Методические указания к диссциплине ИИ и машинное обучение. Тема: конструирование признаков (Ибряева О.Л.)

В этом уроке мы сначала разберем, что делать с так называемыми категориальными признаками, свойствами нашего входного набора данных, которые характеризуют его, но при этом не заданы в виде чисел. Затем приведем некоторые примеры полезных преобразований признаков.

1. Категориальные признаки

Одним из распространенных типов нечисловых данных являются категориальные данные. Например, представьте, что вы изучаете некоторые данные о ценах на жилье, и наряду с числовыми характеристиками, такими как «цена» и «число комнат», у вас также есть информация о районе. Например, ваши данные могут выглядеть примерно так:

In [1]:
```
data = [
    {'price': 850000, 'rooms': 4, 'neighborhood': 'Queen Anne'},
    {'price': 700000, 'rooms': 3, 'neighborhood': 'Fremont'},
    {'price': 650000, 'rooms': 3, 'neighborhood': 'Wallingford'},
    {'price': 600000, 'rooms': 2, 'neighborhood': 'Fremont'}
]
```

Можно, на первый взгляд, закодировать эти данные с помощью простого числового сопоставления:

In [2]:
```
{'Queen Anne': 1, 'Fremont': 2, 'Wallingford': 3};
```

Это будет означать, например, что Queen Anne < Fremont < Wallingford, или что Wallingford - Queen Anne = Fremont, что не имеет большого смысла.

В этом случае одним из проверенных методов является использование one-hot encoding, которое создает дополнительные столбцы, указывающие на наличие или отсутствие категории со значением 1 или 0 соответственно. Ниже пример создания таких столбцов из 0 и 1 с помощью DictVectorizer из Scikit-Learn:

In [3]:
```
from sklearn.feature_extraction import DictVectorizer
vec = DictVectorizer(sparse=False, dtype=int)
vec.fit_transform(data)
```

Out[3]:
```
array([[     0,      1,      0, 850000,      4],
       [     1,      0,      0, 700000,      3],
       [     0,      0,      1, 650000,      3],
       [     1,      0,      0, 600000,      2]])
```

Обратите внимание, что столбец «neighborhood» был расширен на три отдельных столбца, представляющих три метки районов, и что каждая строка имеет 1 в столбце, соответствующем ее району.

Посмотрим на значения каждого столбца:

In [4]:
```
vec.get_feature_names()
```

Out[4]:
```
['neighborhood=Fremont',
 'neighborhood=Queen Anne',
 'neighborhood=Wallingford',
 'price',
 'rooms']
```

Разумеется, если ваш категориальный признак имеет много возможных значений, это приведет к значительному увеличению размера вашего набора данных.

2. Преобразование признаков

Мы уже видели, как добавление признаков, возведенных в квадрат или куб, улучшает линейные модели. Иногда требуется не добавить признаки, а подкорректировать существующие. Большинство моделей работают лучше, когда признаки имеют нормальное распределение, то есть гистограмма каждого признака должна в определенной степени иметь сходство с «колоколообразной кривой». Использование преобразований типа log и exp является банальным, но в то же время простым и эффективным способом добиться более симметричного распределения.

Воспользуемся синтетическим набором данных:

In [2]:
```
import numpy as np

rnd = np.random.RandomState(0)
X_org = rnd.normal(size=(1000, 3))
w = rnd.normal(size=3)
X = rnd.poisson(10 * np.exp(X_org))
y = np.dot(X_org, w)
```

In [4]:
```
X.shape
```

Out[4]:
```
(1000, 3)
```

Давайте посмотрим на первые 10 элементов первого признака. Все они являются положительными и целочисленными значениями, однако выделить какую-то определенную структуру сложно.

In [3]:
```
X[:10,0]
```

Out[3]:
```
array([ 56,  81,  25,  20,  27,  18,  12,  21, 109,   7])
```

Если посчитать частоту встречаемости каждого значения, распределение значений становится более ясным:

