Лабораторная работа №2

Препроцессинг данных

Препроцессинг данных включает в себя широкий круг методов для очистки, выбора и преобразования данных с целью улучшения качества последующего интеллектуального анализа данных.

Программные средства для препроцессинга данных имеются как в библиотеке Pandas, так и основной библиотеке машинного обучения scikit-learn (sklearn).

Качество наборов данных

Плохое качество данных оказывает негативное воздействие на процесс анализа данных. Наиболее часто встречающиеся проблемы включают шум, выбросы, отсутствующие значения и дублирующиеся данные.

Начнем с примера набора данных из репозитария UCI с информацией о пациентах с раком груди.

In [ ]:
```
import pandas as pd
data = pd.read_csv('https://archive.ics.uci.edu/ml/machine-learning
                   'breast-cancer-wisconsin/breast-cancer-wisconsin
data.columns = ['Sample code', 'Clump Thickness', 'Uniformity of Ce
                'Marginal Adhesion', 'Single Epithelial Cell Size',
                'Normal Nucleoli', 'Mitoses','Class']

data = data.drop(['Sample code'],axis=1)        # удаляем ненужный
print('Число записей = %d' % (data.shape[0]))
print('Число признаков = %d' % (data.shape[1]))
data.head()
```

Отсутствующие (пропущенные) значения

Достаточно часто в записи отсутствуют одно или несколько значений признаков. Иногда не хватает информации, а иногда некоторые значения не подходят для данных. Существуют различные подходы для работы с отсутствующими значениями.

В наборах данных репозитария UCI отсутствующие значения часто кодируются как символьная строка '?'. Первая задача состоит в конвертации отсутствующих значений в значение NaNs (NaN - Not a Number). Далее можно подсчитать количество отсутствующих значений в каждом столбце набора данных.

In [ ]:
```
import numpy as np # для использования np.NaN

data = data.replace('?',np.NaN)

print('Число записей = %d' % (data.shape[0]))
print('Число признаков = %d' % (data.shape[1]))

print('Число отсутствующих значений:')
for col in data.columns:
    print('\t%s: %d' % (col,data[col].isna().sum()))
```

Среди всех столбцов только столбец 'Bare Nuclei' содержит отсутствующие значения. Отсутствующие значения в столбце 'Bare Nuclei' могут быть заменены на медиану столбца при помощи метода `fillna()` (значения до и после замены показаны на подмножестве записей).

In [ ]:
```
data2 = data['Bare Nuclei']

print('До замены отсутствующих значений:')
print(data2[20:25])
data2 = data2.fillna(data2.median())

print('\nПосле замены отсутствующих значений:')
print(data2[20:25])
```

Вместо замены отсутствующих значений можно удалить записи (строки), содержащие отсутствующие значения. Для этого можно использовать метод `dropna()`:

In [ ]:
```
print('Число записей в исходных данных = %d' % (data.shape[0]))

data2 = data.dropna()
print('Число записей после удаления отсутствующих значений = %d' %
```

Выбросы

Выбросами (outliers) называются записи (строки) с характеристиками, которые существенно отличаются от характеристик остальных записей набора данных.

Ниже мы изобразим диаграммы размаха (boxplot) столбцов, чтобы найти столбцы таблицы, которые содержат выбросы. Так как столбец 'Bare Nuclei' идентифицирован Pandas как строковый (из-за отсутствующих значений, представленных строками '?'), нам придется конвертировать столбец в числовые значения для того, чтобы использовать диаграмму размаха. В противном случае столбец не будет отображаться на рисунке.

In [ ]:
```
%matplotlib inline

data2 = data.drop(['Class'],axis=1)
data2['Bare Nuclei'] = pd.to_numeric(data2['Bare Nuclei'])
data2.boxplot(figsize=(20,3))
```

Диаграммы размаха показывают, что только пять столбцов (Marginal Adhesion, Single Epithetial Cell Size, Bland Cromatin, Normal Nucleoli, Mitoses) содержат ненормально большие значения.

