# 華南師範大學

# 本科学生实验（实践）报告

院　　系：计 算 机 学 院

实验课程：数据挖掘

实验项目：基于多种算法解决糖尿病分类问题

指导老师：汤娜

开课时间：2021～ 2022 年度第 1 学期

专　　业：计算机科学与技术

班　　级：19 级 3 班

学　　生：吴宇涛

学　　号：20192131089

华南师范大学教务处

目录

# 1. 问题重述

## 1.1. 问题背景

本数据集来自美国国家糖尿病、消化和肾脏疾病研究所。数据集的目的是根据数据集中包含的某些诊断测量值，诊断预测患者是否患有糖尿病。在从较大的数据库中选择这些实例时受到了一些限制。尤其值得一提的是，这里所有的患者都是女性，至少 21 岁，有印度皮马血统。

## 1.2. 问题提出

数据集包括几个医学预测变量和一个目标变量，diabetes.预测变量包括患者怀孕次数、BMI、胰岛素水平、年龄等。能否建立机器学习模型或者深度学习模型，以准确预测数据集中的患者是否具有糖尿病？

# 2. 数据分析

## 2.1. 获取数据

```
#读取数据
#可以通过此网站     https://github.com/ytWu1314/Data-Mining
diabetes_data  =  pd.read_csv('./diabetes_data.csv')

diabetes_data.head()
```

| pregnancies | glucose | diastolic | triceps | insulin | bmi | dpf | age | diabetes |
|---|---|---|---|---|---|---|---|---|
| 6 | 148 | 72 | 35 | 0 | 33.6 | 0.627 | 50 | 1 |
| 1 | 85 | 66 | 29 | 0 | 26.6 | 0.351 | 31 | 0 |
| 8 | 183 | 64 | 0 | 0 | 23.3 | 0.672 | 32 | 1 |
| 1 | 89 | 66 | 23 | 94 | 28.1 | 0.167 | 21 | 0 |
| 0 | 137 | 40 | 35 | 168 | 43.1 | 2.288 | 33 | 1 |

使用 pandas 库读取 diabetes_data.csv，得到一个 dataframe，从上面的数据我们可以得到数据集中的特征值和目标值分别是什么。目标值 diabetes 的值为 0 或者 1。

## 2.2. 数据处理

### 2.2.1. 分析特征值和目标

通过整理数据集和列属性，我们可以得出下表，共有 8 个特征值和一个目标值：

| | |
|---|---|
| pregnancies | 怀孕次数 |
| glucose | 葡萄糖测试值 |
| diastolic | 血压 |
| triceps | 皮肤厚度 |
| insulin | 胰岛素 |
| bmi | 身体质量指数 |
| dpf | 糖尿病遗传函数 |
| age | 年龄 |
| diabetes | 是否患有糖尿病 |

### 2.2.2. EDA 和统计分析

```
1. diabetes_data.info(verbose=True)
2. #verbose=True  显示详细信息
```

调用库函数我们可以查看数据集中是否有空值，数据集列属性的类型，内存使用情况。

| # | Column | Non-Null Count | Dtype |
|---|---|---|---|
| 0 | pregnancies | 768 non-null | int64 |
| 1 | glucose | 768 non-null | int64 |
| 2 | diastolic | 768 non-null | int64 |

| | | | |
|---|---|---|---|
| 3 | triceps | 768 non-null | int64 |
| 4 | insulin | 768 non-null | int64 |
| 5 | bmi | 768 non-null | float64 |
| 6 | dpf | 768 non-null | float64 |
| 7 | age | 768 non-null | int64 |
| 8 | diabetes | 768 non-null | int64 |

我们可以得到数据集中是不存在缺失值的，即没有 NAN 或者？等值。

```
1. diabetes_data.describe().T
2. #只显示数字列，要查看其他详细的数据可以设置 include='all'参数
```

| | count | mean | std | min | 25% | 50% | 75% | max |
|---|---|---|---|---|---|---|---|---|
| pregnancies | 768.0 | 3.845052 | 3.369578 | 0.000 | 1.00000 | 3.0000 | 6.00000 | 17.00 |
| glucose | 768.0 | 120.894531 | 31.972618 | 0.000 | 99.00000 | 117.0000 | 140.25000 | 199.00 |
| diastolic | 768.0 | 69.105469 | 19.355807 | 0.000 | 62.00000 | 72.0000 | 80.00000 | 122.00 |
| triceps | 768.0 | 20.536458 | 15.952218 | 0.000 | 0.00000 | 23.0000 | 32.00000 | 99.00 |
| insulin | 768.0 | 79.799479 | 115.244002 | 0.000 | 0.00000 | 30.5000 | 127.25000 | 846.00 |
| bmi | 768.0 | 31.992578 | 7.884160 | 0.000 | 27.30000 | 32.0000 | 36.60000 | 67.10 |
| dpf | 768.0 | 0.471876 | 0.331329 | 0.078 | 0.24375 | 0.3725 | 0.62625 | 2.42 |
| age | 768.0 | 33.240885 | 11.760232 | 21.000 | 24.00000 | 29.0000 | 41.00000 | 81.00 |
| diabetes | 768.0 | 0.348958 | 0.476951 | 0.000 | 0.00000 | 0.0000 | 1.00000 | 1.00 |

观察 describe 中显示的数据,我们可以察觉到一些问题,0 是没有意义的值(人体的 bmi，血压等数据不应该出现 0),所以,我们将 0 当作缺失值.可以先将这些列出现的 0 值改成 NAN，这样比较

好处理.再统计空值的个数，通过’hist’的方法可以查看每一个属性它的值的分配情况，根据分布图，可以使用均值或者中位数来填充缺失值。数据符合均匀分布，应该使用变量的均值填补缺失值。数据存在倾斜分布，采用中位数填补缺失值。