In [5]:
```
print("Частоты значений:\n{}".format(np.bincount(X[:, 0])))
```

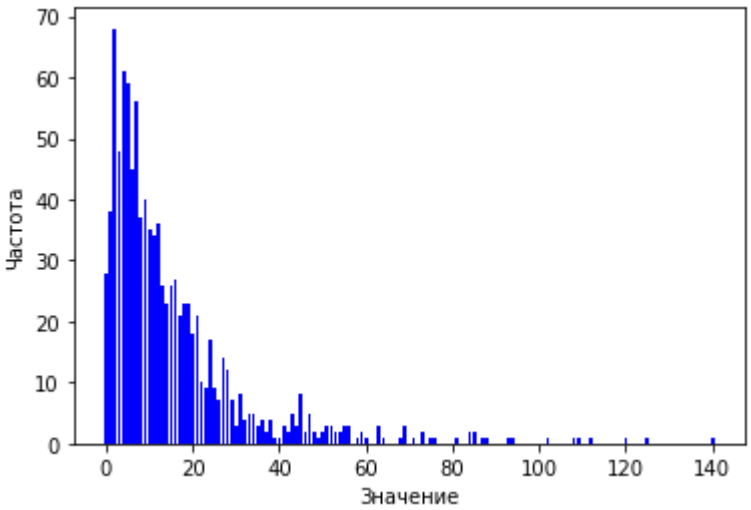
```
Частоты значений:
[28 38 68 48 61 59 45 56 37 40 35 34 36 26 23 26 27 21 23 23 18 21 10  9
 17  9  7 14 12  7  3  8  4  5  5  3  4  2  4  1  1  3  2  5  3  8  2  5
  2  1  2  3  3  2  2  3  3  0  1  2  1  0  0  3  1  0  0  0  1  3  0  1
  0  2  0  1  1  0  0  0  0  1  0  0  2  2  0  1  1  0  0  0  0  1  1  0
  0  0  0  0  0  0  1  0  0  0  0  0  1  1  0  0  1  0  0  0  0  0  0  0
  1  0  0  0  0  1  0  0  0  0  0  0  0  0  0  0  0  0  0  0  1]
```

Значение 2является наиболее распространенным, оно встречается 68 раз (bincount всегда начинает считать с 0), а частоты более высоких значений быстро падают. Однако есть несколько очень высоких значений, например, 84 и 85, которые встречаются два раза. Мы визуализируем частоты на рис

In [8]:
```
import matplotlib.pyplot as plt

bins = np.bincount(X[:, 0])
plt.bar(range(len(bins)), bins, color='b')
plt.ylabel("Частота")
plt.xlabel("Значение")
```

Out[8]:
```
Text(0.5, 0, 'Значение')
```



Признаки X[:, 1] и X[:, 2] имеют аналогичные свойства. Полученное распределение значений (высокая частота встречаемости маленьких значений и низкая частота встречаемости больших значений) является очень распространенным явлением в реальной практике. Однако для большинства линейных моделей оно может представлять трудность. Давайте попробуем использовать гребневую регрессию:

In [11]:
```
from sklearn.model_selection import train_test_split
from sklearn.linear_model import Ridge

X_train, X_test, y_train, y_test = train_test_split(X, y, random_state=0)
score = Ridge().fit(X_train, y_train).score(X_test, y_test)
print("Правильность на тестовом наборе: {:.3f}".format(score))
```

```
Правильность на тестовом наборе: 0.622
```

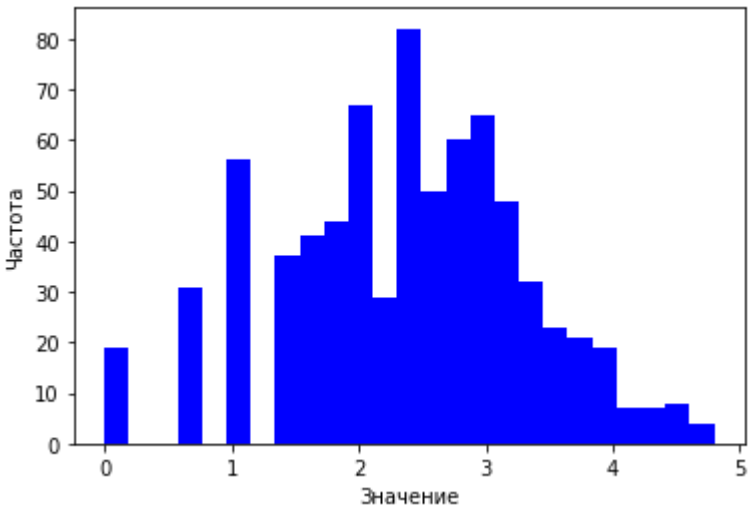
Применим логарифмическое преобразование к данным. Поскольку в данных появляется значение 0 (а логарифм 0 не определен), мы вычислим не log X, а log(X + 1):