Чтобы убрать выбросы, можно посчитать стандартизованную оценку (Z-score) для каждого признака и убрать записи, содержащие атрибуты с ненормально высоким или низким Z-score (например, $Z > 3$ или $Z < -3$). Для нормального распределения вероятность отклонения случайной величины от своего математического ожидания более чем на три стандартных отклонения практически равна нулю (правило трех сигм).

Следующий код показывает результаты стандартизации стоблцов с данными. Отсутствующие значения (NaN) не затрагиваются процессом стандартизации.

In [ ]:
```
Z = (data2-data2.mean())/data2.std()
Z[20:25]
```

Следующий код показывает результаты удаления строк, для которых $Z > 3$ или $Z < -3$. Число 9 соответствует количеству столбцов в Z.

In [ ]:
```
print('Число записей до удаления выбросов = %d' % (Z.shape[0]))

Z2 = Z.loc[((Z >= -3).sum(axis=1)==9) & ((Z <= 3).sum(axis=1)==9),:
print('Число записей после удаления выбросов = %d' % (Z2.shape[0]))
```

Дублирующиеся данные

Некоторые наборы данных, особенно полученные слиянием данных из нескольких источников, могут содержать дублирующиеся записи.

In [ ]:
```
dups = data.duplicated()
print('Число дублирующихся записей = %d' % (dups.sum()))
data.loc[[11,28]]
```

Метод `duplicated()` возвращает булевский массив, который показывает является ли запись дубликатом какой-либо предыдущей записи в таблице. Результат означает, что в наборе данных пациентов с раком груди имеется 236 дублирующихся записей. Например, строка с индексом 11 имеет те же значения признаков, что и строка с индексом 28.

Хотя дублирующиеся записи могут соответствовать данным различных пациентов, допустим, что дублирующиеся записи соответствуют одному и тому же пациенту и удалим их:

In [ ]:
```
print('Число записей до удаления дубликатов = %d' % (data.shape[0]))
data2 = data.drop_duplicates()
print('Число записей после удаления дубликатов = %d' % (data2.shape
```

Трансформация (преобразование) данных

Стандартизация признака

Стандартизацией случайной величины X называют ее линейное преобразование, приводящее к случайной величине с математическим ожиданием 0 и стандартным отклонением 1:

$$\tilde{X} = \frac{X - \mathbb{E}[X]}{\sqrt{\mathbb{V}[X]}},$$

где $\mathbb{E}$ – операция вычисления математического ожидания, $\mathbb{V}$ – операция вычисления дисперсии.

Стандартизация набора данных является существенным условием для применения многих алгоритмов машинного обучения, а именно, алгоритмы дают приемлемый результат, только если отдельные признаки распределены примерно как стандартные нормальные величины (с нулевым матожиданием и единичной дисперсией).

Для стандартизации признаков набора данных может быть использована функция `scale()` из модуля `preprocessing`:

In [ ]:
```
from sklearn import preprocessing
import numpy as np
X = np.array([[ 1., -1.,  2.],
              [ 2.,  0.,  0.],
              [ 0.,  1., -1.]])
X_scaled = preprocessing.scale(X)
print(X_scaled)
```

Стандартизованный набор данных имеет признаки с нулевыми средними и единичной дисперсией:

In [ ]:
```
print(X_scaled.mean(axis=0))
print(X_scaled.std(axis=0))
```

Также модуль preprocessing содержит класс `StandardScaler`, который позволяет сохранить математическое ожидание и стандартное отклонение обучающей выборки и затем применять то же преобразование к другим выборкам.

Альтернативным вариантом стандартизации является масштабирование признака между заданным минимальным и максимальным значениями (нормализация). Этот эффект может быть достигнут при помощи функций `MinMaxScaler()` или `MaxAbsScaler()`:

In [ ]:
```
X = np.array([[ 1., -1.,  2.],
              [ 2.,  0.,  0.],
              [ 0.,  1., -1.]])
min_max_scaler = preprocessing.MinMaxScaler()
X_minmax = min_max_scaler.fit_transform(X)
X_minmax
```

Функция `MaxAbsScaler()` используется аналогично, но масштабирует данные в диапазон $[-1, 1]$.