## 2.3. 数据可视化

通过数据可视化，可以查看目标值的分布情况，没有糖尿病的患者是糖尿病患者的两倍。

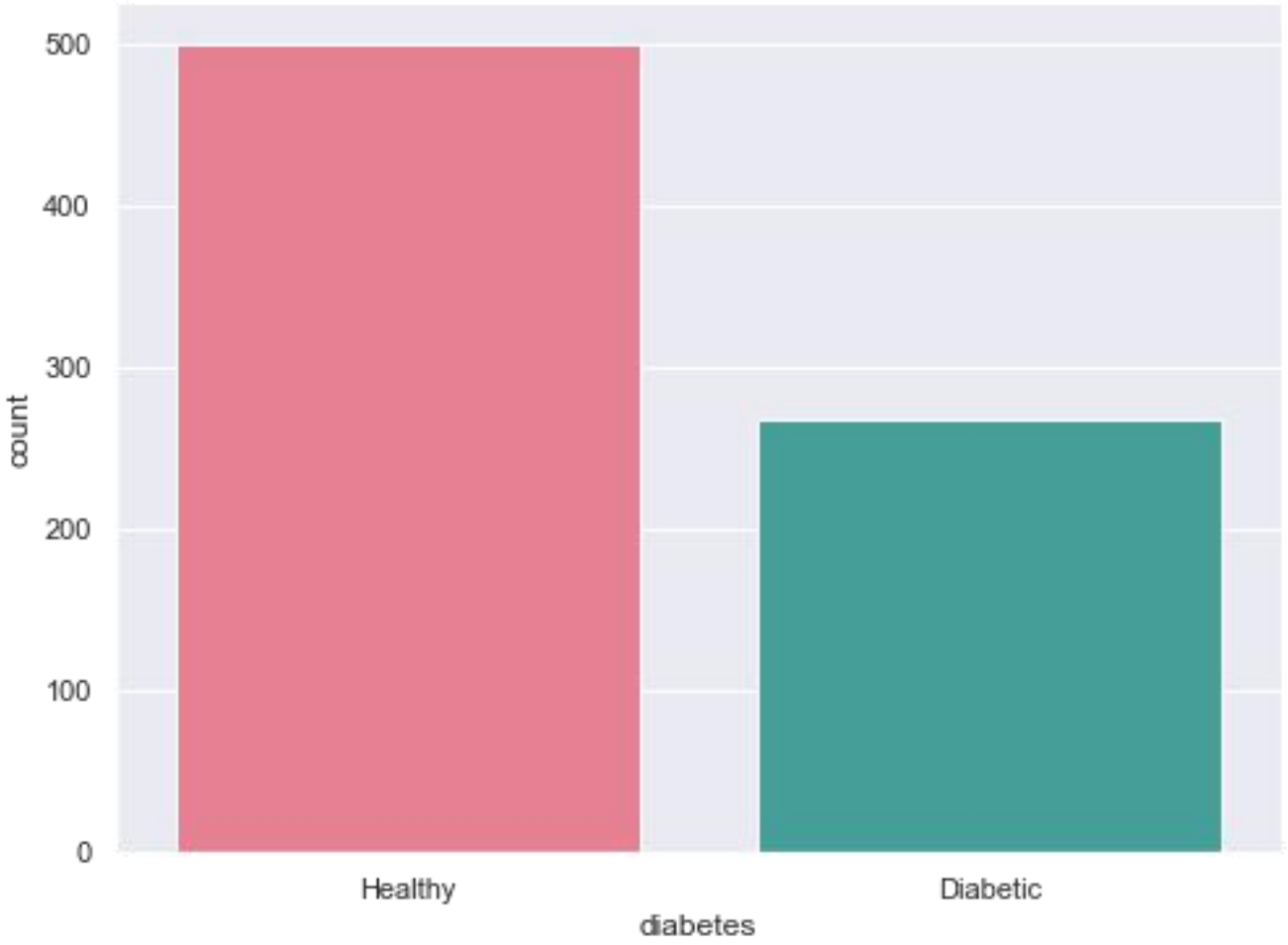



图 1 特征值条形图

处理完缺失值的数据可以通过热图查看各个特征值的相关性，是正相关还是负相关。热图是借助颜色对信息的二维表示。热图可以帮助用户可视化简单或复杂的信息。

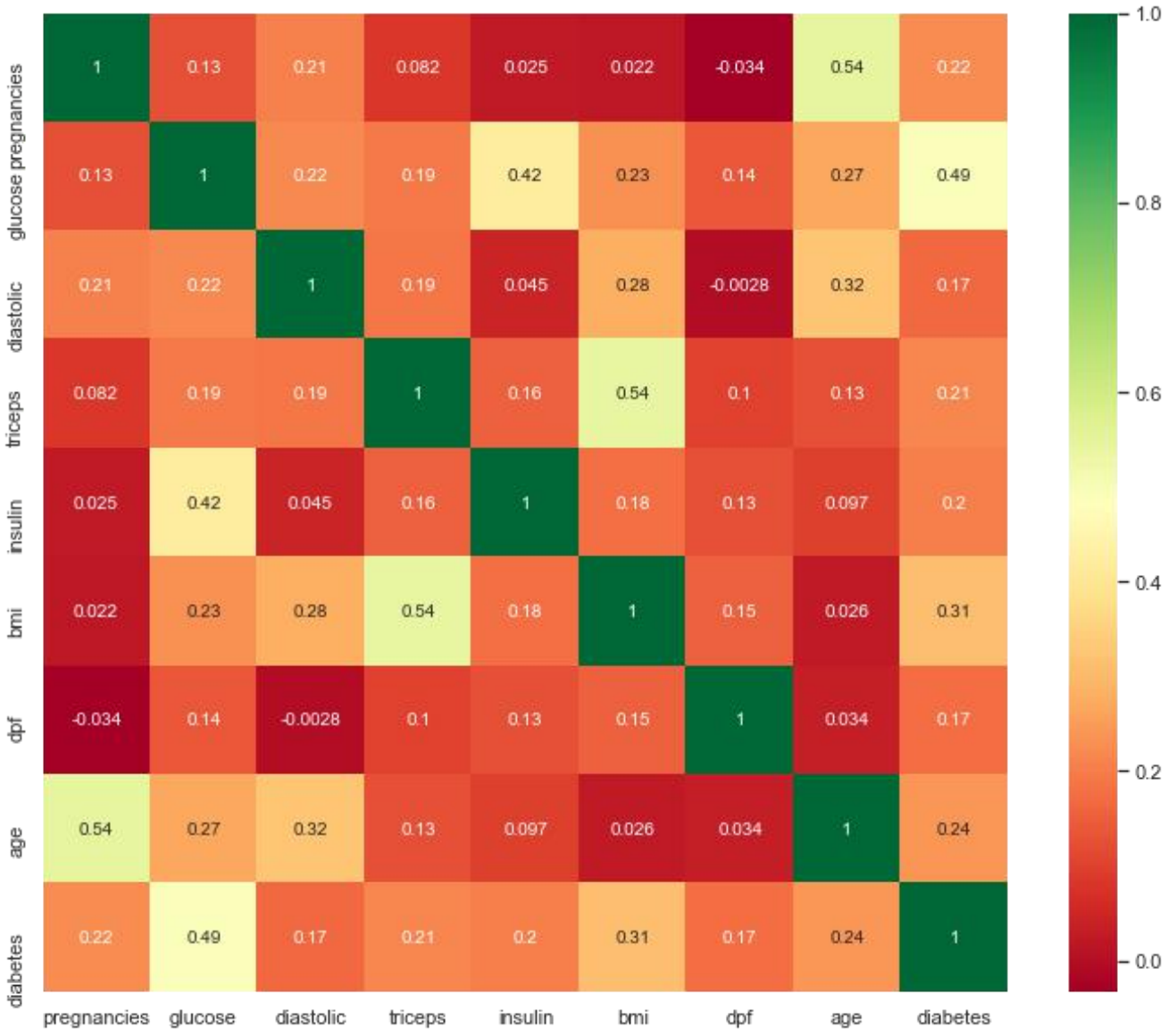



图 2 热图

## 2.4. 特征标准化

- Z-score 规范化（标准差标准化 / 零均值标准化）
- 通过求 z-score 的方法，转换为标准正态分布（均值为 0，标准差为 1）
- 和整体样本分布相关，每个样本点都能对标准化产生影响，通过均值（μ \mu μ）和标准差（σ \sigma σ）体现出来。
- 也能取消由于量纲不同引起的误差，使不同度量之间的特征具有可比性
- 输出范围：[−∞, +∞]
- 优点（相对于最值归一化）：即使原数据集中有极端值，归一化有的数据集，依然满足均值为 0 方差为 1，不会形成一个有偏的数据；

```
from sklearn.preprocessing import StandardScaler
sc_X = StandardScaler()
X = pd.DataFrame(sc_X.fit_transform(diabetes_data_copy.drop(["diabetes"],axis = 1),),
                 columns=['pregnancies', 'glucose', 'diastolic', 'triceps', 'insulin',
                'bmi', 'dpf', 'age'])
X.head()
```

| | pregnancies | glucose | diastolic | triceps | insulin | bmi | dpf | age |
|---|---|---|---|---|---|---|---|---|
| 0 | 0.639947 | 0.865108 | -0.033518 | 0.670643 | -0.181541 | 0.166619 | 0.468492 | 1.425995 |
| 1 | -0.844885 | -1.206162 | -0.529859 | -0.012301 | -0.181541 | -0.8522 | -0.365061 | -0.190672 |
| 2 | 1.23388 | 2.015813 | -0.695306 | -0.012301 | -0.181541 | -1.3325 | 0.604397 | -0.105584 |
| 3 | -0.844885 | -1.074652 | -0.529859 | -0.695245 | -0.540642 | -0.633881 | -0.920763 | -1.041549 |
| 4 | -1.141852 | 0.503458 | -2.680669 | 0.670643 | 0.316566 | 1.549303 | 5.484909 | -0.020496 |

## 2.5. 数据集划分

调用 sklearn 中的库函数，对已经标准化后的数据集进行划分，分为特征值训练集，特征值测试集，目标值训练集，目标值测试集，按照 7:3 的比列进行划分。

# 3. 建立模型

## 3.1. 决策树

### 3.1.1. 决策树的定义

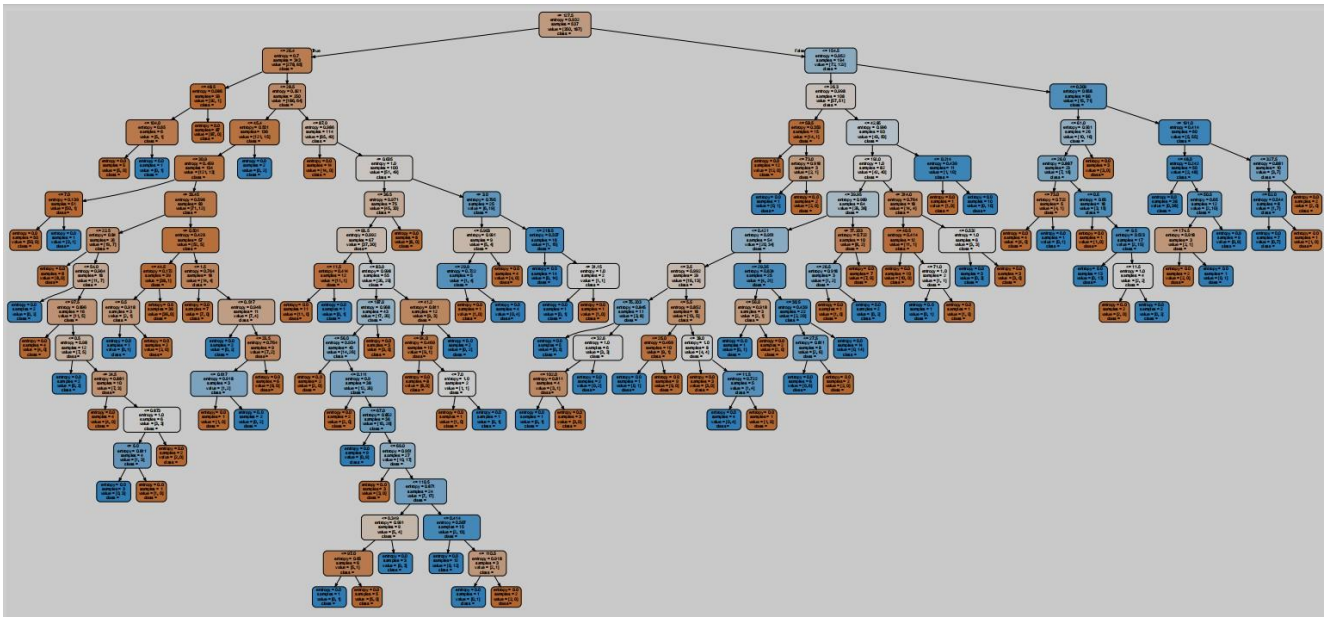
决策树（Decision Tree）是一种非参数的有监督学习方法，它能够从一系列有特征和标签的数据中总结出决策规
则，并用树状图的结构来呈现这些规则，以解决分类和回归问题。决策树算法容易理解，适用各种数据，在解决各
种问题时都有良好表现，尤其是以树模型为核心的各种集成算法，在各个行业和领域都有广泛的应用。
决策树算法的本质是一种图结构，我们只需要问一系列问题就可以对数据进行分类了。

### 3.1.2. 模型建立与评分

```
clf=tree.DecisionTreeClassifier(criterion='entropy')
clf=clf.fit(x_train,Y_train)
score=clf.score(x_test,Y_test)
score
```

### 3.1.3. 决策树可视化(数据未标准化)

```
feature_name = ['怀孕次数','葡萄糖测试值','血压','皮肤厚度','胰岛素','身体质量指数','糖尿病遗传函数','年龄'] # 特征名
import graphviz
dot_data = tree.export_graphviz(clf
                                ,out_file = None
                                ,feature_names= feature_name
                                ,class_names=["健康","糖尿病"] # 标签名
                                ,filled=True #填充颜色（不同颜色不同分类）
                                ,rounded=True # 圆角
                                )
graph = graphviz.Source(dot_data) # 可视化
graph
```



### 3.1.4. 参数调优

```
dt = DecisionTreeClassifier()
para_dt = {'criterion':['gini','entropy'],'max_depth':np.arange(1, 50), 'min_samples_leaf':[1,2,5,10,25,30,40,50,100]}
grid_dt = GridSearchCV(dt, param_grid=para_dt, cv=5) #grid search decision tree for 5 fold cv
#"gini" for the Gini impurity and "entropy" for the information gain.
#min_samples_leaf: The minimum number of samples required to be at a leaf node, have the effect of smoothing the model
```

```
grid_dt.fit(X_train, y_train)
print("Best parameters for Decision Tree:", grid_dt.best_params_)
```

### 3.1.5. 模型评估

```
dt = DecisionTreeClassifier(criterion='gini', max_depth=6, min_samples_leaf=10)
dt = dt.fit(X_train, y_train)
score = dt.score(X_test, y_test) #返回预测的准确度
score
```

```
y_pred_dt = dt.predict(X_test)
accuracy = accuracy_score(y_test,y_pred_dt)
print('{:s} : {:.2f}'.format('Decision Tree 的准确率为:', accuracy))
```

### 3.1.6. 建立混淆矩阵

在分类任务下，预测结果(Predicted Condition)与正确标记(True Condition)之间存在四种不同的组合，构成混淆矩阵(适用于多分类)

| | | 预测结果 | |
|---|---|---|---|
| | | 正例 | 假例 |
| 真实结果 | 正例 | 真正例TP | 伪反例FN |
| | 假例 | 伪正例FP | 真反例TN |

图 3 混淆矩阵图示

```
from sklearn.metrics import confusion_matrix
m_dt = confusion_matrix(y_test,y_pred_dt)
print('预测结果混淆矩阵：\n',m_dt)
```

```
%pylab inline
import seaborn as sns
sns.heatmap(m_dt,cmap='Blues',annot=True)
```

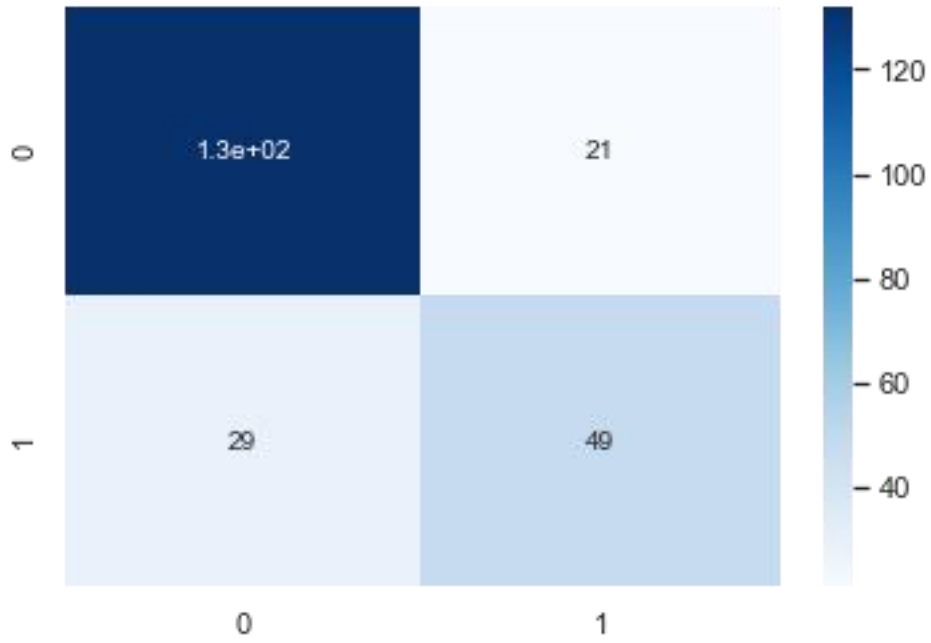



图 4 混淆矩阵

### 3.1.7. 分类评估报告

sklearn.metrics.classification_report(y_true, y_pred, labels=[], target_names=None )

y_true：真实目标值
y_pred：估计器预测目标值
labels:指定类别对应的数字
target_names：目标类别名称
return：每个类别精确率与召回率

```
1.  from sklearn.metrics import classification_report
2.
3.  print(classification_report(y_test, y_pred_dt))
```

| | precision | recall | f1-score | support |
|---|---|---|---|---|
| 0 | 0.82 | 0.86 | 0.84 | 153 |
| 1 | 0.7 | 0.63 | 0.66 | 78 |
| accuracy | | | 0.78 | 231 |
| macro avg | 0.76 | 0.75 | 0.75 | 231 |
| weighted avg | 0.78 | 0.78 | 0.78 | 231 |

### 3.1.8. 总结

- 优点：
  - 简单的理解和解释，树木可视化。
- 缺点：
  - 决策树学习者可以创建不能很好地推广数据的过于复杂的树，会出现过拟合。
- 改进：
  - 剪枝 cart 算法
  - 随机森林

## 3.2. 朴素贝叶斯算法

### 3.2.1. 朴素贝叶斯算法定义

朴素贝叶斯方法是在贝叶斯算法的基础上进行了相应的简化，即假定给定目标值时属性之间相互条件独立。也就是说没有哪个属性变量对于决策结果来说占有着

较大的比重，也没有哪个属性变量对于决策结果占有着较小的比重。虽然这个简化方式在一定程度上降低了贝叶斯分类算法的分类效果，但是在实际的应用场景中，极大地简化了贝叶斯方法的复杂性。

### 3.2.2. 模型建立

```
from sklearn.naive_bayes import GaussianNB
gnb = GaussianNB()
gnb.fit(X_train,y_train)
score_gnb = gnb.score(X_test, y_test) #返回预测的准确度
score_gnb
```

### 3.2.3. 模型评估

```
y_pred_gnb = gnb.predict(X_test)
accuracy_gnb = accuracy_score(y_test,y_pred_gnb)
print('{:s} : {:.2f}'.format('GaussianNB 的准确率为:', accuracy_gnb))
```

### 3.2.4. 建立混淆矩阵

```
from sklearn.metrics import confusion_matrix
m_gnb = confusion_matrix(y_test,y_pred_gnb)
print('预测结果混淆矩阵：\n',m_gnb)
```

### 3.2.5. 可视化混淆矩阵

```
%pylab inline
import seaborn as sns
sns.heatmap(m_gnb,cmap='Blues',annot=True)
```

### 3.2.6. 分类评估报告

```
from sklearn.metrics import classification_report

print(classification_report(y_test, y_pred_gnb))
```

### 3.2.7. 总结

- 优点：
  - 朴素贝叶斯模型发源于古典数学理论，有稳定的分类效率。
  - 对缺失数据不太敏感，算法也比较简单，常用于文本分类。
  - 分类准确度高，速度快
- 缺点：
  - 由于使用了样本属性独立性的假设，所以如果特征属性有关联时其效果不好

## 3.3. 随机森林

### 3.3.1. 随机森林的定义

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。

### 3.3.2. 模型建立

```
1.  rf = RandomForestClassifier()
2.  params_rf = {'n_estimators':[100, 350, 500], 'min_samples_leaf':[2, 10, 30]}
3.  grid_rf = GridSearchCV(rf, param_grid=params_rf, cv=5)
4.  grid_rf.fit(X_train, y_train)
5.  print("Best parameters for Random Forest:", grid_rf.best_params_)
```

### 3.3.3. 模型评估

```
1.  rf = RandomForestClassifier(n_estimators=100, min_samples_leaf=2, random_state=42)
2.  rf.fit(X_train, y_train)
3.  y_pred_rf = rf.predict(X_test)
4.  accuracy_rf = accuracy_score(y_test,y_pred_rf)
5.  print('{:s} : {:.2f}'.format('RandomForest 的准确率为:', accuracy_rf))
```

### 3.3.4. 建立混淆矩阵

```
1.  from sklearn.metrics import confusion_matrix
2.  m_rf = confusion_matrix(y_test,y_pred_rf)
3.  print('预测结果混淆矩阵：\n',m_rf)
```

### 3.3.5. 混淆矩阵可视化

```
1.  %pylab inline
2.  import seaborn as sns
3.  sns.heatmap(m_rf,cmap='Blues',annot=True)
```

### 3.3.6. 分类评估报告

```
1.  from sklearn.metrics import classification_report
2.
3.  print(classification_report(y_test, y_pred_rf))
```

### 3.3.7. 总结

- 随机森林在当前所有算法中，具有极好的准确率
- 能够有效地运行在大数据集上，处理具有高维特征的输入样本，而且不需要降维
- 能够评估各个特征在分类问题上的重要性

## 3.4. 逻辑回归

### 3.4.1. 逻辑回归的原理

$$h(w) = w_1x_1 + w_2x_2 + w_3x_3 \ldots + b$$

图 5 逻辑回归的输入

逻辑回归的输入就是一个线性回归的结果。

$$g(\theta^T x) = \frac{1}{1 + e^{-\theta^T x}},$$

sigmoid 函数

- 分析
  - 回归的结果输入到 sigmoid 函数当中
  - 输出结果：[0,1]区间中的一个概率值，默认为 0.5 为阈值

逻辑回归最终的分类是通过属于某个类别的概率值来判断是否属于某个类别，并且这个类别默认标记为 1(正例),另外的一个类别会标记为 0(反例)。（方便损失计算）

输出结果解释：假设有两个类别 A，B，并且假设我们的概率值为属于 A(1)这个类别的概率值。现在有一个样本的输入到逻辑回归输出结果 0.6，那么这个概率值超过 0.5，意味着我们训练或者预测的结果就是 A(1)类别。那么反之，如果得出结果为 0.3 那么，训练或者预测结果就为 B(0)类别。

### 3.4.2. 模型建立

```
lr = LogisticRegression(random_state=42)
lr.fit(X_train,y_train)
```

### 3.4.3. 模型评估

```
y_pred_lr = lr.predict(X_test)
accuracy_lr = accuracy_score(y_test,y_pred_lr)
print('{:s} : {:.2f}'.format('LogisticRegression 的准确率为:', accuracy_lr))
```

### 3.4.4. 建立混淆矩阵

```
from sklearn.metrics import confusion_matrix
m_lr = confusion_matrix(y_test,y_pred_lr)
print('预测结果混淆矩阵：\n',m_lr)
```

### 3.4.5. 混淆矩阵可视化

```
## 可视化混淆矩阵
%pylab inline
import seaborn as sns
sns.heatmap(m_lr,cmap='Blues',annot=True)
```

### 3.4.6. 分类评估报告

```
from sklearn.metrics import classification_report

print(classification_report(y_test, y_pred_lr))
```

### 3.4.7. 总结

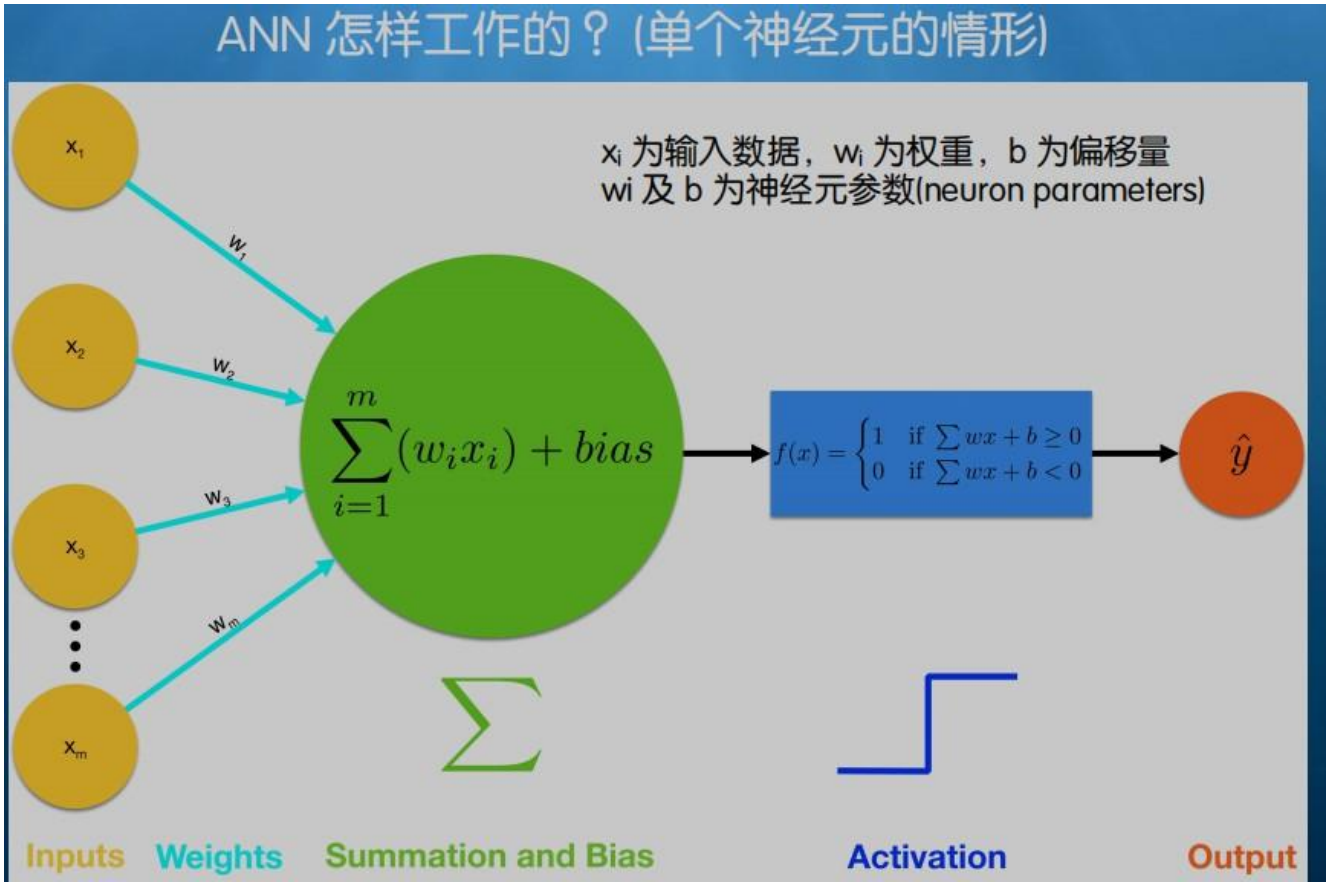
Logistic 回归的一般过程：

- 收集数据：采用任意方法收集数据。
- 准备数据：由于需要进行距离计算，因此要求数据类型为数值型。另外，结构化数据格式则最佳。
- 分析数据：采用任意方法对数据进行分析。
- 训练算法：大部分时间将用于训练，训练的目的是为了找到最佳的分类回归系数。
- 测试算法：一旦训练步骤完成，分类将会很快。
- 使用算法：首先，我们需要输入一些数据，并将其转换成对应的结构化数值；接着，基于训练好的回归系数，就可以对这些数值进行简单的回归计算，判定它们属于哪个类别；在这之后，我们就可以在输出的类别上做一些其他分析工作。
- 优点：实现简单，易于理解和实现；计算代价不高，速度很快，存储资源低。
- 缺点：容易欠拟合，分类精度可能不高。

## 3.5. ANN 人工神经网络

### 3.5.1. ANN 算法由来

在传统机器学习中，我们建立模型，对模型输入特征矩阵，模型将特征矩阵转化为判断结果后为我们输出，如果将模型看作是单个神经元，特征矩阵看作是神经元接受的”信号“，模型对特征矩阵的处理过程看作神经元受到的刺激，最后输出的结果看成是神经元通过轴突传递出的信号， 那人类学习的模式是可以一定程度上被算法模仿的。人脑通过构建复杂的网络可以进行逻辑，语言，图像的学习，而传统机器学习算法不具备和人类相似的学习能力。机器学习研究者们相信，模拟大脑结构可以让机器的学 习能力更上一层楼，于是**人工神经网络**算法应运而生，现在基本都简称为“神经网络”。

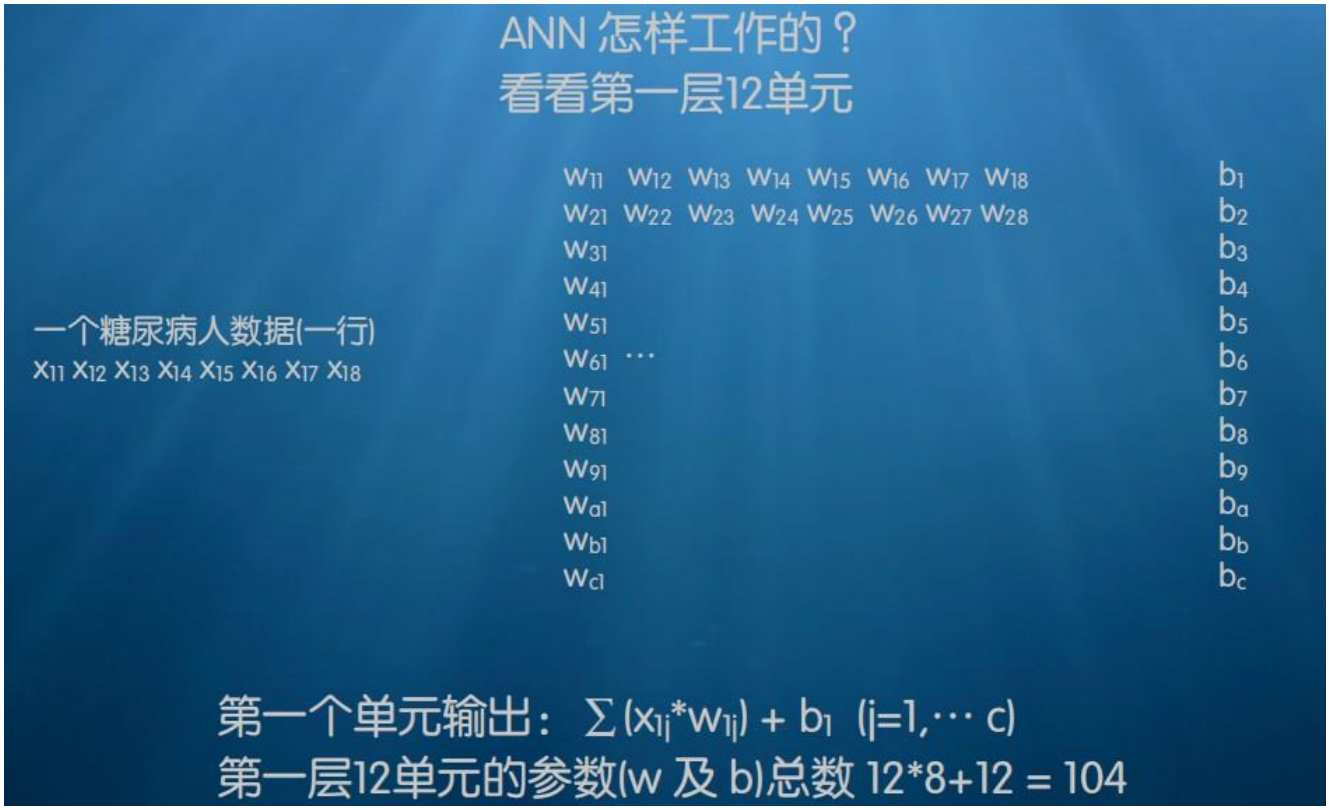



## 3.5.2. 通过代码构造和创建 ANN 模型

```
#创建模型
model = Sequential()

#输入层
model.add(Dense(12, input_dim=8, activation='relu'))
#12 个神经元和 8 个变量（数据有 8 个特征值）


#隐藏层

model.add(Dense(8,activation='relu'))

model.add(Dense(1, activation='sigmoid'))
model.summary()
#  输出层是有一个神经元来预测它的类别

adam = optimizers.Adam(lr=0.001)      # 默认 learning rate 0.001
```

```
loss = losses.binary_crossentropy

# 编译模型
# 需要接受三个参数
    # 优化器 optimizer。它可以是现有优化器的字符串标识符，如 rmsprop 或 adagrad，用的较多的是 adam，
    # 损失函数 loss，模型试图最小化的目标函数。它可以是现有损失函数的字符串标识符，
    # 多分类时为 categorical_crossentropy 或 mse，二分类时为 binary_crossentropy，也可以是一个目标函数。
    # 评估标准 metrics。对于任何分类问题，你都希望将其设置为 metrics = ['accuracy']。
    # 评估标准可以是现有的标准的字符串标识符，也可以是自定义的评估标准函数。

model.compile(loss='binary_crossentropy', optimizer='adam', metrics=['accuracy'])
```

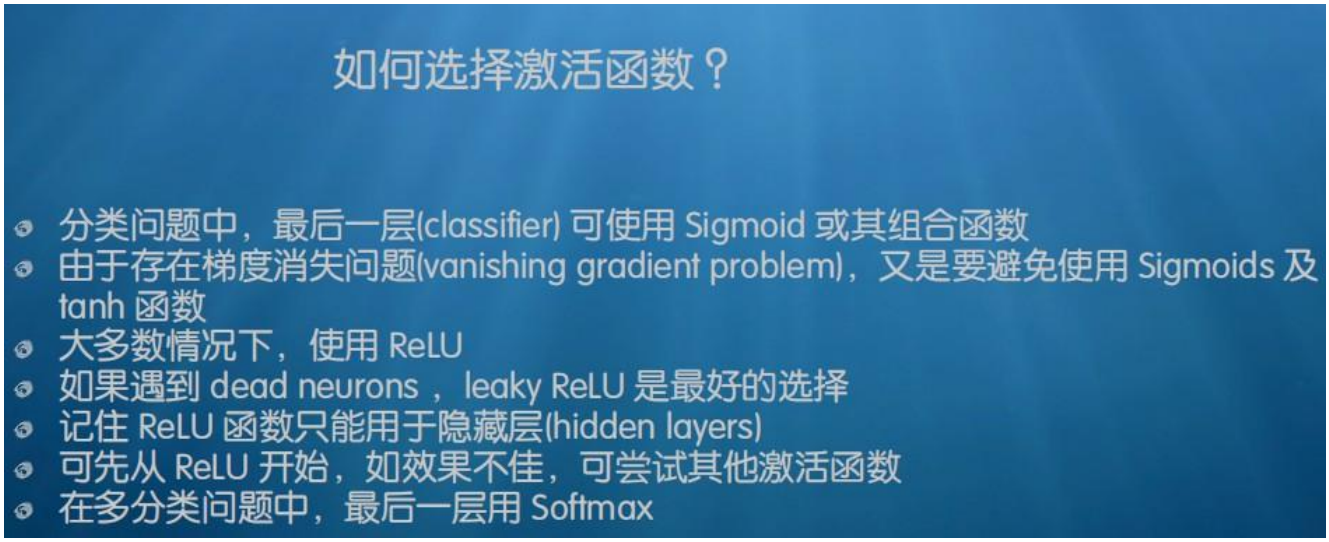
```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense (Dense)                (None, 12)                108
_________________________________________________________________
dense_1 (Dense)              (None, 8)                 104
_________________________________________________________________
dense_2 (Dense)              (None, 1)                 9
=================================================================
Total params: 221
Trainable params: 221
Non-trainable params: 0
_________________________________________________________________
```




## 3.5.3. 激活函数的作用

激活函数的作用是决定本神经元的输出是否传递到下一单元
使用激活函数的原因：

- 各层的输出 wx+b 是线性变换，当网络层数很大时，各神经元输出的总和会非常大，会引起严重的计算问题
- 激活函数最大的作用是将非线性引入 ANN，这对分类问题很重要，0、1 表明类别，0~1 之间值表示了可能性
- 非线性激活函数作用于输入数据可使 ANN 学习和完成更复杂的任务
- 没有激活函数的模型就是一个线性回归模型，它使多层多单元的网络行为如同一个神经元




## 3.5.4. 模型训练

```
history=model.fit(X_train, y_train, epochs=150, batch_size=32,validation_data=(X_test,y_test))

Epoch 1/150
17/17 [==============================] - 3s 33ms/step - loss: 0.7140 - accuracy: 0.4708 - val_loss: 0.6747 - val_accuracy: 0.6667
Epoch 2/150
17/17 [==============================] - 0s 7ms/step - loss: 0.6778 - accuracy: 0.6459 - val_loss: 0.6386 - val_accuracy: 0.6926
Epoch 3/150
17/17 [==============================] - 0s 8ms/step - loss: 0.6581 - accuracy: 0.6104 - val_loss: 0.6046 - val_accuracy: 0.6926
Epoch 4/150
17/17 [==============================] - 0s 9ms/step - loss: 0.6213 - accuracy: 0.6598 - val_loss: 0.5726 - val_accuracy: 0.6970
Epoch 5/150
17/17 [==============================] - 0s 7ms/step - loss: 0.5842 - accuracy: 0.7094 - val_loss: 0.5469 - val_accuracy: 0.6926

```

## 3.5.5. 模型评估

```
y_pred_ann=model.predict_classes(X_test)
accuracy_ann = accuracy_score(y_test,y_pred_ann)
print('{:s} : {:.2f}'.format('ANN 的准确率为:', accuracy_ann))
```

### 3.5.6. 建立混淆矩阵

```
from sklearn.metrics import confusion_matrix
m_ann = confusion_matrix(y_test,y_pred_ann)
print('预测结果混淆矩阵：\n',m_ann)
```

### 3.5.7. 混淆矩阵可视化

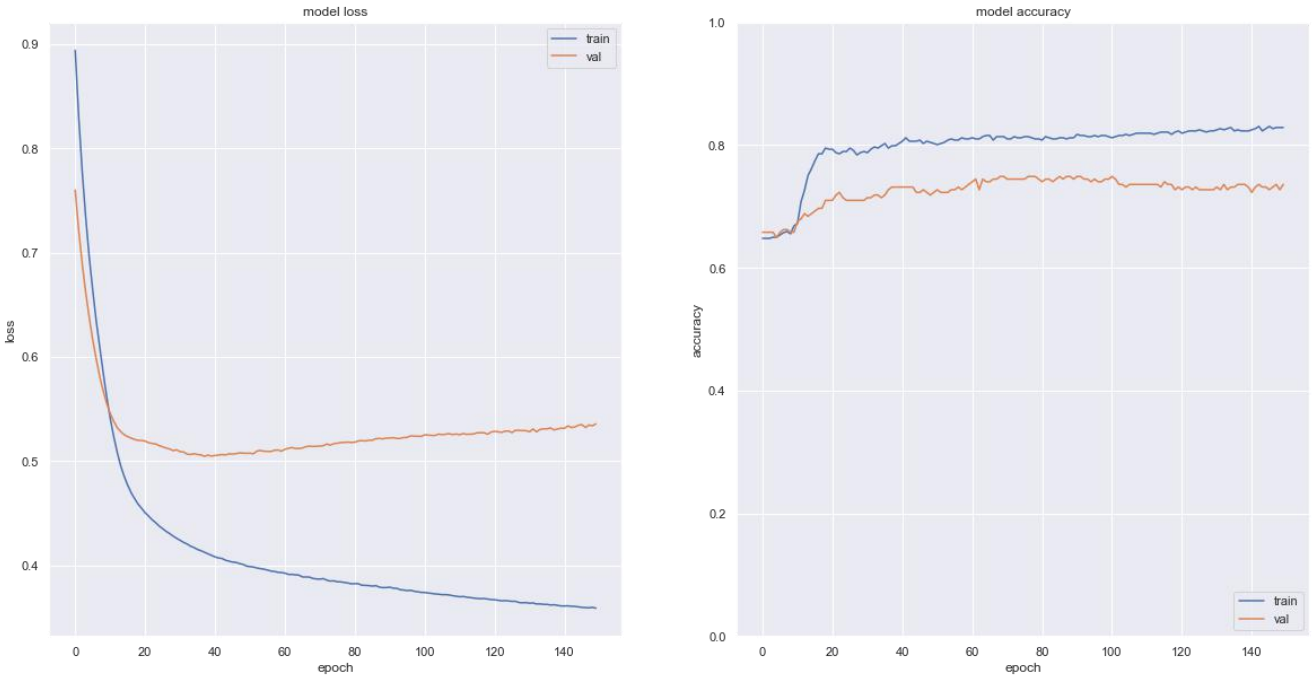
```
%pylab inline
import seaborn as sns
sns.heatmap(m_ann,cmap='Blues',annot=True)
```

### 3.5.8. 分类评估报告

```
from sklearn.metrics import classification_report

print(classification_report(y_test, y_pred_ann))
```

### 3.5.9. 训练集、测试集准确率、损失对比图

```
# summarize history for accuracy or loss
plt.figure(figsize=(20,10))
plt.subplot(121)
plt.plot(history.history['loss'])
plt.plot(history.history['val_loss'])
plt.title('model loss')
plt.ylabel('loss')
plt.xlabel('epoch')
#plt.ylim(0.0,6.0)
plt.legend(['train','val'], loc='upper right')

plt.subplot(122)
plt.plot(history.history['accuracy'])
plt.plot(history.history['val_accuracy'])
plt.title('model accuracy')
plt.ylabel('accuracy')
plt.xlabel('epoch')
plt.ylim(0.0,1.0)
plt.legend(['train','val'], loc='lower right')
plt.show()
```

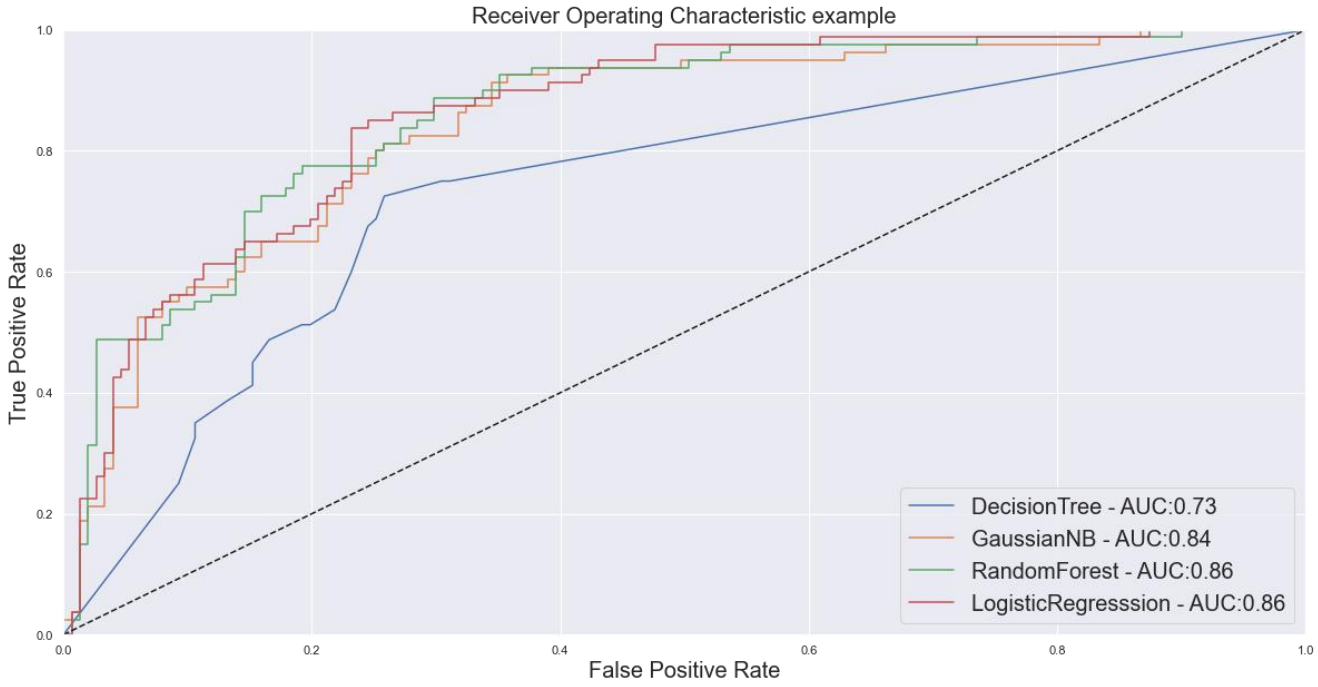



### 3.5.10. 总结

人工神经网络(ANN) 是人脑神经系统的简单抽象，它像人类大脑一样处理信息。它由大量相互连接的神经元(neuron)组成并协同处理某个特定问题。

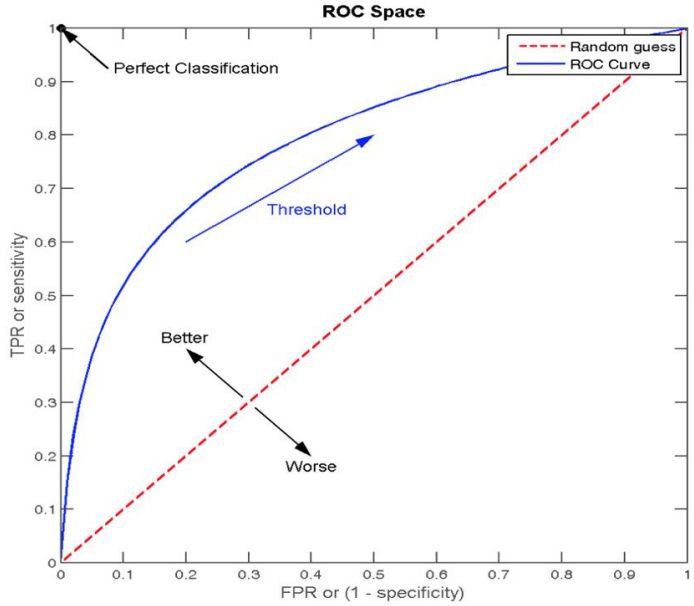
## 4. ROC-AUC 进行评估

```
import matplotlib.pyplot as plt
from sklearn.metrics import roc_curve,auc
plt.figure(figsize=[20,10])
for clf,title in zip([dt,gnb,rf,lr],['DecisionTree','GaussianNB','RandomForest','LogisticRegresssion']):
        probas_ = clf.fit(X_train,y_train).predict_proba(X_test)
        fpr,tpr,threshold = roc_curve(y_test,probas_[:,1])
        plt.plot(fpr,tpr,label='%s - AUC:%.2f'%(title,auc(fpr,tpr)))
plt.plot([0,1],[0,1],'k--')
plt.xlim([0.0,1.0])
plt.ylim([0.0,1.0])
plt.xlabel('False Positive Rate',fontsize=20)
plt.ylabel('True Positive Rate',fontsize=20)
plt.title('Receiver Operating Characteristic example',fontsize=20)
plt.legend(loc='lower right',fontsize=20)
plt.show()
```




## 4.1. ROC 曲线

- ROC 曲线的横轴就是 FPRate，纵轴就是 TPRate，当二者相等时，表示的意义则是：对于不论真实类别是 1 还是 0 的样本，分类器预测为 1 的概率是相等的，此时 AUC 为 0.5



## 4.2. AUC 指标

- AUC 的概率意义是随机取一对正负样本，正样本得分大于负样本的概率
- AUC 的最小值为 0.5，最大值为 1，取值越高越好
- **AUC=1，完美分类器，采用这个预测模型时，不管设定什么阈值都能得出完美预测。绝大多数预测的场合，不存在完美分类器。**
- **0.5<AUC<1，优于随机猜测。这个分类器（模型）妥善设定阈值的话，能有预测价值。**

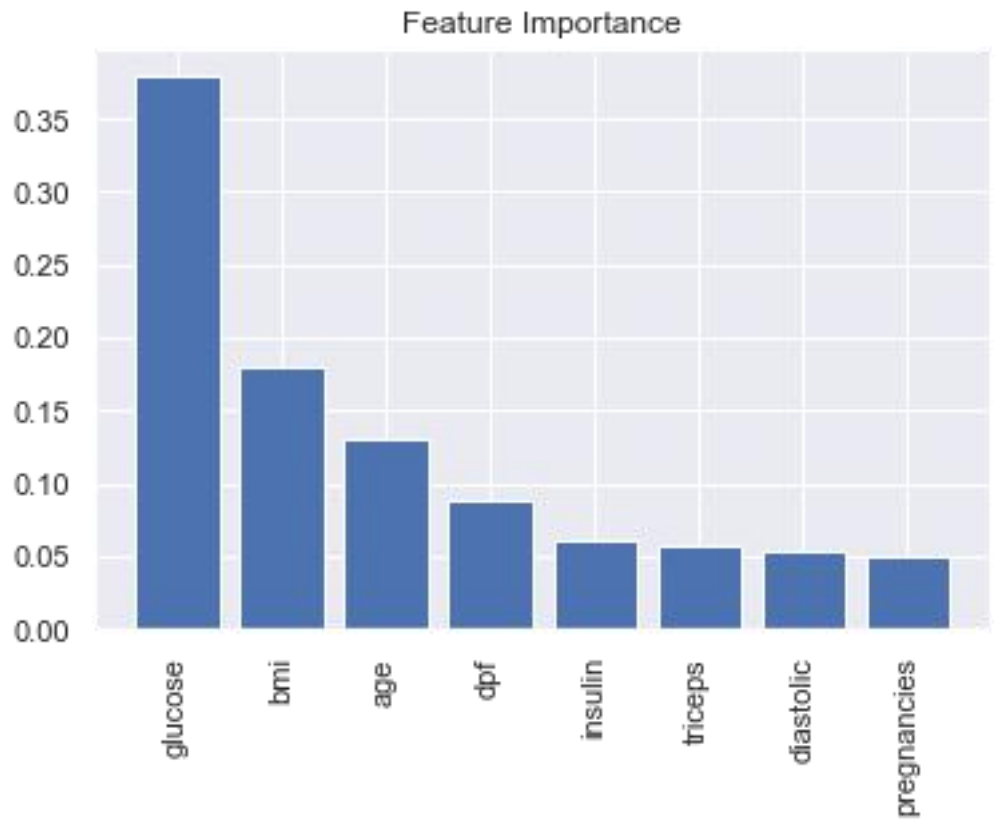
最终 AUC 的范围在[0.5, 1]之间，并且越接近 1 越好

## 4.3. 总结

- AUC 只能用来评价二分类
- AUC 非常适合评价样本不平衡中的分类器性能

# 5. 评估特征重要性

```
#  列出各个特征的重要性
rf.feature_importances_

#特征重要性从小到大返回其位置信息，并倒序从大到小输出
rf.feature_importances_.argsort()[::-1]
diabetes_data_copy.columns[rf.feature_importances_.argsort()[::-1]]

#特征的重要性可视化
import  matplotlib.pyplot  as  plt
importance  =  rf.feature_importances_
names  =  diabetes_data_copy.columns
plt.title('Feature  Importance')
plt.bar(range(1,len(names)),importance[importance.argsort()[::-1]])
plt.xticks(range(1,len(names)),names[importance.argsort()[::-1]],rotation=90)
plt.show()
```




## 6. 项目总结

通过完成本次课程项目以及书写报告文档，基本了解了对于一个数据如何进行分类的问题。首先拿到一个数据集需要分析那个目标值，那些是特征值，这是我们数据挖掘的第一步。对无关的特征需要进行降维处理，缺失值怎么处理，还有数据标准化等等问题。运用了多个算法，在汇报项目和书写文档的时候也重新复习了一下几个经典算法。其实对于本次项目特会最深的还是数据处理的部分，毕竟实现其算法大部分情况下都是调用 sklearn 中的 api，所以处理数据挖掘的问题最重要的还是对于数据怎么处理，怎么从中挖掘出重要的信息。最后，非常感谢汤老师对我们一学期的教学，祝老师身体健康，工作顺利！