Лабораторная работа № 1.1 Табличное представление данных

Введение

Обычно в задаче анализа данных имеется некоторая прямоугольная таблица. Ее строки соответствуют объектам, а столбцы – признакам этих объектов. Объекты также называются наблюдениями или примерами (samples), а признаки – атрибутами (features).

Признаки бывают количественными (как, например, доход в рублях или рост в сантиметрах и т.д.) или категориальными (как, например, марка автомобиля, модель телефона и т.д.).

Один из признаков (столбцов) выделен. Этот признак называется ответом. Остальные признаки – входные. Требуется по имеющейся таблице научиться по новому объекту, которого нет в таблице, но для которого известны значения входных признаков, по возможности с небольшой ошибкой предсказывать значение выделенного признака (ответа).

Если ответ количественный, то задача называется задачей регрессии. Если ответ категориальный, то задача называется задачей классификации.

Для решения задачи анализа данных воспользуемся библиотеками numpy, pandas. Библиотека numpy содержит реализации многомерных массивов и алгоритмов линейной алгебры. Библиотека pandas предоставляет широкий спектр функций по обработке табличных данных. Кроме того, нам понадобится библиотека matplotlib для научной визуализации.

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
```

Этап загрузки данных

В качестве примера рассмотрим задачу, ставшую уже классической, о пассажирах Титаника. Здесь объектами являются пассажиры, а признаками могут являться их пол, номер билета, имена и т.д.

В качестве выделенного признака (ответа) выступает информация о том, выжил ли пассажир или нет в результате аварии. По этим данным требуется научиться предсказывать, выживет ли пассажир в крушении лайнера или нет.

Таким образом, речь идет о задаче классификации: требуется определить, какому классу: положительному (пассажир выжил) или отрицательному (пассажир не выжил) – принадлежит пассажир.

Данные имеют формат csv, загрузим датасет с помощью следующего участка кода:

```
data = pd.read_csv('train.csv', na_values='?')
```

Итак, мы загрузили данные в таблицу data. Объект data имеет тип DataFrame – это основной тип данных в библиотеке pandas, предназначенный для представления табличных данных.

In [3]:
```
data
```

Out[3]:

	PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare	Cabin	Embarked
0	1	0	3	Braund, Mr. Owen Harris	male	22.0	1	0	A/5 21171	7.2500	NaN	S
1	2	1	1	Cumings, Mrs. John Bradley (Florence Briggs Th...	female	38.0	1	0	PC 17599	71.2833	C85	C
2	3	1	3	Heikkinen, Miss. Laina	female	26.0	0	0	STON/O2. 3101282	7.9250	NaN	S
3	4	1	1	Futrelle, Mrs. Jacques Heath (Lily May Peel)	female	35.0	1	0	113803	53.1000	C123	S
4	5	0	3	Allen, Mr. William Henry	male	35.0	0	0	373450	8.0500	NaN	S
...	...	...	...	...	...	...	...	...	...	...	...	...
886	887	0	2	Montvila, Rev. Juozas	male	27.0	0	0	211536	13.0000	NaN	S
887	888	1	1	Graham, Miss. Margaret Edith	female	19.0	0	0	112053	30.0000	B42	S
888	889	0	3	Johnston, Miss. Catherine Helen "Carrie"	female	NaN	1	2	W./C. 6607	23.4500	NaN	S
889	890	1	1	Behr, Mr. Karl Howell	male	26.0	0	0	111369	30.0000	C148	C
890	891	0	3	Dooley, Mr. Patrick	male	32.0	0	0	370376	7.7500	NaN	Q

891 rows × 12 columns

In [ ]:
```
```

Этап анализа данных

Данные загружены. Попытаемся вначале их качественно проанализировать. Узнаем размеры таблицы:

```
data.shape
```

(891, 12)

Итак, таблица содержит 891 строк (объектов) и 12 столбцов (признаков), включая выходной (целевой) признак.

Столбец PassengerId содержит числа от 1 до 891 и не несет никакой информации, кроме того эта информация фактически дублирует номер записи. Установим этот столбец в качестве индекса, это избавит наши данные от одного лишнего признака и упростит нам в дальнейшем задачу:

```
data = data.set_index('PassengerId')
```

Мы можем посмотреть на несколько первых и несколько последних строк этой таблицы, чтобы получить представление об имеющихся данных:

```
data.head()
```

```
data.tail()
```

Как можно заметить, теперь столбец PassengerId выступает в роли индекса и выделен жирным шрифтом.

С помощью метода describe() получим некоторую сводную информацию по всей таблице. По умолчанию будет выдана информация только для количественных признаков. Это общее их количество (count), среднее значение (mean), стандартное отклонение (std), минимальное (min), максимальное (max) значения, медиана (50%) и значения нижнего (25%) и верхнего (75%) квартилей:

```
data.describe()
```

Заметим, что количество элементов в столбце Age меньше общего количества объектов, что говорит о том, что этот столбец содержат пропущенные значения. Заполним пропущенные значения средним значением по этому столбцу:

```
data['Age'] = data['Age'].fillna(data['Age'].mean())
```

Выделим числовые и категориальные признаки. Для того, чтобы выделить категориальные признаки - проверим, являются ли значения в столбцах объектами типа ‘object’. Чтобы выделить числовые признаки, можно воспользоваться методом DataFrame’a .mode(numeric_only=True), который возвращает только столбцы с числовыми значениями:

```
categorical_columns = [c for c in data.columns if data[c].dtype.name ==
'object']
numerical_columns   =
data.mode(numeric_only=True).columns.values.tolist()
numerical_columns.remove('Survived')
print(categorical_columns)
print(numerical_columns)
```

В дополнение можно из числовых столбцов удалить столбец ‘Survived’.

Теперь мы можем получить некоторую общую информацию по категориальным признакам:

```
data[categorical_columns].describe()
```

В таблице для каждого категориального признака приведено общее число заполненных ячеек (count), количество значений, которые принимает данный признак (unique), самое популярное (часто встречающееся) значение этого признака (top) и количество объектов, в которых встречается самое частое значение данного признака (freq).

Вот немного другой способ получить ту же информацию:

```
data.describe(include=[object])
```

Определить полный перечень значений категориальных признаков можно, например, так:

```
for c in categorical_columns:
    print(data[c].unique())
```

Здесь nan означают пропущенные значения.

Подготовка данных

Алгоритмы машинного обучения из библиотеки scikit-learn не работают напрямую с категориальными признаками и данными, в которых имеются пропущенные значения. Поэтому вначале подготовим наши данные.

Пропущенные значения

Узнать количество заполненных (непропущенных) элементов можно с помощью метода count. Параметр axis = 0 указывает, что мы двигаемся по размерности 0 (сверху вниз), а не размерности 1 (слева направо), т.е. нас интересует количество заполненных элементов в каждом столбце, а не строке:

```
data.count(axis=0)
```

Если данные содержат пропущенные значения, то имеется две простые альтернативы:

- удалить столбцы с такими значениями (data = data.dropna(axis=1)),
- удалить строки с такими значениями (data = data.dropna(axis=0)).

После этого, к сожалению, данных может стать совсем мало, поэтому рассмотрим простые альтернативные способы.

Количественные признаки

Заполнить пропущенные значения можно с помощью метода fillna. Заполним, например, медианными значениями.

axis=0 по-прежнему указывает, что мы двигаемся сверху вниз:

```
data = data.fillna(data.median(axis=0), axis=0)
```

Проверим, что теперь все столбцы, соответствующие количественным признакам, заполнены.

```
data.count(axis=0)
```

Категориальные признаки

Теперь рассмотрим пропущенные значения в столбцах, соответствующих категориальным признакам. Из информации, полученной ранее, можно сделать вывод, что среди категориальных признаков столбцы с пропущенными значениями - Cabin и Embarked. Начнем с Embarked:

```
data['Embarked'].describe()
```

В столбце 'Embarked' имеются пропущенные значения. Наиболее частым (встречается 644 раз) является S. Заполняем все пропуски этим значением:

```
data['Embarked'] = data['Embarked'].fillna('S')
```

Столбец “Cabin” содержит слишком мало значений, по отношению к общему количеству пассажиров. Заполнять его самым частым значением не целесообразно, т.к. Это может привести к неверной интерпретации данных и впоследствии только навредить. Следует его **удалить**.

```
data.describe()
```

In [32]:
```
categorical_columns = [c for c in data.columns if data[c].dtype.name == 'object']

data[categorical_columns].describe()
```

Out[32]:

	Name	Sex	Ticket	Embarked
count	891	891	891	891
unique	891	2	681	3
top	Braund, Mr. Owen Harris	male	347082	S
freq	1	577	7	646

Векторизация

Как уже отмечалось, библиотека scikit-learn не умеет напрямую обрабатывать категориальные признаки. Поэтому прежде чем подавать данные на вход алгоритмов машинного обучения преобразуем категориальные признаки в количественные.

Категориальные признаки, принимающие два значения (т.е. бинарные признаки) и принимающие большее количество значений будем обрабатывать по-разному.

Вначале выделим бинарные и небинарные признаки:

```
data_describe = data.describe(include=[object])
binary_columns    = [c for c in categorical_columns if
data_describe[c]['unique'] == 2]
nonbinary_columns = [c for c in categorical_columns if
data_describe[c]['unique'] > 2]
print(binary_columns, nonbinary_columns)
```

Бинарные признаки

Значения бинарных признаков просто заменим на 0 и 1. В рассматриваемых данных бинарный столбец только “Sex”. Осуществить замену можно с помощью метода replace или loc.

Небинарные признаки

К небинарным признакам применим метод векторизации, который заключается в следующем.

Признак j, принимающий s значений, заменим на s признаков, принимающих значения 0 или 1, в зависимости от того, чему равно значение исходного признака j.

Например, в нашей задаче признак ‘Embarked’ принимает 3 различных значения:

```
data['Embarked'].unique()
```

Заменим признак Embarked тремя признаками: C, Q, S:

- Если признак Embarked принимает значение C, то признак C равен 1, Q равен 0, S равен 0.
- Если признак Embarked принимает значение Q, то признак C равен 0, Q равен 1, S равен 0.
- Если признак Embarked принимает значение S, то признак C равен 0, Q равен 0, S равен 1.

Такую векторизацию осуществляет в pandas метод get_dummies:

```
data_nonbinary = pd.get_dummies(data['Embarked'])
data_nonbinary
```

Столбцы Name и Ticket содержат слишком большое количество уникальных значений, поэтому в случае векторизации образуется слишком большое количество дополнительных столбцов, это может негативно повлиять на дальнейшую работу, поэтому **удалим** их.

Нормализация количественных признаков

Многие алгоритмы машинного обучения чувствительны к масштабированию данных. К таким алгоритмам, например, относится метод ближайших соседей, машина опорных векторов и др.

В этом случае количественные признаки полезно нормализовать. Это можно делать разными способами. Например, каждый количественный признак приведем к нулевому среднему и единичному среднеквадратичному отклонению:

```
data_numerical = data[numerical_columns]
data_numerical = (data_numerical - data_numerical.mean()) /
data_numerical.std()
data_numerical.describe()
```

Соединим все столбцы в одну таблицу:

```
survived = data['Survived']
data = pd.concat((data_numerical, data[binary_columns], data_nonbinary),
axis=1)
data = pd.DataFrame(data, dtype=float)
print(data.shape)
print(data.columns)
```

Для удобства отдельно рассмотрим столбцы, соответствующие входным признакам (это будет матрица X), а отдельно – выделенный признак (вектор y):

```
X_train = data  # Выбрасываем столбец 'Survived'.
y_train = survived
feature_names = X_train.columns
print(feature_names)
```

```
print(X.shape)
print(y.shape)
N, d = X.shape
```

Теперь у нас 9 входных признака.

Обучающая и тестовая выборки

Почти все готово, чтобы запустить алгоритмы машинного обучения.

Обучаться, или, как говорят, строить модель, мы будем на обучающей выборке, а проверять качество построенной модели – на тестовой.

Проделанные ранее операции над DataFrame'ом необходимо обернуть в специальный метод, можно назвать его data_prep(data). С помощью этого метода необходимо выполнить подготовку данных для дальнейшего тестирования модели, находящихся в файле test.csv.

Обучение модели

Воспользуемся моделью случайного леса из библиотеки Sklearn

```
from sklearn.ensemble import RandomForestClassifier
model = RandomForestClassifier(random_state=0)
model.fit(X_train, y_train)
```

Чтобы проверить значения, предсказываемые моделью - можно воспользоваться методом predict():

```
y_train_pred=model.predict(X_train)
```

Для оценки качества полученной модели используем метрику accuracy:

```
from sklearn import metrics

print("Accuracy:",metrics.accuracy_score(y_train, y_train_pred))
```

После обучения модели можно исследовать важность признаков, т.е. На сколько они влияют на результат работы модели, для этого можно воспользоваться следующим участком кода:

```
importances = model.feature_importances_

feature_names = X_train.columns
forest_importances = pd.Series(importances, index=feature_names)

fig, ax = plt.subplots()
forest_importances.plot.bar(ax=ax)
ax.set_title("Feature importances using MDI")
ax.set_ylabel("Mean decrease in impurity")
fig.tight_layout()
```

В некоторых случаев удаление мало значимых признаком может повысить итоговые показатели качества работы модели. Для проверки таких гипотез - необходимо удалить соответствующие столбцы и обучить модель заново.

Задание для выполнения лабораторной работы:

Исследовать представленный набор данных, составить описание отдельных признаков, произвести их нормализацию и векторизацию. Полностью подготовить набор данных для дальнейшего их использования при обучении модели. Подготовить обучающие и тестовые данные для модели, обучить модель, проверить показатели качества на обучающей и тестовой выборках. Исследовать важность признаков, проверить, на сколько влияет удаление незначимых признаков на итоговый результат классификации.