Агрегирование данных

Агрегирование данных – это процесс преобразования данных с высокой степенью детализации к более обобщенному представлению, когда значения двух и более объектов комбинируются в один объект.

Целями агрегирования являются:

1. уменьшение размера обрабатываемых данных
2. изменение (укрупнение) масштабов анализа
3. улучшение стабильности данных

В примере ниже мы используем дневные временные ряды с данными об атмосферных осадках от метеорологической станции. Временные ряды с дневными данными об осадках будут сравниваться с месячными значениями.

Программный код загружает дневные данные об осадках и рисует график значений.

In [ ]:
```
daily = pd.read_csv('DTW_prec.csv', header='infer')
daily.index = pd.to_datetime(daily['DATE'])
daily = daily['PRCP']
ax = daily.plot(kind='line',figsize=(15,3))
ax.set_title('Дневные осадки (дисперсия = %.4f)' % (daily.var()));
```

Дневные данные об осадках выглядят весьма хаотическими и изменяются существенно от одного момента времени к другому. Дневные данные могут быть сгрупированы и агрегированы по месяцам, чтобы получить общие месячные значения осадков. Полученные в результате агрегирования данные выглядят более гладкими по сравнению с дневными данными.

In [ ]:
```
monthly = daily.groupby(pd.Grouper(freq='M')).sum()
ax = monthly.plot(kind='line',figsize=(15,3))
ax.set_title('Месячные осадки (дисперсия = %.4f)' % (monthly.var())
```

В примере ниже дневные данные сгруппированы и агрегированы по годам.

In [ ]:
```
annual = daily.groupby(pd.Grouper(freq='Y')).sum()
ax = annual.plot(kind='line',figsize=(15,3))
ax.set_title('Годовые осадки (дисперсия = %.4f)' % (annual.var()));
```

Семплирование данных

Семплирование (от англ. sample — выборка), или методы управления выборкой данных, – это подход, направленный на:

1. сокращение объема данных для анализа данных и масштабирования алгоритмов для приложений с большими данными
2. количественную оценку неопределенностей из-за различного распределения данных

Существуют различные методы выборки данных, такие как выборка без замены, когда каждый выбранный экземпляр удаляется из набора данных, и выборка с заменой, где каждый выбранный экземпляр не удаляется, что позволяет выбирать его более одного раза.

В примере ниже мы применим выборку с заменой и без замены с набору данных пациентов с раком груди.

Выведем первые пять записей набора:

In [ ]:
```
data.head()
```

Далее данные для выборки размера 3 (без замены) выбираются случайным образом из исходных данных.

In [ ]:
```
sample = data.sample(n=3)
sample
```

В следующем примере мы случайным образом выбираем 1% данных (без замены) и выводим выбранные записи. Параметр `random_state` задает начальное значение для генератора случайных чисел.

In [ ]:
```
sample = data.sample(frac=0.01, random_state=1)
sample
```

Наконец, выполним выборку с заменой размером, равным 1% всех данных. Можно увидеть повторяющиеся записи в выборке, если увеличить ее размеры.

In [ ]:
```
sample = data.sample(frac=0.01, replace=True, random_state=1)
sample
```

Дискретизация данных

Дискретизация – это этап препроцессинга, который часто используется при преобразовании непрерывного признака в категориальный.

Пример ниже иллюстрирует два простых, но часто применяемых метода дискретизации (равной ширины, равных частот) для признака 'Clump Thickness' набора данных пациентов с раком груди.

Вначале нарисуем гистограмму, которая показывает распределение значений признака. Метод `value_counts()` также может быть использован, чтобы подсчитать частоты каждого значения признака.

In [ ]:
```
data['Clump Thickness'].hist(bins=10)
data['Clump Thickness'].value_counts(sort=False)
```

При использовани метода равной ширины можно использовать метод `cut()`, чтобы дискретизировать признак в 4 бина, имеющих равную ширину. Метод `value_counts()` может быть использован для определения числа записей в каждом из бинов.