In [12]:
```
X_train_log = np.log(X_train + 1)
X_test_log = np.log(X_test + 1)
```

После преобразования распределение данных стало менее асимметричным и уже не содержит очень больших выбросов:

In [14]:
```
plt.hist(X_train_log[:, 0], bins=25, color='b')
plt.ylabel("Частота")
plt.xlabel("Значение")
```

Out[14]:
```
Text(0.5, 0, 'Значение')
```



Построение модели гребневой регрессии на новых данных дает гораздо более лучше качество:

In [15]:
```
score = Ridge().fit(X_train_log, y_train).score(X_test_log, y_test)
print("Правильность на тестовом наборе: {:.3f}".format(score))
```

```
Правильность на тестовом наборе: 0.875
```

Поиск преобразования, которое наилучшим образом сработает для конкретного сочетания данных и модели – это в некоторой степени искусство. В этом примере все признаки имели одинаковые свойства. Такое редко бывает на практике, и как правило, лишь некоторые признаки нуждаются в преобразовании, либо в ряде случаев каждый признак необходимо преобразовывать по-разному.

Эти преобразования не имеют значения для моделей на основе дерева, но могут иметь важное значение для линейных моделей.

3. Автоматический отбор признаков

Не всегда добавление новых признаков является хорошей идеей. Это делает модели более сложными и поэтому увеличивает вероятность переобучения. Добавляя новые признаки или работая с высокоразмерными наборами данных, неплохо бы уменьшить количество признаков и оставить только наиболее полезные из них. Это позволит получить более простые модели с лучшей обобщающей способностью. Однако как узнать, насколько полезен каждый признак? Существуют три основные стратегии: `одномерные статистики` (univariate statistics), `отбор на основе модели` (model-based selection) и `итеративный отбор` (iterative selection).

3.1 Одномерные статистики

С помощью одномерных статистик мы определяем наличие статистически значимой взаимосвязи между каждым признаком и зависимой переменной. Затем отбираем признаки, сильнее всего связанные с зависимой переменной. "Одномерность" заключается в том, что каждый признак рассматривается по отдельности. Следовательно признак будет исключен, если он становится информативным лишь в сочетании с другим признаком.

Чтобы осуществить одномерный отбор признаков в scikit-learn, вам нужно выбрать тест, обычно либо `f_classif` (по умолчанию) для классификации или `f_regression` для регрессии, а также один из методов исключения признаков, основанных на использовании порогового значения. Самыми простыми из них являются SelectKB, выбирающий фиксированное число k признаков, и SelectPercentile, выбирающий фиксированный процент признаков.

Давайте применим отбор признаков для классификационной задачи к набору данных `cancer`. Чтобы немного усложнить задачу, мы добавим к данным некоторые неинформативные шумовые признаки. Мы предполагаем, что отбор признаков сможет определить неинформативные признаки и удалит их:

In [23]:
```
from sklearn.datasets import load_breast_cancer
from sklearn.feature_selection import SelectPercentile
from sklearn.model_selection import train_test_split

cancer = load_breast_cancer()

# задаем определенное стартовое значение для воспроизводимости результата
rng = np.random.RandomState(11)
noise = rng.normal(size=(len(cancer.data), 50))

# добавляем к данным шумовые признаки
# первые 30 признаков являются исходными, остальные 50 являются шумовыми
X_w_noise = np.hstack([cancer.data, noise])
X_train, X_test, y_train, y_test = train_test_split(
X_w_noise, cancer.target, random_state=0, test_size=.5)
```

```
# используем f_classif (по умолчанию)
# и SelectPercentile, чтобы выбрать 50% признаков
select = SelectPercentile(percentile=50)
select.fit(X_train, y_train)

# преобразовываем обучающий набор
X_train_selected = select.transform(X_train)
print("форма массива X_train: {}".format(X_train.shape))
print("форма массива X_train_selected: {}".format(X_train_selected.shape))
```

```
форма массива X_train: (284, 80)
форма массива X_train_selected: (284, 40)
```

