# 数学知识



微积分、线性代数、概率论、最优化方法。

推荐书籍：《数学分析新讲》-张筑生，《线性代数》-史蒂文 J.利昂，《凸优化》-Stephen Boyd，《非线性规划》-Dimitri P. Bertsekas

# 编程能力

推荐书籍：《C++ Primer》，《Effective C++》，《算法导论》

# 机器学习与深度学习

推荐书籍：《Pattern Recognition and Machine Learning》，《深度学习》，《Reinforcement Learning》

# 开源库

livsvm，liblinear，XGBoost，OpenCV，HTK，Weka，TensorFlow，lightgbm

# ML流程

1. 数据清洗
   1. 空值的填充：字符类型 – 用’None’、众数填充；数值类型 – 用0、均值填充。
   2. 字符变成数值：LabelEncoder，pandas.get\_dummies。
   3. 管道建设pipeline实现数据预处理：对于skew较大的数值进行np.log1p处理。

|  |
| --- |
| 1. ##自己写一个转换函数 2. **class** labelenc(BaseEstimator, TransformerMixin): 3. **def** \_\_init\_\_(self): 4. **pass** 6. **def** fit(self,X,y=None): 7. **return** self 8. ##对三个年份来进行一个标签编码,这里可以随便自己添加 9. **def** transform(self,X): 10. lab=LabelEncoder() 11. X["YearBuilt"] = lab.fit\_transform(X["YearBuilt"]) 12. X["YearRemodAdd"] = lab.fit\_transform(X["YearRemodAdd"]) 13. X["GarageYrBlt"] = lab.fit\_transform(X["GarageYrBlt"]) 14. X["BldgType"] = lab.fit\_transform(X["BldgType"]) 16. **return** X 18. #转换函数 19. **class** skew\_dummies(BaseEstimator, TransformerMixin): 20. **def** \_\_init\_\_(self,skew=0.5):#偏度 21. self.skew = skew 23. **def** fit(self,X,y=None): 24. **return** self 26. **def** transform(self,X): 27. X\_numeric=X.select\_dtypes(exclude=["object"])#而是去除了包含了对象数据类型，取出来绝大部分是数值型 28. skewness = X\_numeric.apply(**lambda** x: skew(x))#匿名函数，做成字典的形式 29. skewness\_features = skewness[abs(skewness) >= self.skew].index#通过条件来涮选出skew>=0.5的索引的条件，取到了全部数据，防止数据的丢失 30. X[skewness\_features] = np.log1p(X[skewness\_features])#求对数，进一步让他更符合正态分布 31. X = pd.get\_dummies(X)##一键独热，独热编码，（试错经历） 32. **return** X 34. # 构建管道 35. pipe = Pipeline([##构建管道的意思 36. ('labenc', labelenc()), 37. ('skew\_dummies', skew\_dummies(skew=2)), 38. ]) 40. # 保存原来的数据以备后用，为了防止写错 41. full2 = full.copy() 42. pipeline\_data = pipe.fit\_transform(full2) |

d）利用Lasso等算法进行训练根据coef\_绘制特征的重要性图，然后对特征进行选择，删除不重要的特征，及重做。

## 数据清洗

理解特征：数据特征变量、目标变量的分布，分析变量之间的相关度，文本数据清洗，

数据观察：

1. Outliers-sns.boxplot观察每个特征的分布状况-剔除或者用中位数替换。
2. 如果数据类型时np.float64，为减少内存消耗可转换类型为np.float32。

数据重做：

1）如日期格式变换。

### 正态性检验

若随机变量X服从一个数学期望为、方差为的正态分布，记为，其概率密度函数为：

1. 直方图初判

|  |
| --- |
| 1. s = pd.DataFrame(np.random.randn(1000) + 10, columns=['value']) 2. fig = plt.figure(figsize=(10, 6)) 3. ax1 = fig.add\_subplot(1,1,1) 4. s.hist(bins=50, alpha=0.5, ax=ax1) 5. s.plot(kind='kde', secondary\_y=True, ax=ax1) 6. plt.grid() |

1. K-S检验：将样本数据的累计频数分布与特定的理论分布比较，如果两者差距小，则推论样本分布取自特定的分布。

假设检验问题：H0 - 样本的总体分布服从某特定分布；H1 – 样本的总体分布不服从某特定分布。

其中，D是与差值绝对值的最大值。。

相比较，若则接受H0；则拒绝H0，接受H1。

|  |
| --- |
| 1. **from** scipy **import** stats 2. data = [87,77,92,68,80,78,84,77,81,80,80,77,92,86, 3. 76,80,81,75,77,72,81,72,84,86,80,68,77,87, 4. 76,77,78,92,75,80,78] 5. df = pd.DataFrame(data, columns=['value']) 6. u = df['value'].mean() 7. std = df['value'].std() 9. stats.kstest(df['value'], 'norm', (u, std)) 10. # KstestResult(statistic=0.1590180704824098, pvalue=0.3066297258358026) |

针对一些长尾分布的特征，需进行幂变换或者对数变换，使得模型能更好的优化。

### 缺失值填充

用均值、中位数填充，或把缺失值视为一种特殊的值如-1（针对xgboost）。

### 类别型特征处理

1）独热编码OneHotEncoding

对于每一个特征，如果它有m个可能值，那么经过独热编码后，就变成了m个二元特征。并且，这些特征互斥，每次只有一个激活。因此，数据会变成稀疏的。

|  |
| --- |
| 1. **from** sklearn **import** preprocessing 3. encoder = preprocessing.OneHotEncoder() 4. x = [[0, 2, 1, 12], 5. [1, 3, 5, 3], 6. [2, 3, 2, 12], 7. [1, 2, 4, 3]] 8. encoder.fit(x) 9. encoder\_vector = encoder.transform(x).toarray() 10. # 或 11. encoder\_vector = encoder.fit\_transform(x).toarray() |

上述代码中，第一个特征（即第一列）为[0,1,2,1]，共有三类特征值，One-Hot Code表示为[100, 010, 001]。

2）序号编码

3）二进制编码

对血型类别（A、B、AB、O），一）先用序号编码给每个类别赋予一个类别ID，如A的ID为1，二进制表示为001，B的ID为2，二进制表示为010，AB表示为011，O表示为100。

本质上是利用二进制对ID进行哈希映射，最终得到0/1特征向量，且维数少于独热编码，节省了存储空间。

4）一个单一的类别特征，往往类别太多会让人迷乱，一般不想超过8~10列，即尽量找到重要的类别。

参考链接：

1）回归模型中的哑变量：http://www.sohu.com/a/199698358\_489312

### 连续变量处理

对连续变量分组，能够更加简便的了解观测值的分布。

Bining、Regularization

### 类别不均衡

调节权重、under sampling、over sampling、增加惩罚项

### 相关性分析

计算correlation coefficient来确定特征之间的两两关系，还可以计算每个特征和目标值之间的相关性，绝对值越大说明这个特征的作用越大。

### 模型权重分析

在训练完成之后通过接口获取特征权重，其直接反映了一个特征在这个模型中的作用。

## 特征工程

1. 特征筛选/降维

共线性分析、IV（Information Value）值处理、Gini/Info Gain、stepwise、PCA/AHP、Variable Threshold、卷积

一）Info Gain

熵：表示一个系统的混乱程度。系统的不确定性越高，熵就越大。设D为用类别对训练元组进行的划分，则D的熵（Entropy）为：

其中，表示第个类别在整个训练元组中出现的概率。

现假设将训练元组D按属性A进行划分，则A对D划分的期望信息为：

则信息增益为两者的差值：

二）Chi-squared test（卡方检验）

三）Correlation Coefficient Scores（相关系数）

1. 特征构造（Feature Construction）
2. 特征提取（Feature Extraction）

### 多重共线性

指回归模型中存在两个或两个以上的自变量彼此相关。

存在的影响：

1. 变量之间高度相关，可能使回归的结果混乱，甚至把分析引入歧途。
2. 在线性回归模型中，对参数估计值的正负号产生影响，特别是估计系数的符号可能与预期的正相反。

共线性问题的检验方法：

1. 相关性分析：检验变量之间的相关系数。

规避共线性问题的方法：

1. PCA等降维法
2. 岭回归：引入L2惩罚项，或称权值衰减（Weight Decay）
3. Lasso回归：引入L1惩罚项
4. ElasticNet回归：同时引入L1惩罚项和L2惩罚项

参考链接：

讲讲共线性问题：<https://www.jianshu.com/p/ef1b27b8aee0?from=timeline>

### IV值处理

可以用来衡量自变量的预测能力，类似的指标还有信息增益、基尼系数等。

WOE（Weight of Evidence），是对原始自变量的一种编码形式。要对一个变量进行WOE编码，需首先把这个变量进行分组处理。分组后，对于第组其WOE为：

在违约模型中，预测变量取值为“是”或“否”。则指组中响应客户占样本中所有响应客户的比例，指组中未响应客户占样本中所有未响应客户的比例。WOE越大，这个分组里的样本响应的可能性就越大。

对于第组，对应的值为：

则该自变量的IV值为：

IV的特点：

1. 对于变量的一个分组，其响应和未响应的比例与样本整体响应和未响应的比例相差越大，IV值越大；
2. 极端情况下，当前分组的响应和未响应的比例与样本整体响应和未响应的比例相等时，IV值为0；
3. IV值的取值范围是，且当当前分组只包含响应客户或未响应客户时，。

为什么用IV而不是用WOE：

1. 当衡量一个变量的预测能力时，所使用的指标值不应该是负数。
2. 乘以后，体现出了变量当前分组中个体的数量占整体数量的比例对变量预测能力的影响。如变量A只取0或1，取1时其客户数太少，即使响应比例很高，对于样本整体来说，变量A的预测能力并没有那么强。系数很好的考虑了分组中样本占整体样本的比例，该比例越低，这个分组对整体预测能力的贡献越低。

当IV为时的处理方式：

1. 把这个分组做成一个规则，作为模型的前置条件。
2. 重新对变量进行离散化或分组。
3. 若上述两种方法都无法使用，人工把该分组的响应数或非响应数由0调整为1。
4. 将响应客户、未响应客户均加上0.5，防止分母为0。

参考链接：

1. 数据挖掘模型中的IV和WOE详解：

https://blog.csdn.net/kevin7658/article/details/50780391

### 特征归一化

为了消除数据特征之间的量纲影响，需进行特征归一化（Normalization）处理，使各指标处于同一数值量级，以便进行分析。

1. 线性函数归一化（Min-Max Scaling）
2. 零均值归一化（Z-Score Normalization）

Why？假设有两种数值类型特征和，如果将其归一化到相同的数值区间后，和的更新速度变得更为一致，容易更快地通过梯度下降找到最优解。

信息增益比跟特征是否经过归一化是无关的，因为归一化并不会改变样本在特征上的信息增益。

如果使用的模型是基于决策树的，则不需要对特征进行归一化处理。

### 实现方法

1. 去掉取值变化小的特征。

离散型变量：假设某特征的值只有0和1，并且在所有的输入样本中，95%的实例的该特征取值都是1，那就可以认为这个特征的作用不大。

连续型变量：需要将连续变量离散化之后才能用。

1. 单变量特征选择方法。

独立的衡量每个特征与响应变量之间的关系。

一、Pearson相关系数：衡量变量之间的线性相关性。其主要用于连续型特征的筛选，不适用于离散型特征的筛选。

二、互信息和最大信息系数。

熵与条件熵之间的差称为互信息。

互信息法评价定性自变量对定性因变量的相关性，其只能用于离散型特征的选择，连续型特征需要离散化才能用互信息进行特征选择。

|  |
| --- |
| 1. x = np.random.normal(0,10,300) 2. z = x \*x 3. **print**(pearsonr(x,z)) # 输出-0.1 4. **from** minepy **import** MINE 5. m = MINE() 6. m.compute\_score(x, z) 7. **print**(m.mic()) # 输出1.0 |

三、距离相关系数：克服Pearson相关系数只能识别线性相关性的弱点而产生的。

四、基于学习模型的特征排序

直接使用要用的机器学习算法，针对每个单独的特征和响应变量建立预测模型。

|  |
| --- |
| 1. **from** sklearn.model\_selection **import** cross\_val\_score, ShuffleSplit 2. **from** sklearn.datasets **import** load\_boston 3. **from** sklearn.ensemble **import** RandomForestRegressor 5. #Load boston housing dataset as an example 6. boston = load\_boston() 7. X = boston["data"] 8. Y = boston["target"] 9. names = boston["feature\_names"] 11. rf = RandomForestRegressor(n\_estimators=20, max\_depth=4) 12. scores = [] 13. # 使用每个特征单独训练模型，并获取每个模型的评分来作为特征选择的依据。 14. **for** i **in** range(X.shape[1]): 15. score = cross\_val\_score(rf, X[:, i:i+1], Y, scoring="r2", cv=ShuffleSplit(len(X), 3, .3)) 16. scores.append((round(np.mean(score), 3), names[i])) 17. **print**(sorted(scores, reverse=True)) |

五、卡方检验

卡方值描述两个事件的独立性或者描述实际观测值与期望值的偏离程度。卡方值越大，表明实际观测值与期望值偏离越大，即两个事件的相关程度越高。

a.对于特征变量，以及分类变量，只需要计算，并按照CHI的值从大到小进行排序。

b. 选择合适的阈值，大于阈值的特征留下，小于阈值的特征删除。

只适用于分类问题中离散型特征筛选，不能用于分类问题中的连续型特征的筛选，也不能用于回归问题的特征筛选。

参考链接：

1、特征筛选的原理与实现（上）：https://mp.weixin.qq.com/s/WZv1wtkCXFG1eg1LHt3Kog

## 调参

Grid Search：网格搜索/穷举搜索：给定每个待调参数的几个选择，然后排列组合出所有可能性，做Cross Validation，然后挑选出最好的那组参数组合。

Random Search：随机搜素：不直接给定参数的有限个取值可能，而是给出一个参数分布，从这个分布中随机采样一定个数的取值。

Heuristic Tuning：手动调参

Aotomatic Hyperparameter Tuning：自动调参

1. ##定义交叉方式，先指定模型后指定参数，方便测试多个模型，网格交叉验证
2. **class** grid():
3. **def** \_\_init\_\_(self,model):
4. self.model = model
6. **def** grid\_get(self,X,y,param\_grid):
7. grid\_search = GridSearchCV(self.model,param\_grid,cv=5, scoring="neg\_mean\_squared\_error")
8. grid\_search.fit(X,y)
9. **print**(grid\_search.best\_params\_, np.sqrt(-grid\_search.best\_score\_))
10. grid\_search.cv\_results\_['mean\_test\_score'] = np.sqrt(-grid\_search.cv\_results\_['mean\_test\_score'])
11. **print**(pd.DataFrame(grid\_search.cv\_results\_)[['params','mean\_test\_score','std\_test\_score']])
13. grid(Lasso()).grid\_get(X\_scaled,y\_log,{'alpha': [0.0004,0.0005,0.0007,0.0006,0.0009,0.0008],'max\_iter':[10000]})

## 模型集成

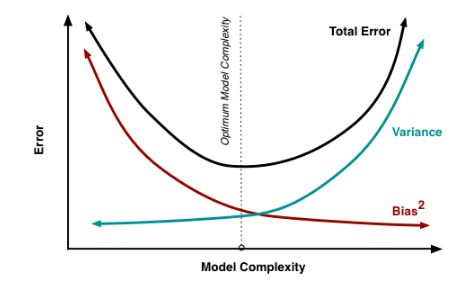
Boosting：主要关注降低偏差，能基于泛化性能相当弱的学习器构建出很强的集成。

Bagging：一系列弱分类器之间不存在强依赖关系，可以并行生成。其主要关注降低方差，因此它在不剪枝的决策树、神经网络等学习器上效用更为明显。

就机器学习算法来说，其泛化误差可分解为两部分：偏差（Bias）和方差（Variance）。

偏差指的是算法的期望预测与真实预测之间的偏差程度，反应了模型本身的拟合能力；

方差度量了同等大小的训练集的变动导致学习性能的变化，刻画了数据扰动所导致的影响。



对于Boosting来说，每一步我们都会在上一轮的基础上更加拟合原数据，所以可以保证偏差。因此，对于每个基分类器来说，问题就在于如何选择Variance更小的分类器，即更简单的分类器，所以选择了深度很浅的决策树。

对于Bagging来说，并行地训练很多不同的分类器目的就是降低方差，所以可以保证方差。因此，对于每个基分类器来说，目标就是如何降低偏差，即可采用深度更深甚至不剪枝的决策树。

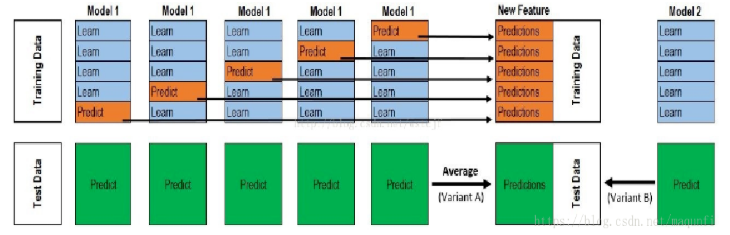
### 加权平均法

1. #定义交叉验证的策略，以及评估函数
2. **def** rmse\_cv(model,X,y):
3. rmse = np.sqrt(-cross\_val\_score(model, X, y, scoring="neg\_mean\_squared\_error", cv=5))
4. **return** rmse
6. ##定义加权平均值，就相当于自己写fit\_transform（）
7. **class** AverageWeight(BaseEstimator, RegressorMixin):
8. **def** \_\_init\_\_(self,mod,weight):
9. self.mod = mod##模型的个数
10. self.weight = weight##权重
12. **def** fit(self,X,y):
13. self.models\_ = [clone(x) **for** x **in** self.mod]
14. **for** model **in** self.models\_:
15. model.fit(X,y)
16. **return** self
18. **def** predict(self,X):
19. w = list()
20. pred = np.array([model.predict(X) **for** model **in** self.models\_])
21. # 针对于每一个数据点，单一的模型是乘以权重，然后加起来
22. **for** data **in** range(pred.shape[1]):
23. single = [pred[model,data]\*weight **for** model,weight **in** zip(range(pred.shape[0]),self.weight)]
24. w.append(np.sum(single))
25. **return** w
27. #指定每一个算法的参数
28. lasso = Lasso(alpha=0.0005,max\_iter=10000)
29. ridge = Ridge(alpha=60)
30. svr = SVR(gamma= 0.0004,kernel='rbf',C=13,epsilon=0.009)
31. ker = KernelRidge(alpha=0.2 ,kernel='polynomial',degree=3 , coef0=0.8)
32. ela = ElasticNet(alpha=0.005,l1\_ratio=0.08,max\_iter=10000)
33. bay = BayesianRidge()
35. ##6个权重
36. w1 = 0.02
37. w2 = 0.2
38. w3 = 0.25
39. w4 = 0.3
40. w5 = 0.03
41. w6 = 0.2
43. weight\_avg = AverageWeight(mod = [lasso,ridge,svr,ker,ela,bay],weight=[w1,w2,w3,w4,w5,w6])
44. rmse\_cv(weight\_avg,X\_scaled,y\_log),  rmse\_cv(weight\_avg,X\_scaled,y\_log).mean()##计算出交叉验证的均值

### Stacking

主要思想：训练模型来学习使用底层学习器的预测结果。

下图为5折Stacking中单个基模型在所有数据集上生成预测结果的过程，次学习器会基于基模型的预测结果进行再训练。



单个基模型生成预测预测结果的过程：

* + 1. 首先将所有数据集生成训练集和测试集（如训练集为10000行，测试集为2500行），对训练集进行5折交叉验证，即使用训练集中的8000行作为喂养集，剩余2000行作为验证集。
    2. 每次交叉验证时，使用蓝色的8000行数据训练出一个模型，该模型对验证集进行验证得到2000行数据，并对测试集测试得到2500行数据。这样经过5次交叉验证，即可得到中间橙色的5\*2000行验证集的预测结果，5\*2500行测试集的预测结果。
    3. 将验证集5\*2000行预测结果拼接成10000行的矩阵，标记为A1。而对5\*2500行的测试集结果进行加权平均，得到2500行的矩阵，标记为B1。
    4. 当使用3个基模型进行集成的话，相当于按步骤1~3得到了A1、A2、A3、B1、B2、B3。
    5. 将A1、A2、A3并列在一起成10000行3列的矩阵作为训练数据；B1、B2、B3合并在一起成2500行3列的矩阵作为测试数据，让次学习器进行再训练。次学习器会学习如何往这些基模型的预测结果上赋予权重w，来使得最后得到的预测更为准确。

1. **class** stacking(BaseEstimator, RegressorMixin, TransformerMixin):
2. **def** \_\_init\_\_(self,mod,meta\_model):
3. self.mod = mod
4. self.meta\_model = meta\_model#元模型
5. self.kf = KFold(n\_splits=5, random\_state=42, shuffle=True)##这就是堆叠的最大特征进行了几折的划分
7. **def** fit(self,X,y):
8. self.saved\_model = [list() **for** i **in** self.mod]
9. oof\_train = np.zeros((X.shape[0], len(self.mod)))
11. **for** i,model **in** enumerate(self.mod):#返回的是索引和模型本身
12. **for** train\_index, val\_index **in** self.kf.split(X,y):##返回的是数据本省
13. renew\_model = clone(model)##模型的复制
14. renew\_model.fit(X[train\_index], y[train\_index])#对数据进行训练
15. self.saved\_model[i].append(renew\_model)##把模型添加进去
16. oof\_train[val\_index,i] = renew\_model.predict(X[val\_index])##用来预测验证集
18. self.meta\_model.fit(oof\_train,y)#元模型
19. **return** self
21. **def** predict(self,X):
22. whole\_test = np.column\_stack([np.column\_stack(model.predict(X) **for** model **in** single\_model).mean(axis=1)
23. **for** single\_model **in** self.saved\_model]) ##得到的是整个测试集
24. **return** self.meta\_model.predict(whole\_test)#返回的是利用元模型来对整个测试集进行预测
26. **def** get\_oof(self,X,y,test\_X):
27. oof = np.zeros((X.shape[0],len(self.mod)))##初始化为0
28. test\_single = np.zeros((test\_X.shape[0],5))##初始化为0
29. test\_mean = np.zeros((test\_X.shape[0],len(self.mod)))
30. **for** i,model **in** enumerate(self.mod):##i是模型
31. **for** j, (train\_index,val\_index) **in** enumerate(self.kf.split(X,y)):##j是所有划分好的的数据
32. clone\_model = clone(model)##克隆模块，相当于把模型复制一下
33. clone\_model.fit(X[train\_index],y[train\_index])##把分割好的数据进行训练
34. oof[val\_index,i] = clone\_model.predict(X[val\_index])##对验证集进行预测
35. test\_single[:,j] = clone\_model.predict(test\_X)##对测试集进行预测
36. test\_mean[:,i] = test\_single.mean(axis=1)##测试集算好均值
37. **return** oof, test\_mean
39. ##经过预处理之后才能放到堆叠的模型里面去计算
40. a = Imputer().fit\_transform(X\_scaled)#相当于x
41. b = Imputer().fit\_transform(y\_log.values.reshape(-1,1)).ravel()#相当于y
43. #定义了第一层的和第二层的模型
44. stack\_model = stacking(mod=[lasso,ridge,svr,ker,ela,bay],meta\_model=ker)
45. X\_train\_stack, X\_test\_stack = stack\_model.get\_oof(a,b,test\_X\_scaled)#将数据进行变换
46. X\_train\_add = np.hstack((a,X\_train\_stack))
47. X\_test\_add = np.hstack((test\_X\_scaled,X\_test\_stack))
49. stack\_model = stacking(mod=[lasso,ridge,svr,ker,ela,bay],meta\_model=ker)
50. stack\_model.fit(X\_train\_add,b)#模型进行训练
51. pred = np.exp(stack\_model.predict(X\_test\_add))#进行预测

参考链接：

Stacking和Blending比较：<https://blog.csdn.net/maqunfi/article/details/82220115>

模型融合：<https://blog.csdn.net/qq_36330643/article/details/78576503>

### Blending

Blending的训练集不是通过K-FOLD的CV策略来获得预测值从而生成第二阶段模型的特征，而是建立一个HoldOut集，如10%的训练数据，第二阶段的次学习器就基于第一阶段的基模型对这10%训练数据的预测值进行拟合。

## 模型验证

### RMSE

均方根误差（Root Mean Square Error）

### MSE

均方误差（Mean Square Error）

### MAE

平均绝对误差（Mean Absolute Error）

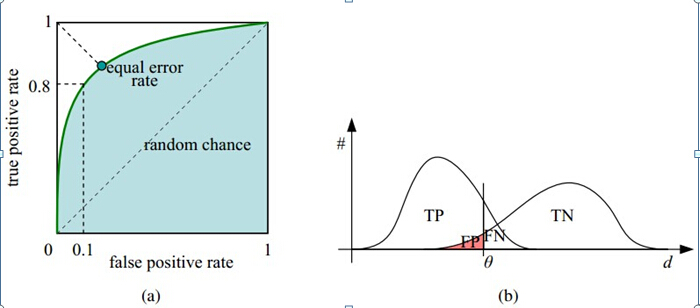
### ROC

接受者操作特征（Receiver Operating Characteristic）

横轴：负正类率（False Positive Rate，FPR），代表分类器预测的正类中实际负实例占所有负实例的比例。

纵轴：真正类率（True Positive Rate，TPR），代表分类预测的正类中实际正实例占所有正实例的比例。TPR就是Recall。

假设使用逻辑斯蒂回归分类器，其给出针对每个实例为正类的概率。则通过设定一个阈值如0.6，大概大于0.6的为正类，小于0.6的为负类，就可算出一组（FPR，TPR），对应ROC平面中的坐标点。



理想目标：，即上图中(0,1)点，故ROC曲线越靠拢(0.1)点，越偏离45度对角线越好。

### AUC

Area Under Curve，ROC曲线下面的面积就是AUC的值，介于0.1到1之间。可直观的评价分类器的好坏，值越大越好。

AUC的本质是：任取一对正负样本，正样本的预测值大于负样本的预测值的概率。

优点：1）当测试集中正负样本的分布变化的时候，ROC曲线能够保持稳定，即对问题的类别分布差异不敏感。2）如逻辑斯蒂回归模型中，抵消阈值对模型的影响。

### Recall

### Precision

### F1 score

### PRC

其纵轴为Precision，横轴为Recall的曲线。

优点：PRC曲线在正负样本比例悬殊较大时更难反应分类的真实性能。

# 算法类

## 线性模型

### 线性判别分析LDA

## 决策树

### ID3

核心思想：以信息增益度量属性选择，选择分裂后信息增益最大的属性进行分裂。

### C4.5

### Cart

## 贝叶斯分类器

贝叶斯公式为：

由于对所有类都是一样的，简化后的分类器的判别函数为：

如果假设样本特征向量的各个分量之间相互独立，则称为朴素贝叶斯分类器。如果假设特征向量服从多维正态分布，则称为正态贝叶斯分布，其预测函数为：

## K近邻法

## 相似度模型

常用方法：根据实体之间的距离确定相似度。

应用场景：商铺选址、广告宣传、个性化推荐。

为了避免量级大的特征对最终结果的影响太大，应该把item的各特征值标准化。

距离度量：

1. 欧氏距离
2. 明氏距离
3. 曼哈顿距离
4. 切比雪夫距离

## 降维算法

### 主成分分析PCA

KPCA

### 流形学习

LLE

拉普拉斯特征映射

等距映射

局部保持投影

## 聚类算法

聚类（Clustering）：就是将数据对象分组成为多个类或者簇。

目标：在同一个簇中的对象之间具有较高的相似度，而不同簇中的对象差别较大。

不同的聚类问题：

1. 聚类结果是排他的还是可重叠的。

问题的核心是，对于一个元素，它是否可以属于聚类结果中的多个簇中。如果是，则是一个可重叠的聚类问题；如果否，是一个排他的聚类问题。

1. 基于层次还是基于划分的。

划分的：拿到一组对象，按照一定的原则将它们分成不同的组。

层次的：将这些对象分等级，“自顶向下”即一步步的细化分组，“自底向上”即一步步的归并分组。

1. 簇数目固定的，还是无限制的聚类。

无限制的：指根据数据本身的特征，由聚类算法选择合适的簇的数目。

1. 基于距离还是基于概率分布的模型。

基于概率分布的模型：就是在一组对象中，找到能符合特定分布模型的点的集合，它们不一定是距离最近的或者最相似的，而是能完美的呈现出概率分布模型所描述的模型。

在基于概率分布模型的聚类方法里，核心是模型的定义，不同的模型可能导致完全不同的聚类结果。

### K均值聚类算法

使用场景：

1. 在大量文本中发现隐含的话题；
2. 发现图像中包含的颜色种类，压缩图像；
3. 从销售数据中发现不同特征顾客的分类。

算法步骤：

1. 假设希望将训练数据集分为K类；
2. 在中，随机选择K个作为初始分类的圆心（centroids）：；
3. 遍历，计算出和每个距离最近的圆心，记录当前属于第类；
4. 遍历K种分类，分别计算上一步中，划归其中的所有点的中心点，将该点设置为分类中心的圆心；
5. 迭代上面两步，直到圆心位置收敛。

其中，表示所属的那个簇的聚类中心。在调试K均值聚类算法的时候，可看其是否收敛来判断算法是否正常工作。

由于初始的圆心是随机选择的，可能陷入局部最优而导致最终的圆心无法收敛到合适的位置。可使用随机初始化来解决该问题。

聚类数量K的选择

1）使用肘方法（Elbow Method），逐渐增加K，并分别计算Cost Function J，寻找J较小的K。

2）为什么要使用K-means？充分理解聚类的需求以及聚类后能向下游贡献什么东西？

### Canopy聚类算法

### 模糊K均值聚类算法

### 狄利克雷聚类算法

参开链接

1. 深入推进引擎相关算法 - 聚类：

<https://blog.csdn.net/kevin7658/article/details/23826899>

1. K均值聚类算法：<https://zhuanlan.zhihu.com/p/32300636>

### 层次聚类

### DBSCAN

### OPTICS

### Mean Shift

### 谱聚类

### EM算法

## 异常检测

应用：检测异常的信用卡转账以防欺诈，检测制造缺陷。

## 关联规则学习

### Apriori算法

### Eclat算法

## 支持向量机

## Boosting算法

从弱学习算法出发，反复学习，得到一系列弱分类器，然后组合这些弱分类器，构成一个强分类器。其中，一系列弱分类器之间存在强依赖关系，需要串行生成。

核心问题：

1. 在每一轮如何改变训练数据的权值或概率分布？
2. 如何将弱分类器组合成一个强分类器？

### Adaboost

Adaptive boosting的缩写。

核心思想：针对同一个训练集训练不同的分类器（弱分类器），然后把这些弱分类器集合起来，构成一个更强的最终分类器（强分类器）。

特点：

1. 提高那些被前一轮弱分类器错误分类样本的权值，而降低那些被正确分类样本的权值。
2. 采取加权多数表决的方法。具体的，加大分类误差率小的弱分类器的权值，使其在表决中起较大的作用；减小分类误差率大的弱分类器的权值，使其在表决中起较小的作用。

输入：训练数据集，类标记。

算法步骤：

1. 初始化训练数据的权值分布，假设每个训练样本在基本分类器的学习中作用相同：

，

1. 对
   1. 使用具有权值分布的训练数据集学习，得到基本分类器；
   2. 计算在训练数据集上的分类误差率，其为被误分类样本的权值之和：
   3. 计算的系数，表示在最终分类器中的重要性，并且随着的减小而增大：
   4. 更新训练数据集的权值分布。其中，被基本分类器误分类样本的权值得以扩大，而被正确分类的样本权值却得以缩小。
2. 构建基本分类器的加权线性组合，得到最终分类器为：

缺点：

1. 训练时间过长；
2. 执行效果依赖于弱分类器的选择。

参考链接：

1. Adaboost算法的原理与推导：<https://blog.csdn.net/v_JULY_v/article/details/40718799>
2. Adaboost原理详解：<https://www.cnblogs.com/ScorpioLu/p/8295990.html>
3. Adaboost基本原理：https://blog.csdn.net/qq\_24519677/article/details/81910112

### 提升树

不允许更改原来模型的参数，如何进一步来提高模型的拟合能力呢？

建立一个新的模型来拟合未完全拟合真实样本的残差，即。所以，对于每个样本来说，拟合的样本集就变成了：

其中，称为残差。

当Adaboost算法中的基本分类器是Cart回归树时，就是提升树。损失函数变为平方误差损失函数。

其负梯度为：

算法步骤：

1. 初始化
2. 对
   1. 计算残差：计算损失函数的负梯度在当前模型的值作为残差的估计，；对于平方损失函数来说，即所谓的残差；
   2. 拟合残差学习一个回归树，得到
   3. 更新
3. 得到回归问题提升树：

### 梯度提升树GBDT

与提升树算法类似，只不过GBDT的损失函数不是平方损失函数，而是一般的损失函数，所以用损失函数的负梯度来代替残差。

提升树和GBDT，每一轮的树都是二层的（根节点+叶子节点）。

GBDT算法核心：在决策树的基础上引入GB和shrinkage两种思想，从而提升普通决策树的泛化能力。其结果是多颗决策树预测值的累加，而残差则是每颗决策树的学习目标。

如何选择特征

如何构建特征

如何用于分类

通过什么方式减少误差

如何加速训练

参数有哪些，如何调参

优点：不需要做特征的归一化；自动进行特征选择；模型可解释性较好；可以适应多种损失函数，如SquareLoss、LogLoss等。

缺点：boosting是个串行的过程，不能并行化；计算复杂度高；不太适合高维稀疏特征，通常采用稠密的数值特征。

#### sklearn.ensemble.GradientBoostingClassifier()

#### sklearn.ensemble.GradientBoostingRegressor()

n\_estimators：最大的弱学习器的个数。

learning\_rate：每个弱学习器的权重衰减系数。加上了正则化项，强学习器的迭代公式变为。的取值范围为。对于同样的训练集拟合效果，较小的意味着需要更多的弱学习器的迭代次数。通常，n\_estimators和learning\_rate要一起调参。

subsample：子采样，取值为。其为不放回抽样。如果取值为1，则全部样本都使用。如果取值小于1，则只有一部分样本会去做GBDT的决策树拟合。选择小于1的比例可以减小方差，即防止过拟合。推荐取之间的数。

loss：GBDT算法中的损失函数。

对于分类模型，默认为对数似然损失函数deviance，以及指数损失函数exponential。一般来说，推荐使用默认的deviance，其对二元分类和多元分类都有比较好的优化。

对于回归模型，默认为均方差ls，以及绝对损失lad、Huber损失huber、分位数损失quantile。如果数据的噪音点不多，用默认的ls比较好。如果噪音点较多，则推荐用抗噪音的huber损失函数。

alpha：只适用于回归模型，当损失函数使用huber或quantile时，需指定分位数的值。默认为0.9。若噪音点较多，可适当降低该值。

参考链接：

1. Boosting-梯度提升树：<https://zhuanlan.zhihu.com/p/33052697>
2. 机器学习-一文理解GBDT的原理：<https://zhuanlan.zhihu.com/p/29765582>
3. Boosting算法原理，GBDT&XGBoost：<https://www.cnblogs.com/csyuan/p/6537255.html>
4. scikit-learn梯度提升树调参小结：<https://www.cnblogs.com/pinard/p/6143927.html>
5. 基于树的模型：<https://blog.csdn.net/hy592070616/article/details/81628956>
6. gbdt基本原理：https://blog.csdn.net/qq\_24519677/article/details/82020863

### xgboost

使用pre-sorted算法，能够更精确的找到数据分割点：

* 1. 首先，对所有特征按数值进行预排序；
  2. 其次，在每次的样本分裂时，用O(#data)的代价找到每个特征的最优分割点；
  3. 最后，根据找到的特征以及分割点，将数据分裂成左右两个子节点。

决策树生长策略：采用的是按层生长策略（level-wise tree growth），能够同时分裂同一层的叶子，从而进行多线程优化，不容易过拟合。但不加区分的对待同一层的叶子，带来了很多不必要的开销。因为实际上很多叶子的分裂增益较低，没必要进行搜索和分裂。



优点：pre-sorted算法能够准确找到分割点。

缺点：在时间和空间上有很大的开销。

1. 由于需要对特征进行预排序，并且需要保存排序后的索引值（为了后续快速的计算分割点），因此内存需要训练数据的两倍。
2. 在遍历每个分割点的时候，都需要进行分裂增益的计算，消耗的代价大。

决策树节点分裂时是如何选择特征的？

Gini Index和Information Gain的公式，并举例说明？

回归树和分类树的区别？

与Random Forest作比较，并以此介绍什么是模型的Bias和Variance？

XGBoost的参数调优有哪些经验？

XGBoost的正则化是如何实现的？

XGBoost的并行化部分是如何实现的？

参考链接：

1）XGBoost参数调优：https://www.cnblogs.com/mfryf/p/6293814.html

### lightgbm

使用histogram算法，将连续的浮点特征离散成k个离散值，并构造宽度为k的histogram。然后遍历训练数据，统计每个离散值在直方图中的累计统计量。在进行特征选择时，只需要根据直方图的离散值，遍历寻找最优的分割点。

决策树生长策略：采用leaf-wise生长策略。每次从当前所有叶子中找到分裂增益最大的一个叶子，然后分裂，如此循环。

同level-wise相比，在分裂次数相同的情况下，leaf-wise可以降低更多的误差，得到更好的精度。leaf-wise的缺点是可能会长出比较深的决策树，产生过拟合。因此lightgbm在leaf-wise之上增加了一个最大深度的限制，在保证高效率的同时防止过拟合。



优缺点：

1. 由于特征被离散化后，找到的并不是很精确的分割点，所以会对结果产生影响。但在实际数据集上表明，离散化的分割点对最终的精度影响不大，甚至会好一些。原因在于决策树本身就是一个弱学习器，采用histogram算法会起到正则化的效果，有效地防止模型的过拟合。
2. 时间上的开销由xgboost的O(#data \* #features)降到O(k \* features)，在时间上有很大的提升。

lightgbm优化了对类别特征的支持，可以直接输入类别特征，不需要额外的0/1展开。

参考链接：

1. LightGBM算法：<https://blog.csdn.net/luanpeng825485697/article/details/80236759>
2. LightGBM调参方法：https://www.imooc.com/article/43784

## Bagging

Bootstrap aggregating的缩写。其各个弱学习器之间没有依赖关系，可以并行拟合。

每个分类器都随机从原样本中做有放回的采样，然后分别在这些采样后的样本上训练分类器，然后再把这些分类器组合起来。

一般随机采样集和训练样本数m是一样的，这样得到的采样集和训练集样本的个数相同，但是样本内容不同。

关于Bagging的集合策略，对于分类问题，通常使用简单投票法，得到最多票数的类别或者类别之一为最终的模型输出。对于回归问题，通常使用简单平均法，对T个弱学习器得到的回归结果进行算术平均得到最终的模型输出。

#### sklearn.ensemble.BaggingClassifier()

#### sklearn.ensemble.BaggingRegressor()

base\_estimator：默认None，表示决策时。用于指定基本分类器。

n\_estimators：默认10，集成的弱分类器的个数。

max\_samples：默认1.0。训练样本集中用于训练弱分类器的样本比例。

max\_features：默认1.0。训练样本集中用于训练弱分类器的特征比例。

bootstrap：默认True，决定训练样本集为有放回的随机抽样。

bootstrap\_features：默认False，决定训练样本集特征为无放回的随机抽取方式。

oob\_score：是否采用袋外样本来评估模型的好坏，默认False。

### 随机森林

使用Cart决策树作为弱学习器。

RF通过随机选择节点上的一部分样本特征，这个数字小于n，假设为，然后在这些随机选择的个样本特征中，选择一个最优的特征来做决策树的左右子树划分。这样进一步增强了模型的泛化能力。

越小，则模型越健壮，当然此时对于训练集的拟合程度会变差。即越小，模型的方差会减小，但是Bias会增大。在实际案例中，一般会通过交叉验证调参来获取一个合适的的值。

应用：分类回归、特征转换、异常点检测等。

优点：

1. 训练可以高度并行化，对于大数据时代的大样本训练速度更有优势。
2. 由于可以随机选择决策树节点划分特征，这样在样本特征维度很高的时候，仍然能高效的训练模型。
3. 由于采用了随机采样，训练出的模型方差小、泛化能力强。
4. 对部分特征确实不敏感。

缺点：

1. 在某些噪音比较大的样本集上，RF模型容易陷入过拟合。
2. 取值划分比较多的特征容易对RF的决策产生更大的影响，从而影响拟合模型的效果。

#### sklearn.ensemble.RandomForestClassifier()

#### sklearn.ensemble.RandomForestRegressor()

n\_estimators：最大的弱学习器的个数。一般来说，n\_estimators太小，容易欠拟合；太大，计算量会太大。并且n\_estimators达到一定的数量后，再增大获得的模型提升效果会很小。默认是100。

oob\_score：即是否采用袋外样本来评估模型的好坏。默认为False。推荐设置为True，因为袋外分数反应了一个模型拟合后的泛化能力。

criterion：即Cart树做划分时对特征的评估标准。分类RF对应的Cart分类树默认是基尼系数gini，另一个是信息增益entropy；回归RF对应的Cart回归树默认是均方差mse，另一个是绝对值差mae。一般选择默认的标准即可。

max\_features：划分时考虑的最大特征数。一般用默认的auto即可，即个特征。

max\_depth：决策树最大深度。默认可以不输入，即决策树在建立子树的时候不会限制子树的深度。一般来说，数据少或特征少的时候可不管这个值。如果模型样本多、特征也多，推荐限制这个最大深度，常用的取10~100之间。

min\_samples\_split：内部节点再划分所需最小样本数。默认为2，如果样本量不大，不需要管这个值；如果样本量数量级非常大，则推荐增大这个值。

min\_samples\_leaf：叶子节点最少样本数。如果某叶子节点数目小于该值，则会和兄弟节点一起被剪枝。默认为1，如果样本量不大，不需要管这个值；如果样本量数量级非常大，则推荐增大这个值。

min\_weight\_fraction\_leaf：叶子节点最小的样本权重和。限制了叶子节点所有样本权重和的最小值。如果小于这个值，会和兄弟节点一起被剪枝。默认是0，即不考虑权重问题。一般来说，如果有较多样本缺失值，或分类树样本的分布类别偏差较大，才会使用。

max\_lead\_nodes：最大叶子节点数，可防止过拟合。默认为None，即不限制最大的叶子节点数。如果特征不多，可以不考虑这个值；如果特征较多的话，可加以限制。具体的值可通过交叉验证得到。

参考链接

1. 刘建平Pinard：<https://www.cnblogs.com/pinard/category/894692.html>
2. Scikit-learn随机森林调参小结：https://www.cnblogs.com/pinard/p/6160412.html

### 非常随机化树Extra trees

原理与RF的区别为：

1. 对于每个决策树的训练，RF采用的是随机有放回的采样来选择采样集作为每个决策树的训练集，而extra trees一般不采用随机采样，即每个决策树采用原始训练集；
2. 在选定了划分特征后，RF会选择一个最优的特征值划分点，但extra trees比较激进，会随机选择一个特征值来划分决策树。

由于随机选择一个特征值的划分点，这样会导致生成的决策树的规模一般会大于RF所生成的决策时。即模型的方差相对于RF进一步减少，但是Bias相对于RF进一步增大。

#### sklearn.ensemble.ExtraTreesRegressor()

#### sklearn.ensemble.ExtraTreesClassifier()

## Stacking

参考链接：

1. 集成学习总结&Stacking方法详解：

https://blog.csdn.net/willduan1/article/details/73618677

## CTR预估模型

softmax函数的定义是什么？

神经网络为什么会产生梯度消失现象？

常见的激活函数有哪些？都有什么特点？

挑一种激活函数推导梯度下降的过程。

Attention机制是什么？

聚类算法

模型评估

GAN

Online learning

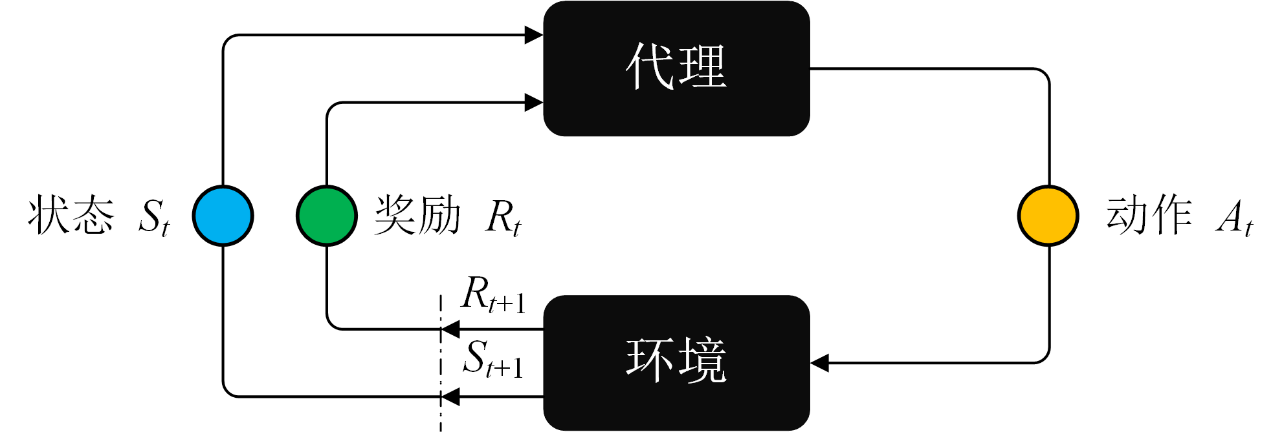
Embedding方法

Reinforcement learning

# 强化学习

基本内涵：将问题用代理（Agent）和环境进行建模。其中，代理能够对环境执行一些特定的动作，从而到达这个状态，然后就可以根据该状态为代理赋予特定的奖励。

代理的目标是最大化未来的奖励总和，它通过将未来可获得的最大奖励添加到当前的奖励来实现这一点，从而通过潜在的奖励影响当前行为。这个潜在奖励是从当前状态开始的所有未来动作的奖励期望值的加权和。



用马儿可夫决策过程（Markov Decision Process，MDP）来描述强化学习，一个基本的MDP问题可以用一个五元组来表示，各个符号的含义为：

S：表示有限状态集；

A：表示有限动作集；

P：表示状态转移概率矩阵（如高三努力学习，从年级前100名到年级前20名的概率）；

R：表示奖励（Reward）函数。表示在状态下执行动作后，从状态s转移到，所得到的奖励。

：表示折扣因子，该因子主要用于平衡当前的奖励与未来的奖励，可以理解为权重。

这样，MDP的目标就是找到一种策略，使得代理在状态s下能够做出对应的动作a，使得回报（Return）能够达到最大：

定义价值函数（Value Function），其定义为回报的期望，用于表示当前状态的未来潜在价值（如小明当前的状态是考上了一所好大学，那么他未来的潜在价值应该比较高）：

Reward（奖励）：指代理采取一个动作之后的奖励，可理解为短期的、即时的奖励。

Return（回报）：是各个短期奖励加权之和，可视为长远的奖励。

Value（价值）：是上述长远奖励的期望。

当前强化学习的两种思路：

1. 基于价值（Value）函数的强化学习：需首先对价值进行估计，然后间接地去求解如何选择动作。
2. 基于策略（Policy）梯度的强化学习：通过反馈调整策略，具体来说就是在得到正向奖励时，增加相应的动作的概率；得到负向的奖励时，降低相应动作的概率。

## 贝尔曼（Bellman）方程

从Bellman方程可得当前状态的价值（Value）与两个因素有关：

1. 当前的奖励（Reward）；
2. 下一个迭代时刻的价值（Value）。

## 策略迭代

## 价值迭代

## 蒙特卡洛算法

## 时序差分算法

## Q-Learning

参考链接：

1）深度强化学习研究笔记（1）https://blog.csdn.net/discoverer100/article/details/88266631

# 深度学习

## 反向传播（back propagation）

在计算百万级别的参数时，使用BP能够使计算梯度时更加有效率。

梯度时一个方向向量，它表示其函数在该点变化率最大的方向。

为了求函数的极小值问题，其变化量可表示成下述公式：

其中，是该点上的微小变化，我们可随意指定这些微小变化，只需要保证即可。但是为了更快的下降，所以选在梯度方向上做变化。即使得

从而保证一直递减，而对于而言，每次按下式进行更新即可：

参考链接

1）反向传播算法：https://www.cnblogs.com/wlzy/p/7751297.html

CNN一系列算法：LeNet5、AlexNet、VGG、GoogleNet、ResNet

激活函数：为了解决日常生活中不能用线性方程所概括的问题。

## 深度Q网络（DQN）

## 策略梯度算法

## Online learning

# NLP

应用范围智能问答、机器翻译、搜索引擎等。

RNN

LSTM

Seq2Seq

# CV

读取返回值的最后一个字节来保证只提取ASCII码：

|  |
| --- |
| 1. keycode = cv2.waitKey(1) 2. **if** keycode != -1: 3. keycode &= 0xFF |

计算机视觉中三种常用的色彩空间：灰度、BGR、HSV（Hue-Saturation-Value）。

# 工具

将machine learning知识应用于实际业务的工具。

Spark

# Tensorflow

|  |
| --- |
| 1. **import** tensorflow as tf 2. tf.\_\_version\_\_ 3. tf.\_\_path\_\_  # 查询安装路径 |

|  |
| --- |
| 1. a = tf.placeholder(tf.float32) 2. b = tf.placeholder(tf.float32) 3. adder\_node = a + b 5. # Session打开模式1 6. sess = tf.Session() 7. # 使用feed\_dict参数传递具体的值到run方法的占位符 8. **print**(sess.run(adder\_node, {a:3, b:4.5})) 9. **print**(sess.run(adder\_node, {a:[1,3], b:[2,4]})) 10. sess.close() 12. # Session打开模式2 13. with tf.Session() as sess: 14. **print**(sess.run(adder\_node, {a:2, b:3.5})) |

当调用tf.constant时常量被初始化，它们的值是不可以改变的。而当你调用tf.Variable时没有被初始化，要想初始化这些变量，必须调用特定的操作：

|  |
| --- |
| 1. init = tf.global\_variables\_initializer() 2. sess.run(init) |

改变Tensor的值：

|  |
| --- |
| 1. w = tf.Variable([.3], dtype=tf.float32) 2. w = tf.assign(w, [1.]) |

例子：

|  |
| --- |
| 1. # 创建数据 2. x\_data = np.random.rand(100).astype(np.float32) 3. y\_data = x\_data \* 0.1 + 0.3 5. # 搭建模型 6. Weights = tf.Variable(tf.random\_uniform([1], -1.0, 1.0)) 7. biases = tf.Variable(tf.zeros([1])) 8. y = Weights \* x\_data + biases 10. # 计算误差 11. loss = tf.reduce\_mean(tf.square(y - y\_data)) 13. # 传播误差 14. optimizer = tf.train.GradientDescentOptimizer(0.5) 15. train = optimizer.minimize(loss) 17. # 训练 18. init = tf.global\_variables\_initializer() 19. sess = tf.Session() 20. sess.run(init) 22. **for** step **in** range(201): 23. sess.run(train) # run每一次training的数据，逐步提升预测准确性 24. **if** step % 20 == 0: 25. **print**(step, sess.run(Weights), sess.run(biases)) 27. sess.close() |

构造神经网络

|  |
| --- |
| 1. **def** add\_layer(inputs, in\_size, out\_size, activation\_function=None): 2. # 构造添加神经层的函数 3. Weights = tf.Variable(tf.random\_normal([in\_size, out\_size])) 4. biases = tf.Variable(tf.zeros([1, out\_size]) + 0.1) 5. Wx\_plus\_b = tf.matmul(inputs, Weights) + biases 6. **if** activation\_function **is** None: 7. **return** Wx\_plus\_b 8. **else**: 9. **return** activation\_function(Wx\_plus\_b) 10. x\_data = np.linspace(-1, 1, 300, dtype=np.float32)[:, np.newaxis] 11. noise = np.random.normal(0, 0.05, x\_data.shape).astype(np.float32) 12. y\_data = np.square(x\_data) - 0.5 + noise 14. # 利用占位符定义神经网络的输入 15. xs = tf.placeholder(tf.float32, [None, 1]) # None代表无论输入有多少都可以 16. ys = tf.placeholder(tf.float32, [None, 1]) 18. # 搭建网络 19. l1 = add\_layer(xs, 1, 10, activation\_function=tf.nn.relu) 20. prediction = add\_layer(l1, 10, 1, activation\_function=None) 21. # 真实值与预测值差的平方和再取平均 22. loss = tf.reduce\_mean(tf.reduce\_sum(tf.square(ys - prediction), reduction\_indices=[1])) 23. train\_step = tf.train.GradientDescentOptimizer(0.1).minimize(loss) 25. init = tf.global\_variables\_initializer() 26. sess = tf.Session() 27. sess.run(init) 28. # 训练 29. **for** i **in** range(1001): 30. sess.run(train\_step, feed\_dict={xs:x\_data, ys:y\_data}) 31. **if** i % 50 == 0: 32. **print**(i, sess.run(loss, feed\_dict={xs:x\_data, ys:y\_data})) |

显示预测数据：

|  |
| --- |
| 1. **import** matplotlib.pyplot as plt 3. fig = plt.figure() 4. ax = fig.add\_subplot(1,1,1) 5. ax.scatter(x\_data, y\_data) 7. **for** i **in** range(1001): 8. sess.run(train\_step, feed\_dict={xs:x\_data, ys:y\_data}) 9. **if** i % 50 == 0: 10. **try**: 11. ax.lines.remove(lines[0]) 12. **except** Exception as e: 13. **print**(e) 15. prediction\_value = sess.run(prediction, feed\_dict={xs:x\_data}) 16. lines = ax.plot(x\_data, prediction\_value, 'r-', lw=5) 17. plt.pause(0.1) |

## TensorBoard可视化

|  |
| --- |
| 1. with tf.name\_scope('graph') as scope: 2. matrix1 = tf.constant([[3., 3.]],name ='matrix1')  #1 row by 2 column 3. matrix2 = tf.constant([[2.],[2.]],name ='matrix2') # 2 row by 1 column 4. product = tf.matmul(matrix1, matrix2,name='product') 5. sess = tf.Session() 6. writer = tf.summary.FileWriter("logs/", sess.graph) #第一个指定生成文件目录 7. init = tf.global\_variables\_initializer() 8. sess.run(init) |

最后，在终端中，在logs/文件根目录使用如下命令：

|  |
| --- |
| 1. > tensorboard --logdir logs |

同时，将终端中输出的网址<http://localhost:6006>复制到浏览器中，即可看到定义的视图框架。

重新编译代码，需更新图时：要去logs目录下把之前生成的文件删除，再运行代码，并重新执行tensorboard命令，最后刷新tensorborad界面。

### 可视化训练过程

### 参考链接

1. [谢小小XH](https://me.csdn.net/xierhacker)博客：<https://blog.csdn.net/xierhacker/article/category/6511974>
2. 深度学习7：TensorBoard使用方法：

<https://blog.csdn.net/lin453701006/article/details/79391088>

3）莫烦PYTHON

<https://morvanzhou.github.io/tutorials/machine-learning/tensorflow/4-1-tensorboard1/>

4）mnist分类：https://blog.csdn.net/xrinosvip/article/details/87385979

Xgboost

Parameter server

# PyTorch

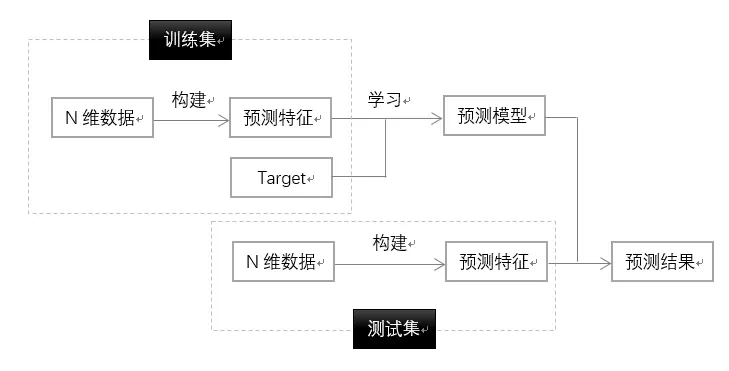
参考链接：

官方中文教程：<https://github.com/fendouai/PyTorchDocs>

PyTorch中文手册：https://github.com/zergtant/pytorch-handbook

# 截面数据

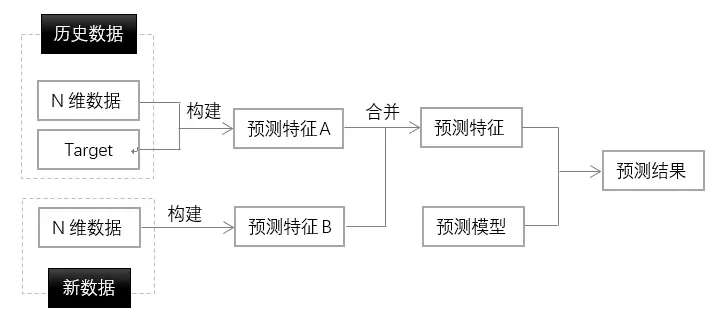
针对截面数据的数据处理思路：



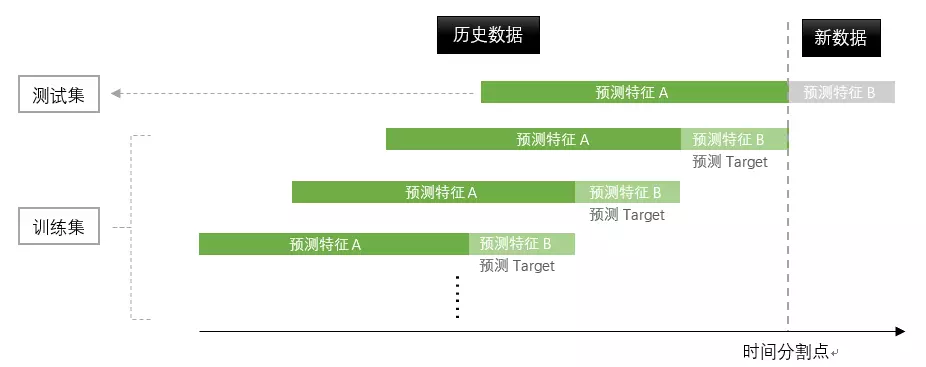
一般来说，在做完数据清洗之后，用N维数据来分别给训练集、测试集构建M维预测特征，然后用机器学习算法在训练集的预测特征和Target上训练模型，最后通过训练出的模型和测试集的预测特征来计算预测结果。为了给模型调优，需从训练集中随机分割一部分出来做为验证集。

# 时间序列数据

针对时间序列数据的处理，需要人为从时间序列中构造截面数据，其思路为：



用机器学习算法构造时间序列预测模型，关键的思路在于，通过时间滑窗，人为地构造未来Target，来给算法进行学习。



在历史数据上，通过截取不同的时间窗口的数据来构造一组或几组数据。比如，历史数据是2017年1月到12月每家店每天的销售数据，那么可以截取3组数据：2017年1月到10月的数据、2017年2月到11月的数据、2017年3月到12月的数据。

然后人为地给每组数据划分历史窗口和未来窗口。比如，2017年1月到10月的数据，把1月到9月作为历史窗口，10月作为未来窗口，以此类推。

分别给每组数据构建预测特征，包括历史特征（预测特征A）和未来特征（预测特征B），以及预测Target。

然后把得到的所有预测特征都合并起来作为训练集特征，把所有预测Target合并起来作为训练集Target（需分割0.2的数据作为验证集），以此来构建机器学习模型。

## 特征的构建

先将预测时间段的value设置为0，并与历史数据合并，然后再构建特征，做预测。

### 时间特征

因为时序数据是与时间相关联的数据，因此可提取出部分时间特征：

* 1. 每天中的第几小时hour\_of\_day
  2. 每个月中的第几天day\_of\_month
  3. 每年中的第几月month\_of\_year
  4. 每年中的第几季度season
  5. 工作日/非工作日weekday/weekend
  6. 是否节假日holiday

### 平移特征（lag特征）

将列值进行平移操作，构建新的特征。

### 窗口特征

将时间和列值结合起来，构建新的特征，如窗口宽度为5：

1. 窗口内均值
2. 窗口内最大值
3. 窗口内最小值

参考链接：

1. 机器学习与时间序列预测：<https://www.jianshu.com/p/e81ab6846214>

# 迁移学习

# 增量学习

# Python

@property：把getter方法变成属性，[其又会创建另一个@XXX.setter](mailto:其又会创建另一个@XXX.setter)装饰器，其负责把一个setter方法变成属性赋值。

@staticmethod：声明函数为静态方法。使类不需要实例化即可调用该函数。

## itertools

### product

|  |
| --- |
| 1. product('ABCD', 'xy') --> Ax Ay Bx By Cx Cy Dx Dy |

## six

## xgboost

## spark

## numpy

参考链接：

1）图解numpy：https://mp.weixin.qq.com/s/\_r1czXpTRL4zFfaBL6XiVA

## pandas

参考链接：

1）DataFrame.groupby()：https://blog.csdn.net/brucewong0516/article/details/78768443

## matplotlib

## seaborn

## skimage

图像处理库

## librosa

音频处理库

## nltk

自然语言处理库

## Python顺序执行多个py文件

|  |
| --- |
| 1. **import** os 3. os.system("python ./test01.py 1>>log.txt") # 指定输出到log.txt文件 4. os.system("python ./test02.py") # 执行test02.py 5. os.system("python ./test03.py") |

## 错误问题集锦

1. TypeError：slice indices must be integers or None or have an \_\_index\_\_ method

一般为数据格式的问题。

* 1. 若错误出现在带除法的那行，需将除法/更改成//。
  2. []中的索引变成了浮点数，不能作为数据的下标，需强制转换成int型数据。

# VIM

按了ctrl+s后完全动不了，按ctrl+q即可。

sudo vim /etc/vim/vimrc 更改vim配置文件。

# Powershell

参考链接

1）在线教程：https://www.pstips.net/powershell-online-tutorials

End：将光标移至当前命令的末尾。

Home：光标移至命令行最左端。

Esc：清空当前命令行。

F7：对话框显示命令行历史记录。F9：根据历史记录编号选择命令。

Ctrl+C：取消正在执行的命令。

Ctrl+Home：删除光标最左端的所有字符。

Ctrl+左/右方向键：在单词之间移动光标。

### 重定向

把命令的输出保存到文件中，>为覆盖，>>为追加。

|  |
| --- |
| 1. > ls >test.txt 2. > "Powershell Routing" >>test.txt 3. > Get-Content .\test.txt # 获取文件内容 |

### 管道

把上一条命令的输出作为下一条目录的输入。

|  |
| --- |
| 1. > ls | sort -Descending Name | Format-Table Name,Mode |

### 外部命令

* route print：查看路由器信息。
* &’cmd.exe’：执行命令，或启动程序。

### 别名

查询别名所指的真实cmdlet命令：

|  |
| --- |
| 1. > get-alias -name ls |

查询所有可用的别名：

|  |
| --- |
| 1. > ls alias: 2. > get-alias 3. > ls alias: | where {$\_.definition.startswith("Remove")} |

where：对数组元素进行遍历，$\_代表当前元素。

创建自己的别名：

|  |
| --- |
| 1. > set-alias -name edit -value notepad # 别名edit，退出时自动清除 2. > $alias:edit 3. > export-alias alias.psl # 将别名导出到文件 4. > import-alias -force alias.psl # 导入别名 |

通过函数设置默认参数，其中$args为参数的占位符：

|  |
| --- |
| 1. > function test-conn {test-connection -count 2 -computername $args} 2. > set-alias tc test-conn 3. > tc localhost |

# 题外内容

### 标签平滑技术

标签平滑方法：通过计算数据集中hard target的加权平均以及平均分布来计算交叉熵。

作用：能够有效防止模型过拟合，可应用于图片分类、机器翻译、语音识别。

### vector unpacking

### word2vec

### soft target

### hard target

# 题目集锦

1、int 64 1000万长度 无序可重复 找出第100万大数。

2、三个有序数组，写个归并排序。

3、已知一个0，1非等概率随机数生成器rand()，要求设计一个函数generate(n)，等概率的输出[1, n]的任意一个数。

## CV

## 机器学习

1、讲一下SVM的原理；

2、讲一下GBDT原理；

3、Xgboost原理什么什么,损失函数怎么构造？为什么选择Xgboost？

# 逻辑

举一反三的能力，解决问题的条理性，发散思维的能力。

# 业务

深入理解所在行业的商业模式，从业务中发现motivation并进而改进模型算法的能力。

# 经验

每个面试的结尾，面试官会问你有没有什么想问的，请注意这个问题也很关键。

比如：这个小组目前在做什么项目/实现项目主要用什么语言和算法/…

在面试中遇到不理解的，比如C++语法不懂，可以问这个C++具体在项目中实现什么功能。如果你提出好问题，能再次引起面试官对你的兴趣，那就能增加面试成功率。

# 资源

Kaggle：[www.kaggle.com](http://www.kaggle.com)

DC竞赛：https://www.dcjingsai.com/static\_page/cmpList.html

讯飞开放平台：<http://challenge.xfyun.cn/2019/gamedetail?type=detail/lifeforecast>

阿里云天池：<https://tianchi.aliyun.com/competition/gameList/activeList>

DataFountain: https://www.datafountain.cn/

Github：[www.github.com](http://www.github.com)

StackOverFlow：[www.stackoverflow.com](http://www.stackoverflow.com)

Csdn：[www.csdn.net](http://www.csdn.net)

Sklearn：scikit-learn.org/stable

Coursera：[www.coursera.org/browse](http://www.coursera.org/browse)

[www.deeplearning.ai](http://www.deeplearning.ai)

[www.fast.ai](http://www.fast.ai)

in.udacity.com

网易云课堂

数据可视化：<http://cs109.github.io/2015/pages/videos.html>

特征工程：<http://www.cnblogs.com/jasonfreak/p/5448385.html>

在线做实验：<https://www.shiyanlou.com/>

Python练习小程序：<https://github.com/Yixiaohan/show-me-the-code>

刘江Django教程：<http://www.liujiangblog.com/course/django/2>

力扣题库：<https://leetcode-cn.com/problemset/all/>

脚本之家：<https://www.jb51.net/>

书籍推荐：<https://github.com/loveunk/Deep-learning-books>

华校专：<http://www.huaxiaozhuan.com/>

Github：https://github.com/

PRML Python代码：<https://github.com/ctgk/PRML>

经验分享，如何入门深度学习：<http://bbs.xfyun.cn/forum.php?mod=viewthread&tid=41602>

Kaggle比赛经验分享：https://blog.csdn.net/leadai/article/details/79466106

天池智慧交通预测：https://github.com/PENGZhaoqing/TimeSeriesPrediction

时间序列分析之随机森林：<https://blog.csdn.net/weixin_41512727/article/details/85706521>

时间序列分析之序列特征：<https://blog.csdn.net/weixin_41512727/article/details/86412059>

O2O Coupon：<https://github.com/wepe/O2O-Coupon-Usage-Forecast>

1、数据科学的研究流程，问题类型，评价指标等

2、二分类问题

3、多分类问题

4、回归问题

5、xgboost、lightgbm模型，如何调参；如何做模型融合

6、时间序列，包括规则方法和模型、滑窗法等

7、ctr预估

1、评价算法的方法

2、算法思想：枚举、贪心、分治、二分

3、排序算法

4、搜索问题：搜索剪枝算法（递归、回溯）

5、动态规划算法：一维问题、二维问题

6、线型数据结构

7、树形数据结构

8、图算法：Floyd算法、Dijskra算法

9、字符串算法：MP算法、Hash算法、KMP算法、字典树、ShiftAnd算法

10、简单数论、进制转换、快速幂