In [ ]:
```
bins = pd.cut(data['Clump Thickness'],4)
bins.value_counts(sort=False)
```

При использовани метода равных частот можно использовать метод `qcut()` для разделения значений признака на 4 бина, имеющих примерно равное число записей.

In [ ]:
```
bins = pd.qcut(data['Clump Thickness'],4)
bins.value_counts(sort=False)
```

Дискретизация также возможна при помощи средств библиотеки scikit-learn.

Кодирование категориальных признаков

В большинстве наборов данных присутствуют категориальные признаки, которые содержат значения в текстовом формате. Примерами являются цвета (“Red”, “Green”, “Yellow”, “Blue”), размеры (“Small”, “Medium”, “Large”, “Extra Large”), географические обозначения (страны, города и т.п.). Независимо от назначения категориальных признаков возникает вопрос, как использовать категориальные признаки при анализе данных. Многие алгоритмы машинного обучения поддерживают категориальные значения без необходимости каких-либо манипуляций с данными, однако есть и такие алгоритмы, которые требуют преобразования текстовых значений в числовые для дальнейшей обработки.

Набор данных

Рассмотрим набор данных Automobile из репозитария UCI, содержащий как категориальные, так и непрерывные признаки.

Импортируем данные, выполняя попутно обработку пропущенных значений:

In [ ]:
```
import pandas as pd
import numpy as np
# определяем метки столбцов
headers = ["symboling", "normalized_losses", "make", "fuel_type", "
    "num_doors", "body_style", "drive_wheels", "engine_location",
    "wheel_base", "length", "width", "height", "curb_weight",
    "engine_type", "num_cylinders", "engine_size", "fuel_system",
    "bore", "stroke", "compression_ratio", "horsepower", "peak_rpm"
    "city_mpg", "highway_mpg", "price"]
# считываем CSV файл и конвертируем значения "?" в NaN
df = pd.read_csv("https://archive.ics.uci.edu/ml/machine-learning-d
    header=None, names=headers, na_values="?" )
df.head()
```

Чтобы понять, с какими типами данным мы имеем дело, рассмотрим свойство

In [ ]:
```
df.dtypes
```

Так как нас интересуют только категориальные признаки, оставим в наборе столбцы с типом “object”. Pandas содержит удобный метод `select_dtypes()`, который можно использовать, чтобы оставить в наборе только столбцы с типом “object” (категориальные признаки):

In [ ]:
```
obj_df = df.select_dtypes(include=['object']).copy()
obj_df.head()
```

Построенный набор содержит несколько строк с пропущенными значениями, которые нужно заполнить:

In [ ]:
```
obj_df[obj_df.isnull().any(axis=1)]
```

В наборе наиболее часто встречается значение "four" (4 двери):

In [ ]:
```
obj_df["num_doors"].value_counts()
```

Для простоты заполним пропущенные значения этим значением:

In [ ]:
```
obj_df = obj_df.fillna({"num_doors": "four"})
obj_df[obj_df.isnull().any(axis=1)]
```

Теперь набор не содержит пропущенных значений и мы можем приступить к кодированию категориальных значений.

Замена значений признаков

В двух столбцах набора данных текстовые значения представляют собой числа, а именно, число цилиндров в двигателе и число дверей в автомобиле.

Признак "num_cylinders" принимает 7 значений, которые легко преобразуются в целые числа:

In [ ]:
```
obj_df["num_cylinders"].value_counts()
```

Метод `replace()` из Pandas имеет множество опций, в частности, опцию словаря, содержащего названия столбцов и словари для отображения старых значений в новые значения.

Словарь для преобразования признаков "num_doors" и "num_cylinders" в числовые значения задается следующим образом:

In [ ]:
```
cleanup_nums = {"num_doors": {"four": 4, "two": 2},
    "num_cylinders": {"four": 4, "six": 6, "five": 5, "eight": 8,
    "two": 2, "twelve": 12, "three":3 }}
```

Для преобразования признаков в числовые значения выполним код:

In [ ]:
```
obj_df.replace(cleanup_nums, inplace=True)
obj_df.head()
```

Pandas автоматически преобразует тип признаков в числовой (int64):

In [ ]:
```
obj_df.dtypes
```

Описанный подход работает в тех случаях, когда имеется понятный способ интерпретации текстовых значений как числовых.

