的标签，并将一部分模型通过五折交叉验证进行验证，比对其准确率。

首先是 SVM 算法，SVM 是在分类与回归分析中分析数据的监督式学习模型与相关的学习算法。给定一组训练实例，每个训练实例被标记为属于两个类别中的一个或另一个，SVM 训练算法建立一个将新的实例分配给两个类别之一的模型，使其成为非概率二元线性分类器。SVM 模型是将实例表示为空间中的点，这样映射就使得单独类别的实例被尽可能宽的明显的间隔分开。然后，将新的实例映射到同一空间，并基于它们落在间隔的哪一侧来预测所属类别。



由于本分类任务属于三分类任务且考虑非线性分类，需要构造 SVM 的多类分类器，其中比较重要的参数有和函数类型 kernel、惩罚系数 C 与样本映射维度 gamma，通过网格搜索得到准确率最高的最有超参数组合：

```
#SVM
params = {'kernel':['linear', 'rbf', 'poly', 'sigmoid'],
          'C': [1e-2, 0.1, 1, 10, 1e2],
          'gamma': ['auto', 0.08, 0.2, 0.4, 0.6, 0.8]}
svm = GridSearchCV(SVC(decision_function_shape='ovr'), params)
svm = svm.fit(X_train_standardized, Y_train)
print('最优超参数组合:', svm.best_params_)
y_pred = svm.predict(X_test_standardized)
num_true = np.sum(y_pred == Y_test)
acc_svm = num_true / Y_test.shape[0]
print('预测正确的结果数目为: ', num_true)
```

```
print('预测错误的结果数目为：',  Y_test.shape[0] - num_true)
print('预测结果准确率为：', acc_svm)
print('使用 SVM 预测数据的分类报告:','\n',
     classification_report(Y_test, y_pred))
```

```
最优超参数组合: {'C': 10, 'gamma': 0.2, 'kernel': 'rbf'}
预测正确的结果数目为：  42
预测错误的结果数目为：  3
预测结果准确率为：  0.9333333333333333
使用SVM预测数据的分类报告:
              precision    recall  f1-score   support

           1       0.83      1.00      0.91        10
           2       0.94      0.89      0.92        19
           3       1.00      0.94      0.97        16

    accuracy                           0.93        45
   macro avg       0.93      0.94      0.93        45
weighted avg       0.94      0.93      0.93        45
```

可见 SVM 模型的最优超参数为核函数使用高斯核函数 rbf，惩罚系数设为 10，gamma 为 0.2，总体预测效果较好，但考虑到数据量较少且会出现过拟合情况，采用五折交叉验证的方法再进行验证：

```
#五折交叉验证
scores = cross_val_score(svm, X, Y, cv=5, scoring='accuracy')
print('五折交叉验证结果：', scores)
```

```
五折交叉验证结果：  [0.86111111 0.94444444 0.97142857 1.         0.97142857]
```

五折交叉验证结果表明模型预测效果较好。

接下来通过 softmax 回归模型进行葡萄酒种类预测。逻辑回归常用于解决二分类问题，而遇到多分类问题时需要修改逻辑回归的损失函数使其适应多分类问题，即 softmax 回归。对于有 k 个标记的分类问题，分类函数如下：

$$h_\theta(x^{(i)}) = \begin{bmatrix} p(y^{(i)}=1|x^{(i)},\theta) \\ p(y^{(i)}=2|x^{(i)},\theta) \\ \vdots \\ p(y^{(i)}=k|x^{(i)},\theta) \end{bmatrix} = \frac{1}{\sum_{c=1}^{k} e^{\theta_c^T x^{(i)}}} \begin{bmatrix} e^{\theta_1^T x^{(i)}} \\ e^{\theta_2^T x^{(i)}} \\ \vdots \\ e^{\theta_k^T x^{(i)}} \end{bmatrix}$$

将上式的所有的$\theta_1, \theta_2, \cdots, \theta_k$组合起来，用矩阵$\theta$来表示，即：

$$\theta = \begin{bmatrix} \theta_1^T \\ \theta_2^T \\ \vdots \\ \theta_k^T \end{bmatrix}$$

此时 softmax 回归算法的代价函数如下所示（其中$\text{sign}(expression\ is\ true) = 1$）：

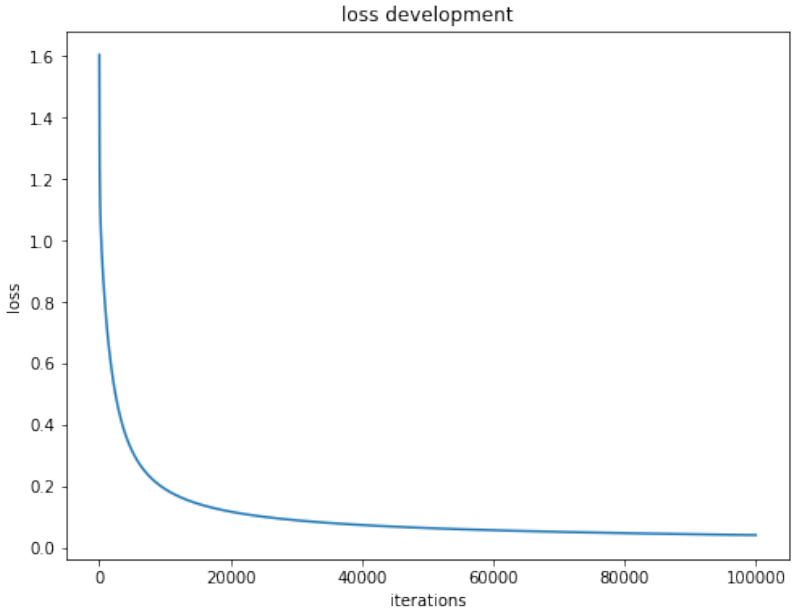
$$J(\theta) = -\sum_{i=1}^{m}\sum_{c=1}^{k} \text{sign}(y^{(i)} = c) \log p(y^{(i)} = c|x^{(i)}, \theta) = -\sum_{i=1}^{m}\sum_{c=1}^{k} \text{sign}(y^{(i)} = c) \log \frac{e^{\theta_c^T x(i)}}{\sum_{l=1}^{k} e^{\theta_l^T x^{(i)}}}$$

```
#softmax 回归
class SoftmaxRegressor:

    def __init__(self):
        pass

    def train(self, X, Y, n_classes, n_iter, learning_rate):
        self.n_samples, n_features = X.shape
        self.n_classes = n_classes
        self.weights = np.random.rand(self.n_classes, n_features)
        self.bias = np.zeros((1, self.n_classes))
        losses = []
        for i in range(n_iter):
            scores = np.dot(X, self.weights.T) + self.bias
            exp = np.exp(scores)
            sum_exp = np.sum(np.exp(scores), axis=1, keepdims=True)
            probs = exp / sum_exp
            y_predict = np.argmax(probs, axis=1)[:, np.newaxis]
            y_one_hot = np.zeros((self.n_samples, self.n_classes))
            y_one_hot[np.arange(self.n_samples), Y.T] = 1
            loss = - (1 / self.n_samples) * np.sum(y_one_hot * np.log(probs))
            losses.append(loss)
            dw = (1 / self.n_samples) * np.dot(X.T, (probs - y_one_hot))
            db = (1 / self.n_samples) * np.sum(probs - y_one_hot, axis=0)
            self.weights = self.weights - learning_rate * dw.T
            self.bias = self.bias - learning_rate * db
            if i % 10000 == 0:
                print(f'Iteration: {i}, loss: {np.round(loss, 4)}')

        return self.weights, self.bias, losses

    def predict(self, X):
        scores = np.dot(X, self.weights.T) + self.bias
        exp = np.exp(scores)
        sum_exp = np.sum(np.exp(scores), axis=1, keepdims=True)
        probs = exp / sum_exp
        return np.argmax(probs, axis=1)[:, np.newaxis]

softmax = SoftmaxRegressor()
w, b, loss = softmax.train(X_train_standardized, Y_train, learning_rate=0.1, n_
iter=100000, n_classes=3)
fig = plt.figure(figsize=(8,6))
plt.plot(np.arange(100000), loss)
```

```
plt.title("loss development")
plt.xlabel("iterations")
plt.ylabel("loss")
plt.show()
```

进行 100,000 次迭代 loss 下降情况及图示：

```
Iteration: 0, loss: 1.6053
Iteration: 10000, loss: 0.1905
Iteration: 20000, loss: 0.1166
Iteration: 30000, loss: 0.0885
Iteration: 40000, loss: 0.0731
Iteration: 50000, loss: 0.0631
Iteration: 60000, loss: 0.0561
Iteration: 70000, loss: 0.0508
Iteration: 80000, loss: 0.0465
Iteration: 90000, loss: 0.0431
```



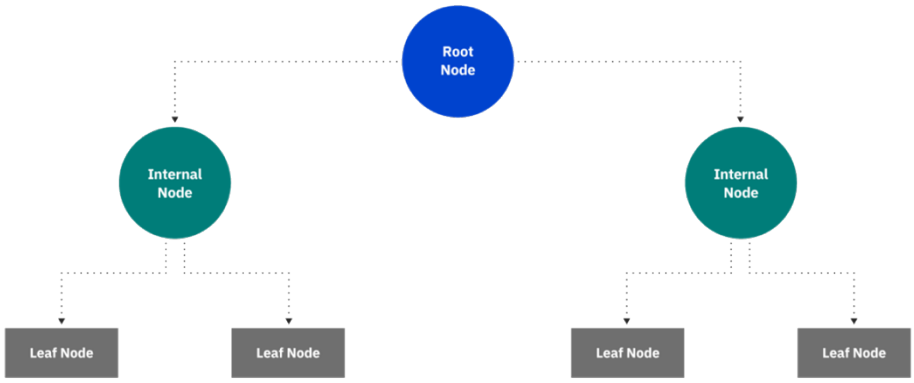
使用该模型对测试集进行预测，得到模型预测的准确率：

```
y_pred = softmax.predict(X_test_standardized)
y_pred = pd.DataFrame(y_pred)
num_true = np.sum(y_pred == Y_test)
acc_softmax = num_true / Y_test.shape[0]
print('预测正确的结果数目为：', num_true)
print('预测错误的结果数目为：',  Y_test.shape[0] - num_true)
print('预测结果准确率为：', acc_softmax)
```

```
预测正确的结果数目为：  43
预测错误的结果数目为：  2
预测结果准确率为：  0.9555555555555556
```

softmax 回归模型效果略优于 SVM 模型，接下来训练决策树模型并进行类别预测。决策树是一种用于分类和回归任务的非参数监督学习算法。 它是一种分层树形结构，由根节点、分支、内部节点和叶节点组成。决策树学习采用"一一击破"的策略，执行贪心搜索来识别决策树内的最佳分割点

。然后以自上而下的回归方式重复此拆分过程，直到所有或者大多数记录都标记为特定的类别标签。模型训练时采用 CART 算法，叶子节点最小样本数为 2，由于数据已经经过 PCA 降维且标签类别较均匀，可以训练出拟合效果较好的模型。



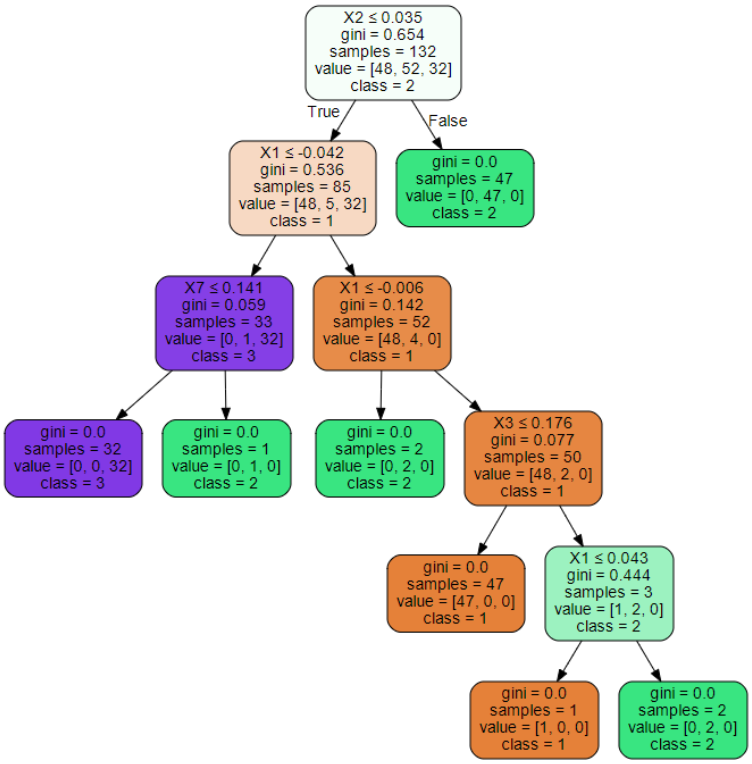

```
tree_model = tree.DecisionTreeClassifier(criterion='gini', max_depth=None, min_
samples_leaf=2, ccp_alpha=0.0)
tree_model.fit(X_train_standardized, Y_train)
y_pred = tree_model.predict(X_test_standardized)
num_true = np.sum(y_pred == Y_test)
acc_tree = num_true / Y_test.shape[0]
print('预测正确的结果数目为：', num_true)
print('预测错误的结果数目为：',  Y_test.shape[0] - num_true)
print('预测结果准确率为：', acc_tree)
print('使用决策树预测数据的分类报告:','\n',
     classification_report(Y_test, y_pred))
#五折交叉验证
scores = cross_val_score(tree_model, X, Y, cv=5, scoring='accuracy')
print(scores)
```

```
预测正确的结果数目为：  40
预测错误的结果数目为：  5
预测结果准确率为：  0.8888888888888888
使用决策树预测数据的分类报告:
              precision    recall  f1-score   support

           1       0.88      0.70      0.78        10
           2       0.82      0.95      0.88        19
           3       1.00      0.94      0.97        16

    accuracy                           0.89        45
   macro avg       0.90      0.86      0.87        45
weighted avg       0.90      0.89      0.89        45

[0.97222222 0.86111111 0.88571429 0.97142857 0.91428571]
```

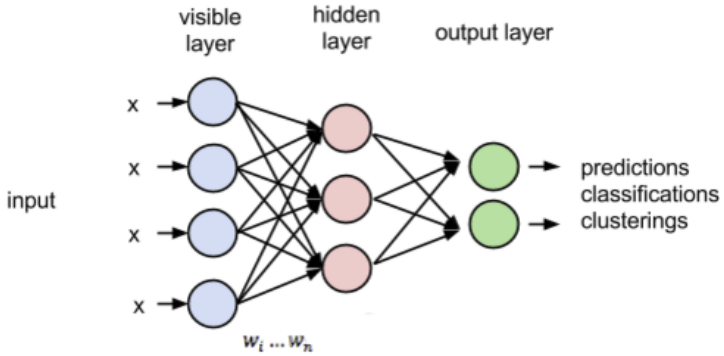
决策树模型的简单验证效果略差于 SVM 与 softmax 回归，但五折交叉验证结果尚可，对决策树进行可视化：

```
#决策树可视化
dot_data = StringIO()
feature_names = ['X1','X2','X3','X4','X5','X6','X7','X8']
```

```
target_names = ['1', '2', '3']
tree.export_graphviz(tree_model, out_file=dot_data, feature_names=feature_names,
class_names=target_names, filled=True,  rounded=True, special_characters=True)
graph = pydotplus.graph_from_dot_data(dot_data.getvalue())
graph.write_pdf("tree.pdf")
```



多层感知机是一种前向结构的人工神经网络，映射一组输入向量到一组输出向量。MLP 可以被看作是一个有向图，由多个的节点层所组成，每一层都全连接到下一层。除了输入节点，每个节点都是一个带有非线性激活函数的神经元（或称处理单元）。将多层感知机模型的 L2 正则化参数设为 1e-5，隐藏层有 2 层，第一层有 6 个神经元，第二层有 2 个神经元。得到预测结果后计算 MSE 并将结果舍入为整数计算准确率。



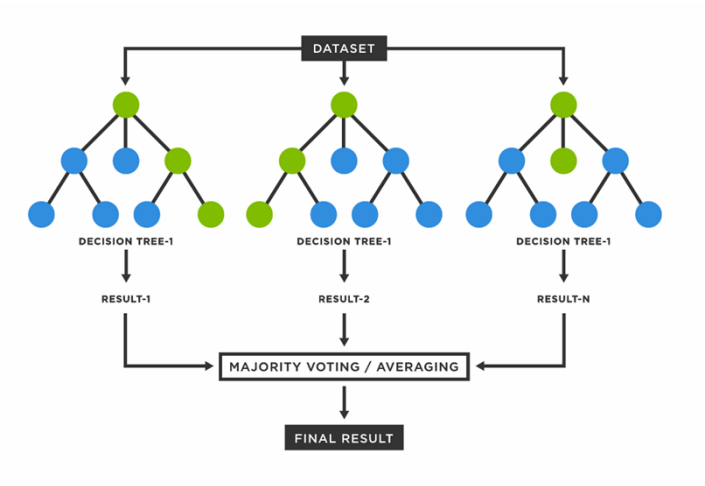

```
#多层感知机
mlp = Pipeline([('scale', StandardScaler()), ('MLPRegressor', MLPRegressor(solv
er='lbfgs', alpha=1e-5, hidden_layer_sizes=(6, 2), random_state=1))])
mlp = mlp.fit(X_train_standardized, Y_train)
y_pred = mlp.predict(X_test_standardized)
print('MSE:', mean_squared_error(y_pred, Y_test))
y_pred = np.around(y_pred)
```

```
num_true = np.sum(y_pred == Y_test)
acc_mlp = num_true / Y_test.shape[0]
print('预测正确的结果数目为：', num_true)
print('预测错误的结果数目为：',  Y_test.shape[0] - num_true)
print('预测结果准确率为：', acc_mlp)
```

```
MSE: 0.24079317128750138
预测正确的结果数目为：  39
预测错误的结果数目为：  6
MLP预测结果准确率为：  0.8666666666666667
```

将回归模型结果离散化，可以看到 MLP 模型的 MSE 为 0.24，模型准确率相比其他模型较低，可能是由于 MLP 对于特征缩放比较敏感所致。

接下来训练随机森林模型，随机森林是一组决策树。但是决策树往往会创建规则，用来做出决策，而随机森林将随机选择要素并进行观测，构建决策森林，然后计算平均结果。与单个决策树相比，大量不相关的决策树会产生更准确的预测，这是因为大量的决策树协同工作，可以相互保护，免受单个错误和过度拟合的影响。随机森林模型训练的最大弱学习器个数设为 100（过小会欠拟合，过大收益较低且耗时长），内部节点再划分所需最小样本数设为 5。



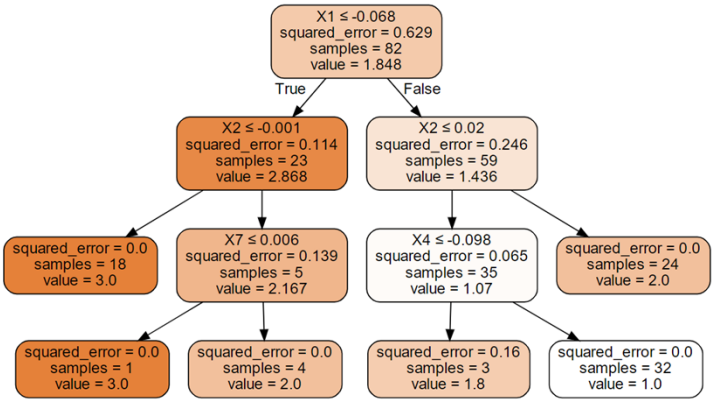

```
#随机森林
forest = RandomForestRegressor(n_estimators=100, min_samples_split=5)
forest = forest.fit(X_train_standardized, Y_train)
y_pred = forest.predict(X_test_standardized)
print('MSE:', mean_squared_error(y_pred, Y_test))
y_pred = np.around(y_pred)
num_true = np.sum(y_pred == Y_test)
acc_forest = num_true / Y_test.shape[0]
print('预测正确的结果数目为：', num_true)
print('预测错误的结果数目为：',  Y_test.shape[0] - num_true)
print('预测结果准确率为：', acc_forest)
```

```
MSE: 0.05235418469387751
预测正确的结果数目为：  41
预测错误的结果数目为：  4
随机森林预测结果准确率为：  0.9111111111111111
```

可见随机森林预测结果的 MSE 远低于 MLP 且准确率更高。由于 scikit-learn 无法对随机森林直

接进行可视化，故随机森林模型可视化需要拆解成多棵决策树（选取最后一次迭代的结果）

```
#随机森林可视化
estimators = forest.estimators_
for index, model in enumerate(estimators):
    dot_data = tree.export_graphviz(model , out_file=None,
                                    feature_names=['X1','X2','X3','X4','X5','X6','X7','X8'],
                                    class_names=['1', '2', '3'],
                                    filled=True, rounded=True,
                                    special_characters=True)
    graph = pydotplus.graph_from_dot_data(dot_data)
    graph.write_pdf("forest.pdf")
```



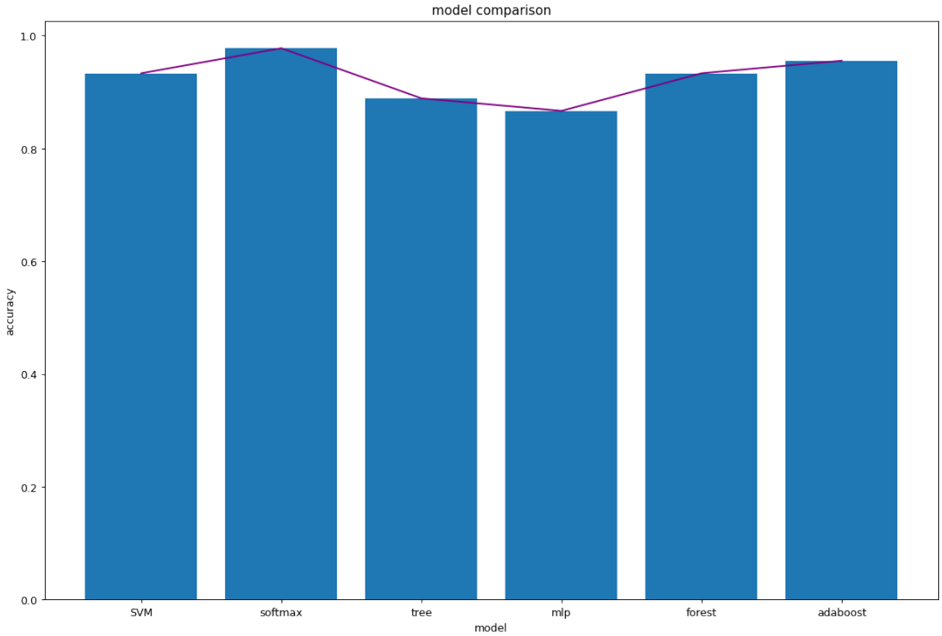
最后训练集成学习 AdaBoostRegressor 回归模型，使用 CART 决策树分类器，弱学习器的最大迭代次数为 100（过小容易欠拟合，过大容易过拟合），每个弱学习器的权重缩减系数设为 0.1：

```
#集成学习 adaboost 回归
adaboost = AdaBoostRegressor(n_estimators=100, learning_rate=0.1)
adaboost = adaboost.fit(X_train_standardized, Y_train)
y_pred = adaboost.predict(X_test_standardized)
print('MSE:', mean_squared_error(y_pred, Y_test))
y_pred = np.around(y_pred)
num_true = np.sum(y_pred == Y_test)
acc_adaboost = num_true / Y_test.shape[0]
print('预测正确的结果数目为：', num_true)
print('预测错误的结果数目为：',  Y_test.shape[0] - num_true)
print('adaboost 回归预测结果准确率为：', acc_adaboost)
```

```
MSE: 0.021692118775899237
预测正确的结果数目为： 44
预测错误的结果数目为： 1
adaboost回归预测结果准确率为： 0.9777777777777777
```

可以看到 adaboost 集成学习回归模型的 MSE 进一步降低且准确率非常高。

综上使用了 SVM、softmax 回归、决策树、多层感知机、随机森林、集成学习回归 6 个模型对 wine 数据集分类进行预测，6 个模型都有 80%以上的准确率，对比 6 个模型的准确率：



可见 softmax 回归与集成学习回归的准确率最高，MLP 多层感知机准确率稍低，但总体来说都对测试集有着很好的预测分类效果。

四、总结

本次实验基于 wine 数据集，首先通过 PCA 对数据进行降维，并进行数据预处理及通过留出法划分训练集与测试集，之后训练了 SVM、softmax 回归、决策树、多层感知机、随机森林、集成学习回归 6 个模型并对 wine 数据进行预测分类，6 个模型均取得较好效果，最后对比了 6 个模型预测的准确率，发现 softmax 回归与集成学习回归模型的效果最好。

本次实验收获颇多，比如学习了如何使 SVM 与逻辑回归模型适应多分类问题，如何对决策树及随机森林进行剪枝优化等等，但由于采用的数据集数据较少，本次实验对模型的调参优化要求较低，在日后的学习中还需要在模型训练及优化方面多多努力。