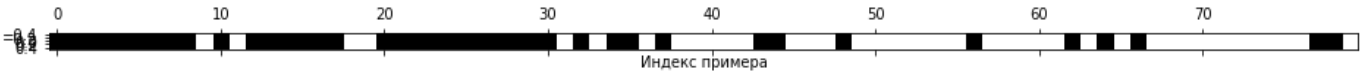
Как видно, количество признаков уменьшилось с 80 до 40 (на 50% от исходного количества признаков). Мы можем выяснить, какие функции были отобраны, воспользовавшись методом `get_support`, который возвращает булевы значения для каждого признака:

In [24]:
```
mask = select.get_support()
print(mask)

# визуализируем булевы значения: черный - True, белый - False
plt.matshow(mask.reshape(1, -1), cmap='gray_r')
plt.xlabel("Индекс примера")
```

```
[ True  True  True  True  True  True  True  True  True False  True False
  True  True  True  True  True  True False False  True  True  True  True
  True  True  True  True  True  True  True False  True False  True  True
 False  True False False False False False  True  True False False False
  True False False False False False False False  True False False False
 False False  True False  True False  True False False False False False
 False False False False False  True  True False]
```

Out[24]:
```
Text(0.5, 0, 'Индекс примера')
```



Видно, что большинство отобранных признаков являются исходными характеристиками, а большинство шумовых признаков были удалены. Тем не менее восстановление исходных признаков далеко от идеала. Давайте сравним правильность логистической регрессии с использованием всех признаков с правильностью логистической регрессии, использующей лишь отобранные признаки:

In [25]:
```
from sklearn.linear_model import LogisticRegression

# преобразовываем тестовые данные
X_test_selected = select.transform(X_test)
lr = LogisticRegression(max_iter=10000)
lr.fit(X_train, y_train)

print("Правильность со всеми признаками: {:.3f}".format(lr.score(X_test, y_test)))
lr.fit(X_train_selected, y_train)
print("Правильность только с отобранными признаками: {:.3f}".format(
lr.score(X_test_selected, y_test)))
```

```
Правильность со всеми признаками: 0.930
Правильность только с отобранными признаками: 0.940
```

В данном случае удаление шумовых признаков повысило правильность, даже несмотря на то, что некоторые исходные признаки отсутствовали. Это был очень простой синтетический пример, результаты, получающиеся на реальных данных, как правило, получаются смешанными. Однако одномерный отбор признаков может быть очень полезен, если их количество является настолько большим, что невозможно построить модель, используя все эти характеристики, или же вы подозреваете, что многие характеристики совершенно неинформативны.

3.2 Отбор признаков на основе модели

Отбор признаков на основе модели использует модель машинного обучения с учителем, чтобы вычислить важность каждого признака, и оставляет только самые важные из них.

Модель, применяющаяся для отбора признаков, требует вычисления определенного показателя важности для всех признаков, с тем чтобы характеристики можно было ранжировать по этой метрике. В деревьях решений и моделях на основе дерева решений такой показатель реализован с помощью атрибута `feature_importances_` , в котором записывается важность каждого признака.

Чтобы применить отбор на основе модели, мы должны воспользоваться инструментом SelectFromModel:

In [62]:
```
from sklearn.feature_selection import SelectFromModel
from sklearn.ensemble import RandomForestClassifier
select = SelectFromModel(RandomForestClassifier(n_estimators=100,
                                                random_state=11),threshold="median")
```

`SelectFromModel` отбирает все признаки, у которых показатель важности (заданный моделью машинного обучения с учителем) превышает установленное пороговое значение. Чтобы вычислить результат, сопоставимый с тем, который мы получили при однофакторном отборе признаков, мы использовали в качестве порогового значения медиану, поэтому будет отобрана половина признаков. Мы используем случайный лес на основе деревьев классификации (100 деревьев), чтобы вычислить важности признаков. Это довольно сложная модель, обладающая гораздо большей прогнозной силой, нежели одномерные тесты. Теперь давайте обучим эту модель:

In [63]:
```
select.fit(X_train, y_train)
X_train_mb = select.transform(X_train)
X_test_mb = select.transform(X_test)
print("X_train.shape: {}".format(X_train.shape))
print("X_train_mb.shape: {}".format(X_train_mb.shape))
```

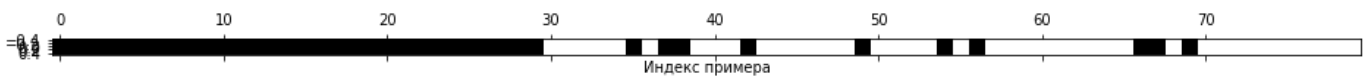
```
X_train.shape: (284, 80)
X_train_mb.shape: (284, 40)
```