Кодирование меток

Кодирование меток (label encoding) – это способ конвертации значений в столбцах в числа.

Например, столбец body_style содержит 5 различных значений. Можно закодировать их так:

```
convertible -> 0
hardtop -> 1
hatchback -> 2
sedan -> 3
wagon -> 4
```

Можно использовать Pandas, чтобы преобразовать столбец в категорию (категория – это тип данных в Pandas, принимающий несколько значений), а потом использовать значения категории для кодирования меток:

In [ ]:
```
obj_df["body_style"] = obj_df["body_style"].astype('category')
obj_df.dtypes
```

Далее можно присвоить закодированные значения признака новому столбцу "body_style_cat" используя свойство `cat.codes` :

In [ ]:
```
obj_df["body_style_cat"] = obj_df["body_style"].cat.codes
obj_df.head()
```

Особенностью этого подхода является то, что появляется возможность использовать преимущества категорий Pandas (компактность данных, возможность упорядочения, поддержка визуализации), при этом категории могут быть легко конвертированы в числовые значения для дальнейшего анализа.

Прямое кодирование (One Hot Encoding)

Кодирование меток имеет преимущество в виде простоты реализации и недостаток, состоящий в том, что числовое значение может быть некорректно интерпретировано алгоритмами машинного обучения. Например, значение 0, очевидно, меньше значения 2, но соответствует ли эта зависимость реальной ситуации для текстовых значений?

Альтернативный подход (прямое кодирование) состоит в том, чтобы конвертировать каждую категорию в новый столбец, принимающий значения 1 или 0 (True/False). Преимуществом этого подхода является то, что между категориальными значениями не устанавливаются несуществующие связи, а недостатком – что в наборе данных появляются дополнительные столбцы.

Pandas поддерживает этот подход в функции `get_dummies()` , которая создает новые столбцы вида “столбец_значение”.

Рассмотрим пример для столбца drive_wheels со значениями 4wd , fwd, rwd. Используя `get_dummies()` мы конвертируем этот столбец в три столбца со значениями 1 или 0, соответствующими правильному значению исходного признака (столбца):

In [ ]:
```
pd.get_dummies(obj_df, columns=["drive_wheels"]).head()
```

Столбец "drive_wheels" пропал, при этом новый набор данных содержит три новых столбца:

- drive_wheels_4wd
- drive_wheels_rwd
- drive_wheels_fwd

В функцию `get_dummies()` можно передать несколько столбцов с категориальными признаками, а также передать префиксы для именования новых столбцов с целью упростить последующий анализ данных:

In [ ]:
```
pd.get_dummies(obj_df, columns=["body_style", "drive_wheels"], pref
```

Прямое кодирование является очень полезным инструментом, однако может приводить к резкому увеличению числа столбцов в наборе, если категориальные признаки имеют большое число различных значений.

Двоичное кодирование, управляемое пользователем

В зависимости от особенностей набора данных можно использовать различные комбинации кодирования меток и прямого кодирования, которые в наибольшей степени соответствуют целям дальнейшего анализа.

Для иллюстрации двоичного кодирования, управляемого пользователем (custom binary encoding), рассмотрим следующий пример. В наборе данных имеется столбец engine_type (тип двигателя), который содержит несколько различных значений:

In [ ]:
```
obj_df["engine_type"].value_counts()
```

Допустим, что требуется выделить в отдельную группу все двигатели с верхней камерой (Overhead Cam или OHC). Другими словами, различные версии OHC эквивалентны для анализа. В это случае можно использовать свойство `str` и функцию `np.where` , чтобы создать новый столбец как индикатор того, что двигатель автомобиля имеет тип OHC.

In [ ]:
```
obj_df["OHC_Code"] = np.where(obj_df["engine_type"].str.contains("o
obj_df["OHC_Code"].value_counts()
```

В результате получаем набор данных, включающий столбец OHC_Code (показываем в наборе только три столбца):

In [ ]:
```
obj_df[["make", "engine_type", "OHC_Code"]].head()
```

Данный подход является по-настоящему полезным, если имеется возможность консолидировать бинарные значения (да/нет) в новом столбце.

Возможности кодирования в библиотеке Scikit-Learn

Библиотека scikit-learn также содержит функционал для кодирования текстовых признаков.

Например, чтобы кодировать метки для производителей автомобиля, используем объект `LabelEncoder` и метод `fit_transform()` для столбца с данными:

In [ ]:
```
from sklearn.preprocessing import LabelEncoder
lb_make = LabelEncoder()
obj_df["make_code"] = lb_make.fit_transform(obj_df["make"])
obj_df[["make", "make_code"]].head(11)
```

Scikit-learn также поддерживает бинарное кодирование при помощи объекта `LabelBinarizer` . Можно использовать процедуру, аналогичную приведенной выше, чтобы преобразовать данные, но требуются некоторые дополнительные шаги.

In [ ]:
```
from sklearn.preprocessing import LabelBinarizer
lb_style = LabelBinarizer()
lb_results = lb_style.fit_transform(obj_df["body_style"])
pd.DataFrame(lb_results, columns=lb_style.classes_).head()
```

На следующем шаге нужно включить эти данные в исходный набор данных.

Отбор признаков

Набор признаков, используемых для обучения модели, оказывает значительное влияние на качество результатов. Присутствие в наборе данных малоинформативных признаков приводит к снижению точности многих моделей, особенно моделей регрессии.

Отбор признаков (feature selection) – это процесс выбора признаков, обеспечивающий более высокое качество модели машинного обучения.

Отбор признаков перед построением модели обеспечивает следующие преимущества:

• Уменьшение переобучения. Чем меньше избыточных данных, тем меньше возможностей для модели принимать решения на основе «шума».

• Повышение точности. Чем меньше противоречивых данных, тем выше точность.

• Сокращение времени обучения. Чем меньше данных, тем быстрее обучается модель.

Будем работать с набором данных, содержащим информацию о качество вина.

Удаление признаков с низкой дисперсией

Простейшим подходом к отбору признаков является исключение признаков с низкой дисперсией. Если дисперсия признака равна нулю, то признак для всех записей имеет одно и то же значение и может не приниматься во внимание при анализе данных. Если дисперсия признака близка к нулю, то признак принимает значения, близкие к некоторому (среднему) значению и, скорее всего, является несущественным.

В качестве примера рассмотрим гипотетический набор данных с булевыми признаками и допустим, что мы хотим удалить все признаки, в которых нули или единицы составляют более чем 80% значений. Булевы признаки могут быть интепретированы как случайные величины с распределением Бернулли, имеющие дисперсию

$$V[X] = p\ (1 - p),$$

поэтому при отборе признаков можем использовать пороговое значение $0.8\,(1 - 0.8)$:

In [ ]:
```
from sklearn.feature_selection import VarianceThreshold
X = [[0, 0, 1],
     [0, 1, 0],
     [1, 0, 0],
     [0, 1, 1],
     [0, 1, 0],
     [0, 1, 1]]
sel = VarianceThreshold(threshold=(.8 * (1 - .8)))
sel.fit_transform(X)
```

Как и ожидалось, метод `VarianceThreshold` удалил первый столбец, для которого вероятность нулевого значения $p = \frac{5}{6} > 0.8$.

Одномерный отбор признаков

Признаки, имеющие наиболее выраженную взаимосвязь с целевой переменной, могут быть отобраны с помощью статистических критериев. Библиотека scikit-learn содержит класс `SelectKBest`, реализующий одномерный отбор признаков (univariate feature selection). Этот класс можно применять совместно с различными статистическими критериями для отбора заданного количества признаков.

В примере ниже используется критерий хи-квадрат (chi-squared test) для неотрицательных признаков, чтобы отобрать 4 лучших признака.

In [ ]:
```
# отбор признаков при помощи одномерных статистических тестов
import pandas as pd
from sklearn.feature_selection import SelectKBest,chi2

# загрузка данных - качество вина
url = "https://archive.ics.uci.edu/ml/machine-learning-databases/wi
df = pd.read_csv(url,sep=";")
print("\nИсходный набор данных:\n",df.head())
array = df.values
X = array[:,0:11] # входные переменные (11 признаков)
Y = array[:,11]   # выходная переменная - качество (оценка между 0

# отбор признаков
test = SelectKBest(score_func=chi2, k=4)
fit = test.fit(X, Y)

# оценки признаков
print("\nОценки признаков:\n",fit.scores_)

cols = test.get_support(indices=True)
df_new = df.iloc[:,cols]
print("\nОтобранные признаки:\n",df_new.head())
```

Мы видим оценки для каждого признака и 4 отобранных признака (с наивысшими оценками): volatile acidity, free sulfur dioxide, total sulfur dioxide и alcohol.

Если выходная (зависимая) переменная представляет собой класс, то можно использовать статистические критерии `chi2` или `f_classif`. Если выходная (зависимая) переменная представляет собой признак, принимающий непрерывные значения, то следует использовать статистический критерий `f_regression`.

Отбор на основе важности признаков

Ансамблевые алгоритмы на основе деревьев решений, такие как случайный лес (random forest), позволяют оценить важность признаков.

В представленном ниже примере мы обучаем классификатор `ExtraTreesClassifier`, чтобы с его помощью определить важность признаков.

In [ ]:
```
# Feature Importance with Extra Trees Classifier
from sklearn.ensemble import ExtraTreesClassifier

# загрузка данных - качество вина
url = "https://archive.ics.uci.edu/ml/machine-learning-databases/wi
df = pd.read_csv(url,sep=";")

array = df.values
X = array[:,0:11] # входные переменные (11 признаков)
Y = array[:,11]   # выходная переменная - качество (оценка между 0

# feature extraction
model = ExtraTreesClassifier()
model.fit(X, Y)
print(model.feature_importances_)
```

Мы получили оценки для каждого признака. Чем больше значение оценки, тем важнее признак. Таким образом, согласно данному методу отбора, двумя наиболее важными признаками являются: total sulfur dioxide и sulphates.

Метод главных компонент

Метод главных компонент (principal component analysis, PCA) позволяет уменьшить размерность данных с помощью преобразования на основе линейной алгебры. Пользователь может задать требуемое количество измерений (главных компонент) в результирующих данных.

Прочитаем набор данных "Ирисы" и сократим его размерность до двух:

In [ ]:
```
import pandas as pd
from sklearn.decomposition import PCA

url = \
"https://archive.ics.uci.edu/ml/machine-learning-databases/iris/iri

# считываем данные в объект data frame
my_data = pd.read_csv( url, header=None, prefix="V", usecols=(0,1,2

pca = PCA(n_components=2)

pcad = pca.fit_transform(my_data) # numpy array

print( "*** Первые 5 строк данных:" )
for x in range(0,5):
  print( pcad[x] )

print( "*** Дисперсии компонент:\n", pca.explained_variance_ratio_
```

Определим уровень объясняемой дисперсии для различных значений параметра `n_components` :

In [ ]:
```
for r in range(1,5):
  pca = PCA( n_components = r )
  pca.fit( my_data )
  print( "r =",r,"\tДисперсия =",
        sum(pca.explained_variance_ratio_)*100,"%" )
```

В примере ниже выделим 3 главных компоненты с помощью PCA.

In [ ]:
```
import numpy
from pandas import read_csv
from sklearn.decomposition import PCA

# загрузка данных - качество красного вина
url = "https://archive.ics.uci.edu/ml/machine-learning-databases/wi
df = pd.read_csv(url,sep=";")

array = df.values
X = array[:,0:11] # входные переменные (11 признаков)

# главные компоненты
pca = PCA(n_components=3)
fit = pca.fit(X)
features = fit.transform(X)

# результаты
print("Объясняемая дисперсия:", sum(fit.explained_variance_ratio_)*
print(features[0:5,:])
```

Результат преобразования (3 главных компоненты) совсем не похож на исходные данные и содержит отрицательные значения.

