

# 机器学习十大算法

# 目录

0. 引言 .....	1
0.1 监督学习 .....	1
0.2 无监督学习 .....	1
0.3 强化学习 .....	1
0.4 通用机器学习算法列表 .....	1
1. 线性回归 (Linear Regression) .....	2
2. 逻辑回归 (Logistic Regression) .....	3
3. 决策树 (Decision Tree) .....	4
4. 支持向量机 (SVM –Support Vector Machine) .....	5
5. 朴素贝叶斯 (Naive Bayes) .....	6
6. K 近邻 (KNN –K- Nearest Neighbors) .....	8
7. K 均值 (K-Means) .....	9
8. 随机森林 (Random Forest) .....	10
9. 降维算法 (Dimensionality Reduction Algorithms) .....	11
10. 梯度提升算法 (Gradient Boosting Algorithms) .....	11
10.1 GBM .....	11
10.2 XGBoost .....	12
10.3 LightGBM .....	12
10.4. Catboost .....	13
11. 总结 .....	14

# 0. 引言

整体来说，机器学习算法可以分为 3 大类：

## 0.1 监督学习

工作原理：该算法由自变量（协变量、预测变量）和因变量（结果变量）组成，由一组自变量对因变量进行预测。通过这些变量集合，我们生成一个将输入映射到输出的函数。训练过程达到我们设定的损失阈值停止训练，也就是使模型达到我们需要的准确度等水平。

监督学习的例子：回归，决策树，随机森林，KNN，逻辑回归等

## 0.2 无监督学习

工作原理：在无监督学习算法中，我们没有目标或结果变量来预测。通常用于不同群体的群体聚类。

无监督学习的例子：Apriori 算法，K-means。

## 0.3 强化学习

工作原理：强化学习(reinforcement learning)，又称再励学习、评价学习，学习不是单一方法，而是一种机器学习方式，在智能控制机器人及分析预测等领域有许多应用。

强化学习例子：马尔可夫决策过程

## 0.4 通用机器学习算法列表

此处列举常用的机器学习算法， 这些算法几乎可以应用于所有的数据问题：

1. 线性回归
2. Logistic 回归
3. 决策树

4. SVM
5. 朴素贝叶斯
6. KNN
7. K 均值
8. 随机森林
9. 降维算法
10. 梯度提升算法
  - 1.GBM
  - 2.XGBoost
  - 3.LightGBM
  - 4.CatBoost

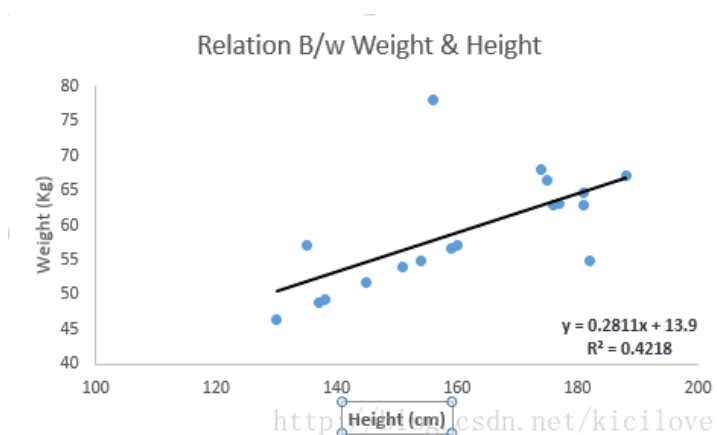
## 1. 线性回归 (Linear Regression)

它用于连续变量的估计（比如说房屋成本，总销售额等）。此处，我们通过拟合自变量和因变量之间的最佳拟合直线来建立它们之间的关系。这条线就被叫做回归线，由线性方程表示为： $y=a \cdot X+b$ 。

在上述等式中，各自代表意义如下：

- Y - 因变量
- a - 斜率，坡度
- X - 自变量
- b - 截距项

系数 a 和 b 是基于数据点与回归线之间的距离的差的平方之和最小化而得出的。看下面的例子。这里我们已经确定了具有线性方程  $y = 0.2811x + 13.9$  的最佳拟合线。现在使用这个等式，可以由一个人的重量，知道一个人的大体身高：



线性回归主要有两种类型：简单线性回归和多元线性回归：

- 简单线性回归的特征是一个自变量。
- 多元线性回归（顾名思义）是以多个（多于一个）自变量为特征的。

Python 代码

```
#Import Library
#Import other necessary libraries like pandas, numpy...
from sklearn import linear_model
#Load Train and Test datasets
#Identify feature and response variable(s) and values must be numeric and numpy arrays
x_train=input_variables_values_training_datasets
y_train=target_variables_values_training_datasets
x_test=input_variables_values_test_datasets
# Create linear regression object
linear = linear_model.LinearRegression()
# Train the model using the training sets and check score
linear.fit(x_train, y_train)
linear.score(x_train, y_train)
#Equation coefficient and Intercept
print('Coefficient: \n', linear.coef_)
print('Intercept: \n', linear.intercept_)
#Predict Output
predicted= linear.predict(x_test)
```

## 2. 逻辑回归（Logistic Regression）

这个算法的名字具有迷惑性，从建模任务的角度来看，这是一种分类而不是回归算法。它用于基于给定的一组自变量估计因变量（离散值（二进制值，如 0/1，是/否，真/假）），当然也用于多分类的任务。简而言之，它通过将数据拟合到 logit 函数来预测事件发生的可能性。因此，它也被称为 logit 回归。因为它预测了概率，其输出值在 0 和 1 之间：

从统计学的角度来看就是自变量的线性组合建模得出因变量 y 的结果，然后，这个线性组合被表示成分类事件发生概率与不发生概率的对数函数，具体公式如下：

$$\text{odds} = \frac{p}{1-p} = \frac{\text{事件发生的概率}}{\text{事件不发生的概率}}$$

$$\ln(\text{odds}) = \ln\left(\frac{p}{1-p}\right) = b_0 + b_1X_1 + b_2X_2 + b_3X_3 \dots + b_kX_k$$

以上， $p$  是存在的我们感兴趣的特征预测发生的概率。

Python 代码

```
#Import Library
from sklearn.linear_model import LogisticRegression

#Assumed you have, X (predictor) and Y (target) for training data set and x_test(predict
or) of test_dataset

# Create logistic regression object
model = LogisticRegression()

# Train the model using the training sets and check score
model.fit(X, y)

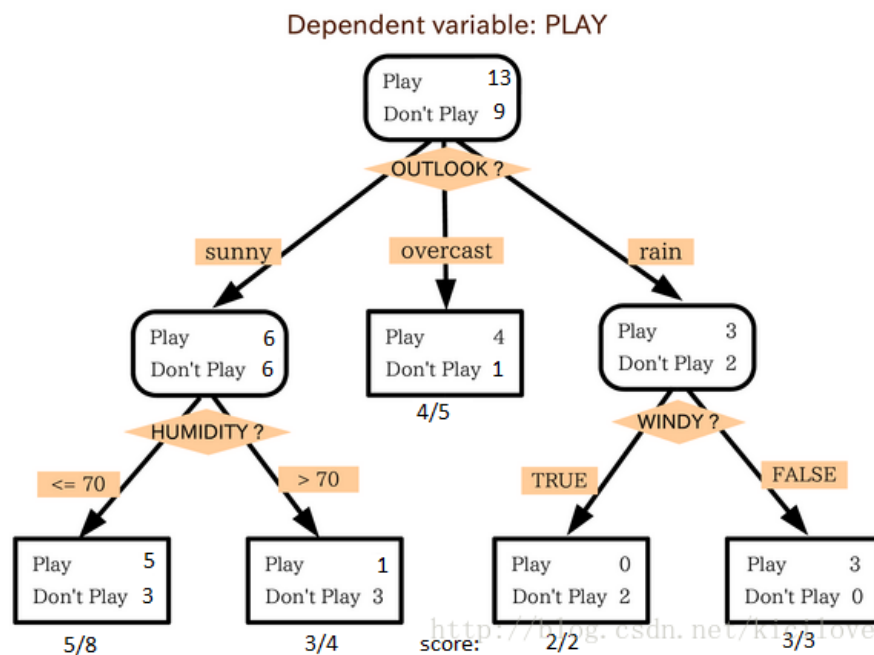
model.score(X, y)

#Equation coefficient and Intercept
print('Coefficient: \n', model.coef_)
print('Intercept: \n', model.intercept_)

#Predict Output
predicted= model.predict(x_test)
```

### 3. 决策树 (Decision Tree)

决策树算法是一种主要用于分类问题的监督学习算法。但是它是个两面身，既适用于分类也适用于连续因变量的预测也就是回归，所以通常会说回归树或者分类树。下面是一棵树的简化版：



在上面的图片中，可以看到根据多个属性最终将人口分为四个不同的组，以识别“他们是否会玩”。这其中根据不同的决策树算法又会使用基尼指数，信息增益，卡方，熵等多种不同的技术。

## Python 代码

```
#Import Library

#Import other necessary libraries like pandas, numpy...

from sklearn import tree

#Assumed you have, X (predictor) and Y (target) for training data set and x_test(predict
or) of test_dataset

# Create tree object

model = tree.DecisionTreeClassifier(criterion='gini') # for classification, here you ca
n change the algorithm as gini or entropy (information gain) by default it is gini
# model = tree.DecisionTreeRegressor() for regression

# Train the model using the training sets and check score

model.fit(X, y)

model.score(X, y)

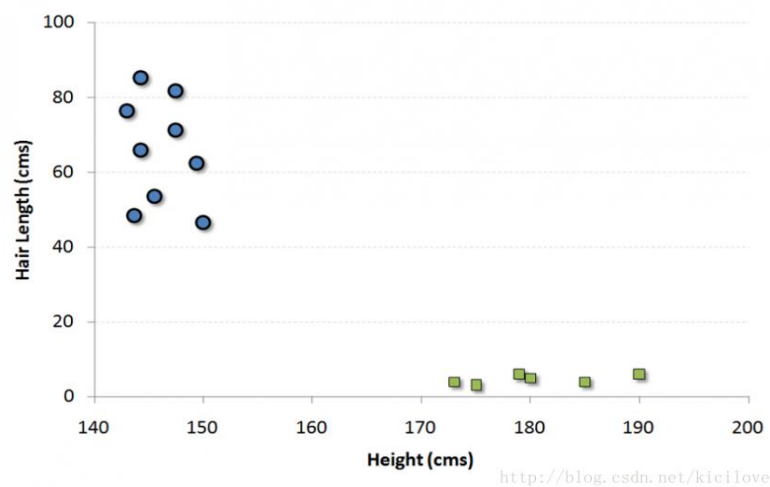
#Predict Output

predicted= model.predict(x_test)
```

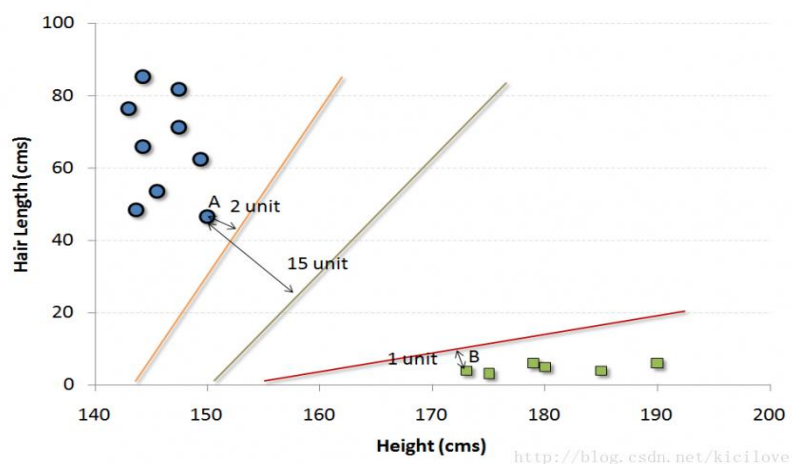
## 4. 支持向量机 (SVM - Support Vector Machine)

支持向量机是一种分类和回归算法。在这个算法中，我们将每个数据项绘制为  $n$  维空间中的一个点（其中  $n$  是您拥有的特征的数量），每个特征的值是特定坐标的值。也就是将数据映射到高维空间。

例如，如果我们只有一个人的身高和头发长度两个特征，我们首先将这两个变量绘制在二维空间中，每个点有两个坐标（这些坐标被称为支持向量）



现在，我们将找到一些将两个不同分类的数据分割的线。 我们真正要找的是这样一条线，两个组中离这条线各自最近的点，整体上相对于所有线来说这两个点到此线的距离最大。



在上面所示的例子中，将数据分成两个不同分类组的线是黑线，因为两个最近的点离线距离和最远。这条线就是我们的分类器。然后，根据测试数据在线两侧的位置，可以将新数据分类到相应的类别。

#### Python 代码

```
#Import Library
from sklearn import svm

#Assumed you have, X (predictor) and Y (target) for training data set and x_test(predict
or) of test_dataset

# Create SVM classification object
model = svm.svc() # there is various option associated with it, this is simple for classi
fication. You can refer link, for more detail.

# Train the model using the training sets and check score
model.fit(X, y)

model.score(X, y)

#Predict Output
predicted= model.predict(x_test)
```

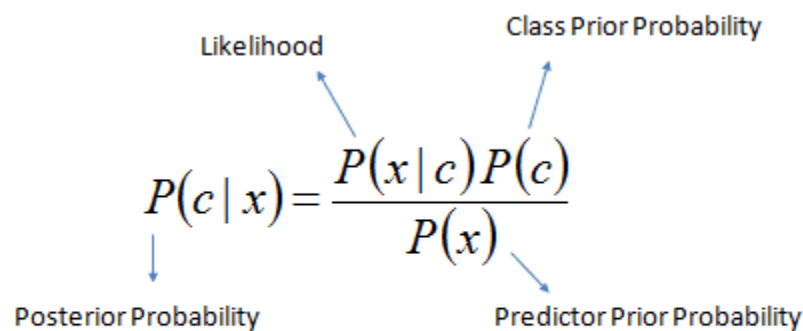
## 5. 朴素贝叶斯 (Naive Bayes)

朴素贝叶斯是一种基于贝叶斯定理的分类算法，假设条件时预测变量之间是相互独立的。简而言之，朴素贝叶斯分类器假设类中特定特征的存在与任何其他特征的存在无关。例如，如果水果是红色的，圆形的，并且直径约 3 英寸，则水果可以被认为是苹果。即使这些特征彼此依赖或者依赖于其他特征的存在，朴素贝叶斯分类器也会考虑所有这些特性来独立地贡献这个果实是苹果的可能性。



朴素贝叶斯模型易于构建，特别适用于非常大的数据集。

贝叶斯定理提供了一种从  $P(c)$ ， $P(x)$  和  $P(x|c)$  计算后验概率  $P(c|x)$  的方法。 看下面的公式：



The diagram shows the formula  $P(c|x) = \frac{P(x|c)P(c)}{P(x)}$  with four labels and arrows: 'Likelihood' points to  $P(x|c)$ , 'Class Prior Probability' points to  $P(c)$ , 'Posterior Probability' points to  $P(c|x)$ , and 'Predictor Prior Probability' points to  $P(x)$ .

$$P(c|X) = P(x_1|c) \times P(x_2|c) \times \cdots \times P(x_n|c) \times P(c)$$

这里：

- $P(c|x)$  是给定预测器（属性）的类（目标）的后验概率。
- $P(c)$  是判断为此类的先验概率。
- $P(x|c)$  是预测器给定类的概率的可能性。
- $P(x)$  是自变量  $X$  发生的先验概率。

例子：让我们用一个例子来理解它。 下面我有一个天气为自变量及相应的目标变量“Play”组成的训练数据集。 现在，我们需要根据天气状况来分类玩家是否玩游戏。 让我们按照下面的步骤来执行它。

步骤 1：将数据集转换为频率表

步骤 2：从表中可以创建右侧似然表格，可以计算出 Overcast 的比例=0.29，Play 的频率 = 0.64。

Weather	Play
Sunny	No
Overcast	Yes
Rainy	Yes
Sunny	Yes
Sunny	Yes
Overcast	Yes
Rainy	No
Rainy	No
Sunny	Yes
Rainy	Yes
Sunny	No
Overcast	Yes
Overcast	Yes
Rainy	No

Frequency Table		
Weather	No	Yes
Overcast		4
Rainy	3	2
Sunny	2	3
Grand Total	5	9

Likelihood table		
Weather	No	Yes
Overcast		4
Rainy	3	2
Sunny	2	3
All	5	9
	=5/14	=9/14
	0.36	0.64

<http://blog.csdn.net/kicilove>

步骤 3: 现在, 使用朴素贝叶斯公式来计算每个类别的后验概率。 后验概率最高的是预测相应类的结果。

问题: 如果天气晴朗, 玩家会出去玩, 这个说法是否正确?

我们可以用上面讨论的方法来解决这个问题,

$$P(\text{Yes}|\text{Sunny}) = P(\text{Sunny}|\text{Yes}) * P(\text{Yes}) / P(\text{Sunny})$$

这里我们有  $P(\text{Sunny}|\text{Yes}) = 3/9 = 0.33$ ,  $P(\text{Sunny}) = 5/14 = 0.36$ ,  $P(\text{Yes}) = 9/14 = 0.64$

现在,  $P(\text{Yes}|\text{Sunny}) = 0.33 * 0.64 / 0.36 = 0.60$ , 其概率较高, 因此我们就认为晴天出去玩的可能性大。

Python 代码

```
#Import Library
from sklearn.naive_bayes import GaussianNB

#Assumed you have, X (predictor) and Y (target) for training data set and x_test(predictor) of test_dataset

# Create SVM classification object model = GaussianNB() # there is other distribution for multinomial classes like Bernoulli Naive Bayes, Refer link

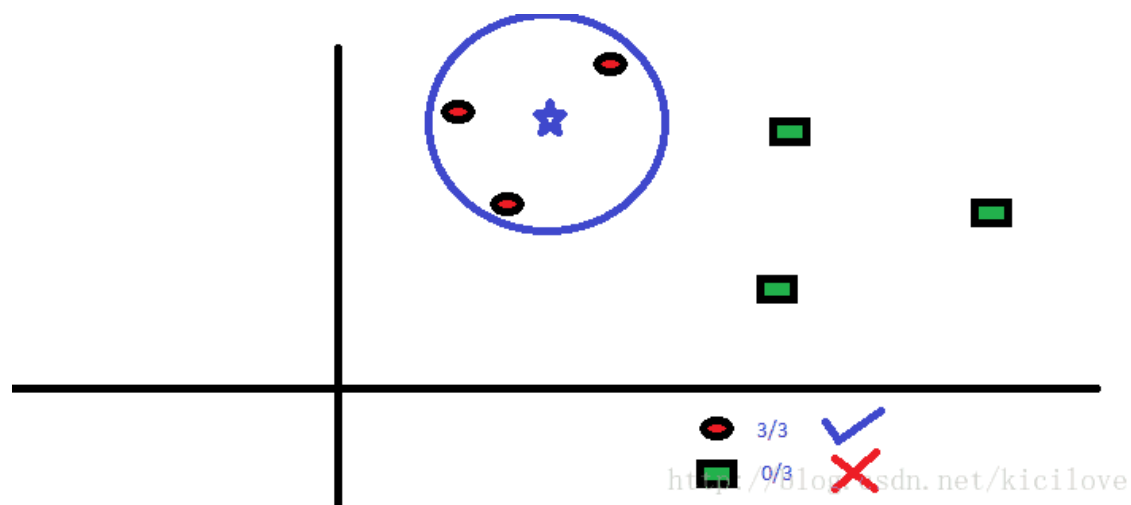
# Train the model using the training sets and check score
model.fit(X, y)

#Predict Output
predicted= model.predict(x_test)
```

## 6. K 近邻 (KNN - K- Nearest Neighbors)

K 近邻算法可以用于分类和回归问题。但是, 它在业内在分类问题上用的比较广泛。K 个最近邻居是对这个算法的简单的描述, 通过其 k 个邻居的多数投票来分类。某数据被分配到某一类是由距离函数测量的 K 个最近的邻居中是最常出现的类别为它相应的类。

这些距离函数可以是欧几里得，曼哈顿，闵可夫斯基和海明距离等。前三个函数用于连续函数，第四个函数（Hamming）用于分类变量。有时候，在执行 KNN 建模时选择 K 是一个挑战。



KNN 可以很容易地映射到我们的真实生活中。如果你想了解一个你没有任何信息的人，你可能想知道他的密友和他的圈子，并因此获得他/她的信息，类似六度空间理论！

#### Python 代码

```
#Import Library
from sklearn.neighbors import KNeighborsClassifier
#Assumed you have, X (predictor) and Y (target) for training data set and x_test(predict
or) of test_dataset
# Create KNeighbors classifier object model
KNeighborsClassifier(n_neighbors=6) # default value for n_neighbors is 5
# Train the model using the training sets and check score
model.fit(X, y)
#Predict Output
predicted= model.predict(x_test)
```

## 7. K 均值 (K-Means)

K 均值是一种解决聚类问题的无监督算法。通过一定数量的聚类（假设  $k$  个聚类）对给定的数据集进行分类。集群内的数据点对同组来说是同质且异构的，就是类里尽量距离最近类间尽量最大距离。

### K-means 如何形成群类：

- 1.K-means 为每个簇选取  $k$  个点，称为质心。
- 2.每个数据点形成具有最接近的质心（即  $k$  个聚类）的聚类。

- 3.根据现有集群成员查找每个集群的质心。 这里我们有新的质心。
- 4.由于我们有新的质心，请重复步骤 2 和 3.从新质心找到每个数据点的最近距离，并与新的 k-簇进行关联。 重复这个过程直到收敛发生，即质心不变。

#### Python 代码

```
#Import Library
from sklearn.cluster import KMeans

#Assumed you have, X (attributes) for training data set and x_test(attributes) of test_d
ataset

# Create KNeighbors classifier object model
k_means = KMeans(n_clusters=3, random_state=0)

# Train the model using the training sets and check score
model.fit(X)

#Predict Output
predicted= model.predict(x_test)
```

## 8. 随机森林（Random Forest）

随机森林是决策树的融合。在随机森林中，我们收集了决策树（所谓的“森林”）。为了根据属性对一个新进来的数据进行分类，每棵树都给出了该数据的一个分类，这些棵树对类别进行“投票”。森林选择票数最多的分类为此新数据的类。每棵树按如下方式生长：

- 1.如果训练集中的样本数为 N，从这 N 个样本中有放回随机抽取样本，这些个样本用来训练生成树。
- 2.如果有 M 个输入变量，则指定一个数  $m \ll M$ ，使得在每个节点处，从 M 中随机选择 m 个变量，并且使用这些 m 上的最佳分割来分割节点。在森林生长期间，m 的值保持不变。
- 3.每棵树都尽可能地长大。 不给修剪。

#### Python 代码

```
#Import Library
from sklearn.ensemble import RandomForestClassifier

#Assumed you have, X (predictor) and Y (target) for training data set and x_test(predict
or) of test_dataset

# Create Random Forest object
model= RandomForestClassifier()

# Train the model using the training sets and check score
model.fit(X, y)

#Predict Output
predicted= model.predict(x_test)
```

## 9. 降维算法 (Dimensionality Reduction Algorithms)

在过去的 4-5 年里，采集的数据呈指数级增长。例如：电子商务公司正在抓住更多关于客户的细节，例如他们的上网历史，他们喜欢什么或不喜欢什么，购买历史等，使他们比杂货店店主更注重客户个性化推荐。降维算法可以帮助我们连同决策树，随机森林，主成分分析，因子分析等进行数据分析。

### Python 代码

```
#Import Library
from sklearn import decomposition

#Assumed you have training and test data set as train and test

# Create PCA object pca= decomposition.PCA(n_components=k) #default value of k =min(n_s
ample, n_features)

# For Factor analysis
#fa= decomposition.FactorAnalysis()

# Reduced the dimension of training dataset using PCA
train_reduced = pca.fit_transform(train)

#Reduced the dimension of test dataset
test_reduced = pca.transform(test)
```

## 10. 梯度提升算法 (Gradient Boosting Algorithms)

### 10.1 GBM

GBM 是一种提升算法，处理大量的数据提高预测性能。Boosting 实际上是一些弱学习算法的集合，它结合了几个弱算法估计量的预测，以提高预测的鲁棒性。它将多个弱预测指标或平均预测指标组合成一个强预测指标。这些提升算法在 Kaggle，AV Hackathon，CrowdAnalytix 等数据科学竞赛中表现突出。

### Python 代码

```
#Import Library
from sklearn.ensemble import GradientBoostingClassifier

#Assumed you have, X (predictor) and Y (target) for training data set and x_test(predict
or) of test_dataset

# Create Gradient Boosting Classifier object
model= GradientBoostingClassifier(n_estimators=100, learning_rate=1.0, max_depth=1, ran
dom_state=0)

# Train the model using the training sets and check score
model.fit(X, y)
```

```
#Predict Output
predicted= model.predict(x_test)
```

GradientBoostingClassifier 和 Random Forest 是两个不同的提升树分类器。

## 10.2 XGBoost

另一个经典的增强算法，被称为 **Kaggle 神器**。因为使用这个算法得当，往往会取得较好的名次。

XGBoost 具有非常好的预测能力，它有线性模型和树模型算法，该算法比现有 GBM 快近 10 倍。

支持包括各种目标函数，包括回归，分类和排序。

XGBoost 也被叫做正则化的提升算法，因为它可以使用正则化减少模型的过拟合。Scala, Java, R, Python, Julia 和 C++等一系列语言支持 XGBoost。

关于 XGBoost 的参数和调参指南，请参考[这篇文章](#)，给出了参数详解以及调参步骤以及实例分析。

Python 代码

```
from xgboost import XGBClassifier
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score

X = dataset[:,0:10]
Y = dataset[:,10:]
seed = 1

X_train, X_test, y_train, y_test = train_test_split(X, Y, test_size=0.33, random_state=
seed)

model = XGBClassifier()

model.fit(X_train, y_train)

#Make predictions for test data
y_pred = model.predict(X_test)
```

## 10.3 LightGBM

LightGBM 是一个梯度提升框架，使用基于树的学习算法。它被设计成分布式，具有以下优点：

- 更快的训练速度和更高的效率
- 降低内存使用量

- 更好的准确性
- 支持并行和 GPU 学习
- 能够处理大规模的数据

该框架是一种基于决策树算法的快速高性能梯度提升算法,用于排序,分类等多种机器学习任务。它是在 Microsoft 的分布式机器学习工具包项目下开发的。

#### Python 代码

```
data = np.random.rand(500, 10) # 500 entities, each contains 10 features
label = np.random.randint(2, size=500) # binary target

train_data = lgb.Dataset(data, label=label)
test_data = train_data.create_valid('test.svm')

param = {'num_leaves':31, 'num_trees':100, 'objective':'binary'}
param['metric'] = 'auc'

num_round = 10
bst = lgb.train(param, train_data, num_round, valid_sets=[test_data])

bst.save_model('model.txt')

# 7 entities, each contains 10 features
data = np.random.rand(7, 10)
ypred = bst.predict(data)
```

## 10.4. Catboost

CatBoost 是 Yandex 最近开源的机器学习算法。它可以很容易地与谷歌的 TensorFlow 和苹果公司的核心 ML 等深度学习框架相结合。

CatBoost 最棒的地方在于它不需要像其他 ML 模型那样要大量数据训练,对于不同的数据格式它也可以应付自如。

#### Python 代码

```
import pandas as pd
import numpy as np

from catboost import CatBoostRegressor

#Read training and testing files
train = pd.read_csv("train.csv")
test = pd.read_csv("test.csv")
```

```

#Imputing missing values for both train and test
train.fillna(-999, inplace=True)
test.fillna(-999, inplace=True)

#Creating a training set for modeling and validation set to check model performance
X = train.drop(['Item_Outlet_Sales'], axis=1)
y = train.Item_Outlet_Sales

from sklearn.model_selection import train_test_split

X_train, X_validation, y_train, y_validation = train_test_split(X, y, train_size=0.7, r
andom_state=1234)
categorical_features_indices = np.where(X.dtypes != np.float)[0]

#importing library and building model
from catboost import CatBoostRegressor(model=CatBoostRegressor(iterations=50, depth=3, l
earning_rate=0.1, loss_function='RMSE')

model.fit(X_train, y_train, cat_features=categorical_features_indices, eval_set=(X_valid
ation, y_validation), plot=True)

submission = pd.DataFrame()

submission['Item_Identifier'] = test['Item_Identifier']
submission['Outlet_Identifier'] = test['Outlet_Identifier']
submission['Item_Outlet_Sales'] = model.predict(test)

```

## 11. 总结

到目前为止，我相信，你会对常用的机器学习算法在脑袋中形成一个大体的框架。那就马上开始，如果你热衷于掌握机器学习，拿起问题，应用这些代码，享受建模的乐趣！