机器学习笔记（初版）

数据集格式

特征值+目标值

DataFrame		列索引			
		Property1	Property2	...	Target
行索引	1	...	...	...	...
	2	...	...	...	...

注：有些数据集可以没有目标值 ### 原始数据处理

- 使用 pandas 对数据进行处理
  - 转化为“特征值+目标值”的方式
  - 对缺失值进行转化

特征工程

- 定义：特征工程是指将原始数据转化为**更好的代表预测模型的潜在问题的特征**的过程，提高对预测未知数据的**准确性。**
- 意义：直接影响预测结果

scikit-learn

介绍

- 机器学习工具
- 包含知名的的机器学习算法
- 容易上手，学术界广泛使用

安装

- 创建虚拟环境并安装 scikit-learn 库

- 使用 scikit-learn 需要 numpy、pandas 的支持

安装方式：

- python3 虚拟环境：
  - mkvirtualenv -p python3.x <envname>
  - pip3 install Scikit-learn
  - import sklearn
- conda 虚拟环境（推荐）：
  - conda create machine-learning
  - conda activate machine-learning
  - conda install numpy pandas scikit-learn

特征抽取

- 特征抽取是对于文本进行特征值化，转化为可供机器进行学习的表格格式
- 为了让计算机更好的理解数据

特征抽取 API： `sklearn.feature_extraction`

字典特征抽取

- 作用：对**字典数据**进行特征值化
- API： `sklearn.feature_extraction.Dictvetorizer`
- Demo：

```
origin_data = [
        {
            'City': 'Beijing',
            'temperature': 30
        },
        {
            'City': "Xi'an",
            'temperature': 34
        },
        {
            'City': "Shanghai",
            'temperature': 29
        }
    ]
    dct = DictVectorizer(sparse=True)
    data = dct.fit_transform(origin_data)
    print(data)
```

- 结果：

```
  (0, 0)        1.0
  (0, 3)        30.0
  (1, 2)        1.0
  (1, 3)        34.0
  (2, 1)        1.0
  (2, 3)        29.0
```

- 保存为此种形式是为了节省内存，因为稀疏矩阵，有效值比较少，只存储矩阵中的非零值更高效
- 如果使得 `sparse=False` ，会存储全部的矩阵形式。
- 打印输出，第一个列表即是列索引，由 `dct.get_feature_names()` 获取
  第二个打印就是矩阵全貌

```
['City=Beijing', 'City=Shanghai', "City=Xi'an", 'temperature']
[[ 1.  0.  0. 30.]
 [ 0.  0.  1. 34.]
 [ 0.  1.  0. 29.]]
[[1.         0.         0.         0.        ]
 [0.         1.         1.         0.83333333]
 [0.5        0.5        0.6        1.        ]]
```

- 矩阵中对于城市属性采用了 **one-hot** 编码，有这种属性则为 **1** ，没有则为 **0** 。
  没有采用对不同城市编号为 *1,2,3...* 的方法，是因为这种编码会引起不同的优先级问题，而 **one-hot** 编码可以规避此种问题

文本特征抽取

to be updated...

数据归一化

- Define：

$$X' = \frac{x - min}{max - min}$$
$$X'' = X' * (mx - mi) + mi$$

  - X'' 为最终值，(mx, mi) 为缩小后的区间范围，(max, min) 为列最大值和最小值
- API： `sklearn.preprocessing.MinMaxScalar`
- 用法： `MinMaxScalar(feature_range=(0,1))`
  - 默认范围为 $[0, 1]$
  - `MinMaxScalar.fit_transform(X)` ， X 为 numpy array 格式的数据
- demo:

```
origin_data = np.array(
        [
            [90, 2, 10, 40],
            [60, 4, 15, 45],
            [75, 3, 13, 46]
        ]
    )
    normal = MinMaxScaler(feature_range=(0,1))
    data = normal.fit_transform(origin_data)
```

结果：

```
[[1.         0.         0.         0.        ]
 [0.         1.         1.         0.83333333]
 [0.5        0.5        0.6        1.        ]]
```

- 归一化的目的：使得每个特征值在**同等标准**下进行操作，不会有某一特征值产生比较大的影响，使得其他特征值的影响被忽略
- 缺点：鲁棒性不好，易受异常点影响，异常点会影响数据最大最小值
- 适用传统精确小数据场景

标准化

- Define:

$$X' = \frac{x - mean}{\sigma}$$

- 特点：将原始数据变换到均值为 0，方差为 1 的范围内
- 优势：少量异常点对于平均值以及方差影响不大
- API： `StandardScaler()`
- 用法： `StandardScalar.fit_transform(X)`，X 是 numpy array 格式数据
  - `StandardScaler.mean_`：计算每列平均值
  - `StandardScaler.std_`：计算每列标准差
- Demo：

```
origin_data = [
        [1, -1, 3],
        [2, 4, 2],
        [4, 6, -1]
    ]
    standard = StandardScaler()
    data = standard.fit_transform(origin_data)
```

结果：

```
[[-1.06904497 -1.35873244  0.98058068]
 [-0.26726124  0.33968311  0.39223227]
 [ 1.33630621  1.01904933 -1.37281295]]
```

两种方法对比

Items	Property	Application
归一化	计算依赖于最大最小值，易受异常点影响	精确的小数据环境
标准化	计算依赖于平均值和标准差，少量异常点影响不大	样本足够多的现代嘈杂大数据环境

缺失值处理

- 策略：填补或者删除
- 通常使用的方法：在缺失值处按特征（列）填补平均值
- API： `Imputer(missing_values='NaN', strategy='mean', axis=0)`
    - 用法： `Imputer.fit_transfrom(X)`
    - 只能填补 `numpy.NaN` 数据，注意在生成特征值时要将缺失值转化为此种类型的数据