Визуализация данных

Используем Matplotlib для визуализации графиков:

In [ ]:
```
%matplotlib inline
import numpy as np
import matplotlib.pyplot as plt

plt.figure( figsize=(8, 6), dpi=80 ) # размер 8x6 дюйма, 80 точек на
plt.subplot( 1, 1, 1 )               # новый график на сетке 1x1

X = np.linspace( -np.pi, np.pi, 256, endpoint=True )
C, S = np.cos(X), np.sin(X)

# рисуем косинус синей сплошной линией шириной 2 пикселя
plt.plot( X, C, color="blue", linewidth=2.0, linestyle="-" )
# рисуем синус красными точками шириной 5 пикселей
plt.plot( X, S, color="red", linewidth=5.0, linestyle=":" )

plt.xlim( -4.0, 4.0 )                                # пределы оси x
plt.xticks( np.linspace(-4, 4, 9, endpoint=True) ) # метки на оси x
plt.ylim( -1.0, 1.0 )                                # пределы оси y
plt.yticks( np.linspace(-1, 1, 5, endpoint=True) );# метки на оси y

# plt.savefig('plot0.png', dpi=72) # сохранение в файл 72 точки на
```

Используем Matplotlib с настройкой меток осей и легендой:

In [ ]:
```
x = np.linspace(0, 2, 100)
plt.plot(x, x, label='линейная функция')
plt.plot(x, x**2, label='квадратичная функция')
plt.plot(x, x**3, label='кубическая функция')
plt.xlabel('Ось x')
plt.ylabel('Ось y')
plt.title('Графики трех функций')
plt.legend();
```

Используем Matplotlib для визуализации набора данных со сниженной размерностью (разными цветами):

In [ ]:
```
from urllib.request import urlopen
from contextlib import closing
import numpy as np
import matplotlib.pyplot as plt
from sklearn.decomposition import PCA

url = "https://archive.ics.uci.edu/ml/machine-learning-databases/ir
with closing(urlopen(url)) as u, open("iris.csv", "w") as f:
    f.write(u.read().decode())

data = np.genfromtxt( "iris.csv", delimiter=",", usecols=(0,1,2,3)
target = np.genfromtxt( "iris.csv", delimiter=",", usecols=(4), dty

pca = PCA(n_components=2)
pcad = pca.fit_transform( data )

plt.figure( figsize=(8, 6), dpi=200 )
plt.plot(pcad[target=="Iris-setosa",0],pcad[target=="Iris-setosa",1
plt.plot(pcad[target=="Iris-versicolor",0],pcad[target=="Iris-versi
plt.plot(pcad[target=="Iris-virginica",0],pcad[target=="Iris-virgin
```

Задание на лабораторную работу №2

Задание (10 баллов)

Для закрепленного за Вами варианта лабораторной работы:

1. Используя функционал библиотеки Pandas, считайте заданный набор данных из репозитария UCI.
2. Проведите исследование набора данных, выявляя числовые признаки. Если какие-то из числовых признаков были неправильно классифицированы, то преобразуйте их в числовые. Если в наборе для числовых признаков присутствуют пропущенные значения ('?'), то заполните их медианными значениями.
3. Определите признак, содержащий метку класса. Определите числовой признак, имеющий максимальную дисперсию.
4. При помощи класса SelectKBest библиотеки scikit-learn найдите два признака, имеющих наиболее выраженную взаимосвязь с признаком, имеющим максимальную дисперсию.
5. Визуализируйте набор данных в виде точек плоскости с координатами, соответствующими найденным признакам, отображая точки различных классов разными цветами.
6. Оставляя в наборе данных только числовые признаки, найдите и выведите на экран размерность метода главных компонент (параметр `n_components`), для которой доля объясняемой дисперсии будет не менее 99%.
7. Пользуясь методом главных компонент, снизьте размерность набора данных до двух признаков и изобразите полученный набор данных в виде точек на плоскости, отображая точки различных классов разными цветами.

In [ ]: