第一次上机作业

1. 练习用 Python 的 dropna 语句丢弃缺失数据。

One of the easiest ways to deal with missing data is to simply remove the corresponding features (columns) or samples (rows) from the dataset entirely; rows with missing values can be easily dropped via the `dropna` method:

```
>>> df.dropna(axis=0)
     A    B    C    D
0  1.0  2.0  3.0  4.0
```

Similarly, we can drop columns that have at least one `NaN` in any row by setting the `axis` argument to `1`:

```
>>> df.dropna(axis=1)
     A    B
0  1.0  2.0
1  5.0  6.0
2  10.0 11.0
```

The `dropna` method supports several additional parameters that can come in handy:

```
# only drop rows where all columns are NaN
# (returns the whole array here since we don't
# have a row with where all values are NaN
>>> df.dropna(how='all')
     A    B    C    D
0  1.0  2.0  3.0  4.0
1  5.0  6.0  NaN  8.0
2 10.0 11.0 12.0  NaN


# drop rows that have less than 4 real values
>>> df.dropna(thresh=4)
     A    B    C    D
0  1.0  2.0  3.0  4.0


# only drop rows where NaN appear in specific columns (here: 'C')
>>> df.dropna(subset=['C'])
     A    B    C   D
0  1.0  2.0  3.0 4.0
2 10.0 11.0 12.0 NaN
```

2. 练习填补缺失数据。

(1) 用 sklearn 中的 Imputer 填补缺失数据。

Often, the removal of samples or dropping of entire feature columns is simply not feasible, because we might lose too much valuable data. In this case, we can use different interpolation techniques to estimate the missing values from the other training samples in our dataset. One of the most common interpolation techniques is **mean imputation**, where we simply replace the missing value with the mean value of the entire feature column. A convenient way to achieve this is by using the `Imputer` class from scikit-learn, as shown in the following code:

```
>>> from sklearn.preprocessing import Imputer
>>> imr = Imputer(missing_values='NaN', strategy='mean', axis=0)
>>> imr = imr.fit(df.values)
>>> imputed_data = imr.transform(df.values)
>>> imputed_data
array([[  1.,   2.,   3.,   4.],
       [  5.,   6.,  7.5,   8.],
       [ 10.,  11.,  12.,   6.]])
```

Here, we replaced each `NaN` value with the corresponding mean, which is separately calculated for each feature column. If we changed the `axis=0` setting to `axis=1`, we'd calculate the row means. Other options for the `strategy` parameter are `median` or `most_frequent`, where the latter replaces the missing values with the most frequent values. This is useful for imputing categorical feature values, for example, a feature column that stores an encoding of color names, such as red, green, and blue, and we will encounter examples of such data later in this chapter.

(2) 不用现成脚本，编程实现上述“mean”、“median”和“most_frequent”的数据填补过程，熟悉 pandas dataframe 的使用。（在实验报告中附编码）

3. 练习特征转化。

(1) 练习转化序数特征。

数据构建:

```
>>> import pandas as pd
>>> df = pd.DataFrame([
...             ['green', 'M', 10.1, 'class1'],
...             ['red', 'L', 13.5, 'class2'],
...             ['blue', 'XL', 15.3, 'class1']])
>>> df.columns = ['color', 'size', 'price', 'classlabel']
>>> df
   color size  price classlabel
0  green    M   10.1     class1
1    red    L   13.5     class2
2   blue   XL   15.3     class1
```

编码过程:

```
>>> size_mapping = {
...                 'XL': 3,
...                 'L': 2,
...                 'M': 1}
>>> df['size'] = df['size'].map(size_mapping)
>>> df
   color  size  price classlabel
0  green     1   10.1     class1
1    red     2   13.5     class2
2   blue     3   15.3     class1
```

解码过程:

```
>>> inv_size_mapping = {v: k for k, v in size_mapping.items()}
>>> df['size'].map(inv_size_mapping)
0     M
1     L
2    XL
Name: size, dtype: object
```

(2) 练习独热编码

方法一:

```
>>> from sklearn.preprocessing import LabelEncoder

>>> X = df[['color', 'size', 'price']].values
>>> color_le = LabelEncoder()
>>> X[:, 0] = color_le.fit_transform(X[:, 0])
>>> X
array([[1, 1, 10.1],
       [2, 2, 13.5],
       [0, 3, 15.3]], dtype=object)
```

After executing the preceding code, the first column of the NumPy array `X` now holds the new `color` values, which are encoded as follows:

- `blue` = 0
- `green` = 1
- `red` = 2

```
>>> from sklearn.preprocessing import OneHotEncoder

>>> ohe = OneHotEncoder(categorical_features=[0])
>>> ohe.fit_transform(X).toarray()
array([[  0. ,   1. ,   0. ,   1. ,  10.1],
       [  0. ,   0. ,   1. ,   2. ,  13.5],
       [  1. ,   0. ,   0. ,   3. ,  15.3]])
```

方法二:

```
>>> pd.get_dummies(df[['price', 'color', 'size']])
   price  size  color_blue  color_green  color_red
0   10.1     1           0            1          0
1   13.5     2           0            0          1
2   15.3     3           1            0          0
```

(3) 不用现成脚本，编程实现独热编码，并与上述结果做比较。(在实验报告中附编码)

4. 对红酒数据集进行特征缩放。

(1) 用 pandas 加载红酒数据集（数据集和数据集说明见所附文件）

```
df = pd.read_csv('your/local/path/to/wine.data',
                 header=None)
```

```
>>> df_wine.columns = ['Class label', 'Alcohol',
...                    'Malic acid', 'Ash',
...                    'Alcalinity of ash', 'Magnesium',
...                    'Total phenols', 'Flavanoids',
...                    'Nonflavanoid phenols',
...                    'Proanthocyanins',
...                    'Color intensity', 'Hue',
...                    'OD280/OD315 of diluted wines',
...                    'Proline']
```

(2) 通过观察直方图，了解每个特征的数据分布。写明你打算对哪些特征进行归一化？对哪些特征进行标准化？

(3) 进行归一化。

```
>>> from sklearn.preprocessing import MinMaxScaler
>>> mms = MinMaxScaler()
>>> X_train_norm = mms.fit_transform(X_train)
>>> X_test_norm = mms.transform(X_test)
```

(4) 不借助 MinMaxScaler，自己编写程序实现归一化。（在实验报告中附代码段）

(5) 进行标准化。

```
>>> from sklearn.preprocessing import StandardScaler
>>> stdsc = StandardScaler()
>>> X_train_std = stdsc.fit_transform(X_train)
>>> X_test_std = stdsc.transform(X_test)
```

(6) 不借助 StandardScaler，自己编写程序实现标准化。（在实验报告中附代码段）

5. 练习数据集划分

(1) 练习搜索合适的 Python 包：自行在网上搜索可以进行留出法、交叉验证法分裂数据集的 python 包，以及使用教程。(提示：sklearn 里就有合适的)

(2) 用上一步中找到的教程，对红酒数据集进行划分（分别用留出法、交叉验证法划分，并在实验报告中附代码段)。

(3) 不依赖上述包，自己编写程序进行数据集划分。(在实验报告中附代码段)