И снова мы можем взглянуть на отобранные признаки

In [64]:
```
mask = select.get_support()

# визуализируем булевы значения -- черный - True, белый - False
plt.matshow(mask.reshape(1, -1), cmap='gray_r')
plt.xlabel("Индекс примера")
```

Out[64]:
```
Text(0.5, 0, 'Индекс примера')
```



На этот раз были отобраны все исходные признаки, кроме последнего. Поскольку мы задали отбор лишь 40 признаков, некоторые шумовые признаки также будут выбраны. Давайте посмотрим на правильность:

In [65]:
```
X_test_l1 = select.transform(X_test)
score = LogisticRegression(max_iter=10000).fit(X_train_mb, y_train).score(X_test_mb, y_test)
print("Правильность на тестовом наборе: {:.3f}".format(score))
```

```
Правильность на тестовом наборе: 0.951
```

Как видим, прогноз модели улучшился.

3.3 Итеративный отбор признаков

В одномерном отборе признаков мы не использовали модель, а в отборе признаков на основе модели мы построили одну модель, чтобы выбрать характеристики. В итеративном отборе признаков строится последовательность моделей с различным количеством признаков. Существует два основных подхода.

В первом случае метод начинается с шага, когда в модель включена лишь одна константа (входных признаков нет) и затем добавляет признак за признаком до тех пор, пока не будет достигнут критерий остановки. Второй подход начинает с шага, когда все признаки включены в модель, и затем начинает удалять признак за признаком, пока не будет достигнут критерий остановки.

Поскольку строится последовательность модели, эти методы с вычислительной точки зрения являются гораздо более затратными в отличие от ранее обсуждавшихся методов. Одним из таких методов является метод рекурсивного исключения признаков (recursive feature elimination , RFE), который начинается с включения всех признаков, строит модель и исключает наименее важный признак с точки зрения модели. Затем строится новая модель с использованием всех признаков, кроме исключенного, и так далее, пока не останется лишь заранее определенное количество признаков.

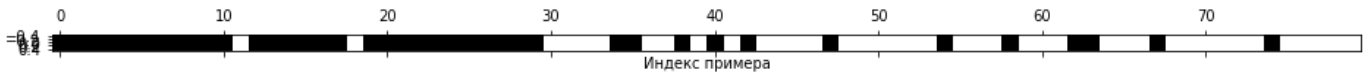
Здесь мы снова воспользуемся той же самой моделью случайного леса, которую применяли ранее:

In [52]:
```
from sklearn.feature_selection import RFE

select = RFE(RandomForestClassifier(n_estimators=100, random_state=12), n_features_to_select=40
select.fit(X_train, y_train)

# визуализируем отобранные признаки:
mask = select.get_support()
plt.matshow(mask.reshape(1, -1), cmap='gray_r')
plt.xlabel("Индекс примера")
```

Out[52]:
```
Text(0.5, 0, 'Индекс примера')
```



Как можно видеть, двух признаков не хватает. Выполнение этого программного кода занимает значительно больше времени в отличие от модельного отбора, поскольку модель случайного леса обучается 40 раз, по одной итерации для каждого отбрасываемого признака. Давайте проверим правильность модели логистической регрессии с использованием RFE для отбора признаков:

In [53]:
```
X_train_rfe= select.transform(X_train)
X_test_rfe= select.transform(X_test)
score = LogisticRegression(max_iter=10000).fit(X_train_rfe, y_train).score(X_test_rfe, y_test)
print("Правильность на тестовом наборе: {:.3f}".format(score))
```

```
Правильность на тестовом наборе: 0.958
```

Автоматический отбор признаков отлично подходит для уменьшения количества необходимых признаков, например, чтобы увеличить скорость вычисления или получить более интерпретируемые модели.

В большинстве реальных примеров применение отбора признаков вряд ли обеспечит большой прирост производительности. Тем не менее, он по-прежнему является ценным инструментом в арсенале специалиста по анализу данных.

Задание. Используйте RFE для выбора трех значимых признаков из четырех (sepal length, sepal width, petal length, petal width) в задаче классификации ирисов Фишера. Приведите код, который выдает значения True/False для каждого признакак. Какой признак менее информативен?

In [ ]: