В. В. Братищенко

ИНФОРМАЦИОННЫЕ ТЕХНОЛОГИИ В БИЗНЕС-АНАЛИТИКЕ С ИСПОЛЬЗОВАНИЕМ ORANGE3 И PYTHON

Учебное пособие

Министерство науки и высшего образования Российской Федерации
Байкальский государственный университет

В. В. Братищенко

ИНФОРМАЦИОННЫЕ ТЕХНОЛОГИИ В БИЗНЕС-АНАЛИТИКЕ С ИСПОЛЬЗОВАНИЕМ ORANGE3 И PYTHON

Учебное пособие

Текстовое электронное издание

Иркутск
Издательский дом БГУ
2021

УДК 681.3
ББК 32.973

Издается по решению редакционно-издательского совета Байкальского государственного университета

Рецензенты: канд. физ.-мат. наук, доц. В. В. Ступин
канд. техн. наук А. В. Родионов

Братищенко, В. В. Информационные технологии в бизнес-аналитике с использованием Orange3 и Python : учеб. пособие / В. В. Братищенко. – Иркутск : Изд. дом БГУ, 2021. – 117 с. – URL: http://lib-catalog.bgu.ru. – Текст : электрон.

Пособие описывает компьютерные технологии и инструменты анализа данных. Приводятся сведения, необходимые для практического применения свободно распространяемых программных продуктов Orange и scikit-learn, разработанных в среде алгоритмического языка Python. Рассмотрены способы и примеры решения наиболее распространенных задач анализа данных и исследования зависимостей: классификация, кластеризация, регрессия, построение моделей временных рядов, выявление ассоциаций, обработка текстов на естественном языке. Представлены схемы решения задач в Orange и примеры использования модулей scikit-learn.

Пособие предназначено для студентов, обучающихся по программе магистратуры по направлению «Прикладная информатика в экономике».

Учебное электронное издание

Минимальные системные требования:
веб-браузер Internet Explorer версии 6.0 и более поздние, Opera версии 7.0 и более поздние, Google Chrome 3.0 и более поздние.
Компьютер с доступом к сети Интернет.
Минимальные требования к конфигурации и операционной системе компьютера определяются требованиями перечисленных выше программных продуктов.

Подготовлено к использованию Т. И. Кочульской

Подписано к использованию 23.11.2021.
Объем 4,5 Мб.

Байкальский государственный университет.
664003, г. Иркутск, ул. Ленина, 11.
http://bgu.ru.

Оглавление

Введение

Применение серверных аналитических технологий, таких как MS SQL Analysis Services [1] или Deductor[1] [2], может быть ограничено по причине отсутствия доступа к соответствующим службам. Кроме того, аналитические возможности данных служб фиксированы разработчиками и в целом уступают возможностям большого количества инициативных разработок, которые не ограничены рамками коммерческого распространения. Свободно распространяемые модули позволяют без существенных затрат реализовать аналитические вычисления и оценить целесообразность их развертывания. Конечно, такие модули проигрывают проприетарным предложениям по удобству использования и, зачастую, требуют знания соответствующего языка программирования, но все-таки являются серьезной альтернативой коммерческим продуктам в реализации бизнес-аналитики.

В данном пособии рассматривается система Orange[2] [3]. Orange позволяет на основе визуального программирования выполнить все основные этапы анализа данных с применением соответствующих модулей Python и специальных модулей Orange. В пособии не приводятся сведения о развертывании Orange. Эти сведения можно получить в Интернете[3]. Изложение материала будет связано с применением модулей Orange и scikit-learn[4] для решения аналитических задач и демонстрацией примеров таких решений.

Проектирование обработки в Orange заключается в размещении на рабочем пространстве элементов обработки данных – виджетов и связывании их потоками передачи данных. Причем параллельные потоки будут исполнятся одновременно.

Orange разработан на языке Python. Его модули могут быть использованы вне системы визуального программирования. Таким образом, Orange можно применять для быстрой апробации моделей, а для встраивания найденных решений с систему программирования можно использовать соответствующие модули. Кроме этого, Orange использует модели, реализованные в scikit-learn. Эти модели можно применять в программах на Python. Для этого можно использовать сборку Anaconda[5].

Ключевым в полезном применении аналитических технологий является понимание того, что могут дать эти технологии пользователям, наличие репрезентативных данных для настройки моделей. Orange позволяет быстро оценить адекватность обнаруженных зависимостей и целесообразность их применения для решения практических задач.

[1] URL: https://basegroup.ru/deductor/description.
[2] URL: https://orange.biolab.si/docs.
[3] URL: https://orangedatamining.com/download/#windows.
[4] URL: https://scikit-learn.org/stable.
[5] URL: https://www.anaconda.com/products/individual.

1. Среда Orange

Виджеты в Orange сгруппированы по панелям (рис. 1) и перетаскиваются на рабочее поле. На рис. 1 представлено решение задачи классификации. Используются следующие виджеты:

– «File» загружает данные из файла;

– «Data Table» обеспечивает просмотр данных;

– «Feature Statistics» позволяет изучить распределение каждой переменной (колонки);

– «Continuize» преобразует текстовые метки в числовые коды;

– «Test and Score» выполняет подбор параметров и вычисление показателей качества классификации – на вход передаются данные и классификаторы («Logistic Regression» – логистическая регрессия, «Neural Network» – нейронная сеть и «Random Forest» – случайный лес);

– «Confusion Matrix» строит матрицу ошибок – количество совпадений / несовпадений результатов классификации и наблюдений;

– «ROC Analysis» строит ROC-кривую качества классификации;

– «Predictions» определяет классы для входных данных и настроенному классификатору;

– «Save Data» сохраняет результаты в виде файла.

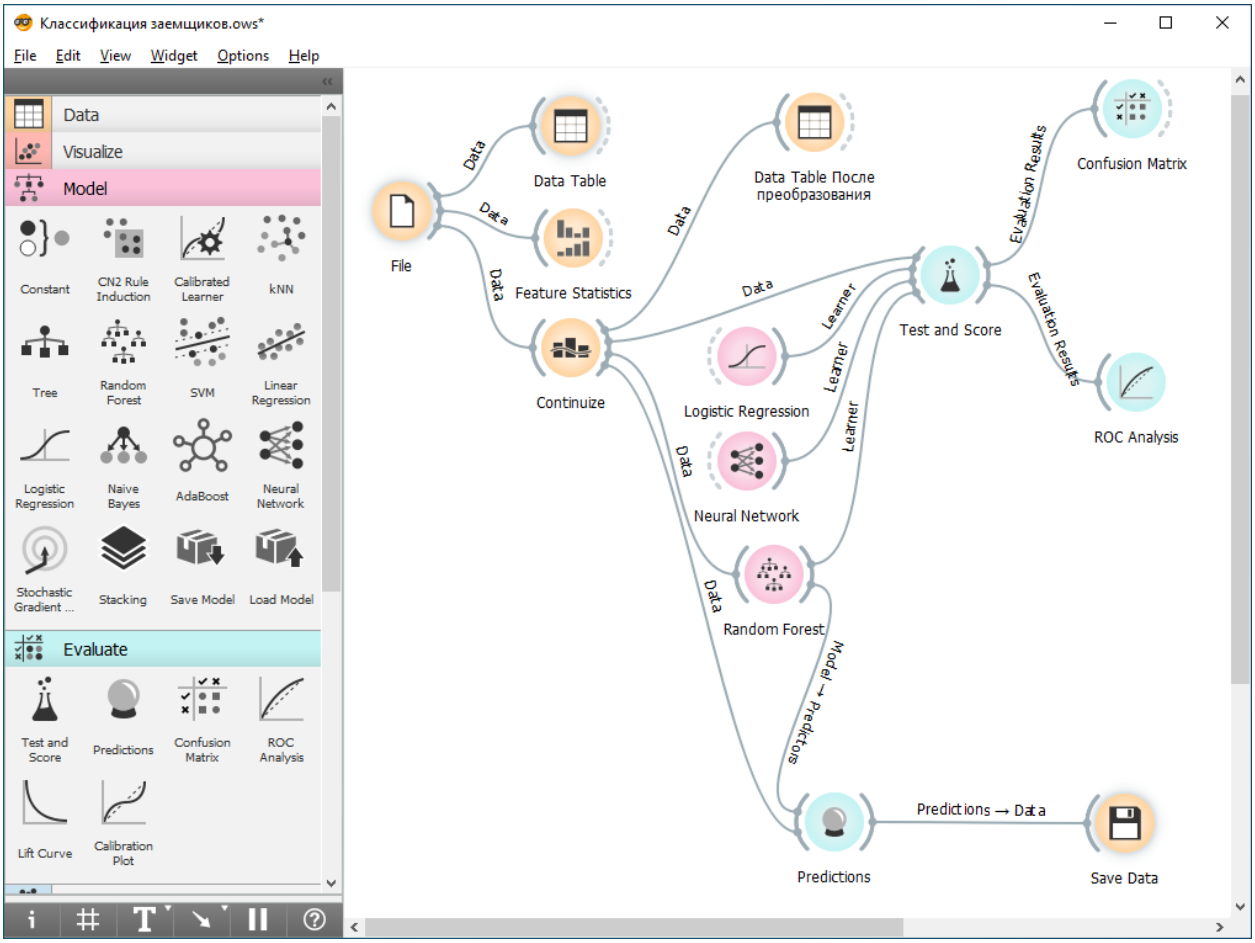


Рис. 1. Применение Orange для решения задачи классификации

Программу, заданную виджетами и потоками данных, можно сохранить и использовать для обработки других исходных данных.

Каждый виджет имеет набор параметров. Для указанного виджета двойной клик или команда «Open» открывает окно параметров виджета. Далее приведен список параметров для нейронной сети (рис. 2). Кроме специфических параметров переключатель «Apply Automatically» заставляет Orange после выполнения работы входных виджетов автоматически запускать обработку текущего. Это может приводить к цепочке запусков иногда ненужной обработки и задержкам времени, например, на этапе визуального программирования. Альтернативой является запуск обработки вручную кнопкой «Apply».

Иногда обработка виджета приводит к ошибке, в этом случае виджет помечается красным крестиком, а в свойствах виджета можно посмотреть сообщение об ошибке.

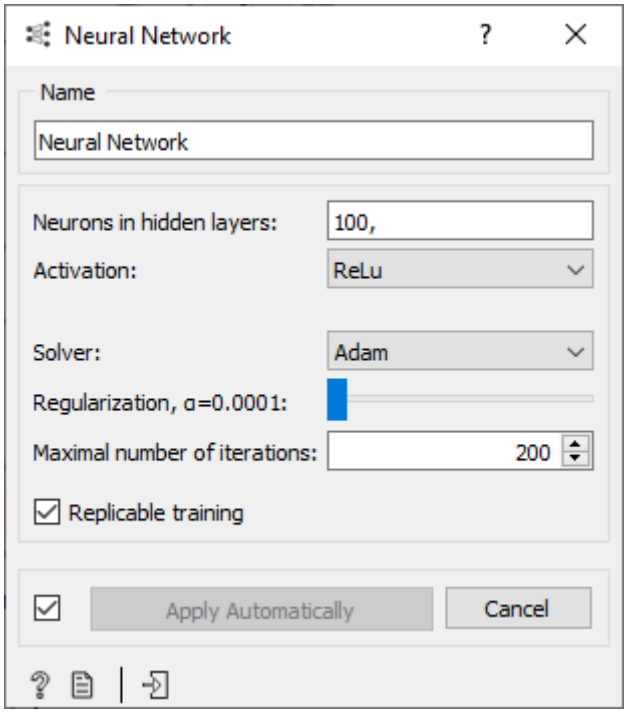


Рис. 2. Свойства виджета «Neural Network» – нейронная сеть

Потоки данных в Orange формируются перетаскиванием. Каждый виджет может иметь несколько входов (расположены слева) и выходов (расположены справа). Для изменения потока можно открыть окно двойным кликом по линии потока данных и в открывшемся окне (рис. 3) соединить выбранный выход одного виджета с выбранным входом другого. Неправильное соединение изображается пунктирной линией.

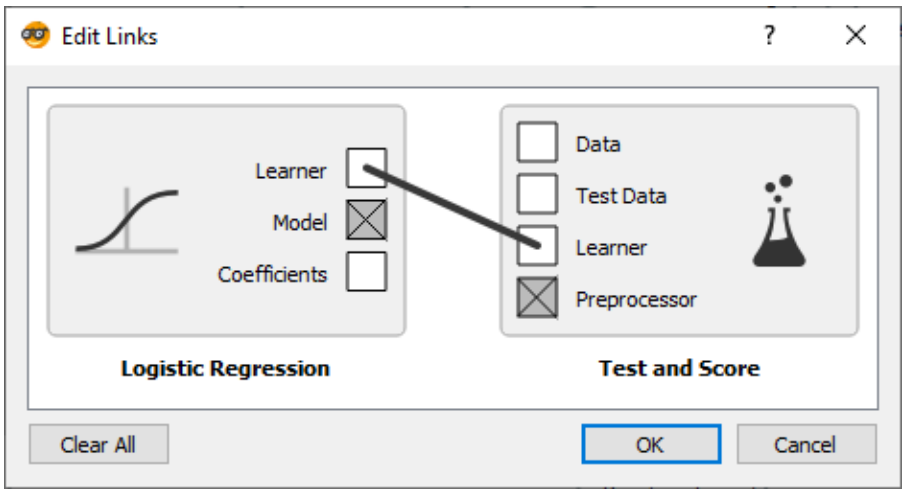


Рис. 3. Изменение потоков данных

2. Загрузка и исследование данных

2.1. Загрузка данных

Применение Orange начинается с загрузки данных с помощью виджета «File» (рис. 4). Загруженные табличные данные анализируются и получают определенный тип. В качестве примера рассмотрим данные о пассажирах Титаника[6] со следующими колонками:

– PassengerId – идентификатор пассажира;

– Survived – поле, в котором указано спасся человек (1) или нет (0);

– Pclass содержит социально-экономический статус (1 – высокий, 2 – средний, 3 – низкий);

– Name – имя пассажира;

– Sex – пол пассажира;

– Age – возраст;

– SibSp содержит информацию о количестве родственников 2-го порядка (муж, жена, братья, сестры);

– Parch содержит информацию о количестве родственников на борту 1-го порядка (мать, отец, дети);

– Ticket – номер билета;

– Fare – цена билета;

– Cabin – каюта;

– Embarked – порт посадки (C – Cherbourg, Q – Queenstown, S – Southampton).

[6] URL: https://habr.com/ru/company/mlclass/blog/270973.

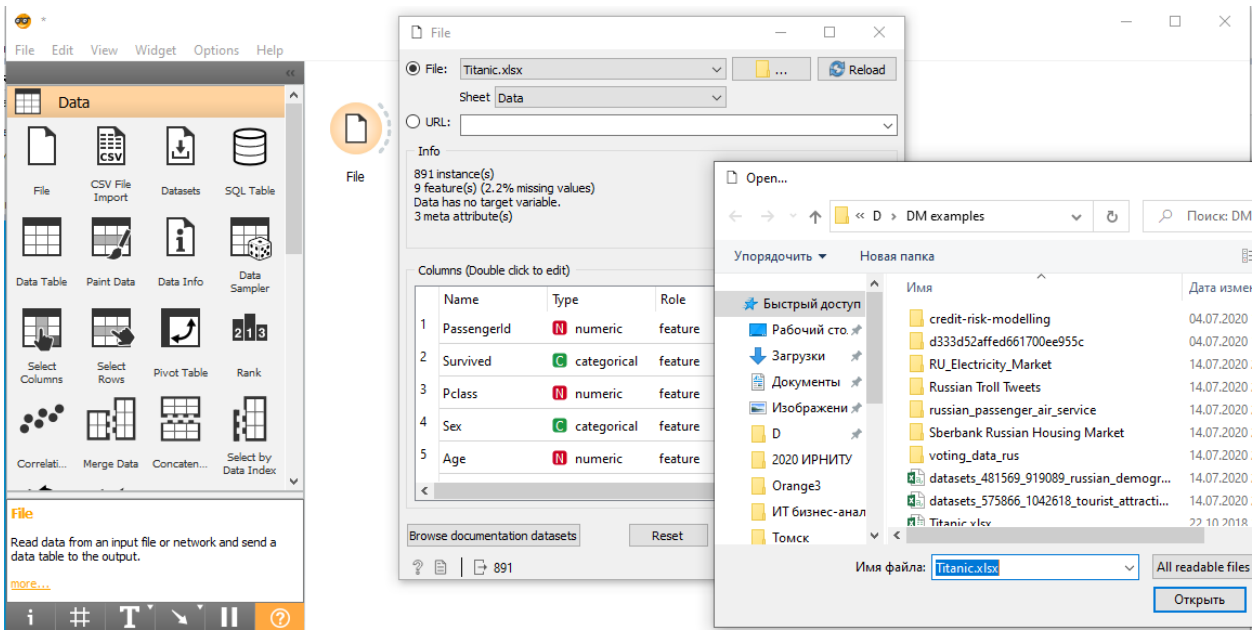


Рис. 4. Виджет «File» получения данных из файла

В результате загрузки создается таблица для каждой колонки которой определяется ее тип: Numeric – числовой, Categorical – категориальный, Datetime – дата – время, Text – текст. Тип можно изменить в зависимости от назначения колонки. Кроме этого каждой колонке можно указать ее роль в предстоящем анализе: Features – входная переменная, Targets – целевая переменная, влияние на которую входных переменных будет изучаться, Meta – описательная переменная, skip – переменная, которая будет проигнорирована.

2.2. Выбор переменных для анализа

Для выбора переменных для исследования зависимостей применяется виджет «Select Columns» (рис. 5). Он позволяет выбрать влияющие переменные (Features), целевые переменные (Target Variables) и атрибуты, служащие метками (Meta Attributes). В случае с Титаником интерес представляет зависимость спасения – переменная Survived от прочих переменных, за исключением PassengerId, Ticket, Cabin и Name.

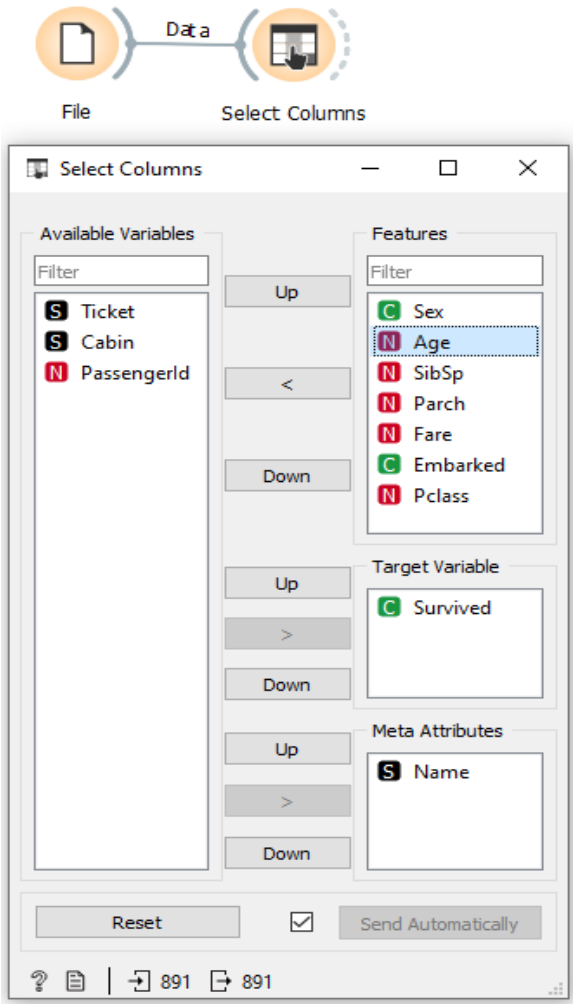


Рис. 5. Виджет «Select Columns» для выбора переменных

2.3. Статистические характеристики переменных

Для понимания свойств каждой переменной можно воспользоваться виджетом «Feature Statistics» (рис. 6). Виджет вычисляет статистические характеристики признаков: гистограммы для числовых значений или полигона для категориальных переменных, Center – математическое ожидание для числовых значений или мода для категориальных переменных, Dispersion – дисперсия для числовых значений или энтропии для категориальных переменных, минимума, максимума и количества пропусков. Переменная в поле «Color» определяет раскраску гистограммы, пропорционально долям различных значений выбранного поля. Например, выбор переменной «Survived» раскрасит каждый столбик пропорционально погибшим и выжившим в катастрофе. Анализируя диаграммы можно выявить влияние значений переменных на целевое значение. На рис. 6 демонстрируется, что класс каюты и пол существенно влияют на долю выживших.

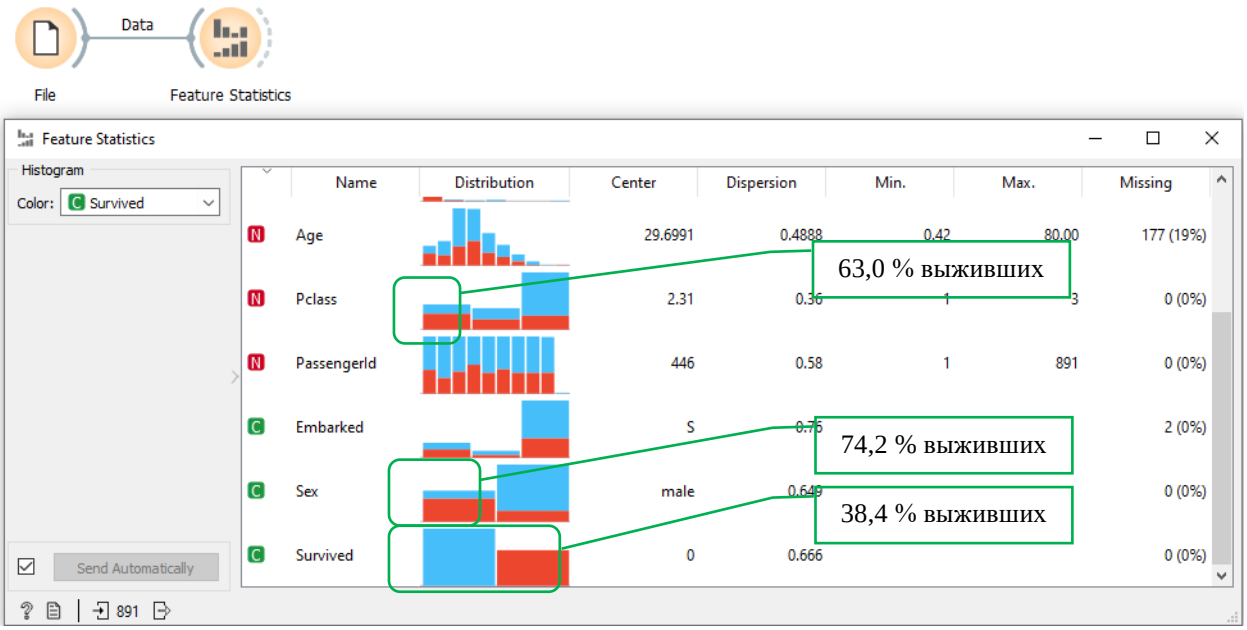


Рис. 6. Виджет «Feature Statistics» для вычисления статистических характеристик признаков

2.4. Распределения вероятностей переменных

Для каждой переменной может быть построено эмпирическое распределение вероятностей с помощью виджета «Distribution» (рис. 7). Для непрерывных переменных указать распределение (нормальное, Релея, экпоненциальное и др.) или применить универсальную оценку плотности с использованием ядер Гаусса (kernel density). Плотность распределения изображается линией и накладывается на гистограмму для визуальной оценки точности подбора. Распределение можно разделить на основе значения выбранного признака (параметр «Split by»). На рис. 7 разделение произведено на основе выживания в катастрофе. Разделение демонстрирует разницу, однако без ярко выраженного влияния.

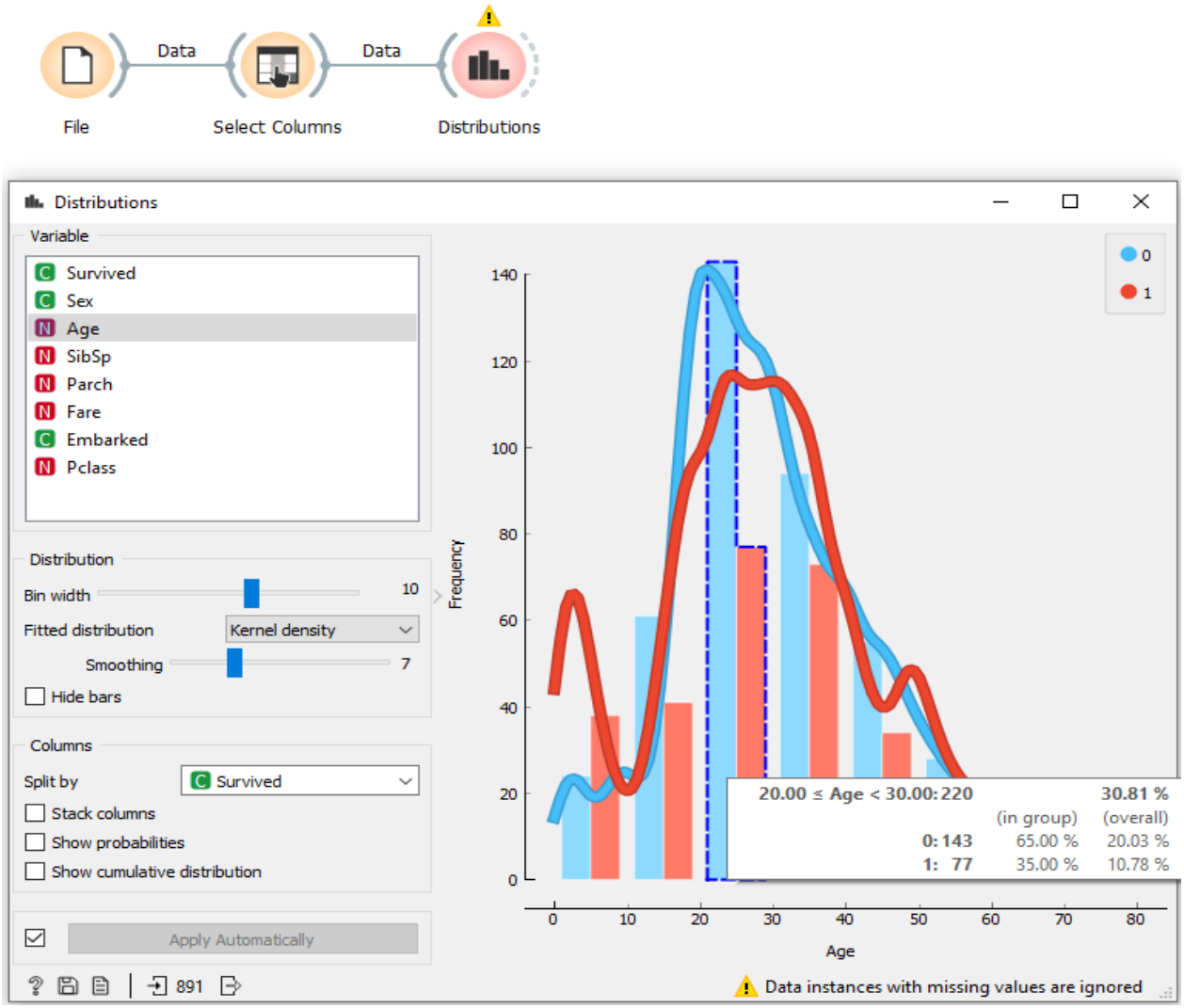


Рис. 7. Виджет «Distribution» для построения распределения вероятностей переменных

Предварительное изучение переменных позволяет выдвинуть ряд предположений о влиянии входных переменных на выходные переменные. На основе этих предположений можно переходить к формализации зависимостей. Однако неполные данные, ошибки в данных, выбросы могут существенно исказить зависимости. Поэтому исходные данные подвергают чистке и преобразованию.

2.5. Разведочный анализ данных на языке Python с помощью модуля Pandas

Визуальные средства Orange скрывают программную реализацию обработки данных. При необходимости разработки программного кода можно воспользоваться соответствующими модулями языка Python. Кроме этого модули предоставляют дополнительные модели и возможности.

Для решения задач исследования зависимостей можно применять модели библиотеки Scikit-Learn[7], которые используют специальную структуру

[7] URL: https://scikit-learn.org/stable/user_guide.html.

DataFrame[8] из модуля Pandas. DataFrame можно считать специальной таблицей, каждая строка которой указывается значением индекса, а каждый столбец содержит проиндексированные данные одного типа и описывается типом данных Series. DataFrame содержит множество аналитических методов и методов экспорта-импорта для различных форматов данных.

Команды

```
# Импорт модулей
import numpy as np
import pandas as pd
# Загрузка Excel таблицы в DataFrame
df_Titanic=pd.read_excel('D:\\Titanic.xlsx')
```

подключают нужные модули, загружают данные в структуру df_Titanic.

Команда

df_Titanic.info() выдает следующую информацию о полях:

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 891 entries, 0 to 890
Data columns (total 12 columns):
PassengerId    891 non-null int64
Survived       891 non-null int64
Pclass         891 non-null int64
Name           891 non-null object
Sex            891 non-null object
Age            714 non-null float64
SibSp          891 non-null int64
Parch          891 non-null int64
Ticket         891 non-null object
Fare           891 non-null float64
Cabin          204 non-null object
Embarked       889 non-null object
dtypes: float64(2), int64(5), object(5)
memory usage: 83.6+ KB.
```

В созданной структуре df_Titanic можно выделить числовые поля (int64, float64) и нечисловые – категориальные поля (object).

Метод df_Titanic.describe() позволяет получить основные статистические характеристики числовых полей (Таблица 1). Левая колонка содержит названия характеристик: count – количество непустых значений (возраст указан для 714 пассажиров из 891), mean – среднее (для индикатора спасения Survived среднее 0,38 соответствует доле спасшихся), std – среднеквадратическое отклонение, пять нижних строк характеризуют диапазоны значений. Например, для возраста минимум составляет 0,42 года, четвертая часть наименьших возрастов заканчивается возрастом 20,13 и т. д.

[8] URL: https://pandas.pydata.org/pandas-docs/stable/reference/api/pandas.DataFrame.html.

Таблица 1

Статистические характеристики числовых полей

Характеристики	PassengerId	Survived	Pclass	Age	SibSp	Parch	Fare
Count	891	891	891	714	891	891	891
Mean	446	0,38	2,31	29,70	0,52	0,38	32,20
Std	257,35	0,49	0,84	14,53	1,10	0,81	49,69
Min	1	0	1	0,42	0	0	0
25 %	223,5	0	2	20,13	0	0	7,91
50 %	446	0	3	28	0	0	14,45
75 %	668,5	1	3	38	1	0	31
Max	891	1	3	80	8	6	512,33

В табл. 1. содержатся характеристики числовых полей. Для категориальных полей можно воспользоваться той же функцией, предварительно выделив только категориальные поля:

categorical_columns = [c for c in df_Titanic.columns if df_Titanic[c].dtype.name == 'object'].

Команда

df_Titanic[categorical_columns].describe()

вернет описание категориальных полей (

Таблица 2), в которой для каждого поля будут приведены следующие характеристики:

– count – количество непустых значений (например, поле «Cabin» существенно неопределено, у 204 пассажиров это поле определено);

– unique – количество уникальных значений;

– top – значение с наибольшей частотой;

– freq – наибольшая абсолютная частота.

Для числовых полей можно построить гистограмму для понимания распределения значений. Метод df_Titanic[['Age']].hist() строит гистограмму для колонки «Age», представленную на рис. 8.

Таблица 2

Статистические характеристики категориальных полей

Характеристики	Name	Sex	Ticket	Cabin	Embarked
Count	891	891	891	204	889
Unique	891	2	681	147	3
Top	Thorneycroft, Mrs. Percival (Florence Kate White)	male	347 082	C23, C25, C27	S
Freq	1	577	7	4	644

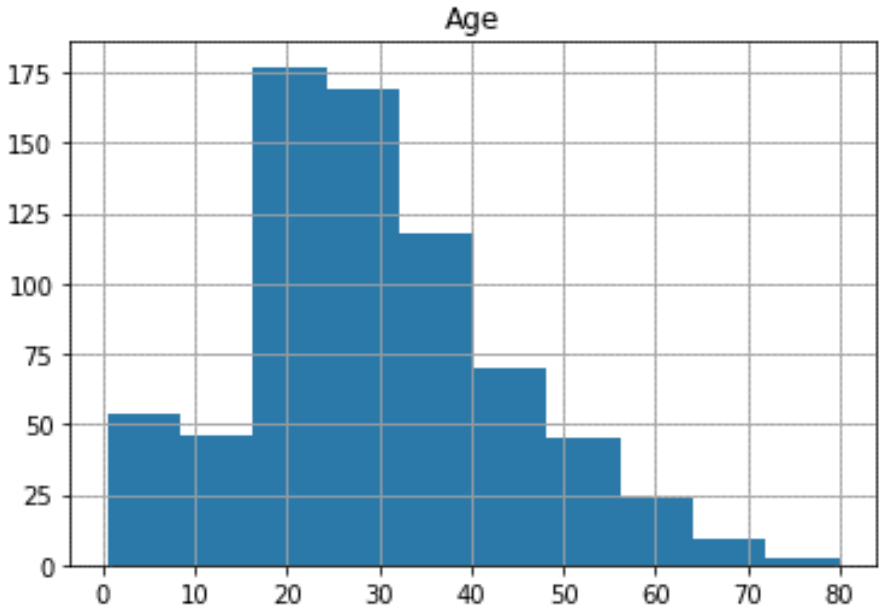


Рис. 8. Гистограмма значений поля возраст («Age»)

Линейную зависимость полей можно оценить с помощью коэффициента корреляции. Команда

df_Titanic[df_Titanic.columns[1:]].corr()

формирует корреляционную матрицу (табл. 3).

Таблица 3

Корреляционная матрица цифровых полей

	Survived	Pclass	Age	SibSp	Parch	Fare
Survived	1,000000	–0,338481	–0,077221	–0,035322	0,081629	0,257307
Pclass	–0,338481	1,000000	–0,369226	0,083081	0,018443	–0,549500
Age	–0,077221	–0,369226	1,000000	–0,308247	–0,189119	0,096067
SibSp	–0,035322	0,083081	–0,308247	1,000000	0,414838	0,159651
Parch	0,081629	0,018443	–0,189119	0,414838	1,000000	0,216225
Fare	0,257307	–0,549500	0,096067	0,159651	0,216225	1,000000

В данном случае можно констатировать отсутствие корреляционных зависимостей, так как модули всех коэффициентов, кроме диагональных, не превышают 0,55. Этот факт, в частности, подтверждает внешний вид облаков рассеивания. Команда

pd.plotting.scatter_matrix(df_Titanic[['Survived', 'Age', 'Sex_id']], alpha=0.7, figsize=(14,8))

строит облака рассеивания для попарных комбинаций полей «Survived», «Age», «Sex_id» (рис. 9). По диагонали размещены гистограммы полей.

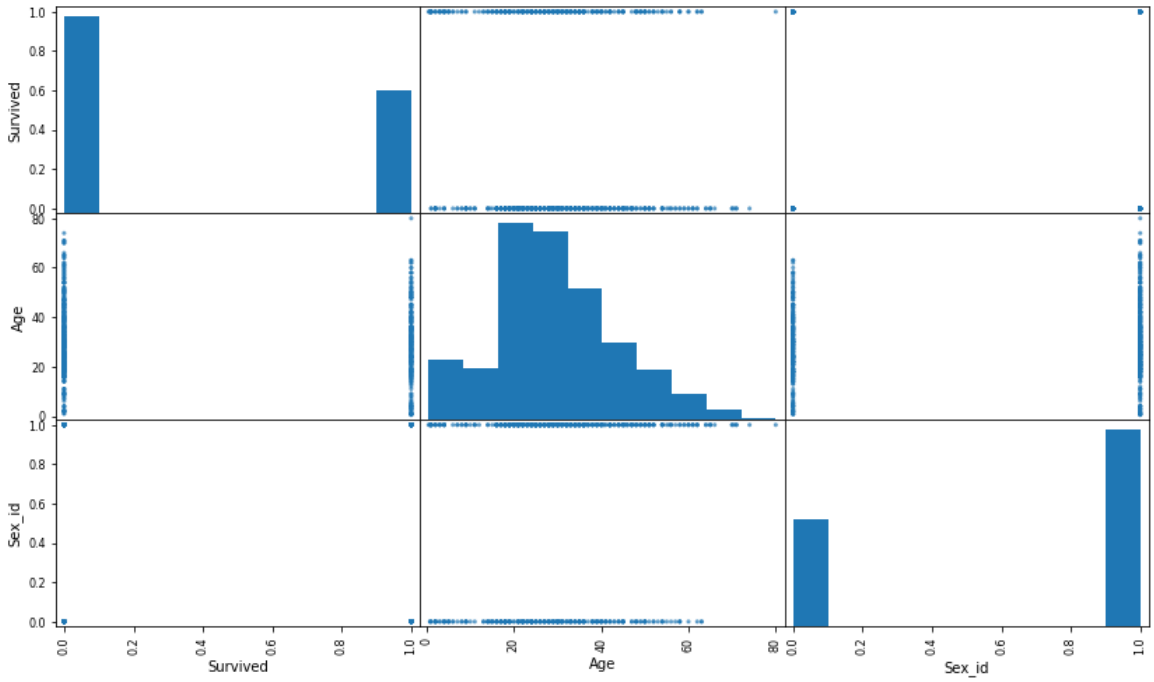


Рис. 9. Облака рассеивания для попарных комбинаций полей «Survived», «Age», «Sex_id»

Сводные таблицы являются универсальным инструментом разведочного анализа. Они позволяют находить сводные числовые характеристики для произвольной комбинации полей. Изучение таких характеристик позволяет выявить возможные зависимости и обосновать применение соответствующих моделей Data Mining.

Схема вызова метода pivot_table объекта DataFrame является следующей: DataFrame.pivot_table(values=, index=, columns=, aggfunc=, fill_value=, margins=, dropna=).

Команда

df_Titanic.pivot_table('Survived', 'Sex', 'Pclass')

передает колонку «Survived» в качестве значений, по которым будет вычислена функция агрегирования (по умолчантю – среднее), индексами строк будут уникальные значения колонки «Sex», заголовками колонок будут значения «Pclass». Таким образом, команда для каждой комбинации пола и класса каюты вернет (табл. 4) среднее значение частоты спасения (среднее поля «Survived»).

Три первых аргумента метода pivot_table могут быть списками. Например, команда

df_Titanic.pivot_table('Survived', 'Pclass', ['Sex','Embarked'])

вернет таблицу, в которой колонки будут представлены комбинацией полей «Sex» и «Embarked» (табл. 5).

Таблица 4

Сводная таблица для переменных «пол» и «класс каюты»

Pclass	1	2	3
Sex			
female	0,968085	0,921053	0,500000
male	0,368852	0,157407	0,135447

Таблица 5

Сводная таблица для комбинации полей

Sex	female			male		
Embarked	C	Q	S	C	Q	S
Pclass						
1	0,976744	1,000000	0,958333	0,404762	0,000000	0,354430
2	1,000000	1,000000	0,910448	0,200000	0,000000	0,154639
3	0,652174	0,727273	0,375000	0,232558	0,076923	0,128302

Параметр aggfunc может быть строкой, содержащей наименование функции агрегирования: *‘sum’*, *‘mean’*, *‘count’*, *‘min’*, *‘max’* и др., или словарем, в котором для поля указывается функция агрегирования (в этом случае параметр values игнорируется. Команда

```
df_Titanic.pivot_table(index='Sex', columns='Pclass', aggfunc={'Survived':sum,
'Age':'mean'})
```

вычислит для каждой комбинации значений 'Sex' и 'Pclass' количество выживших и средний возраст (табл. 6).

Таблица 6

Сводная таблица с выбором функции агрегирования

	Age			Survived		
Pclass	1	2	3	1	2	3
Sex						
female	34,611765	28,722973	21,750000	91	70	72
male	41,281386	30,740707	26,507589	45	17	47

Значение параметра margins=True укажет, что нужно вычислять промежуточные итоги. Команда

```
df_Titanic.pivot_table('Survived', ['Sex'], 'Pclass', aggfunc='mean', margins=True)
```

вычислит табл. 7 с итогами по сталбцам и строкам.

Таблица 7

Сводная таблица с итогами

Pclass	1	2	3	All
Sex				
female	0,968085	0,921053	0,500000	0,742038
male	0,368852	0,157407	0,135447	0,188908
All	0,629630	0,472826	0,242363	0,383838

Другой способ получения сводной информации заключается в применении группировки с последующим агрегированием. Команда

```
df_Titanic[['Survived', 'Pclass', 'Sex', 'Age']].groupby('Sex').aggregate(['count', np.median, max])
```

выбирает поля «Survived», «Pclass», «Sex», «Age», выполняет группировку по полу, а по всем остальным полям вычисляет для каждого количество, медиану и среднее.

3. Преобразование и очистка данных

Предварительная обработка данных является важным этапом аналитики [4]. Собранные в одну таблицу данные из разных источников могут отличаться по форматам и классификаторам, содержать ошибки и пропуски, измеряться в разных единицах и противоречить друг другу. Они могут быть излишне детальными или, наоборот, содержать агрегированные значения. Предварительную обработку рассматривают как отдельный этап анализа – ETL – Extract, Transform, Load – «извлечение, преобразование, загрузка».

3.1. Замена пропущенных значений

Самая распространенная ситуация, с которой приходится сталкиваться при подготовке данных для анализа – это неполные данные – наличие пропусков в определенных полях. Пропуски, интерпретируемые системой как нулевые значения, могут существенно «увеличить» расстояние между объектами. В примере с Титаником из 891 записи возраст не указан для 177 пассажиров (см. рис. 6). Для замены пропусков используется виджет «Impute» (рис. 10). Устанавливается замена по умолчанию и замены для каждой переменной:

– Don’t Impute – без преобразований;

– Average/Most-frequent – пропуски заменяются на средние значения для числовых переменных и на моду для категориальных переменных;

– As a distinct value – пропуск заменяется на новое значение;

– Model-based – замена выполняется на основании наиболее похожей записи (в случае с Титаником это может быть сосед по каюте, что не вполне оправдано);

– Random values – замена случайным значением из множества значений переменной;

– Remove examples – удаление записей с пустым значением переменной;

– Value – замена на указанное значение.

Две гистограммы (рис. 10) демонстрируют изменения в гистограмме возрастов после замены пропусков. Пропущенные значения заменены на средние значения. С одной стороны, после этого для каждого пассажира определен возраст и разница пассажиров по возрасту не будет иметь много выбросов, с другой – распределение возрастов получает серьезные искажения – количество центральных значений увеличивается на 177. Всегда при замене пустых значений исследователь должен решить, что больше исказит зависимости – наличие пустых значений или их замена.

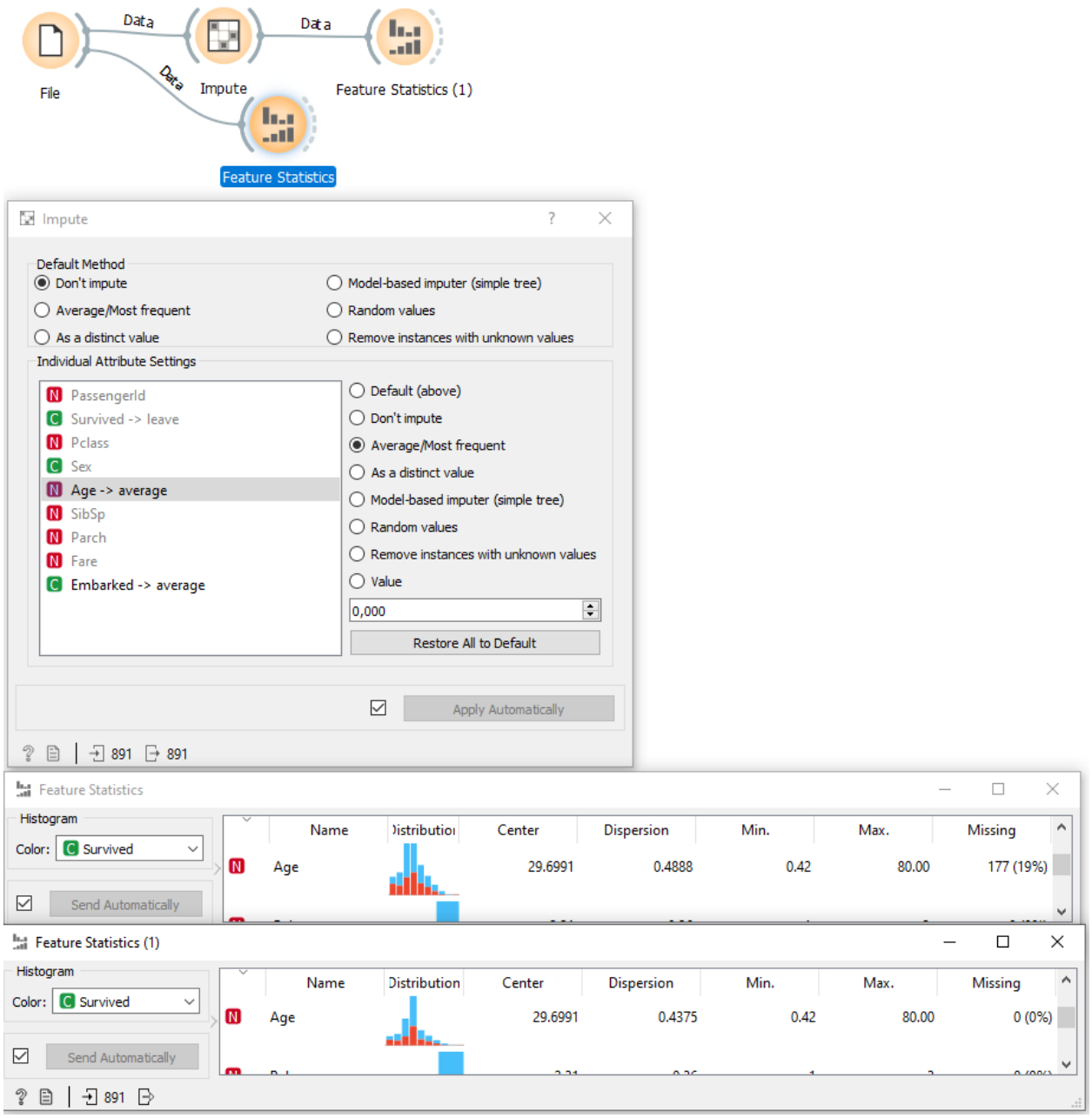


Рис. 10. Виджет «Impute» для замены пропусков

3.2. Квантование переменных

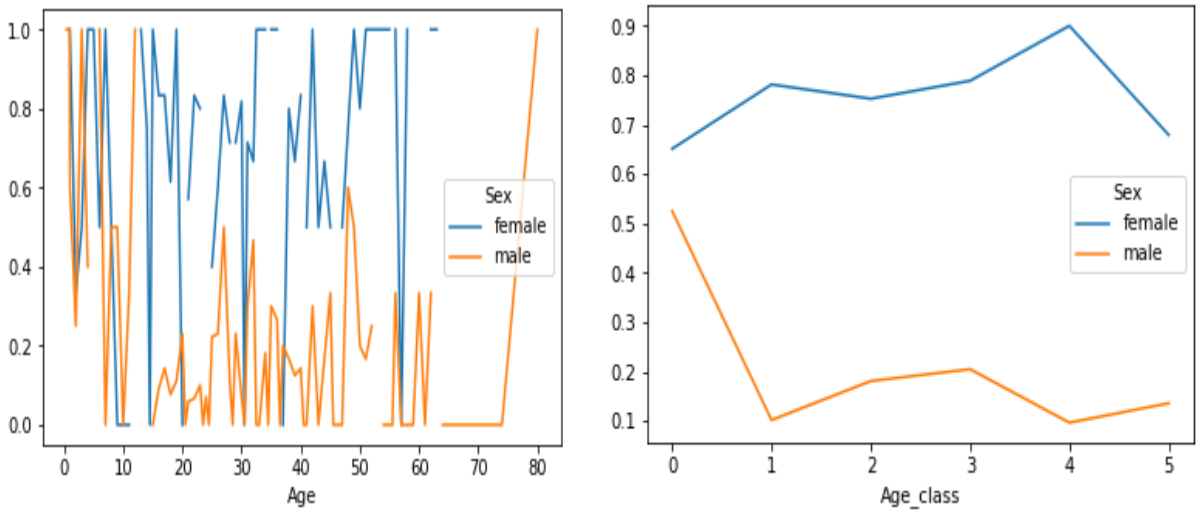
Квантование – еще одно преобразование – заключается в преобразовании одной шкалы в другую с меньшей размерностью. Квантование позволяет исключить случайные колебания, например, по графику доли выживших женщин и мужчин в зависимости от возраста (рис. 11, *а*) трудно выявить закономерности в отличии от графика доли выживших в зависимости от категории возраста (рис. 11, *б*).

Для автоматического квантования можно использовать виджет «Discretize» (рис. 12) с параметрами:

– Entropy-MDL – разбиение с максимизацией количества информации;

– Equal-frequency – разбиение на интервалы с одинаковой частотой попадания;

– Equal-width – разбиение на интервалы одинакового размера.



а *б*

Рис. 11. Зависимость выживания женщин и мужчин
от возраста и категории возраста

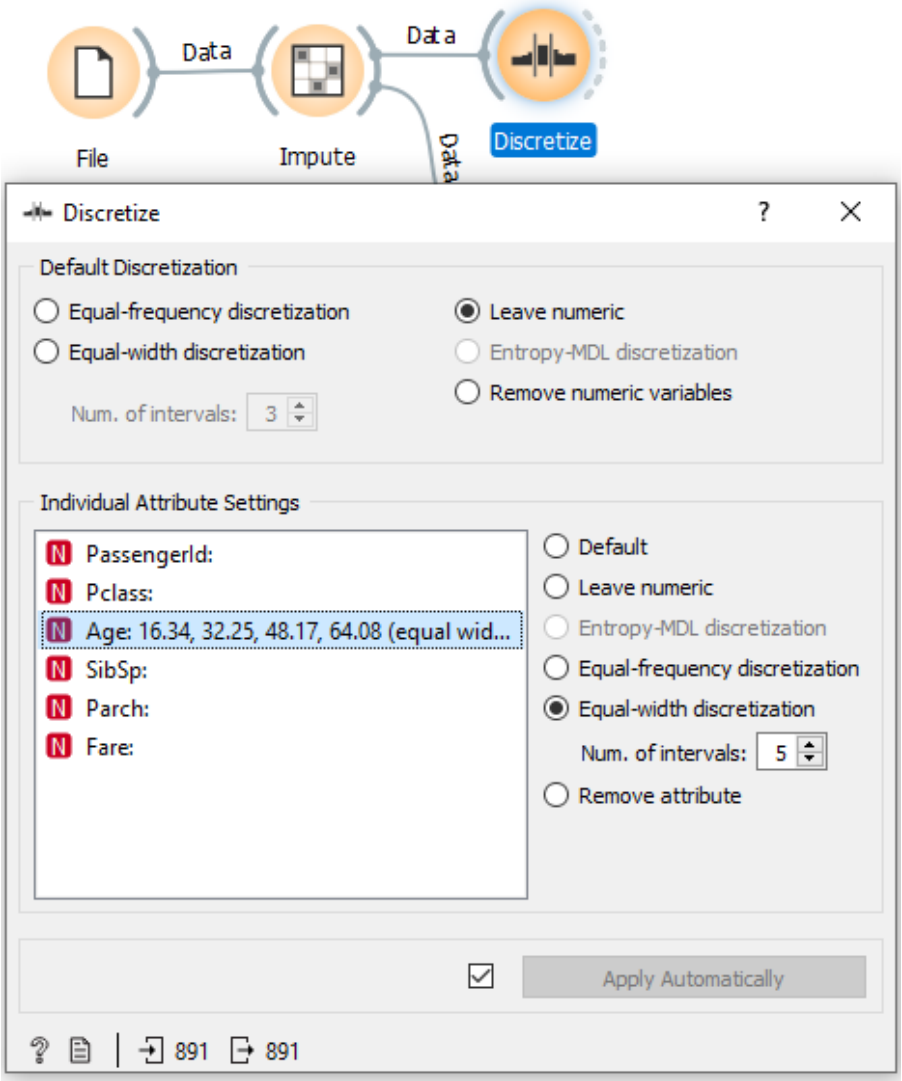


Рис. 12. Квантование возраста (Age) с помощью виджета «Discretize»

3.3. Масштабирование переменных

Масштабирование переменных необходимо в тех случаях, когда сравниваются расстояния между объектами. Если значения одного поля в десятки раз превышают значения другого поля, то влияние второго поля на расстояние между объектами оказывается незначительным по сравнению с первым. Масштабирование позволяет «уравнять» влияние переменных, измеренных в разных шкалах. Выполняется масштабирование виджетом «Continuize» (рис. 13).

Для категориальных атрибутов возможны следующие варианты:

1. First value as base. *N*-значная категориальная переменная $x \in \{x_1, \dots, x_N\}$ преобразуется в *N*-1 индикаторов $x = x_2, \dots, x = x_N$. Первое по порядку значение становиться базовым: если все индикаторы равны нулю, то $x = x_1$. Например, атрибут «Embarked» – порт посадки со значениями: C – Cherbourg, Q – Queenstown, S – Southampton будет заменен на «Embarked=Q» и «Embarked=S».

2. Most frequent value as base. Аналогичен предыдущему с одним отличием – в качестве базового значения выбирается то, которое встречается чаще всего. В данном варианте «Embarked» будет заменено на «Embarked=C» и «Embarked=Q».

3. One attribute per value. *N*-значная категориальная переменная $x \in \{x_1, \dots, x_N\}$ преобразуется в *N* индикаторов $x = x_1, \dots, x = x_N$.

4. Ignore multinomial attributes. Небинарные категориальные переменные исключаются.

5. Treat as ordinal. Ордиальное значение заменяется номером, начиная с нуля.

6. Divide by number of values. Значения предыдущнго варианта делятся на *N*-1.

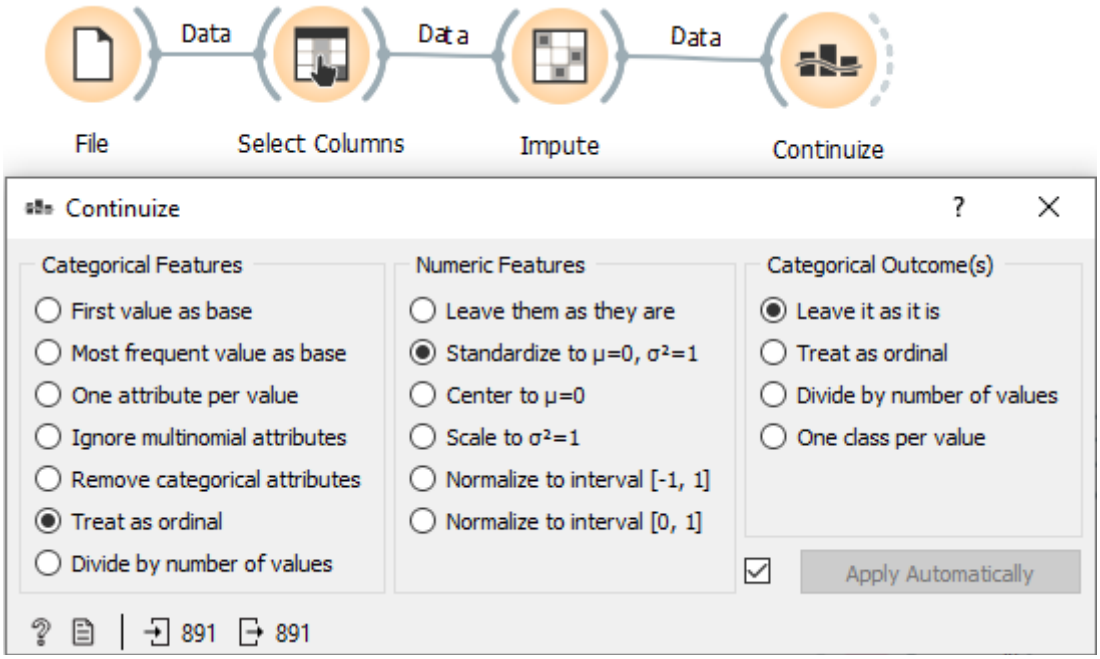


Рис. 13. Масштабирование с помощью виджета «Continuize»

Для непрерывных атрибутов можно выполнять следующие линейные преобразования:

– Leave them as they are – не преобразовывать;

– Standardize to … – преобразовать так, чтобы среднее было нулевым $\mu[x] = 0$, и сумма квадратов отклонений равнялась единице $\sigma[x] = 1$;

– Center to … – преобразовать так, чтобы среднее было нулевым $\mu[x] = 0$;

– Scale to … – преобразовать так, чтобы сумма квадратов отклонений равнялась единице $\sigma[x] = 1$;

– Normalize to interval [0, 1] – преобразование к интервалу [0, 1] $y = \big(x - \min(x)\big)/\big(\max(x) - \min(x)\big)$;

– Normalize to interval [–1, 1] – преобразование к интервалу [–1, 1].

Преобразование выходных переменных:

– Leave them as it is – не преобразовывать;

– Treat as ordinal – ординальное значение заменяется номером, начиная с нуля;

– Divide by number of values – ординальное значение заменяется номером, начиная с нуля, и делится на количество значений;

– One attribute per value – *N*-значная выходная переменная $x \in \{x_1, \dots, x_N\}$ преобразуется в *N* индикаторов $x = x_1, \dots, x = x_N$.

3.4. Редактирование типов переменных с заменой значений

Виджет «Edit Domain» позволяет выполнять специфическое преобразование для каждой переменной. На рис. 14 представлена замена оценок кодами с учетом порядка следования значений. Виджет также позволяет изменить тип переменной.

Для проведения анализа часто возникает задача преобразования переменных, которую можно решить с помощью виджета «Feature Constructor». На рис. 15 выполнено преобразование категориальных значений «Survived» и «Plcass» в числовые, проведена замена названий кодами для переменных «Sex» и «Embarked» и по возрасту «Age» вычислена его категория. Все выражения записываются в соответствии с требованиями языка Python. Этот инструмент дает наибольшие возможности по формированию категорий. Например, возрастные группы меняются с возрастом явно нелинейно.

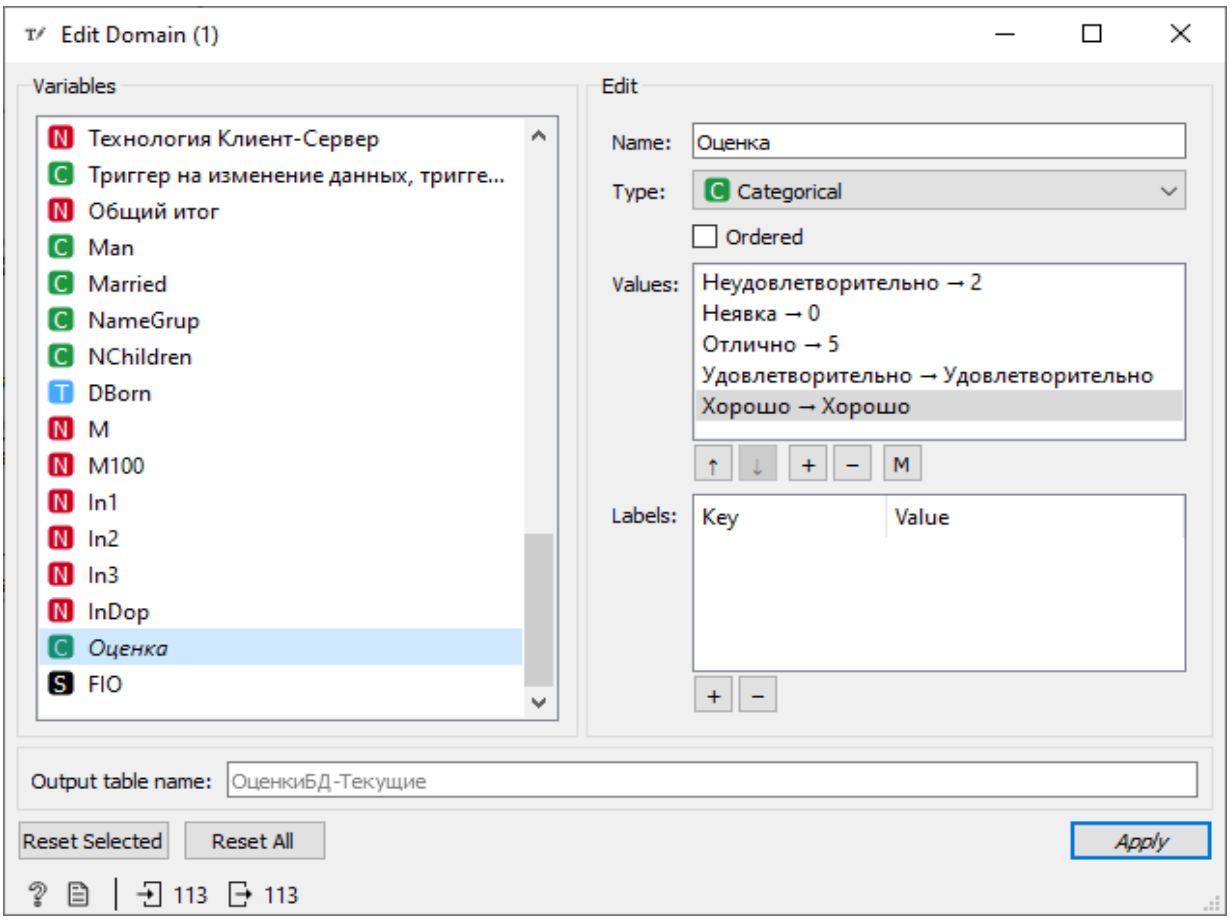


Рис. 14. Преобразование переменных с помощью виджета «Edit Domain»

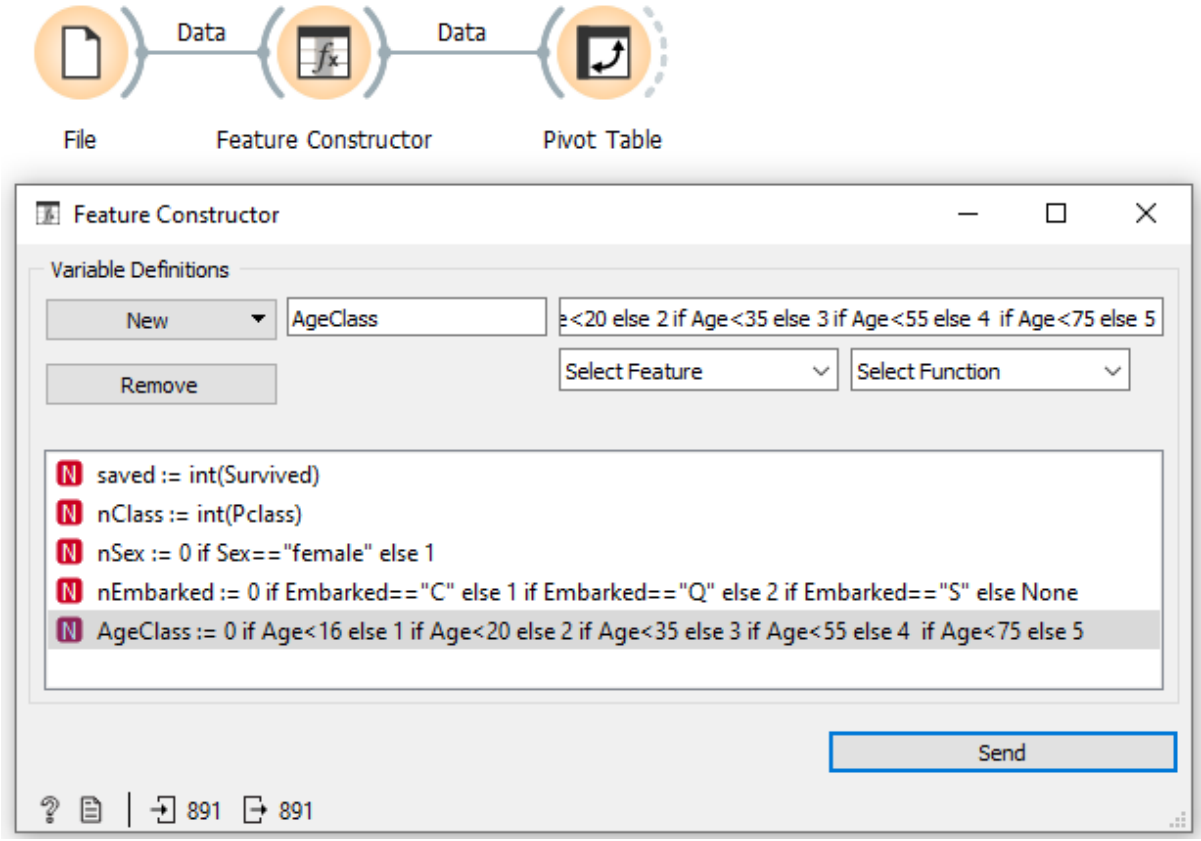


Рис. 15. Виджет «Feature Constructor» для преобразования переменных

3.5. Преобразование и очистка данных на языке Python

Все рассмотренные в данном разделе преобразования можно выполнить в Python-программах.

Замену пропусков можно выполнить, используя методы и свойства DataFrame. Например, для поля «Age» (возраст) вариантом замены является наиболее вероятное значение (мода)

```
x=df_Titanic['Age'].mode()
```

или среднее

```
x=df_Titanic['Age'].mean().
```

Замена пропусков на значение *x* выполняется командой

```
df_ Titanic ['Age'] = df_ Titanic ['Age'].fillna(x).
```

Замена категориальных значений числовыми выполняется специальным модулем LabelEncoder:

```
from sklearn.preprocessing import LabelEncoder # – подключаем модуль кодирования;
label = LabelEncoder() # – создаем объект для выполнения кодирования;
label.fit(df_Titanic['Sex'].drop_duplicates()) # – задаем список категорий для кодирования;
df_Titanic['Sex'+'_id'] = label.transform(df_Titanic['Sex']) # – заменяем категории кодами.
```

Квантование – преобразование одной шкалы в другую с меньшей размерностью можно выполнять простым преобразованием. Например, можно определить квантование возраста:

```
df_Titanic['Age_class'] = [0 if df_Titanic['Age'].iloc[i]<16 else
1 if df_Titanic['Age'].iloc[i]<20 else
2 if df_Titanic['Age'].iloc[i]<35 else
3 if df_Titanic['Age'].iloc[i]<55 else
4 if df_Titanic['Age'].iloc[i]<75 else
5
for i in range(0, len(df_Titanic))].
```

Для масштабирования значений полей применяются процедуры модуля sklearn.preprocessing. Ниже приведены примеры масштабирования[9].

Самым простым является линейное преобразование:

$$y = \big(x - \min(x)\big)/\big(\max(x) - \min(x)\big),$$

которое приводит все значения к интервалу [0, 1]. На рис. 16 приведены результаты линейного преобразования.

[9] URL: http://benalexkeen.com/feature-scaling-with-scikit-learn.

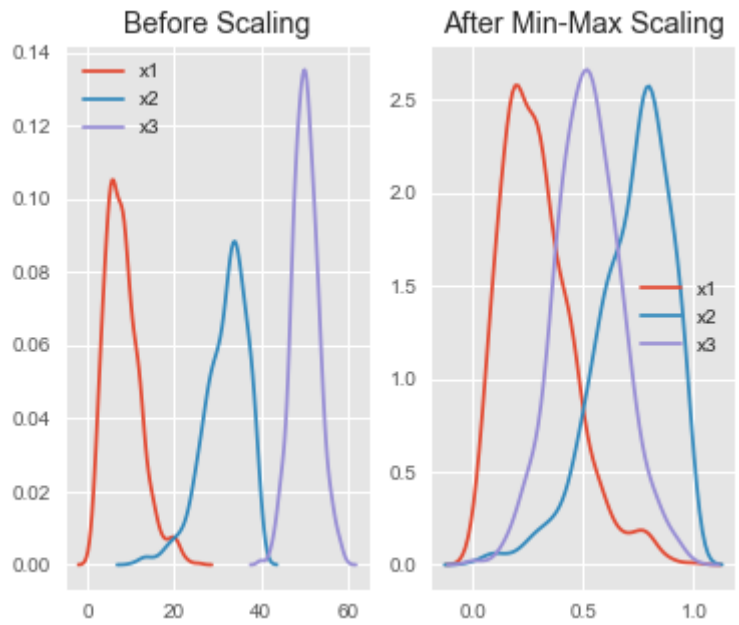


Рис. 16. Пример линейного преобразования

Другой вариант масштабирования выполняют (рис. 17Рис. 17), если требуется, чтобы значения поля соответствовали стандартному нормальному распределению с математическим ожиданием 0 и среднеквадратическим отклонением 1:

$$y = (x - M[x])/(\sigma[x]),$$

где $M[x]$ и $\sigma[x]$ – оценки математического ожидания среднеквадратического отклонения поля, определенные по наблюдениям.

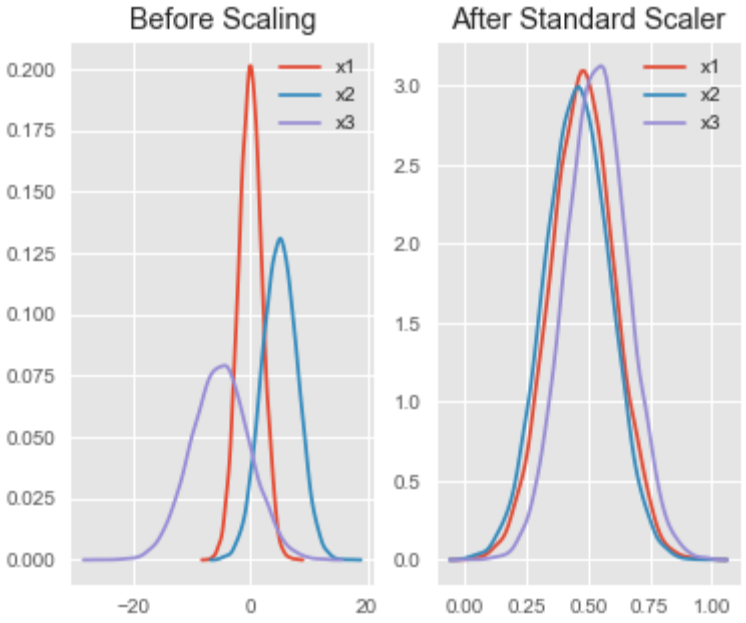


Рис. 17. Пример преобразования к стандартному нормальному распределению

Робастное преобразование применяют, если распределение значений является несимметричным и имеет выбросы в положительную или отрицательную сторону, которые нужно нивелировать:

$$y = (x - Q_1[x])/(Q_3[x] - Q_1[x]),$$

где $Q_1[x]$ – квантиль первой четверти (значения меньшие $Q_1[x]$ составляют четвертую часть всех наблюдений); $Q_3[x]$ – квантиль третьей четверти (значения большие $Q_3[x]$ составляют четвертую часть всех наблюдений).

Интервальное преобразование сохраняет сдвиг двух рядов, а робастное преобразование его исключает (рис. 18), за счет игнорирования выбросов в первую и четвертую четверти значений.

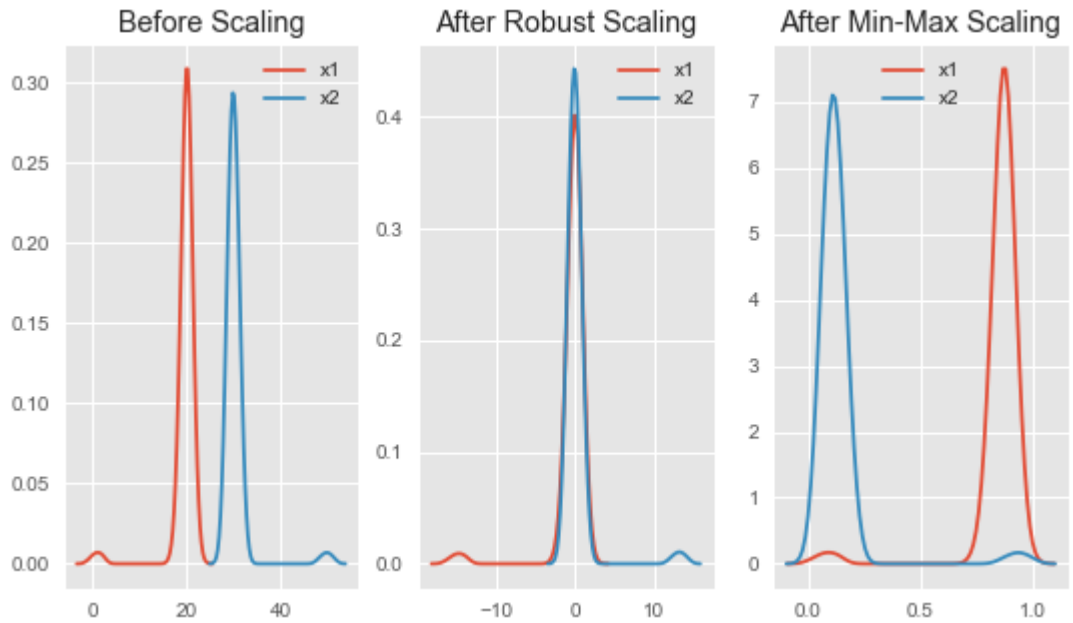


Рис. 1818. Сравнение преобразования к интервалу [0, 1]
и робастного преобразования

Ниже приведены примеры команд преобразования возраста:

```
from sklearn.preprocessing  import MinMaxScaler
df_Titanic['MinMaxScaled_Age'] = MinMaxScaler().fit_transform(df_Titanic[['Age']])
from sklearn.preprocessing  import StandardScaler
df_Titanic['StandardScaled_Age'] = StandardScaler().fit_transform(df_Titanic[['Age']])
from sklearn.preprocessing  import RobustScaler
df_Titanic['RobustScaled_Age'] = RobustScaler().fit_transform(df_Titanic[['Age']]).
```

4. Изучение влияния на выходные переменные

4.1. Применение сводных таблиц (многомерный анализ данных)

Влияние входных переменных на целевые можно определить с помощью сводных таблиц – виджет «Pivot Table» (рис. 19). Для сводной таблицы могут определяться следующие параметры:

1. В поле «Rows» выбирается дискретная или числовая переменная для обозначения строк.

2. В поле «Columns» выбирается дискретная переменная для обозначения столбцов.

3. В поле «Values» выбирается переменная для агрегирования.

4. Методы агрегирования выбираются переключателями:

– *Count* – количество;

– *Count defined* – количество непустых значений.

Для числовых переменных используются следующие переключатели:

– *Sum* – сумма;

– *Mean* – среднее;

– *Mode* – мода – значение с наибольшей частотой;

– *Min* – минимальное значение;

– *Max* – максимальное значение;

– *Median* – медиана – частота меньших и больших значения одинакова и равна 0,5;

– *Var* – дисперсия.

Для дискретных переменных применяется один переключатель: *Majority* – значение с набольшей частотой.

На рис. 19 представлена зависимость количества (Count) и доли выживших (вычисляется как среднее – Mean) в зависимости от класса каюты (Pclass) и пола (sex).

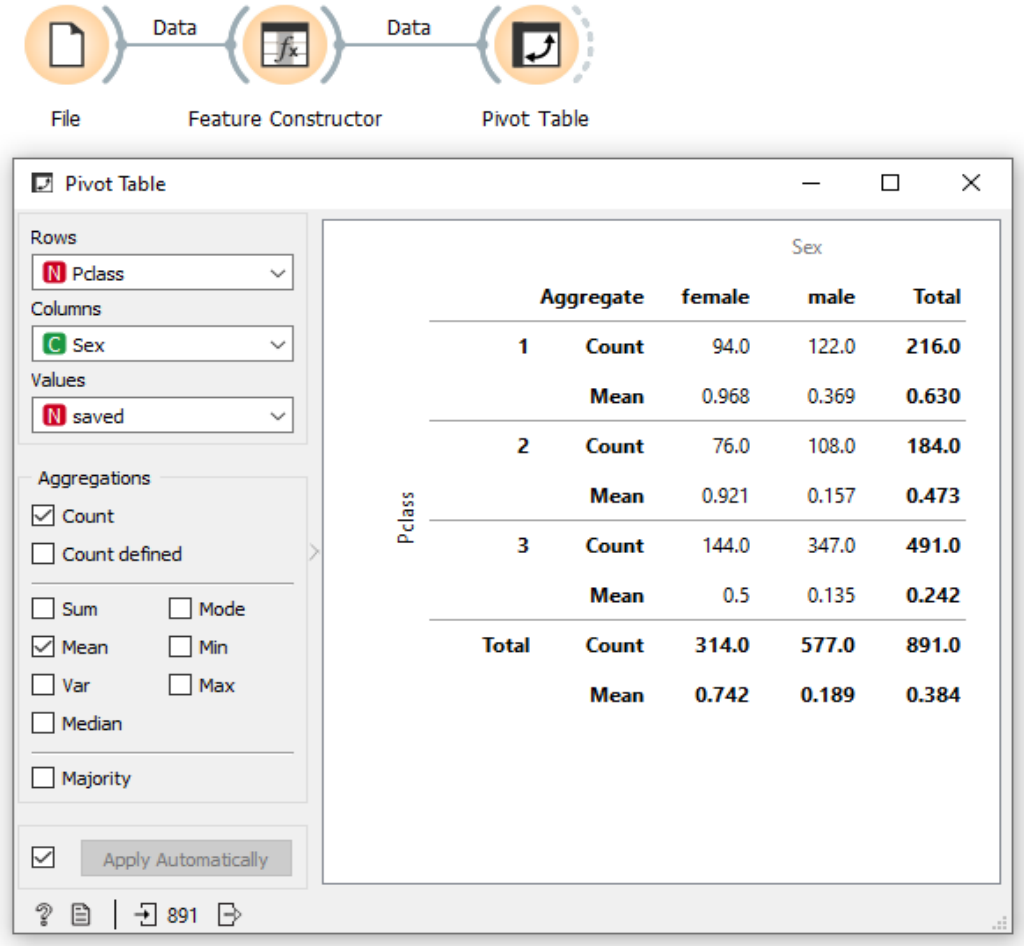


Рис. 19. Виджет «Pivot Table» построения сводной таблицы для исследования зависимости доли выживших от пола и класса каюты

Далее представлена сводная таблица для исследования доли выживших в зависимости от возрастной группы (рис. 20).

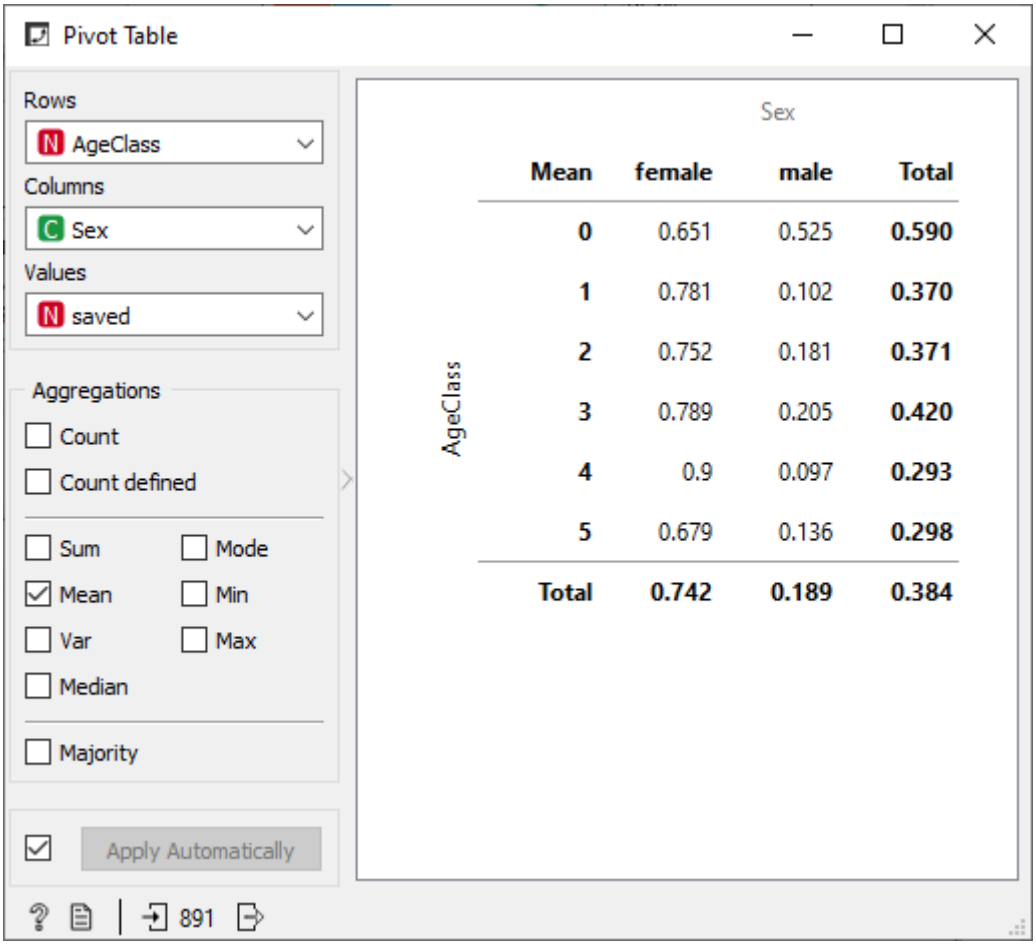


Рис. 19. Виджет «Pivot Table» построения сводной таблицы для исследования зависимости доли выживших от пола и категории возраста

4.2. Коэффициенты корреляции

Другим полезным инструментом анализа данных является изучение попарных коэффициентов корреляции (рис. 21). Коэффициент корреляции характеризует степень линейной зависимости двух случайных переменных. Чем ближе коэффициент корреляции к 1 или к –1 тем сильнее проявляется прямая или обратная линейная зависимость. Крайние значения свидетельствуют о детерминированной зависимости. На рис. 21 такая зависимость связывает переменные pClass и nClass, значения которых просто совпадают, а отличаются только типы переменных. Сильная зависимость возраста (Age) и категории возраста (AgeClass) также появилась в результате замены возрата категорией возраста. Отрицательная зависимость платы за проезд (Fare) и класса каюты (pClass) также вполне очевидна – чем меньше номер класса, тем больше плата. Корреляционная связь пола (Sex) и выживания (Survived) свидетельствует о стремлении сохранить женщин во время катастрофы. Прочие зависимости следует признать малозначимыми. Более глубокий анализ позволит выявить комбинации данных, дающие более точные зависимости.

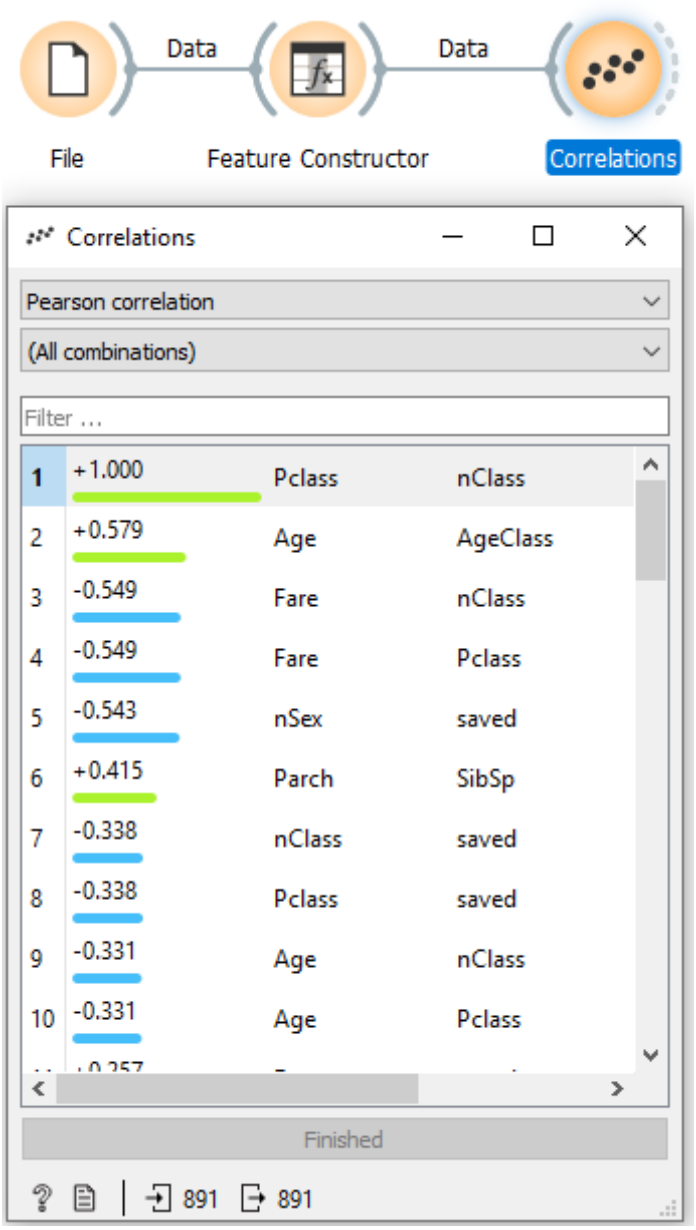


Рис. 20. Виджет «Correlations» вычисления попарных коэффициентов корреляции переменных

4.3. Преобразование пространства переменных

4.3.1. Метод главных компонент

Описанные выше преобразования связаны одной переменной. Дополнительную информацию можно получить, изучая совместное изменение нескольких переменных. К таким методам относится метод главных компонент [4] (Principal Component Analysis, PCA). Идея метода в линейном преобразовании координат, так чтобы изменения вдоль новых осей были максимальными (рис. 22). Оси упорядочиваются по убыванию «изменчивости» вдоль оси. Предполагается, что для выявления зависимостей достаточно нескольких первых осей. Очевидно, что данный метод будет хорошо работать для линейных зависимостей (или близким к линейным зависимостям). Признаком сильных линейных

зависимостей является близость коэффициента корреляции к 1 (положительная линейная зависимость) или к –1 (отрицательная линейная зависимость).

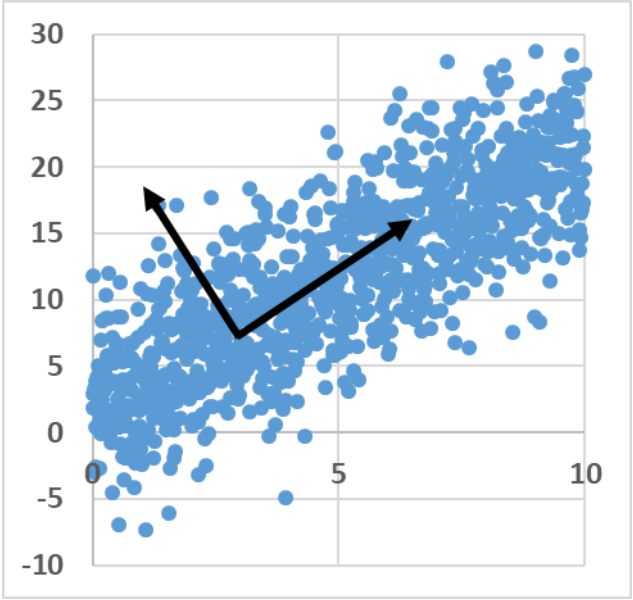


Рис. 21. Преобразование системы координат методом главных компонент

На рис. 23Рис. 22 продемонстрировано применение виджета «Principal Component Analysis». В параметрах выбрано три компоненты, на долю которых приходится 59 % изменчивости. Графики демонстрируют изменение доли суммарной изменчивости (зеленая линия), доли изменчивости последней выбранной компоненты (красная линия) при изменении количества главных компонент.

Можно посмотреть параметры линейного преобразования: как определяются главные компоненты через исходные переменные (рис. 24) и преобразованные данные (рис. 25). Диаграмма разброса по двум первым компонентам (рис. 26) не позволяет сделать однозначные выводы о зависимости спасения от значения главных компонент.

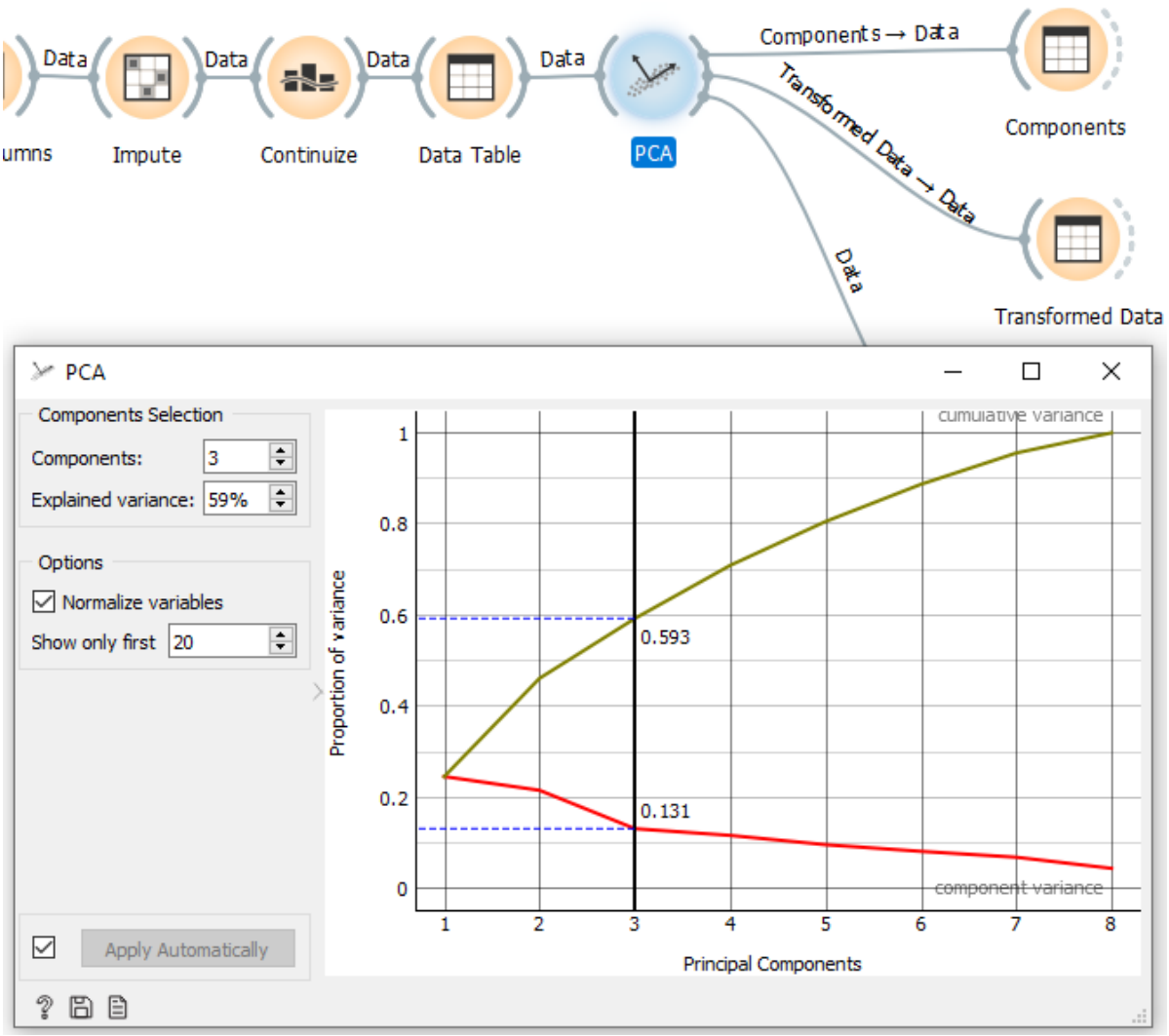


Рис. 22. Преобразование координат методом главных компонет с применением виджета «Principal Component Analysis»

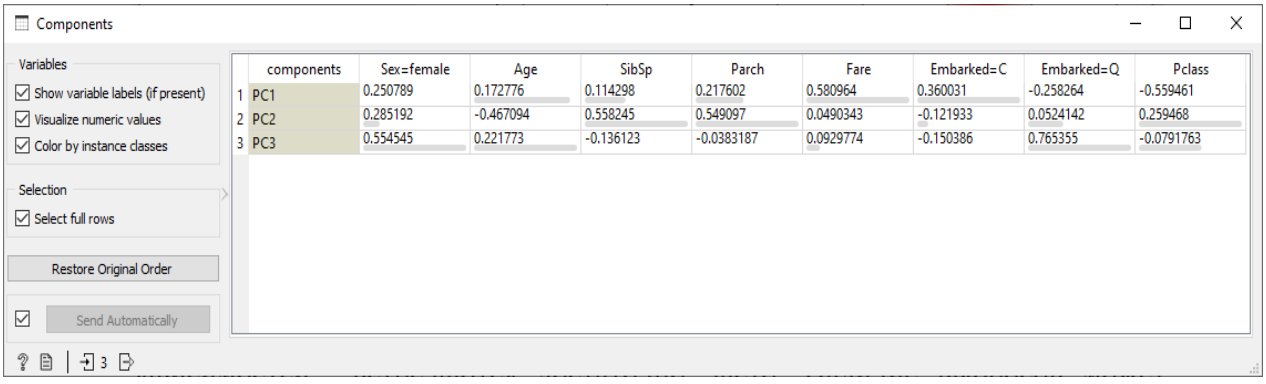
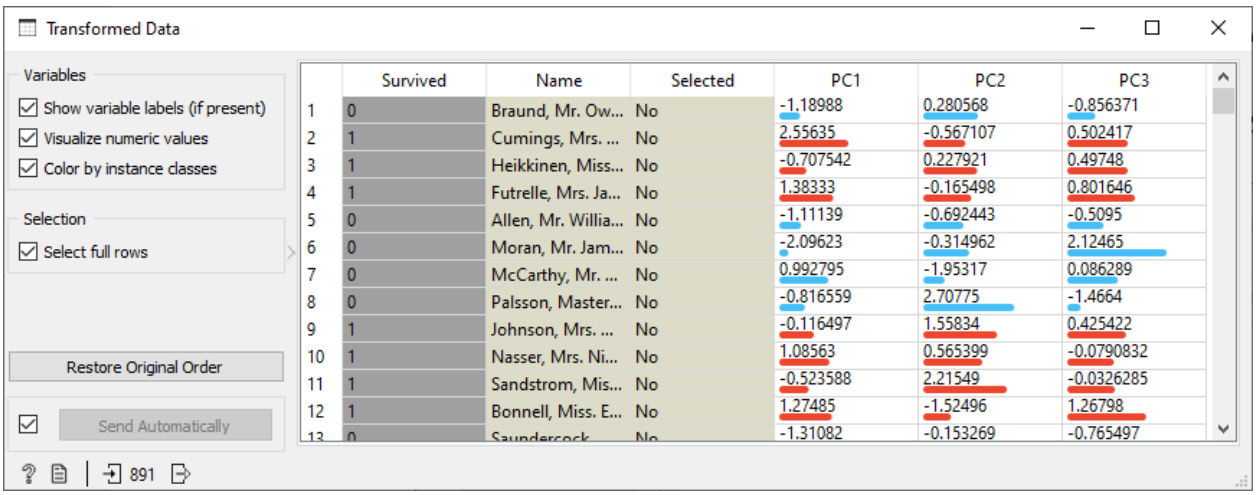


Рис. 23. Параметры линейного преобразования



	Survived	Name	Selected	PC1	PC2	PC3
1	0	Braund, Mr. Ow...	No	-1.18988	0.280568	-0.856371
2	1	Cumings, Mrs. ...	No	2.55635	-0.567107	0.502417
3	1	Heikkinen, Miss...	No	-0.707542	0.227921	0.49748
4	1	Futrelle, Mrs. Ja...	No	1.38333	-0.165498	0.801646
5	0	Allen, Mr. Willia...	No	-1.11139	-0.692443	-0.5095
6	0	Moran, Mr. Jam...	No	-2.09623	-0.314962	2.12465
7	0	McCarthy, Mr. ...	No	0.992795	-1.95317	0.086289
8	0	Palsson, Master...	No	-0.816559	2.70775	-1.4664
9	1	Johnson, Mrs. ...	No	-0.116497	1.55834	0.425422
10	1	Nasser, Mrs. Ni...	No	1.08563	0.565399	-0.0790832
11	1	Sandstrom, Mis...	No	-0.523588	2.21549	-0.0326285
12	1	Bonnell, Miss. E...	No	1.27485	-1.52496	1.26798
13	0	Saundercock...	No	-1.31082	-0.153269	-0.765497

Рис. 24. Результат линейного преобразования

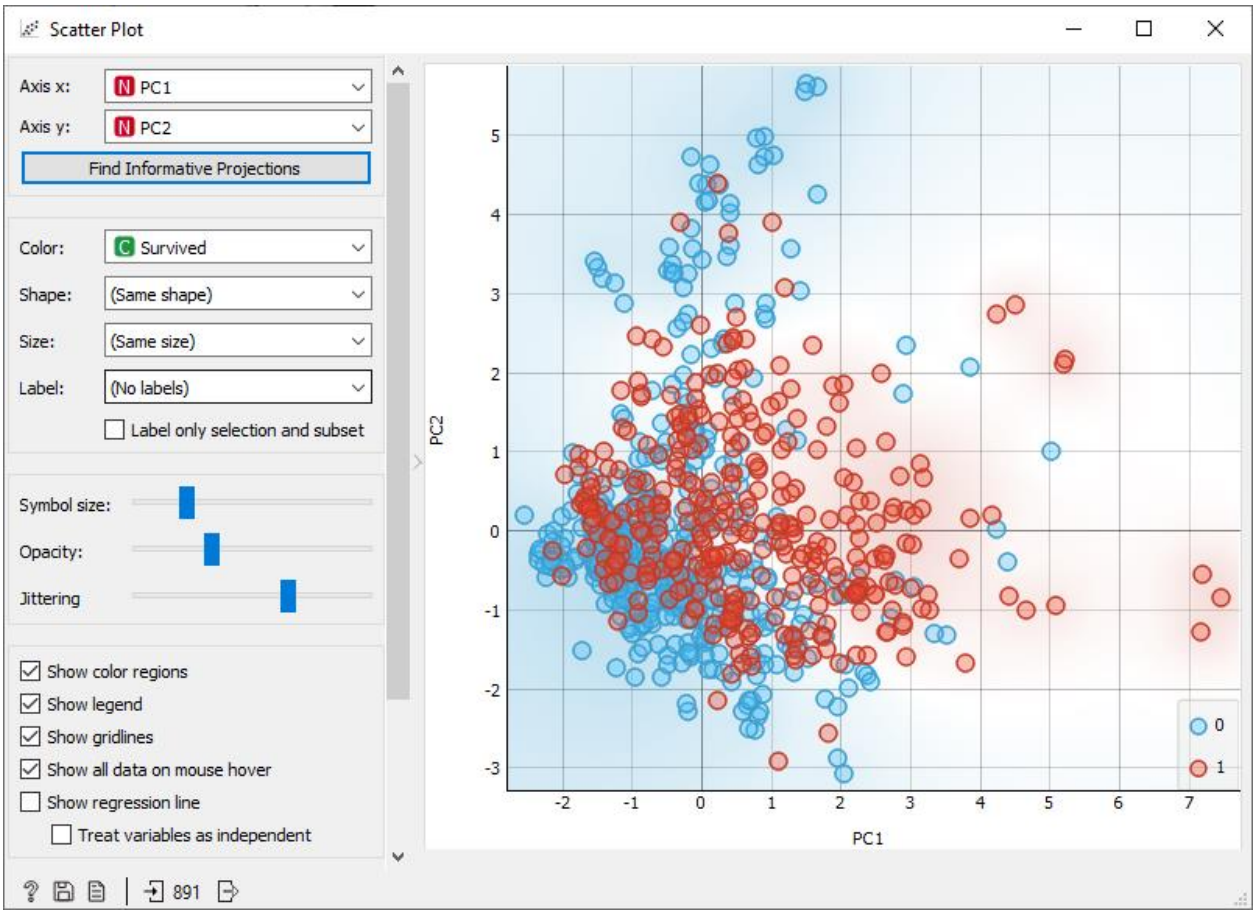


Рис. 25. Диаграмма разброса (Scatter plot) по двум главным компонентам

4.3.2. Снижение размерности алгоритмом t-SNE

Модель t-SNE (t-distributed stochastic neighbor embedding) нелинейного снижения размерности оценивает параметры t-распределения Стьюдента в исходном многомерном пространстве и подбирает аналогичное распределение в

двумерном (или трехмерном) распределении[10]. Виджет t-SNE добавляет две новые координаты (рис. 27) и распределение выживших выглядит более определенным.

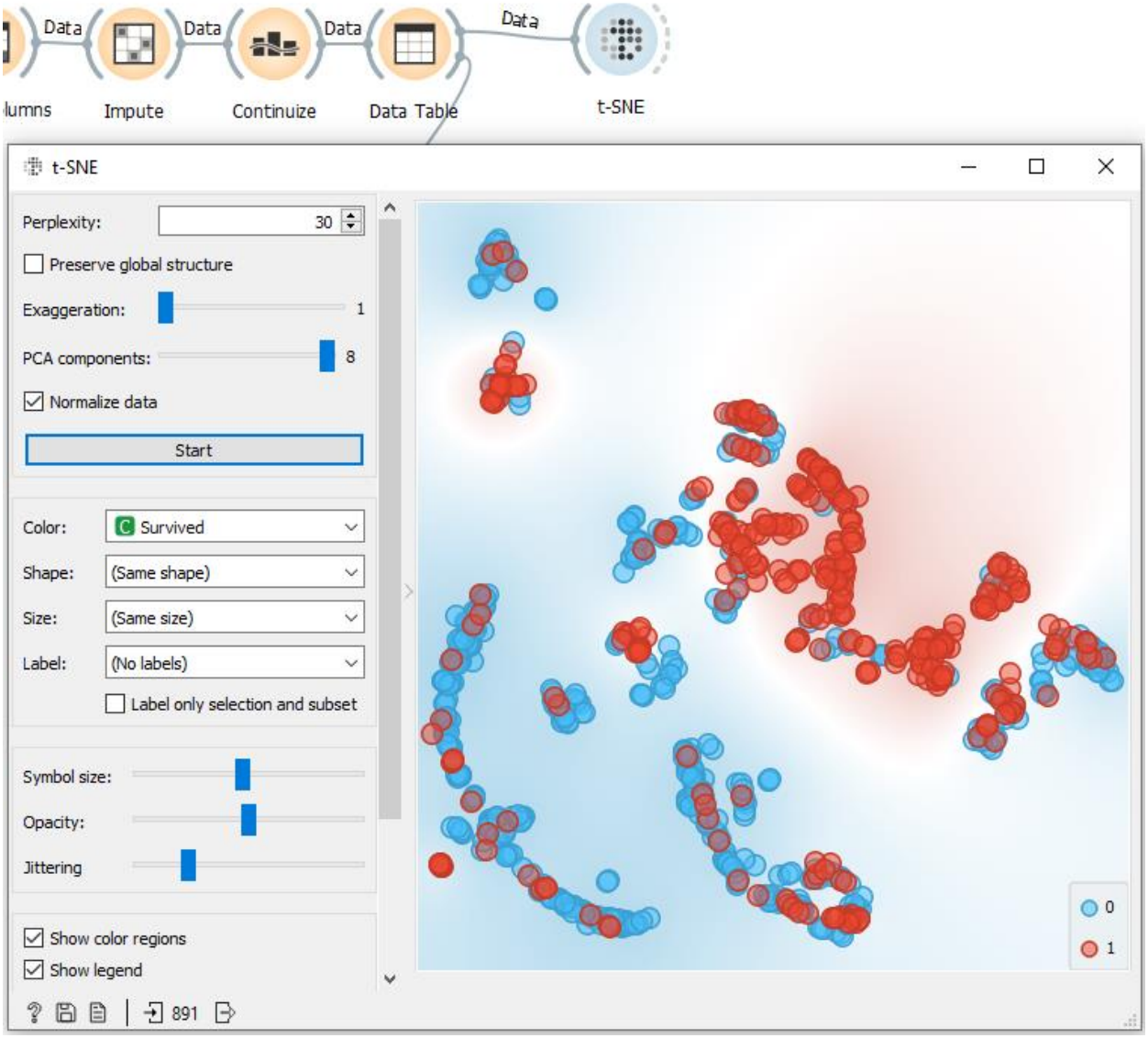


Рис. 26. Преобразование координат с применением виджета t-SNE

5. Методы классификации

5.1. Задача классификации

Задача классификации может быть сформулирована следующим образом [4]. Задано множество объектов $X = \{x_1, \ldots, x_n\}$ и множество классов (названий) Y. Каждый объект x_i характеризуется набором атрибутов $(x_{i1}, x_{i2}, \ldots x_{im})$. Требуется построить алгоритм классификации $X \to Y$ по данным наблюдений $(x_1(x_{11},\ldots,x_{1m}), y_1),\ldots,(x_n(x_{n1},\ldots,x_{nm}), y_n)$.

[10] Хмельков И. Препарируем t-SNE. URL: https://habr.com/ru/post/267041.

Например, в банке для каждого заемщика устанавливают набор таких атрибутов, как доход, семейное положение, владение недвижимостью и т. д. В результате кредитных взаимоотношений по так называемой кредитной истории каждый заемщик получает класс «платежеспособный» или «неплатежеспособный». Класс определяется закономерностями и случайными обстоятельствами (даже самый платежеспособный заемщик может не вернуть кредит под воздействием комбинации неблагоприятных факторов). Таким образом, класс может быть установлен с определенной долей вероятности. Причем, эта вероятность зависит от атрибутов заемщика. В задаче классификации необходимо не только определить принадлежность объекта классу по набору атрибутов, но и оценить вероятность такой классификации.

Предположим, что по данным наблюдений, доля «неплатежеспособных» заемщиков составляет 10 %. Значение некоторого атрибута или комбинация значений атрибутов может существенно повлиять на эту долю, увеличивает или уменьшает вероятность отнесения заемщика к тому или иному классу. Знание таких закономерностей позволяет более успешно решать многие задачи. Если такого влияния атрибутов на платежеспособность нет, то решение задачи классификации не будет лучше выбора наугад.

5.2. Tree – дерево решений

Дерево решений – распространенный способ определения алгоритма (в том числе алгоритма классификации). В каждом узле, кроме терминального, записывается условие, а потомки выбираются в зависимости от выполнения условия. Чаще всего применяются бинарные деревья (например, в узле размещено условие: «Зарплата > = 30 000», с вариантами выполнения «нет» и «да»), хотя можно применять и другие варианты (например, условие «Образование = » с вариантами: «неизвестно», «среднее», «среднее-специальное», «высшее»).

Каждое поддерево в классифицирующем дереве определяет множество объектов. В идеальном случае все объекты поддерева относятся к одному классу, однако в силу вероятностной природы зависимости это не гарантирует точности классификации. Более того, если добиться точной классификации для обучающей выборки, то наблюдается эффект переобучения – случайные отклонения принимаются за закономерности классификации и точность классификации на тестовой выборке существенно снижается. Кроме этого, чем больше узлов в дереве, тем больше времени занимает классификация.

Существует много вариантов построения дерева решений, и каждый вариант обладает своими характеристиками эффективности и точности классификации. Алгоритмы построения деревьев решений строят так, чтобы каждое разбиение максимально уменьшало неопределенность. Неопределенность можно измерять разными способами (индекс Джини, дисперсия, энтропия и др.). Выбор данных для обучения также может существенно повлиять на результат построения дерева.

Известные алгоритмы построения дерева учитывают все эти сложности и позволяют создавать достаточно хорошие классификаторы. Виджет «Tree»

(рис. 28), строящий дерево решений, позволяет определять следующие параметры:

– Induce binary tree строит бинарное дерево;

– Min. number of instances in leaves – алгоритм никогда не построит разбиение, которое содержит меньше указанного количества экземпляров;

– Do not split subsets smaller than запрещает алгоритму разбивать узлы с числом экземпляров меньше заданного;

– Limit the maximal tree depth ограничивает глубину дерева классификации указанным числом уровней узлов;

– Stop when majority reaches [%] – прекратить разбивать узлы после достижения определенного порога.

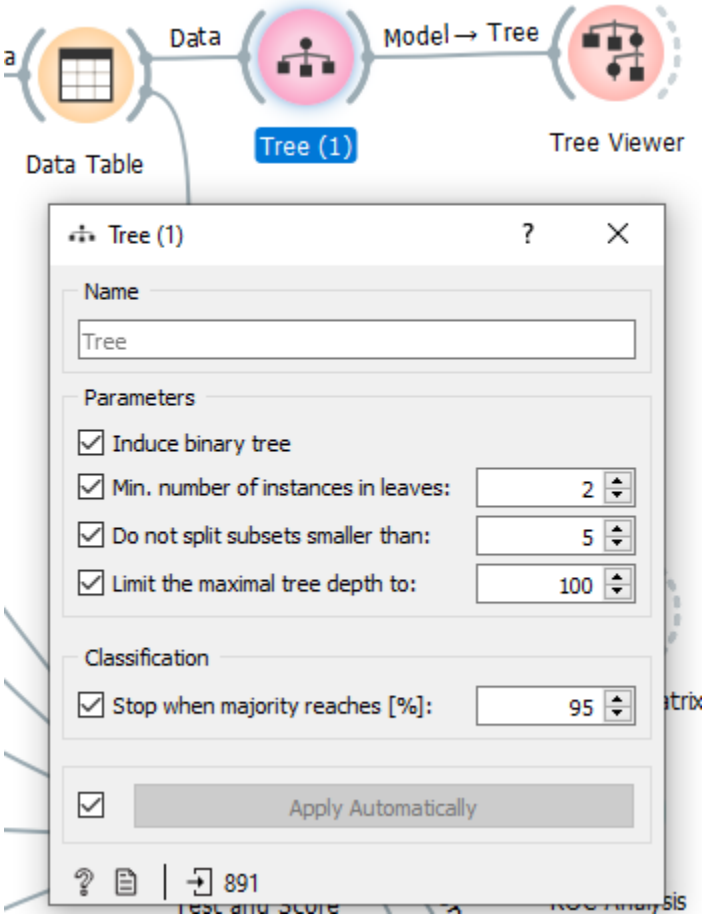


Рис. 27. Параметры дерева решений

Специальный виджет «Tree Viewer» позволяет просматривать полученное дерево решений (рис. 29). Верхнее значение узла соответствует классу (0 – погибшие, 1 – выжившие в катастрофе на Титанике). Класс узла определяется по преобладающему классу наблюдений, соответствующих узлу. Частота наблюдений преобладающего класса – вторая строка в узле. Для корневого узла определен класс 0, так как его доля погибших составляет 61,6 % = 549 / 891 · 100 %. В третьей строке узла записывается условие, например, «Sex=female» («Пол=женский»). Дуги соответствуют вариантам выполнения условия «<=0» («да») и «>0» («нет»). Чем ближе частота к 100 % тем более определенным является выбор

класса. Нет смысла продолжать построение дерева, если достигнута стопроцентная частота.

Достоинством классификации на основе дерева решений является эффективность алгоритма классификации. К недостаткам можно отнести высокую степень неопределенности построения дерева и риски переобучения алгоритма.

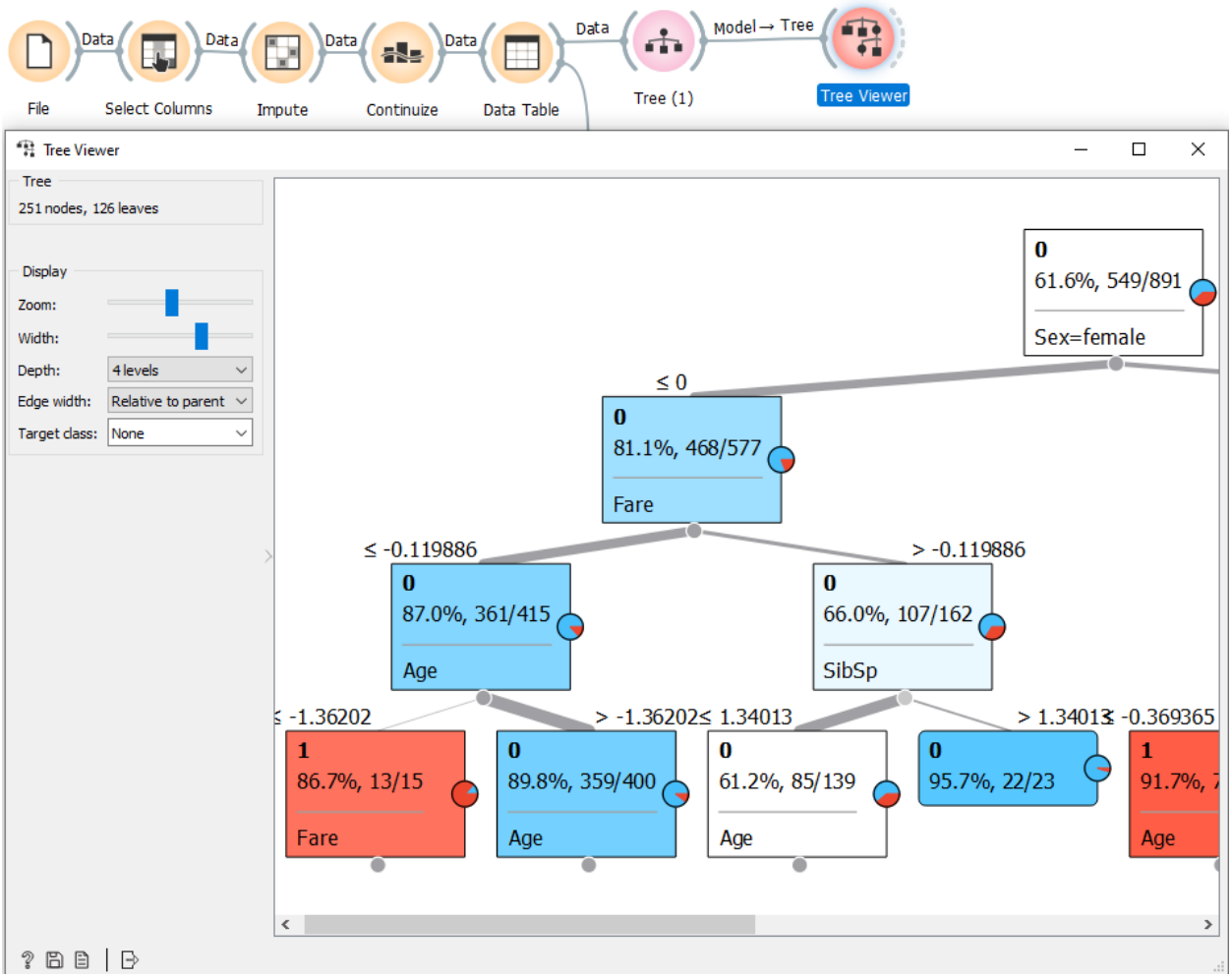


Рис. 28. Просмотр дерева решений с помощью виджета «Tree Viewer»

5.3. Random Forest – случайный лес

В данной модели применяется множество классифицирующих деревьев. Результатом классификации становится класс, за который проголосовало наибольшее количество классификаторов. Количество деревьев является основным параметром модели (рис. 30Рис. 29). Другие параметры определяют стратегию разбиения в узлах дерева и ограничения на рост деревьев.

Случайный лес – это удачный пример построения сильного классификатора на основе ансамбля более слабых. Это распространенный прием, позволяющий сгладить недостатки простых классификаторов. В частности, случайный лес повышает устойчивость к переобучению. К другим достоинствам данной модели можно отнести:

– способность эффективно обрабатывать данные с большим числом признаков и классов;

– нечувствительность к масштабированию (и вообще к любым монотонным преобразованиям) значений признаков;

– одинаково хорошо обрабатываются как непрерывные, так и дискретные признаки. существуют методы построения деревьев по данным с пропущенными значениями признаков;

– существуют методы оценивания значимости отдельных признаков в модели;

– возможность применения параллельной обработки.

Из недостатков следует отметить большой размер получающихся моделей, в связи с чем требуется значительный объем памяти для хранения модели.

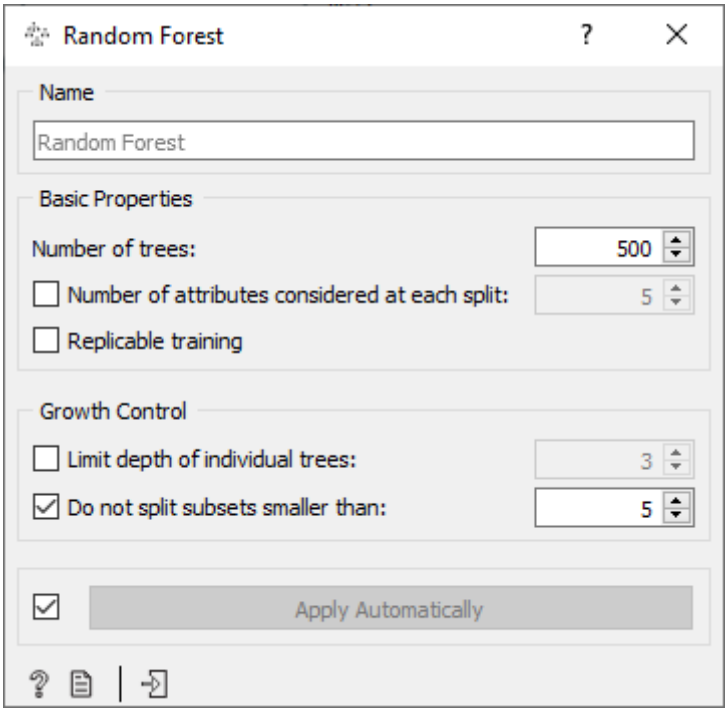


Рис. 29. Параметры модели «Random Forest»

5.4. kNN – метод k-ближайших соседей

Этот алгоритм работает достаточно просто: классифицируемый объект относится к тому классу, которому принадлежат большинство из ближайших к нему *k*-объектов обучающей выборки (рис. 31). В задачах с двумя классами число соседей берут нечетным, чтобы не возникало ситуаций неоднозначности, когда одинаковое число соседей принадлежат разным классам.

Метод опирается на одно важное предположение, называемое гипотезой компактности: если мера сходства объектов введена достаточно удачно, то схожие объекты гораздо чаще лежат в одном классе, чем в разных. В этом случае граница между классами имеет достаточно простую форму, а классы образуют компактно локализованные области в пространстве объектов.

Выбор параметра *k* противоречив. С одной стороны, увеличение его значения повышает достоверность классификации, но при этом границы между классами становятся менее четкими. Несмотря на свою относительную алгоритмическую простоту, метод показывает хорошие результаты. Главным его

недостатком является высокая вычислительная трудоемкость, которая увеличивается квадратично с ростом обучающей выборки.

Основным параметром является количество k-ближайших соседей (рис. 32). Если *k* небольшое, то выбор класса будет в большой степени случайным. Если *k* большое, то алгоритм будет плохо работать на границе классов. На работу алгоритма влияет способ вычисления расстояния между точками:

– Euclidean – евклидово расстояние;

– Manhattan – сумма модулей разностей координат;

– Maximal – наибольшая разность значений среди атрибутов;

– Mahalanobis – расстояние Махаланобиса – расстояние с учетом корреляций атрибутов.

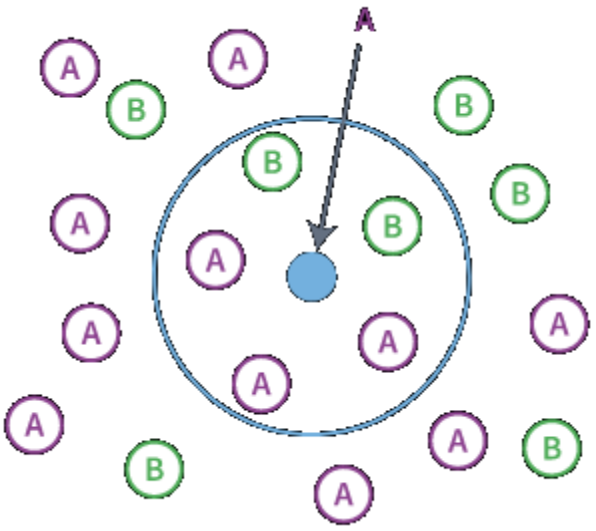


Рис. 30. Определение класса методом k-ближайших соседей

Расстояние может учитывать веса точек:

– Uniform – все веса одинаковы;

– Distance – соседи, расположенные ближе, имеют больший вес.

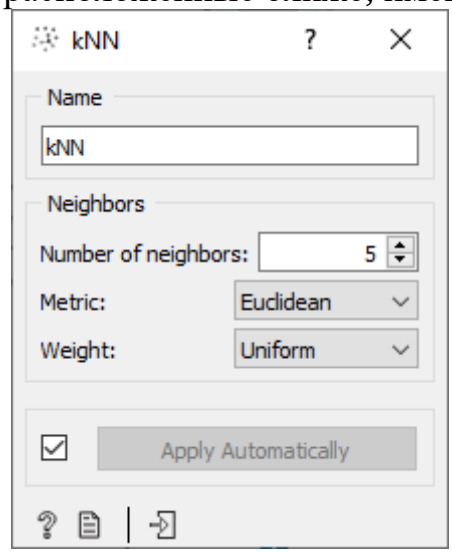


Рис. 31. Параметры модели kNN

Достоинства модели kNN:

– простота реализации;

– очевидность интерпретации и обоснования.

Недостатки модели kNN:

– неэффективный расход памяти и чрезмерное усложнение решающего правила вследствии необходимости хранения обучающей выборки целиком;

– поиск ближайшего соседа предполагает сравнение классифицируемого объекта со всеми объектами выборки, что требует линейного по длине выборки числа операций.

5.5. SVM – метод опорных векторов

Метод находит гиперплоскость, разделяющую классы с максимальным зазором. Метод использует дополнительное измерение (рис. 33), значения которого вычисляются с помощью некоторой функции атрибутов каждой точки – ядра с неизвестными коэффициентами. Вид ядра является значимым параметром метода, от него зависит форма границы классов.

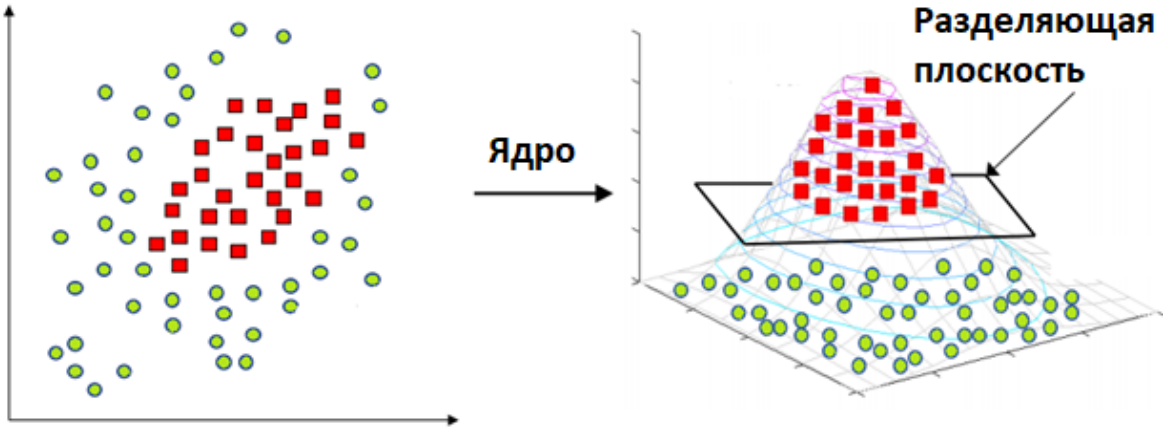


Рис. 32. Использование дополнительного измерения (ядра)
в методе опорных векторов

Возможно использование следующих ядер:

– Linear – линейная функция;

– Polynomial – полином;

– RBF – радиальная базисная функция;

– Sigmoid – сигмоида (рис. 34).

Коэффициент C – параметр настройки метода, который позволяет регулировать отношение между максимизацией ширины, разделяющей полосы, и минимизацией суммарной ошибки в случае перекрывающихся классов.

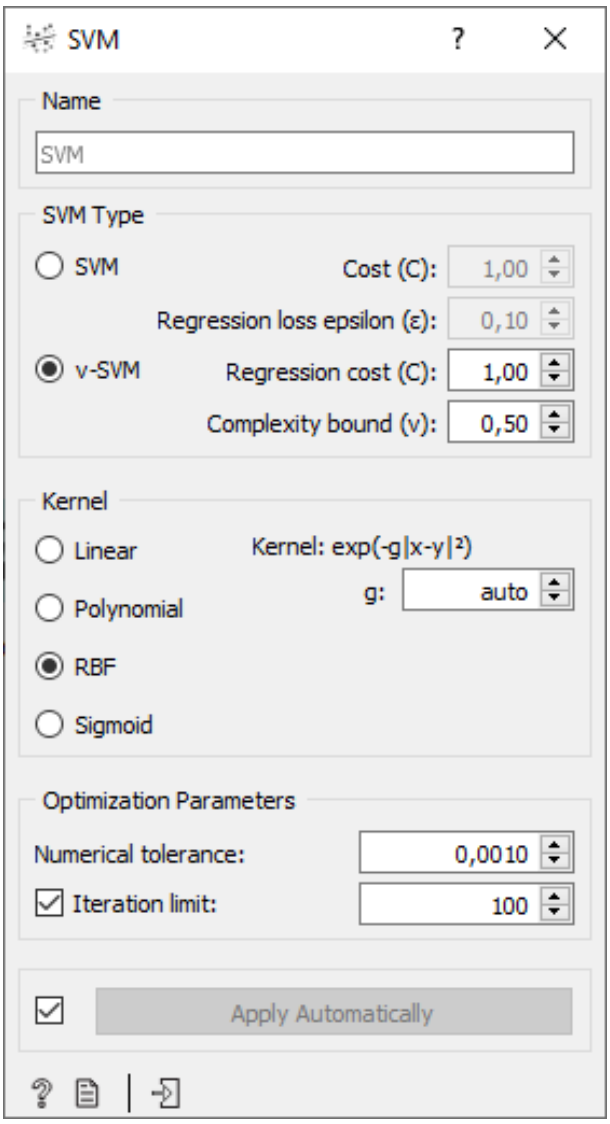


Рис. 33. Параметры метода опорных векторов

Достоинства алгоритма заключаются в том, что за счет применения гиперплоскостей он работает при малых объемах обучающей выборки. За счет использования ядра, описывающего связь между элементами выборки, можно использовать гиперплоскости разной сложности. Недостатками являются большое время обучения и необходимость подбора ядра для каждого конкретного случая.

5.6. Логистическая регрессия

Метод основан на представлении логарифма отношения вероятностей в виде линейной функции атрибутов объекта классификации с коэффициентами, которые определяются на основе минимизации суммы ошибок классификации. Для исключения переобучения применяют специальный метод – регуляризацию, которая характеризуется типом (L1 или L2) и значением параметра C (рис. 35).

Логистическая регрессия – это широко используемый метод, потому что он очень эффективен, хорошо интерпретируется, не требует слишком большого количества вычислительных ресурсов, масштабирования входных функций,

настройки, его легко упорядочить и при этом он выводит хорошо откалиброванные предсказанные вероятности.

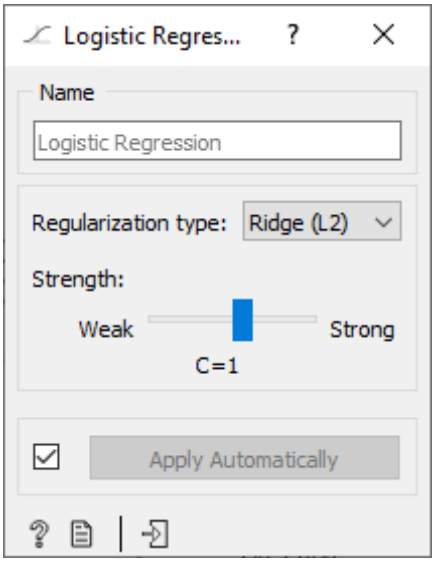


Рис. 34. Параметры логистической регрессии

В тоже время логистическая регрессия не будет хорошо работать с независимыми переменными, которые не связаны с целевой переменной и очень похожи или связаны друг с другом. Поэтому логистическая регрессия не всегда приводит к надежной классификации.

5.7. Naive Bayes – наивная байесовская модель

Данная модель определяет класс по максимуму условной вероятности принадлежности точки определенному классу. Предполагается независимость атрибутов друг от друга и вероятности значений атрибута при условии, что принадлежности классу заменяются соответствующими частотами. Метод сильно зависит от репрезентативности наблюдений. В случае отсутствия наблюдений значений некоторого атрибута в классе соответствующая условная частота будет равна нулю, нулевым будет произведение частот и вероятность соответствующего класса. Независимость атрибутов на самом деле не всегда выполняется. Тем не менее эта модель в вычислительном плане является одной из самых эффективных и в целом показывает неплохие качества классификации по сравнению с другими моделями.

Преимуществом модели является высокая скорость работы алгоритма как на этапе обучения, так и на этапе анализа новых данных, а недостатком – плохая классификация по значениям, которые не использовались (не встретились) при обучении.

5.8. AdaBoost – композиция алгоритмов обучения

Метод строит композицию из базовых алгоритмов обучения для улучшения (Adaptive boosting) их эффективности на основе подбора весовых коэффициентов. В AdaBoost каждая следующая композиция классификаторов строится по

объектам, неверно классифицированным предыдущими композициями. Параметры (рис. 36) определяют количество алгоритмов и скорость обучения.

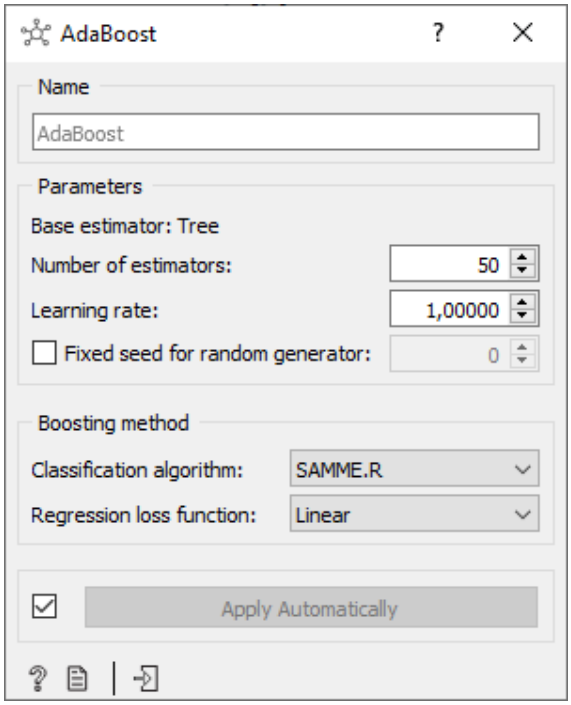


Рис. 35. Параметры AdaBoost

Достоинства модели:

1. Хорошая обобщающая способность. В реальных задачах (не всегда, но часто) удается строить композиции, превосходящие по качеству базовые алгоритмы.

2. Простота реализации.

3. Собственные накладные расходы бустинга невелики. Время построения композиции практически полностью определяется временем обучения базовых алгоритмов.

4. Возможность идентифицировать объекты, являющиеся шумовыми выбросами.

Недостатки модели:

1. AdaBoost склонен к переобучению при наличии значительного уровня шума в данных. Экспоненциальная функция потерь слишком сильно увеличивает веса наиболее трудных объектов, на которых ошибаются многие базовые алгоритмы. Однако именно эти объекты чаще всего оказываются шумовыми выбросами. В результате AdaBoost начинает настраиваться на шум, что ведет к переобучению. Проблема решается путем удаления выбросов или применения менее агрессивных функций потерь.

2. AdaBoost требует достаточно длинных обучающих выборок.

3. Бустинг может приводить к построению громоздких композиций, состоящих из сотен алгоритмов. Такие композиции исключают возможность содержательной интерпретации, требуют больших объемов памяти для хранения базовых алгоритмов и существенных затрат времени на вычисление классификаций.

5.9. Neural Network – нейронная сеть

Искусственная нейронная сеть имитирует поведение нейронов. Каждый нейрон получает на вход сигналы других нейронов или исходные атрибуты объекта и формирует выходной сигнал – вычисленный класс объекта. Входные сигналы складываются с весовыми коэффициентами, и полученная сумма преобразуется нелинейной функцией (обычно сигмоидой) к стандартному интервалу значений (от 0 до 1 или от –1 до 1). Выделяют входной слой нейронов, получающий на вход исходные данные и передающий выходные сигналы следующему слою, промежуточные слои и выходной слой, формирующий выходной сигнал (в случае классификации – вычисленный класс объекта). Данный классификатор (рис. 37) определяется количеством нейронов в скрытом слое (или нескольких слоях через запятую), функцией преобразования взвешенной суммы входных сигналов в выходной и параметрами поиска весовых коэффициентов.

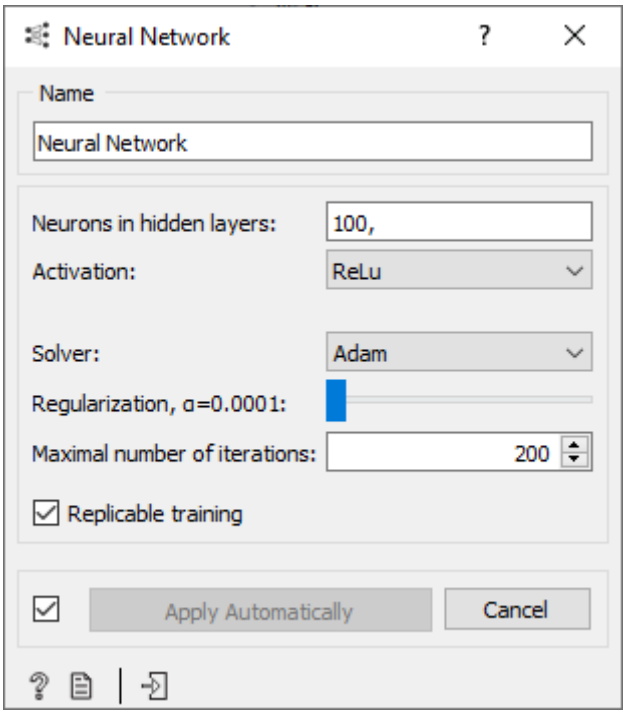


Рис. 36. Параметры нейронной сети

Основными преимуществами нейронных сетей перед традиционными вычислительными методами являются:

1. Решение задач в условиях неопределенности. Благодаря способности к обучению нейронная сеть помогает решать задачи с неизвестными закономерностями и зависимостями между входными и выходными данными, что позволяет работать с неполными данными.

2. Устойчивость к шумам во входных данных. Нейронная сеть может самостоятельно выявлять неинформативные для анализа параметры и производить их отсев, в связи с чем отпадает необходимость в предварительном анализе входных данных.

3. Гибкость структуры нейронных сетей. Компоненты нейрокомпьютеров – нейроны и связи между ними – можно комбинировать различными способами. За счет этого один нейрокомпьютер можно применять для решения различных задач, зачастую никак не связанных между собой.

4. Высокое быстродействие. Входные данные обрабатываются многими нейронами одновременно, благодаря чему нейронные сети решают задачи быстрее, чем большинство других алгоритмов.

5. Адаптация к изменениям окружающей среды. Нейронные сети, обучаясь на данных, способны подстраиваться под изменения.

К недостаткам нейронных сетей можно отнести следующее:

1. Отсутствие связи решаемой задачи и структуры нейронной сети.

2. Большое количество параметров нейронной сети требует большого количества данных для обучения.

3. Минимизация ошибки в процессе обучения не гарантирует выявление наилучших параметров сети.

4. Обучение нейронной сети требует больших вычислительных затрат.

5.10. Stochastic Gradient Descent – метод стохастического градиентного спуска

Метод устанавливает параметры границы классов на основе градиента, определяющего направление наиболее быстрого уменьшения суммы ошибок классификации. Градиент находится численными методами на основе случайного измерения параметров. Параметры (рис. 38) определяют особенности градиентного спуска.

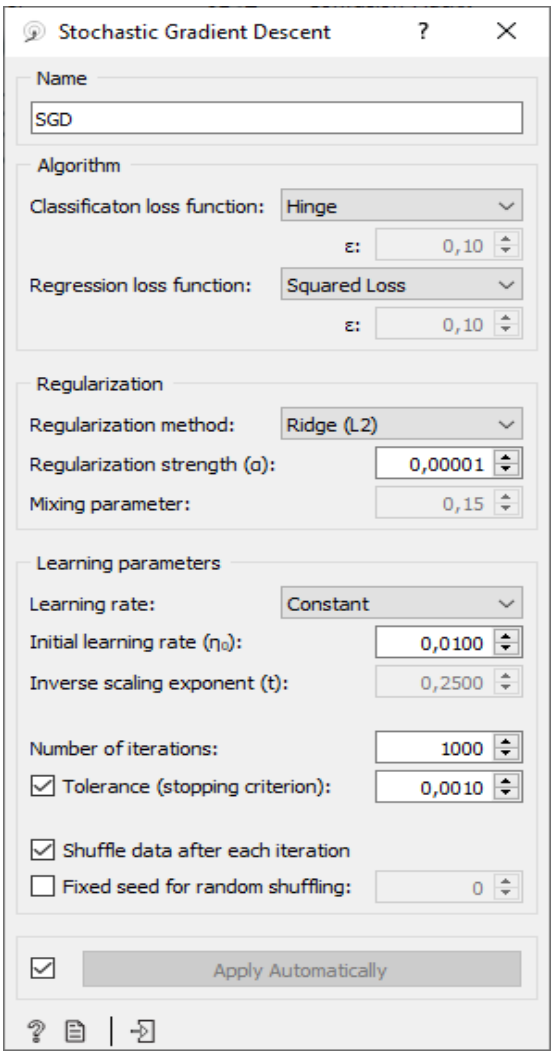


Рис. 37. Параметры стохастического градиентного спуска

Достоинствами стохастического градиентного спуска является простота реализации и возможность использования для обработки больших данных. К недостаткам можно отнести отсутствие универсальной методики подбора параметров.

5.11. Матрица ошибок (Confusion Matrix) классификации

Точность моделирования оценивается матрицей ошибок [4] (рис. 39), которая демонстрирует соотношения наблюдаемых и предсказанных классов. Действительно, может быть четыре результата бинарной классификации:

– истинноположительный (true positive, TP) – положительный класс модели совпадает с наблюдаемым;

– истинноотрицательный (true negative, TN) – отрицательный класс модели совпадает с наблюдаемым;

ложноположительный (false positive, FP) – положительный класс модели не совпадает с наблюдаемым;

– ложноотрицательный (false negative, FN) – отрицательный класс модели не совпадает с наблюдаемым (рис. 40).

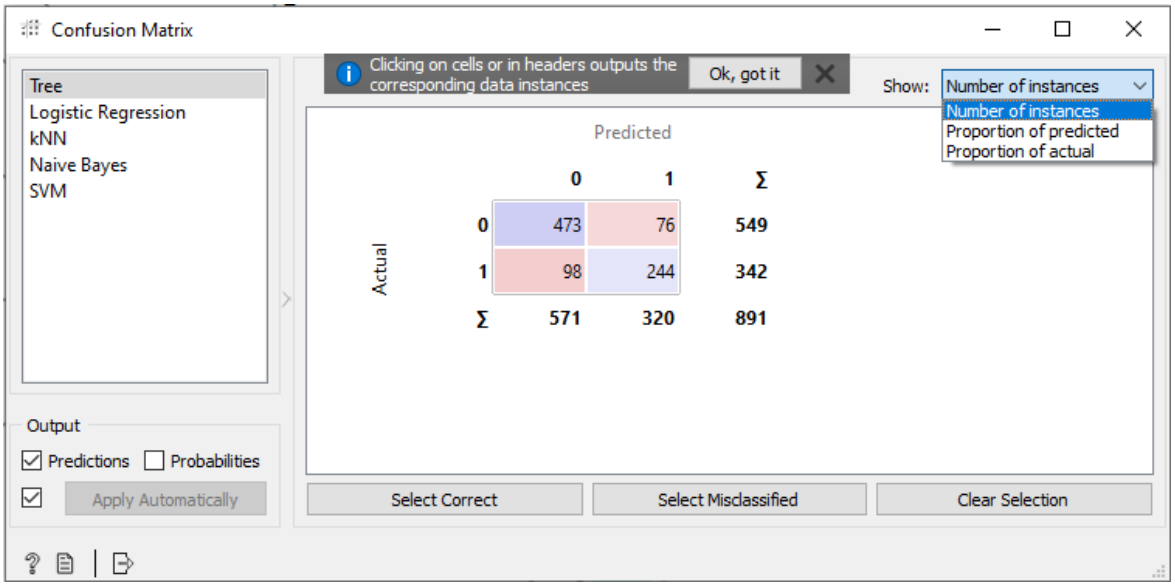


Рис. 38. Матрица ошибок

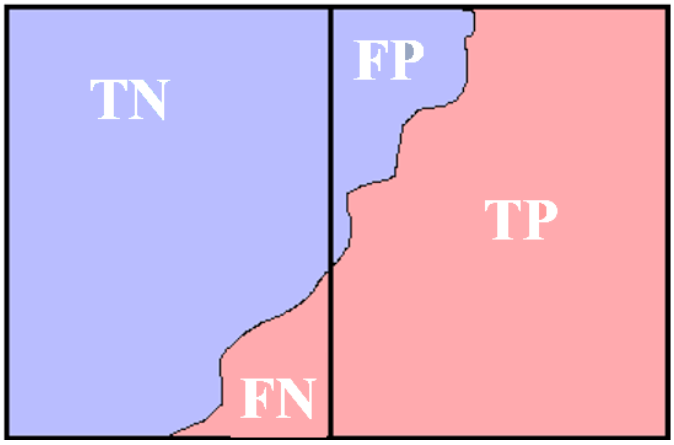


Рис. 39. Пример ошибок бинарной классификации:
синий цвет – «отрицательный» класс; красный – «положительный» класс;
вертикальная линия – деление объектов на классы алгоритмом
(слева – объекты, отнесенные алгоритмом к отрицательному классу,
справа – к положительному классу)

5.12. Показатели качества классификации

По данным матрицы ошибок определяется большое количество показателей классификаторов. Кроме общей точности классификации алгоритмы могут отличаться по точности классификации каждого класса.

Accuracy (точность), показывает долю правильных классификаций:

$$Acc = TP + TN / (TP + TN + FP + FN).$$

Несмотря на очевидность и простоту это одна из самых малоинформативных оценок классификаторов (аналог средней температуры по больнице). Он не показывает насколько хорошо распознаются отдельные классы или как часто совершаются ошибки первого рода (объект положительного класса отнесен классификатором к отрицательному) или второго рода (объект отрицательного класса отнесен классификатором к положительному).

Recall (полнота) или sensitivity (чувствительность), или TPR (true positive rate) показывает долю найденных классификатором объектов положительного класса к общему числу объектов положительного класса:

$$TPR = Recall = TP / (TP + FN).$$

Иначе говоря, полнота характеризует, насколько хорошо классификатор находит объекты из положительного класса.

Precision (точность), показывает долю объектов положительного класса среди объектов, отнесенных классификатором к положительному классу:

$$Prec = TP / (TP + FP).$$

Этот показатель также характеризует точность положительной классификации.

Specificity (специфичность), показывает долю верных классификаций отрицательного класса:

$$Spc = TN / (FP + TN).$$

Это характеристика точности распознавания отрицательного класса.

Fall-out или FPR (false positive rate) показывает долю ошибок распознавания объектов отрицательного класса:

$$FPR = FP / (FP + TN) = 1–Spc.$$

Это характеристика ошибочности распознавания отрицательного класса.

Для сравнения классификаторов применяют некоторые обобщенные характеристики, которые учитывают и точности распознавания классов и частоты ошибок. В качестве такой характеристики можно использовать F_1 меру – среднее гармоническое между precision и recall:

$$F_1 = \frac{2 \cdot \mathrm{Prec} \cdot \mathrm{Recall}}{\mathrm{Prec} + \mathrm{Recal}}.$$

В зависимости от важности классов дополнительно в формулу для F_1 вводят весовой коэффициент:

$$F_\beta = \frac{(1+\beta^2) \cdot \mathrm{Prec} \cdot \mathrm{Recall}}{\beta^2 \cdot \mathrm{Prec} + \mathrm{Recal}}.$$

Если $\beta < 1$, то больший вес получает точность (precision) положительной классификации, если $\beta > 1$, то больший вес получает полнота (recall).

5.13. ROC-функция (ROC-Analysis)

Данная функция определяет зависимость True Positive Rate (TPR)

$$TPR = TP / (TP + FN)$$

от False Positive Rate (FPR)

$$FPR = FP / (FP + TN).$$

ROC-кривая представляет собой линию от (0, 0) до (1, 1) (рис. 41). При построении кривой все наблюдения упорядочивают по убыванию вероятности положительной классификации. Близкой к нулю FPR соответствует большая вероятность положительной классификации и малое количество ошибок (FP) отнесения к положительному классу В идеальном случае, когда классификатор не делает ошибок, FPR = 0, TPR = 1. Чем больше FPR, тем меньше вероятность положительной классификации. Единичное значение FPR означает, что классификатор все наблюдения относит к положительному классу.

Чем ближе ROC-кривая к точке FPR = 0, TPR = 1, тем лучше работает классификатор. Таким образом площадь Area Under Curve (AUC) под ROC-кривой является интегральной характеристикой точности классификации.

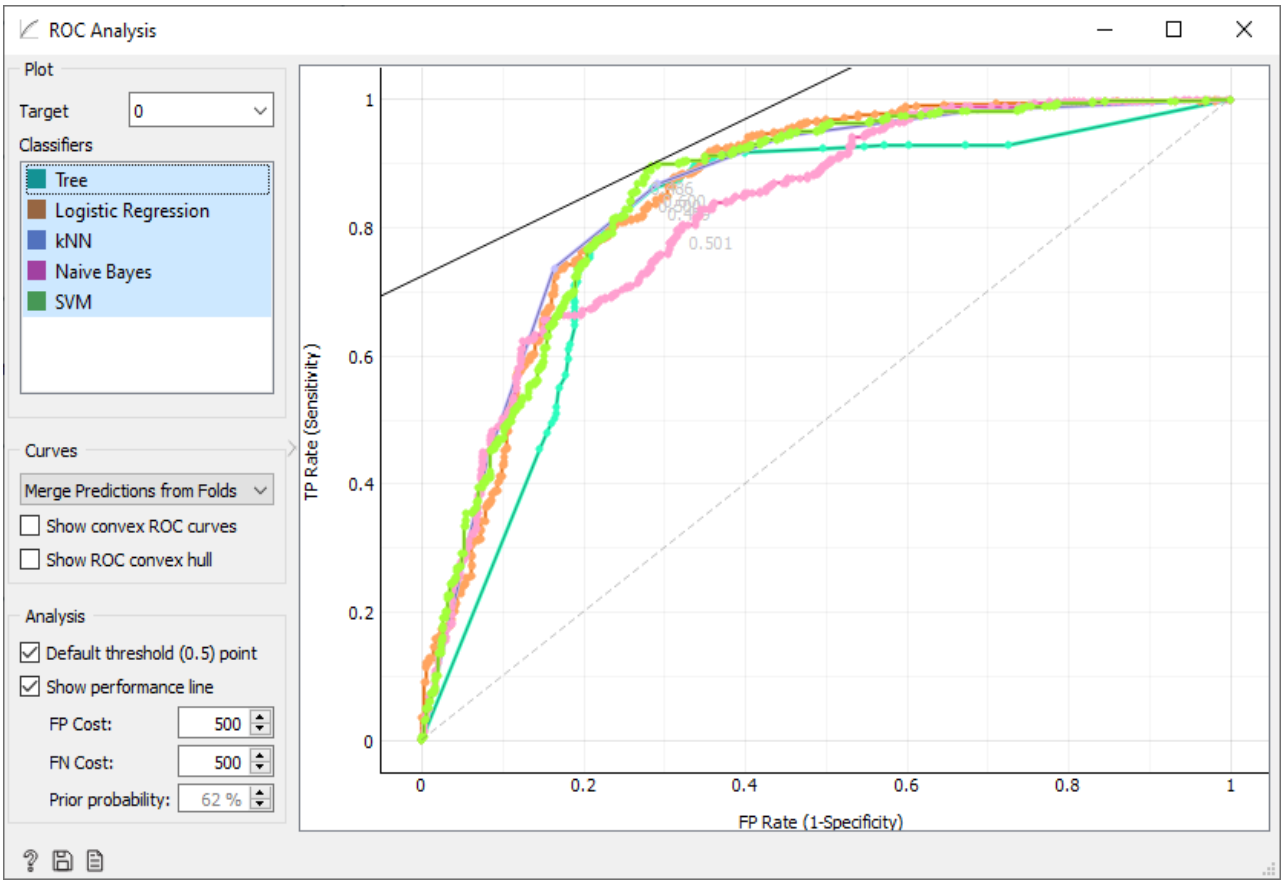


Рис. 40. ROC-кривые алгоритмов классификации

5.14. Лифт-функция (Lift Curve)

Одной из характеристик эффективности алгоритма классификации является лифт (Lift):

$$L = (k / K) / (n / N),$$

где N – количество объектов; n – количество объектов в положительном классе; K – количество «перспективных» объектов, выбранных с помощью соответствующей точки отсечения – порогового значения вероятности отнесения объектов к положительному классу; k – количество верно классифицированных объектов среди K выбранных.

Лифт показывает во сколько раз алгоритм классификации эффективнее выбора наугад. График лифта приведен на рис. 42. По графику видно, что при небольшом количестве наиболее перспективных объектов лифт максимален, а когда выбранные объекты соответствуют всей выборке, то лифт равен своему минимальному значению – 1.

Лифт-диаграмма (рис. 43), также как ROC-кривая, строится по данным, отсортированным в порядке убывания вероятности принадлежности положитель-

ному классу. По оси *x* откладывают значения доли выбранных наиболее перспективных объектов, по оси *y* откладывается доля правильно классифицированных, деленная на долю объектов положительного класса во всей выборке. Для идеально классификатора лифт-кривая растер линейно, пока не будут выбраны все объекты положительного класса, а затем остается постоянно равной единице.

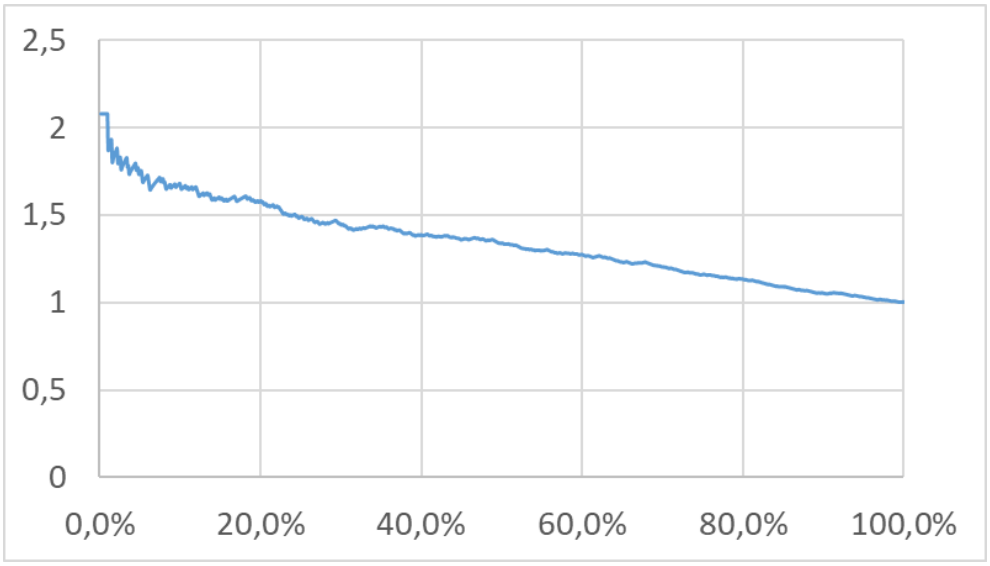


Рис. 41. График лифта в зависимости доли *K* / *N* выбранных объектов

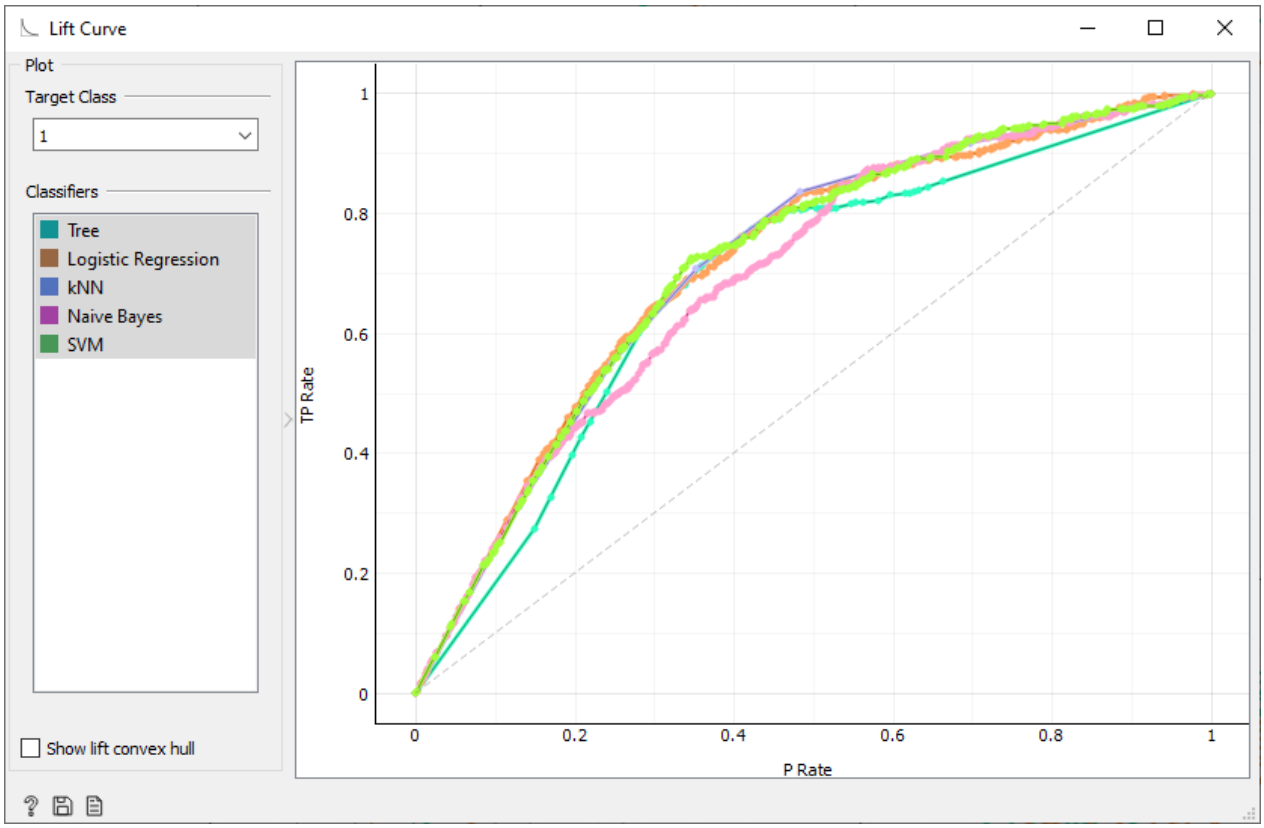


Рис. 42. Графики лифт-функций алгоритмов классификации

5.15. График калибровки вероятности (Calibration Plot)

Классификатор может по-разному ошибаться при определении вероятности положительной классификации для «перспективных» и «неперспективных» объектов. Это приводит к разной точности распознавания положительного и отрицательных классов. Более точную картину дает график калибровки вероятности (рис. 44). Он показывает зависимость наблюдаемой вероятности (частоты) для объектов с определенным алгоритмом вероятности положительной классификации. В идеале они должны совпадать, и соответствующий идеальный график – это диагональ из точки (0, 0) в точку (1, 1). Значения выше диагонали соответствуют ситуации, когда наблюдаемая частота больше предсказанной вероятности – заниженная оценка вероятности. Значения ниже диагонали свидетельствуют о завышенной вероятности положительной классификации.

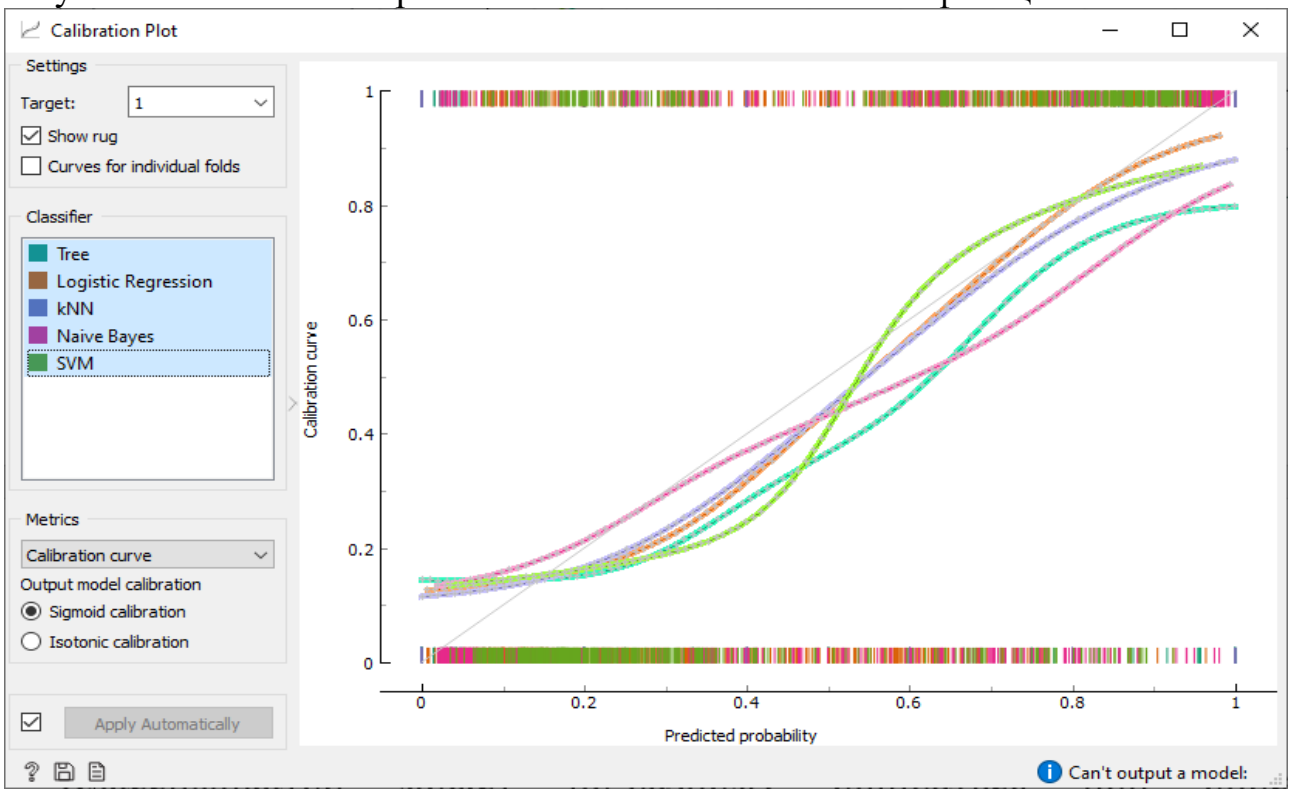


Рис. 43. Графики калибровки вероятности разных алгоритмов классификации

5.16. Сравнение моделей

Большое количество алгоритмов классификации позволяет сравнивать их применение и выбирать наиболее подходящий алгоритм для решения задачи по предложенному набору данных. Виджет «Test and Score» (рис. 45) тестирует и измеряет характеристики работы алгоритмов. Входом для него служат данные для обучения и тестирования алгоритмов и сами алгоритмы. Выходом являются оценки характеристик алгоритмов.

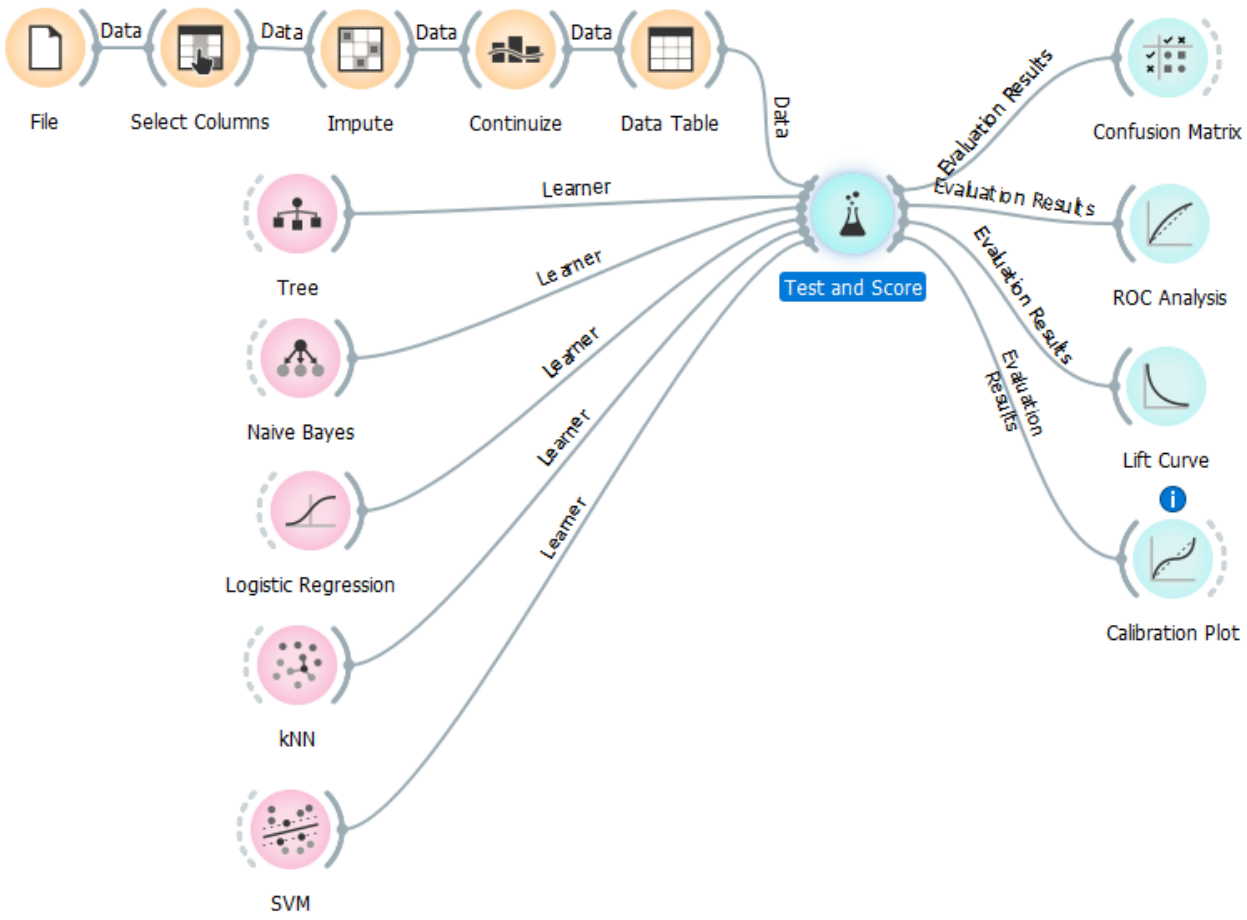


Рис. 44. Подбор параметров и сравнение моделей классификации

Виджет может выполнить обучение алгоритмов по одной из следующих схем:

– Cross-validation разделяет данные на выбранное количество частей (каждая часть по очередности используется как тестовая, а остальные для обучения);

– Cross validation by feature – разделение на части выполняется по значениям выбранной категориальной переменной;

– Random sampling формирует обучающую и тестовые части заданных объемов случайным выбором;

– Leave-one-out фиксирует один образец, обучая модель по остальным;

– Test on train data использует все данные для обучения (практически всегда происходит переобучение);

– Test on test data – данные используются для обучения, тестирование выполняется по данным для тестирования (рис. 46).

Параметр «Target class» определяет класс, по которому будут вычисляться характеристики алгоритмов или усреднение характеристик по всем классам (Average over classes).

Параметр «Model comparison» определяет характеристику, по которой будет происходить сравнение алгоритмов.

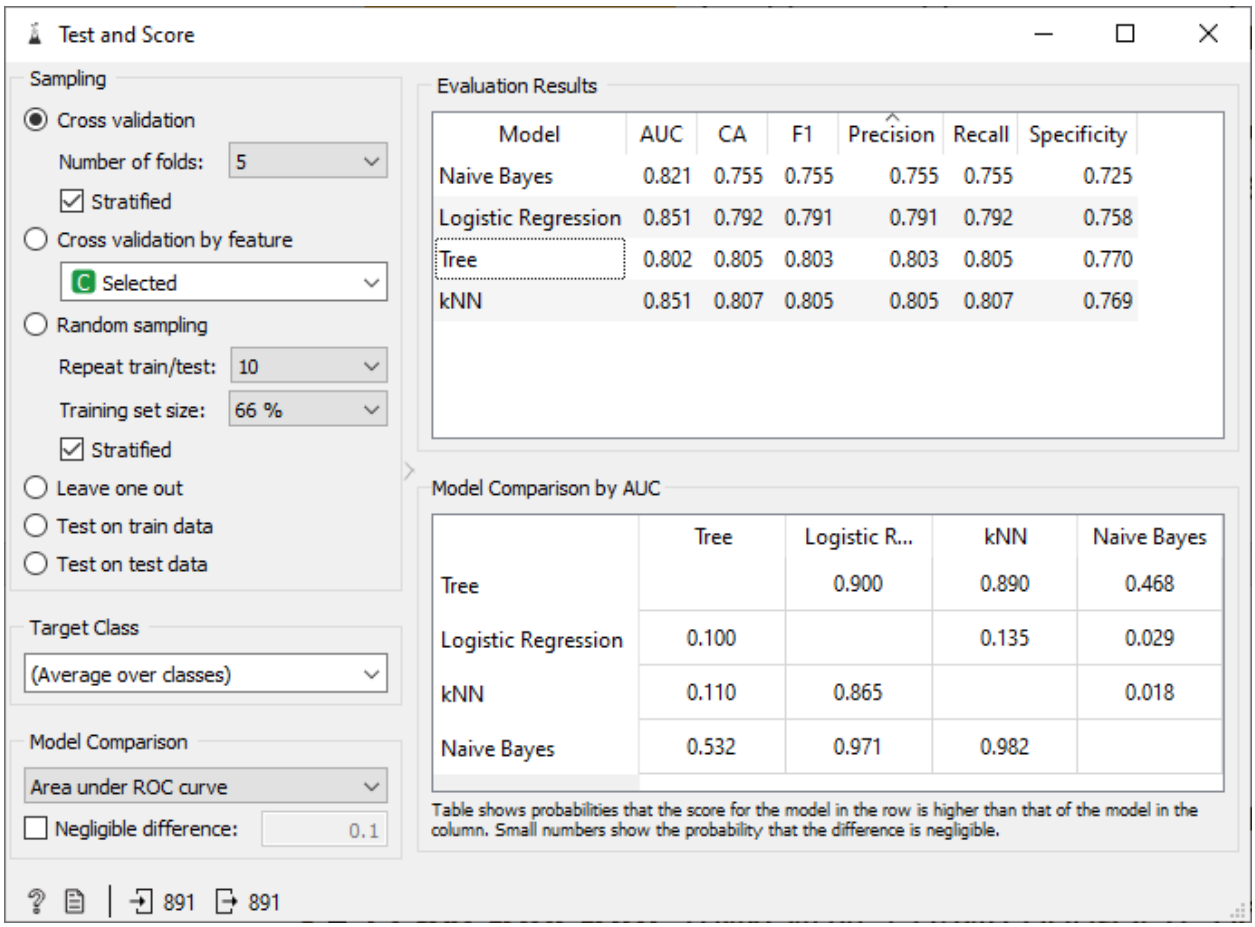


Рис. 45. Параметры и результат виджета «Test and Score»

В таблице Evaluation Results приводятся характеристики алгоритмов:

– AUC – Area under ROC – площадь под ROC-кривой;

– CA – Classification accuracy – доля правильно классифицированных наблюдений;

– F1 – взвешенное гармоническое среднее показателей precision и recall;

– Precision – доля правильно положительно классифицированных наблюдений среди всех положительно классифицированных наблюдений (доля выживших среди пассажиров, отнесенных алгоритмом к выжившим);

– Recall – доля правильно положительно классифицированных наблюдений среди всех наблюдений положительного класса (доля выживших среди выживших пассажиров);

– Specificity – доля правильно отнесенных отрицательного класса среди наблюдений отрицательного класса;

– LogLoss (cross-entropy loss) – неопределенность предсказания по отношению к истинному значению;

– Train time – время обучения в секундах;

– Test time – время тестирования в секундах;

Таблица Model Comparison by… содержит вероятности того, что модель в заголовке столбца лучше по указанной характеристике, чем модель, указанная в заголовке строки[11].

В соответствии с этой таблицей лучшим по AUC оказался алгоритм логистической регрессии (см. рис. 46).

5.17. Предсказание класса

Для выполнения предсказания используется виджет «Predictions». На его вход (рис. 47) передается настроенная модель и данные. На рисунке используются данные, по которым выполнялась настройка модели. На выходе получают данные для прогнозирования, дополненные данными прогноза. На рис. 48 представлены выходные данные: колонка «Random Forest» содержит прогноз класса, колонка «Random Forest (0.0)» – вероятность нулевого класса, колонка «Random Forest (1.0)» – вероятность первого класса.

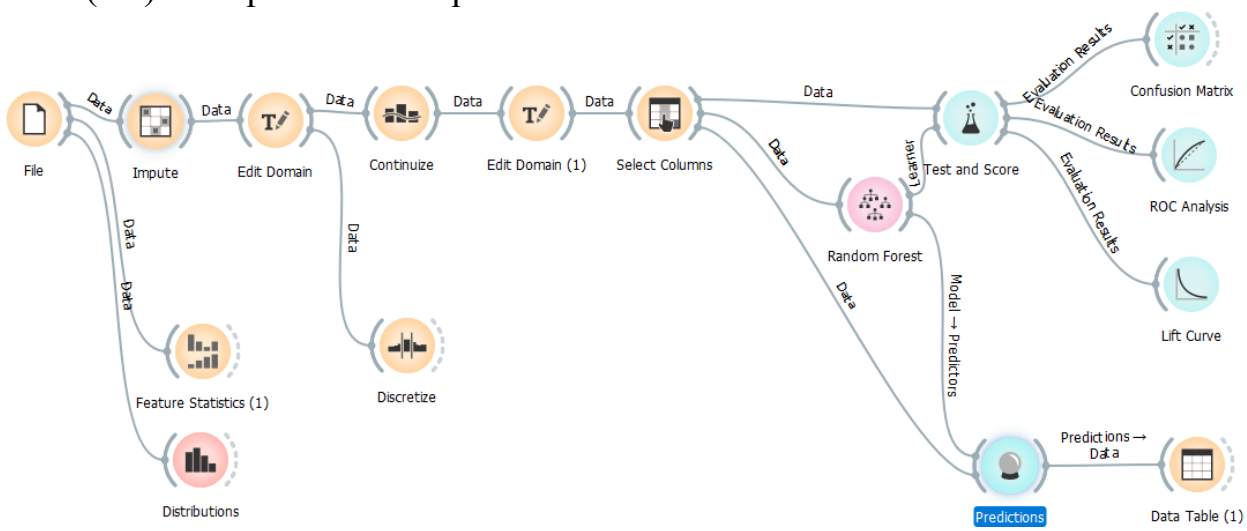
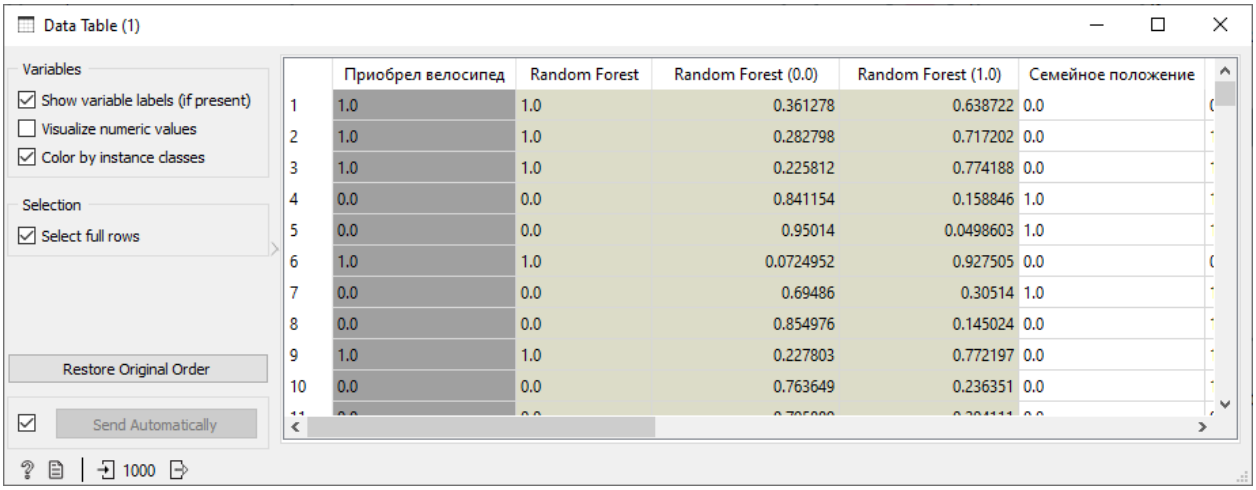


Рис. 46. Предсказание класса с использованием виджета «Predictions»



	Приобрел велосипед	Random Forest	Random Forest (0.0)	Random Forest (1.0)	Семейное положение
1	1.0	1.0	0.361278	0.638722	0.0
2	1.0	1.0	0.282798	0.717202	0.0
3	1.0	1.0	0.225812	0.774188	0.0
4	0.0	0.0	0.841154	0.158846	1.0
5	0.0	0.0	0.95014	0.0498603	1.0
6	1.0	1.0	0.0724952	0.927505	0.0
7	0.0	0.0	0.69486	0.30514	1.0
8	0.0	0.0	0.854976	0.145024	0.0
9	1.0	1.0	0.227803	0.772197	0.0
10	0.0	0.0	0.763649	0.236351	0.0

Рис. 47. Результаты предсказания класса с использование виджета «Predictions»

[11] Giorgio Corani, Alessio Benavoli A Bayesian approach for comparing cross-validated algorithms on multiple data sets. URL: https://link.springer.com/article/10.1007/s10994-015-5486-z.

5.18. Решение задач классификации с использованием модулей библиотеки Scikit-Learn на языке Python

В библиотеке Scikit-learn реализовано много наиболее известных алгоритмов классификации (табл. 8).

До настройки классификатора нужно приготовить данные для обучения и тестирования. Для этого выполняется импорт соответствующей функции
from sklearn.model_selection import train_test_split.

Вызов этой функции
X_train, X_test, y_train, y_test = train_test_split(
df_Titanic.drop(['Survived'], axis=1), # таблица со входными атрибутами объектов и исключенным целевым столбцом
df_Titanic['Survived'], # целевой столбец
test_size=0.75) # доля значений обучающей выборки
создает исходные данные для обучения – (X_train, y_train) и тестирования – (X_test, y_test).

Таблица 8

Алгоритмы машинного обучения, реализованные в Scikit-learn

Метод	Класс
kNN – k-ближайших соседей	sklearn.neighbors.KNeighborsClassifier
Logistic – логистическая регрессия	sklearn.linear_model.LogisticRegression
SVC – машина опорных векторов	sklearn.svm.SVC
Tree – деревья решений	sklearn.tree.DecisionTreeClassifier
RF – случайный лес	sklearn.ensemble.RandomForestClassifier
AdaBoost – адаптивный бустинг	sklearn.ensemble.AdaBoostClassifier
GBT – градиентный бустинг деревьев решений	sklearn.ensemble.GradientBoostingClassifier

Все алгоритмы выполнены в виде классов, обладающих одинаково названными методами, перечисленными в табл. 9.

Таблица 9

Основные методы классов, реализующих алгоритмы машинного обучения

Метод класса	Описание
fit(X, y)	Обучение (тренировка) модели на обучающей выборке X, y
predict(X)	Предсказание на данных X
predict_proba(X)	Определение вероятности (кроме SVC)
set_params(**params)	Установка параметров алгоритма
get_params()	Чтение параметров алгоритма

Кроме этого, нужно включить в программу:

– модуль для кросс-валидации
from sklearn.model_selection import cross_val_score

– модуль поиска параметров классификатора
from sklearn.model_selection import RandomizedSearchCV

– модуль для выполнения ROC-анализа характеристик классификатора
from sklearn.metrics import roc_curve, auc, roc_auc_score.

Модули классификаторов:

– модули алгоритмов опорных векторов
from sklearn import svm
– машина опорных векторов
from sklearn.svm import SVC
алгоритм k-ближайших соседей
from sklearn.neighbors import KNeighborsClassifier
метод случайного леса
from sklearn.ensemble import RandomForestClassifier
– логистическая регрессия
from sklearn.linear_model import LogisticRegression
– адаптивнй бустинг
from sklearn.ensemble import AdaBoostClassifier
– градиентный бустинг
from sklearn.ensemble import GradientBoostingClassifier
– нейронная сеть
from sklearn.neural_network import MLPClassifier
– дерево решений
from sklearn.tree import DecisionTreeClassifier.

Настройку и применение классификатора рассмотрим на примере алгоритма k-ближайших соседей

```
# Создание объекта – классификатора
KNN=KNeighborsClassifier(n_neighbors=5)
# Для подбора параметров создается словарь по числу параметров. Для данного алгоритма – это n_neighbors – количество соседей. Для этого параметра будут выполняться три варианта настройки: [3,5,7].
Params = {'n_neighbors': [3,5,7]}
# Выполнение настройки классификатора для всех комбинаций параметров и определение наилучшей по критерию 'roc_auc' – площадь под ROC-кривой. Применяется кросс-валидация с разбиением на qParts частей.
gridSearch = RandomizedSearchCV(estimator=KNN, param_distributions=Params, n_iter=5,
scoring='roc_auc', cv=qParts, verbose=2).fit(X_train, y_train)
# Наилучшие параметры
Best_Params = gridSearch.best_params_
# Наилучшее значение критерия качества классификатора
best_roc_auc=gridSearch.best_score_
# Создание объекта с наилучшими параметрами
KNN = KNeighborsClassifier(n_neighbors = gridSearch.best_params_['n_neighbors'])
# Обучение классификатора
KNN.fit(X_train,y_train)
# Запись в таблицу колонки с прогнозом класса
df_Titanic['KNN'] = KNN.predict(df_model.drop([targetfield], axis=1))
# Запись в таблицу колонки с вероятностью класса
df_Titanic['p(KNN)'] = KNN.predict_proba(df_model.drop([targetfield], axis=1))[:,1]
```

```
# Вычисление прогнозов класса по тестовой выборке
test_labels = KNN.predict(X_test)
# Вычисление вероятностей классов по тестовой выборке
test_proba = KNN.predict_proba(X_test)
# Вычисление характеристики точности классификации
KNN_score = KNN.score(X_test, y_test) по тестовой выборке
# Вычисление площади под ROC-кривой по тестовой выборке
auc_score = roc_auc_score(y_test, test_proba[:,1], average='macro', sample_weight=None)
# Запоминание точек ROC-кривой
fprKNN, tprKNN, thresholdKNN = roc_curve(y_test, test_proba[:,1])
```

Аналогично выполняется настройка других классификаторов. Полные текст программы применения классификации и кластеризации приведены в Приложении 1.

Для визуального сравнения графиков ROC-кривых выполняются команды

```
fig, ax = plt.subplots()
ax.plot(fprKNN, tprKNN)
ax.plot(fprLR, tprLR)
ax.plot(fprDT, tprDT)
ax.plot(fprSVM, tprSVM)
ax.legend(['K-соседей', 'Логистическая регрессия', 'Дерево решений', 'Машина опорных векторов'], loc='right')
ax.set(xlabel='fpr', ylabel='tpr',title='ROC')
ax.grid()
fig.savefig("KNN LR DT SVM.png")
```

которые строят график ROC-кривой и сохраняют его в файле «KNN LR DT SVM.png» (рис. 49). Аналогично строятся графики для других классификаторов (рис. 50).

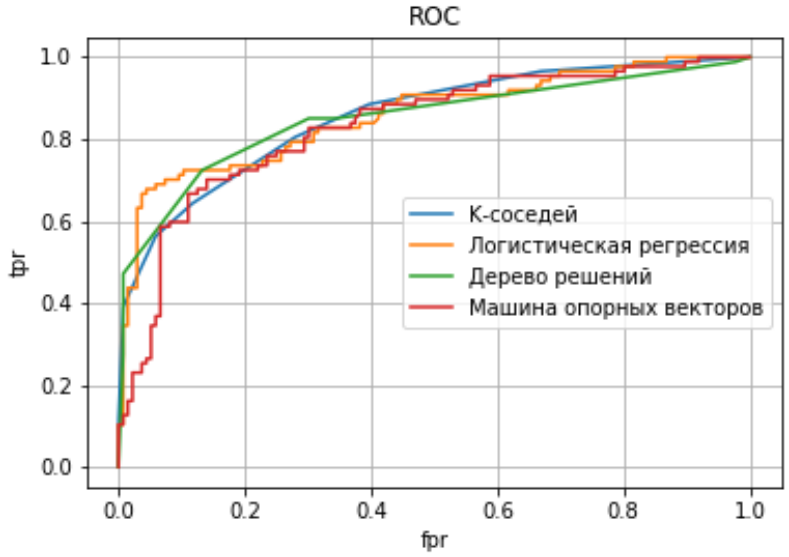


Рис. 48. ROC-кривые классификаторов «K-соседей», «Логистическая регрессия», «Дерево решений», «Машина опорных векторов»

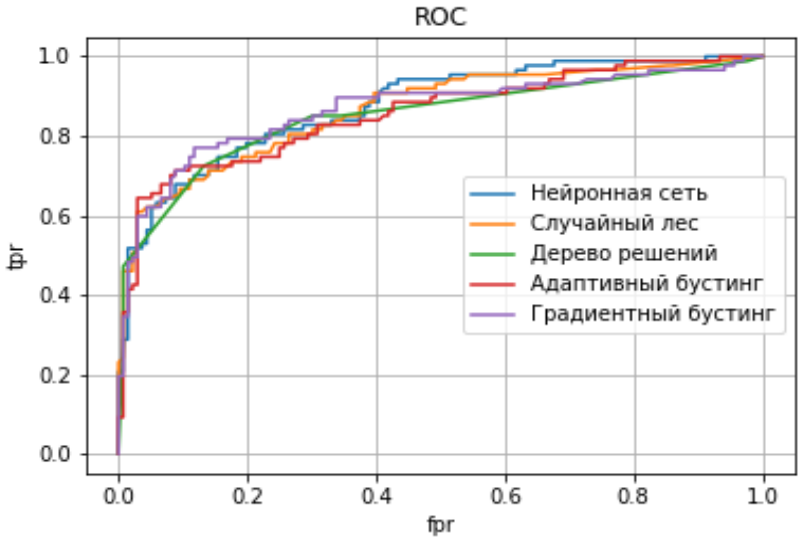


Рис. 49. ROC-кривые классификаторов «Нейронная сеть», «Случайный лес», «Дерево решений», «Адаптивный бустинг», «Градиентный бустинг»

Приведенные примеры демонстрируют достаточно простое применение модулей Scikit-Learn для программирования применения моделей класстфикации.

6. Регрессионные модели

Модель регрессии заключается в подборе вектора a параметров так, чтобы функция $y = f(x,a)$ аппроксимировала зависимость y от x наилучшим образом. Для подбора параметров обычно применяют метод наименьших квадратов, т. е. находят параметры, минимизирующие сумму квадратов ошибок:

$$a^* = \arg\min \sum_{i=1}^{m} \left(f(x_i, a) - y_i \right)^2 .$$

Семейство параметрических кривых $y = f(x,a)$ определяют, исследуя облако рассеивания. Если оно вытянуто вдоль прямой, то применяют линейные уравнения регрессии. В этом случае коэффициент корреляции характеризует рассеивание. Чем он ближе к 1 или –1, тем меньше разброс и больше точность приближения зависимости линейной функции. Нулевое значение коэффициента корреляции свидетельствует либо о независимости функции от значений аргументов, либо о нелинейном характере зависимости.

В Orange для построения регрессионных моделей можно применять такие же алгоритмы, какие применялись для решения задачи классификации. В этом случае алгоритм на выходе будет определять не номер класса, а непрерывное значение. Рассмотрим построение регрессионной модели для данных проката велосипедов (Bike Sharing). Данные содержат входные переменные:

– Season – время года;
– Yr – год;
– Mnth – месяц;
– Holiday – выходной день;

– Weekday – день недели;
– Workingday – рабочий день;
– Weathersit – погода;
– Temp – температура среды;
– Atemp – ощущение температуры;
– Hum – влажность;
– Windspeed – скорость ветра (рис. 51).

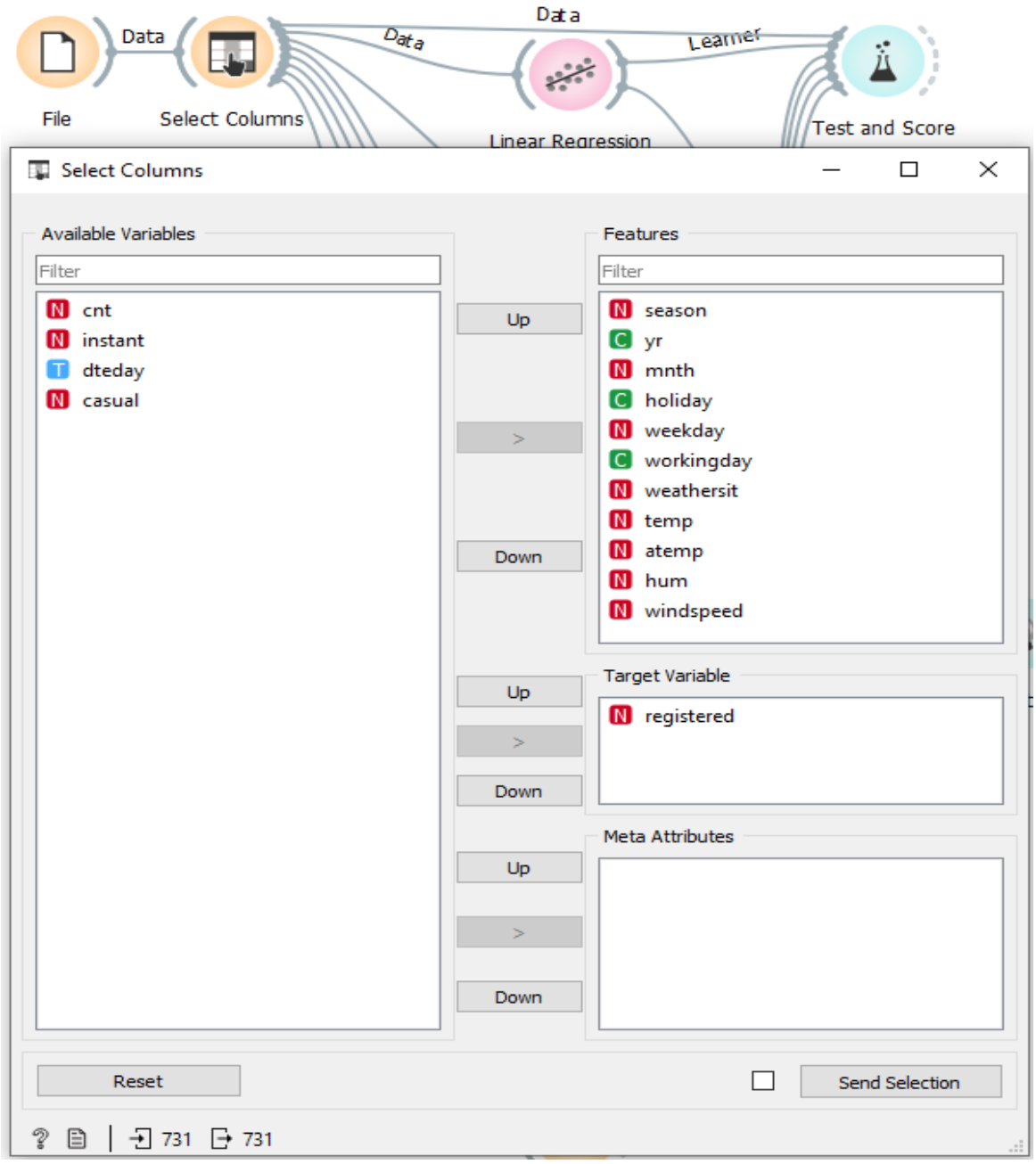


Рис. 50. Определение входных и выходных переменных
для регрессионных моделей проката велосипедов

Выходная переменная «registered» – количество плановых использований велосипедов. Параметры регрессии (рис. 52) позволяют включать регуляризацию сильной зависимости входных переменных. Регуляризация заключается в добавлении условий и ограничений, снижающих эффект переобучения.

Параметры нейронной сети (рис. 53) позволяют задать количество нейронов для внутренних слоев. Приведенная на рисунке конфигурация подобрана эмпирически для получения меньшей ошибки регрессии. Аналогично подобрано количество ближайших соседей для алгоритма k-ближайших соседей (рис. 54). Кроме перечисленных моделей для решения задачи применялась модель дерева решений и случайного леса (рис. 55).

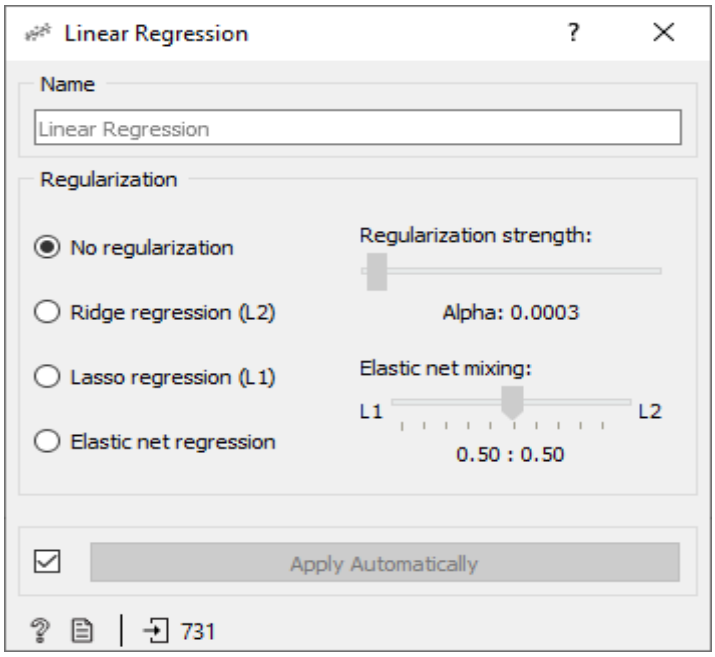


Рис. 51. Параметры линейной регрессии

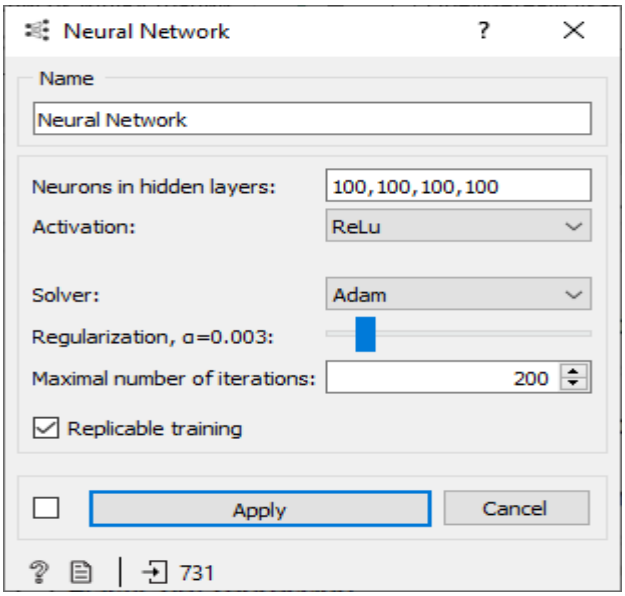


Рис. 52. Параметры нейронной сети

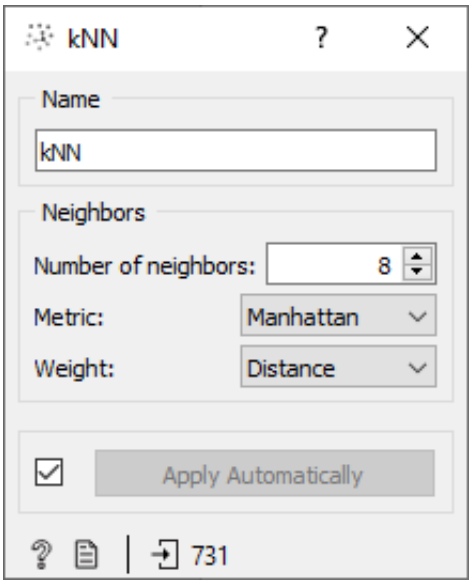
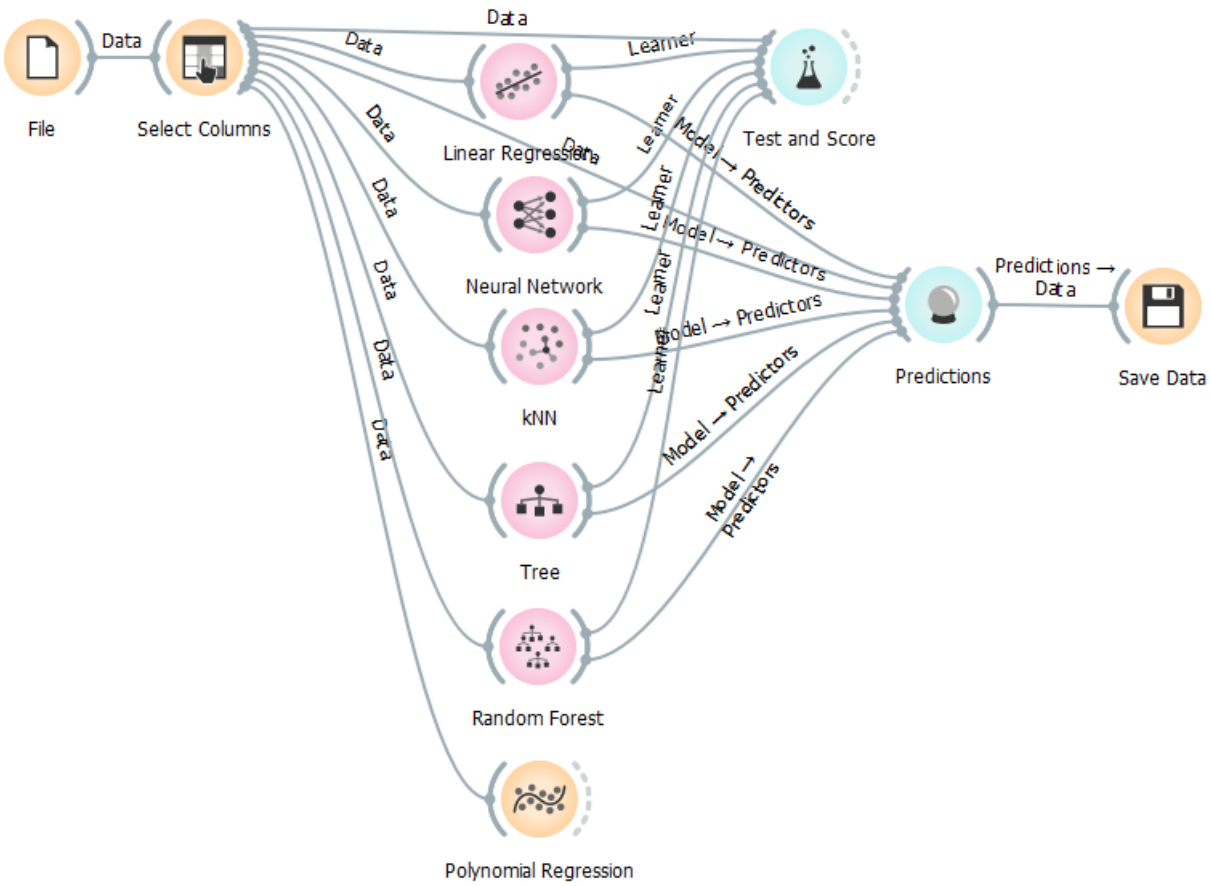


Рис. 53. Параметры алгоритма k-ближайших соседей



.

Рис. 54. Настройка моделей регрессии

На рис. 56 представлены характеристики точности моделей регрессии:

– MSE – дисперсия ошибки;

– RMSE (root mean squared error) – среднеквадратическая ошибка;

– MAE median absolute error – медиана абсолютной ошибки;

– MAPE mean absolute percent error – доля MAE;

– R^2 coefficient of determination – коэффициент детерминации (доля дисперсии, объясняемой моделью).

Сравнение моделей демонстрирует, что модель случайного леса обеспечивает минимальное значение средней ошибки регрессии.

Полиномиальная регрессия использует в качестве функции регрессии полином, подбирая коэффициенты полинома для минимизации ошибки. На рис. 57 представлены параметры виджета полиномиальной регрессии для аппроксимации полиномом 5-го порядка зависимости целевой переменной – количества прокатов велосипедов в зависимости от температуры (температура в исходных данных представлена в нормализованном виде).

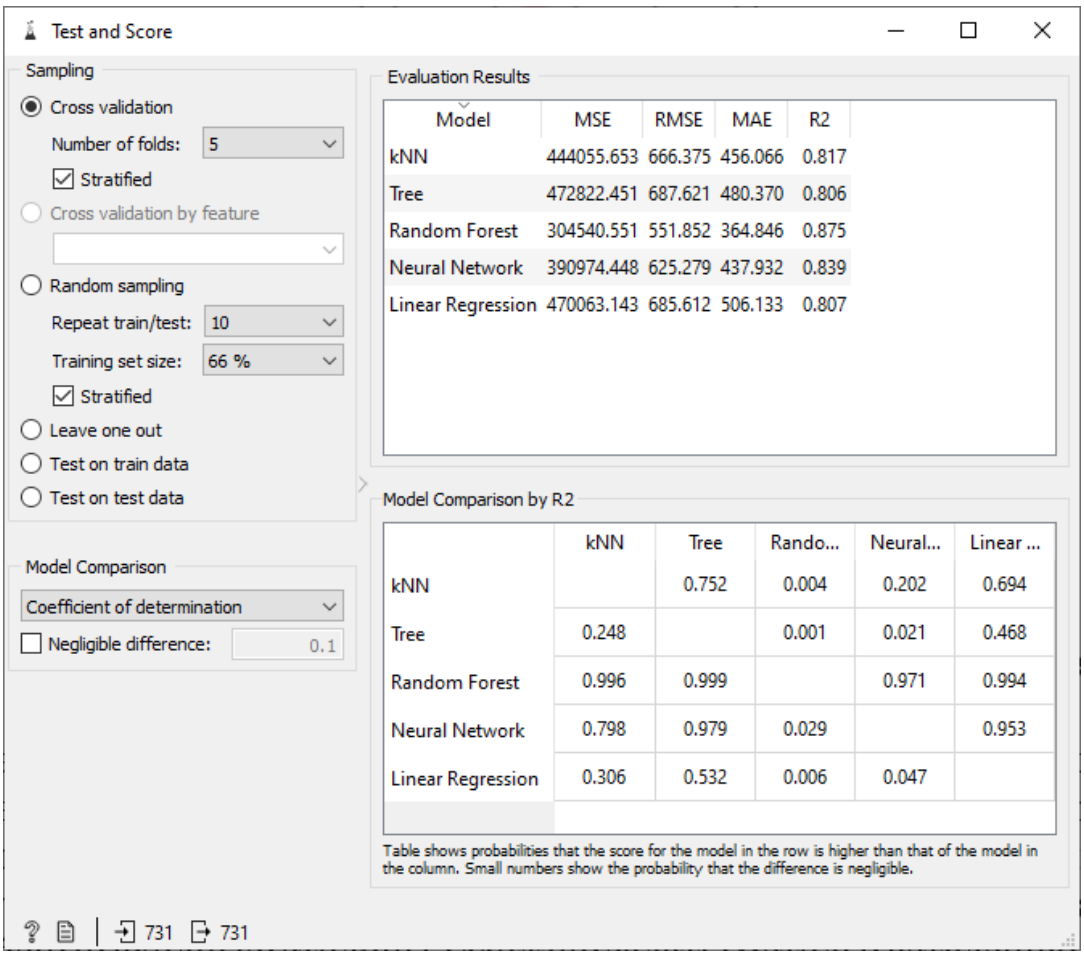


Рис. 55. Характеристики моделей регрессии

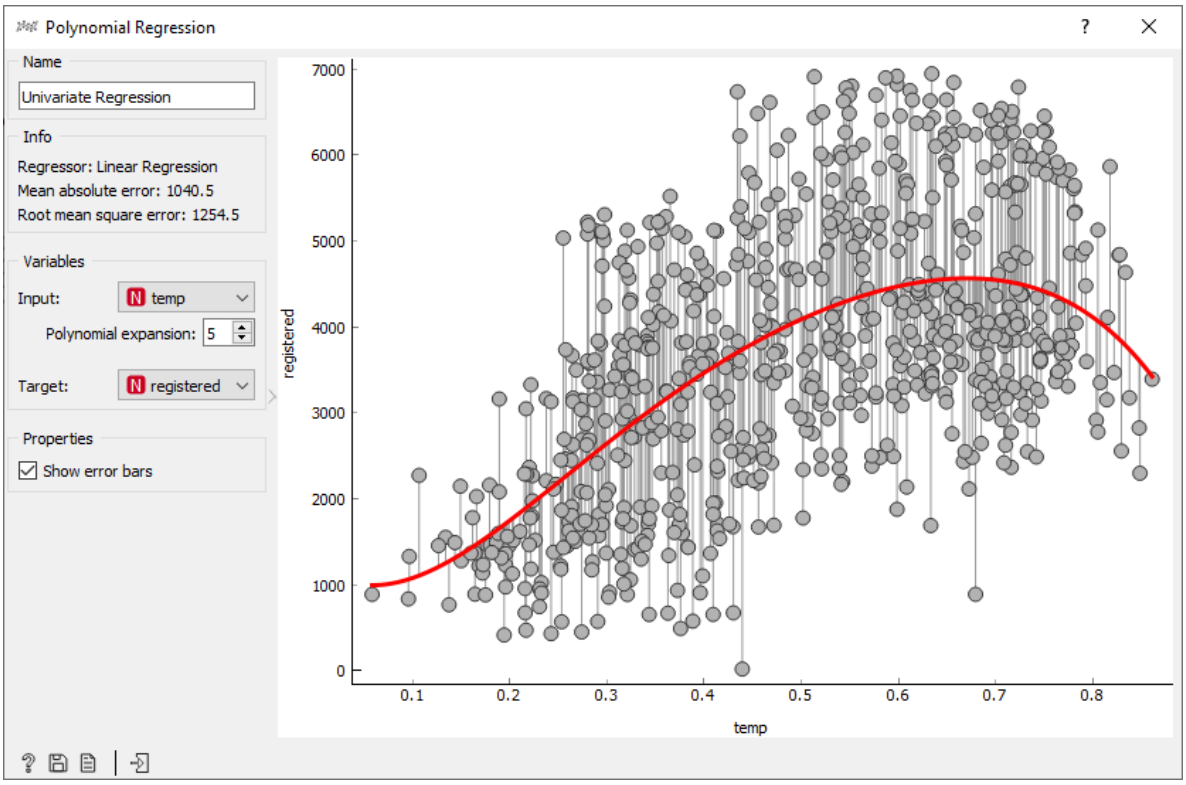


Рис. 56. Полиномиальная регрессия

Настроенную модель можно применять для вычисления выходной переменной по значениям входных переменных. На рис. 55 для этого используется виджет «Prediction».

7. Кластеризация

Кластеризация – объединение объектов в группы (кластеры) по степени похожести, т. е. по расстоянию между объектами [4]. Существует много способов определения расстояния между объектами. Можно находить традиционное расстояние по прямой между точками в многомерном пространстве – евклидово расстояние – квадратный корень из суммы квадратов разностей координат (атрибутов); манхетенское расстояние – сумма модулей разности координат; расстояние Махаланобиса, похожее на евклидову метрику, но учитывающее корреляционные зависимости координат, максимум модуля разности и многие др. Общим является учет разности координат, поэтому необходимо приводить координаты к одинаковым масштабам, чтобы исключить влияние слишком больших различий в масштабах измерений.

Результаты кластеризации неоднозначны уже и потому, что неизвестно количество кластеров. Тем не менее кластеризация разными методами дает приблизительно похожие результаты. Это подтверждает общее предположение о неоднородном распределении объектов в пространстве атрибутов.

Критерии качества кластеризации также разнообразны и ориентированы на определенную конфигурацию кластеров в пространстве, например, шарообразную или вытянутую и изогнутую. Зашумленность данных случайными отклонениями добавляет свой вклад в общую неопределенность. При этом кластеризация нередко дает определенные результаты в отношении свойств объектов, отнесенных к одному кластеру.

7.1. Hierarchical Clustering – иерархическая кластеризация

Для иерархической кластеризации требуется предварительно выполнить вычисление расстояния для каждой пары объектов с помощью виджета «Distances» (рис. 58). В результате получается матрица расстояний.

Виджет «Hierarchical Clustering» по матрице расстояний строит дерево и представляет его в виде дендрограммы (рис. 59). Виджет строит дерево на основе расстояний между кластерами:

– Single linkage – расстояние до ближайших элементов кластеров;

– Average linkage – среднее расстояние между элементами двух кластеров;

– Weighted linkage – вычисляется взвешенное расстояние методом WPGMA;

– Complete linkage – определяется расстояние между наиболее удаленными элементами кластеров.

Слева от дендрограммы выводятся метки атрибута, выбранного в поле «Annotation». Глубина изображения дендрограммы устанавливается парамерами обрезки в поле «Pruning». Обрезка изображения не влияет на построение дерева.

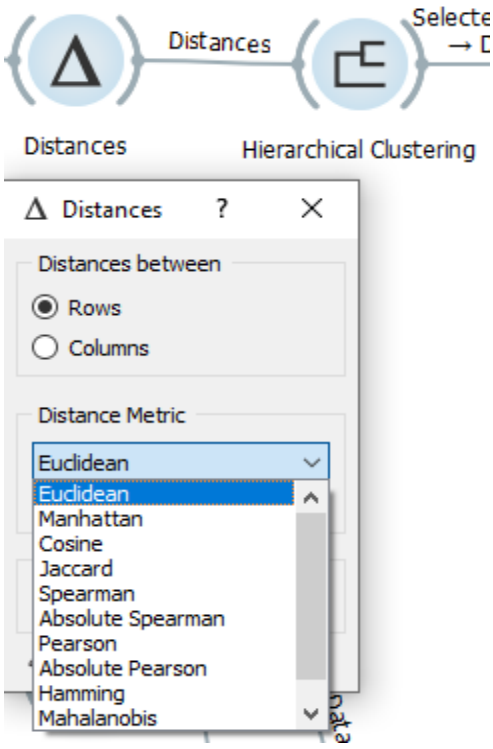


Рис. 57. Вычисление расстояний между объектами

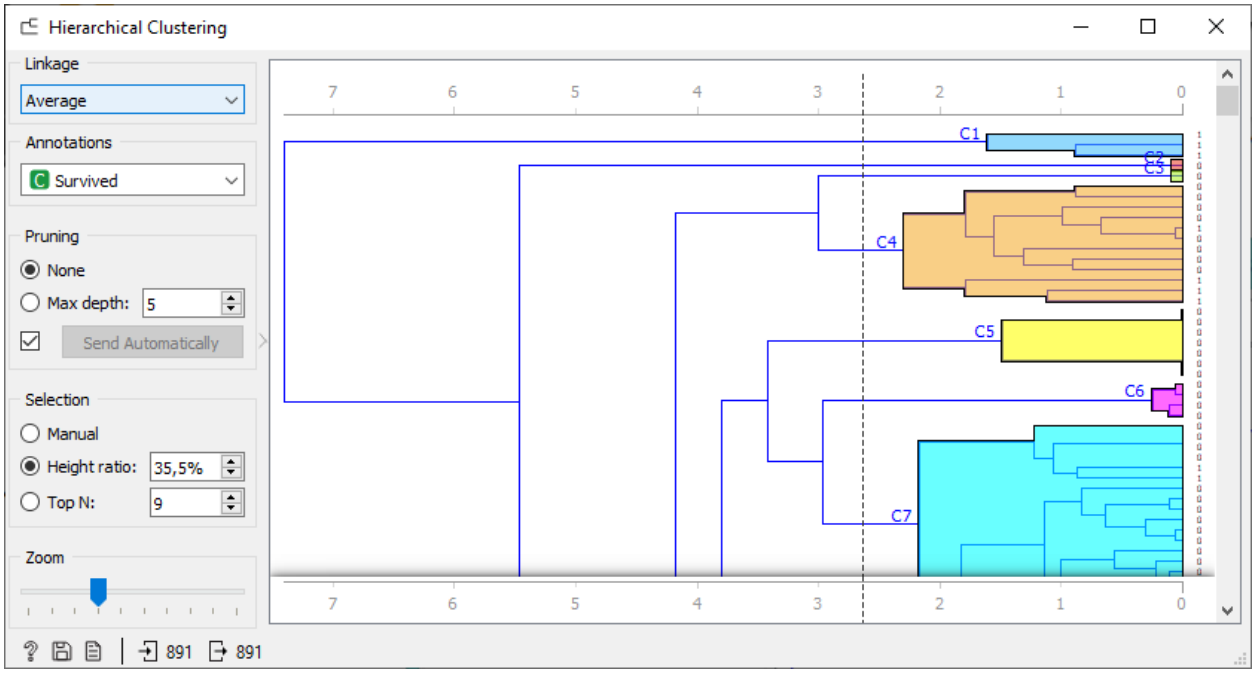


Рис. 58. Иерархическая кластеризация

Выделение кластеров в иерархической кластеризации можно выполнить следующими способами:

– Manual – вручную, выбрав поддерево в дендрограмме (удерживая клавишу Ctrl можно выбрать несколько кластеров);

– Height ratio – выбирая долю высоты дерева явно или с помощью линейки, расположенной сверху и снизу;

– Top N – выбирая указанное количество узлов верхней части дерева.

Результатом выделения кластеров является появление атрибута, указывающего кластер. На рис. 60 представлено распределение кластеров в искусственных координатах t-SNE.

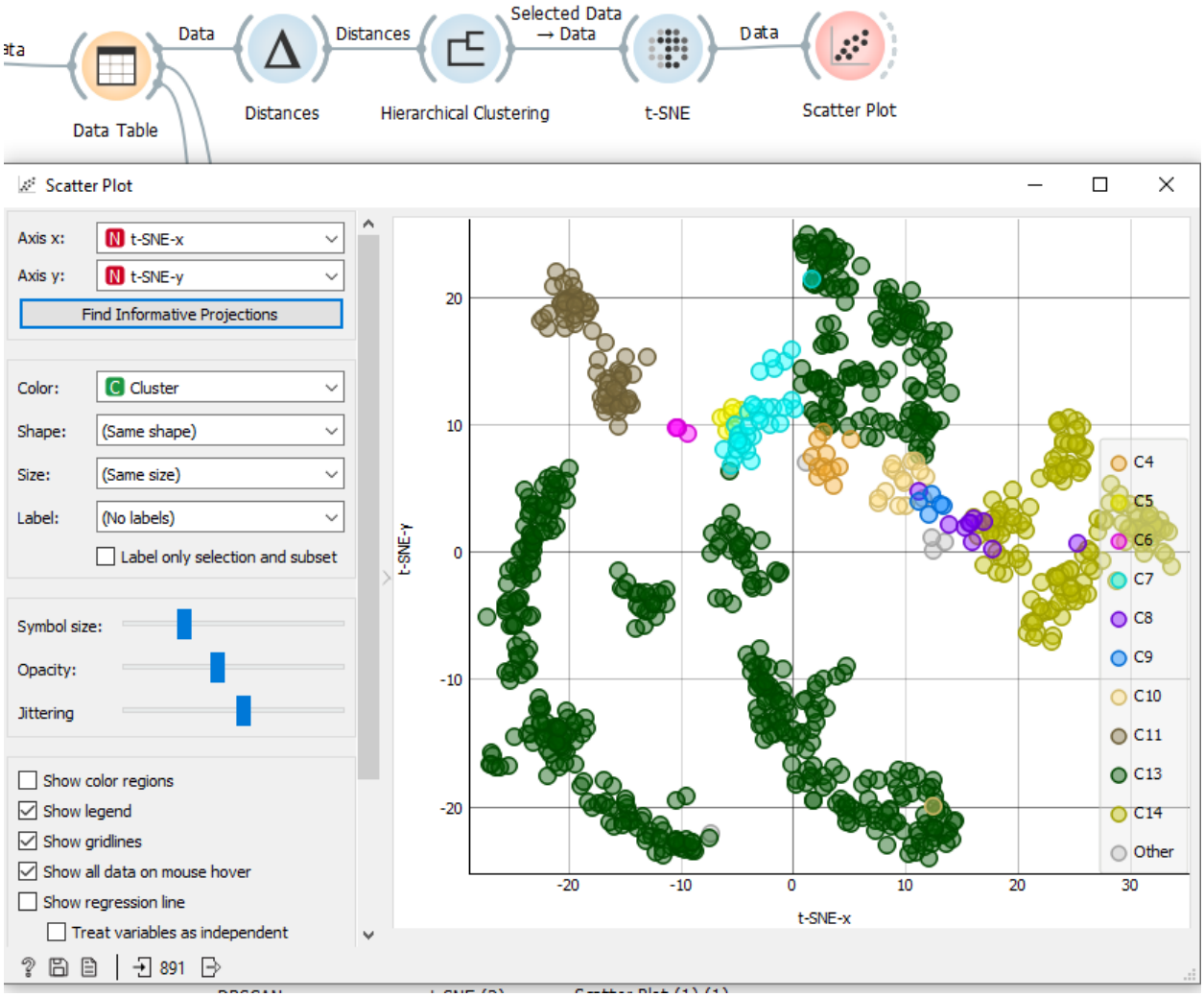


Рис. 59. Результат иерархической кластеризации

7.2. Кластеризация методом k-средних (k-Means)

Кластеризация методом k-средних объединяет объекты на основании расстояний от центров кластеров: минимизируется расстояние между объектами кластера и его центром и максимизируется расстояние между центрами кластеров. Метод не умеет подбирать оптимальное количество кластеров и получает это значение как параметр кластеризации (рис. 61). Другим вариантом работы данного виджета является задание интервала возможных значений количества кластеров и поиск подходящего варианта разбиения. Мерой качества разбиения на кластеры выбран показатель силуэта, характеризующий соотношение средних расстояний от центра кластера до внутренних и внешних его точек. Чем больше силуэт – тем лучше кластеризация. Это справедливо для выпуклых кластеров, чья геометрия близка к сферам. На рис. 61 максимум силуэта достигается для разбиения на 13 кластеров.

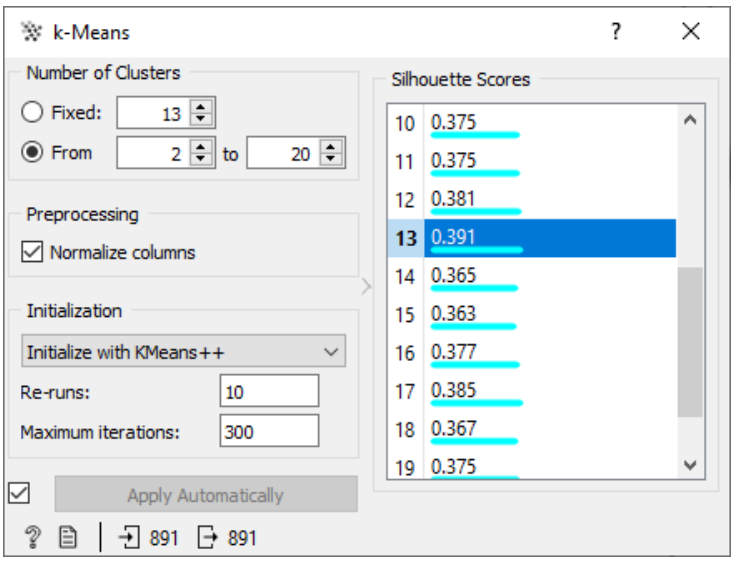


Рис. 60. Параметры кластеризации методом k-средних

Результаты кластеризации пассажиров Титаника представлены на рис. 62.

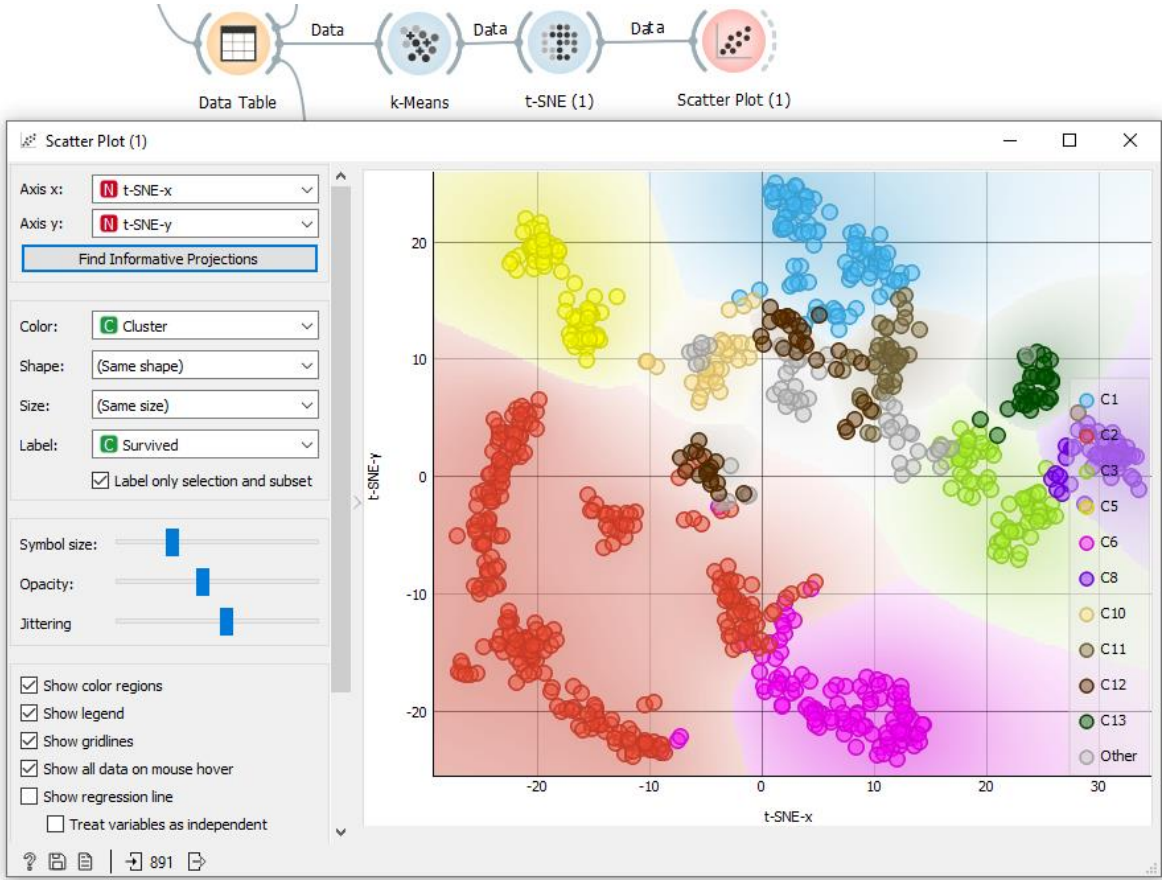


Рис. 61. Результаты кластеризации методом k-средних

7.3. DBSCAN – основанная на плотности пространственная кластеризация с выделением шума

Данный алгоритм исследует плотность расположения точек для выделения «шума» – точек, не отнесенных ни к одному кластеру в областях с низкой плот-

ностью точек и кластеров – областей с высокой плотностью. Алгоритм группирует вместе точки, которые тесно расположены (точки с k-близкими соседями), помечая как выбросы точки, которые находятся одиноко в областях с малой плотностью (ближайшие k-соседей которых лежат далеко).

Параметры алгоритма (рис. 63) устанавливают минимальное количество точек в кластере и k-расстояние – расстояние между точкой и k-й ближайшей точкой, а также способ вычисления расстояния. Данные параметры влияют на количество кластеров и уровень «шума». Для наглядности изображен график расстояния k-й ближайшей и количества точек с k-расстоянием большим, чем данное. Все точки с бóльшим k-расстоянием относятся к шуму (серые точки на рис. 64), прочие точки входят в какой-либо кластер.

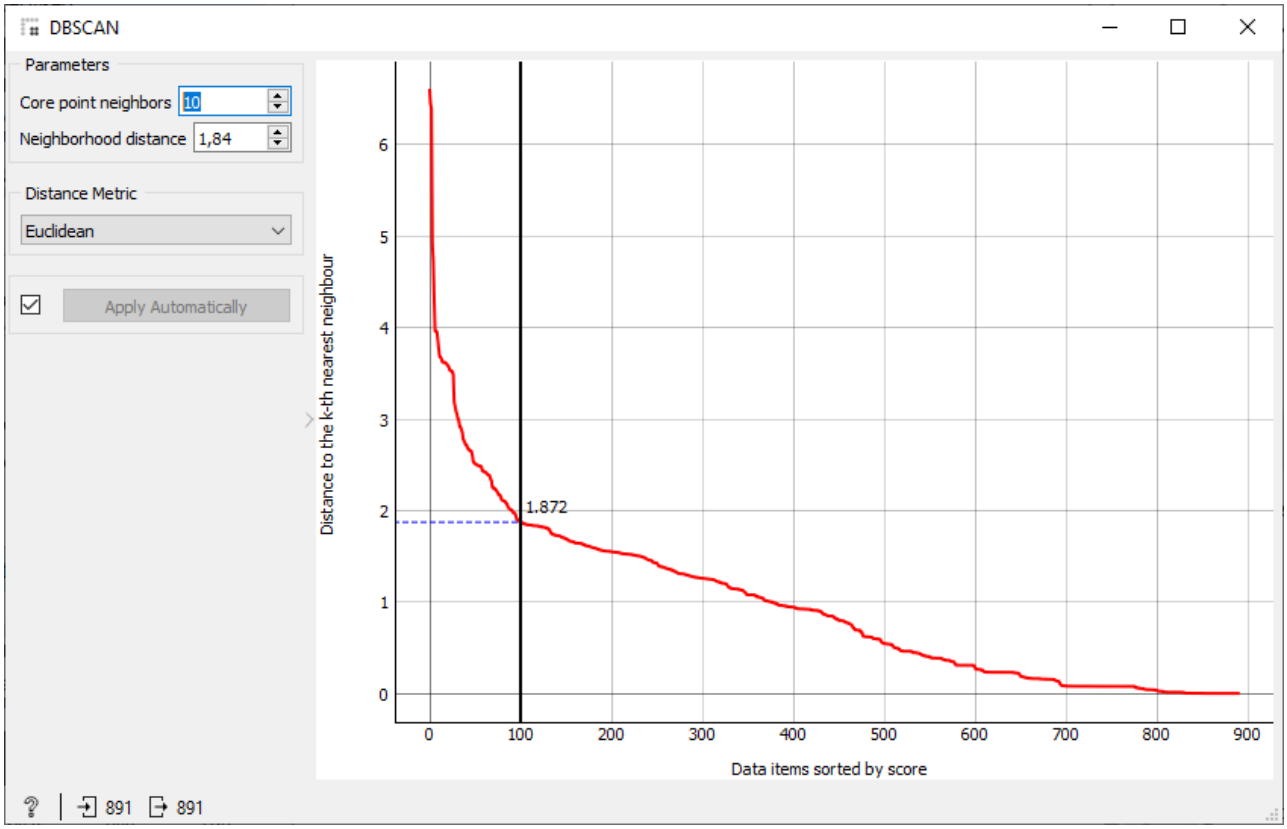


Рис. 62. Параметры кластеризации методом DBSCAN

Данный метод подходит для выделения «нешарообразных» кластеров. Мера силуэта может давать плохие результаты для кластеров DBSCAN из-за ее ориентации на кластеры точек, сгруппированных вокруг центров.

Для рассмотренного примера с Титаником можно отметить, что кластеры DBSCAN (рис.Рис. 63 64) похожи на кластеры, полученные другими методами. Это свидетельствует о том, что применение кластеризации для выбранных данных выделяет существующие группы.

Изучение характеристик объектов кластера может дать дополнительные сведения. На Рис. 64рис. 65 представлены доли выживших в разных кластерах: доля выживших мужчин в кластере C6 существенно больше среднего значения.

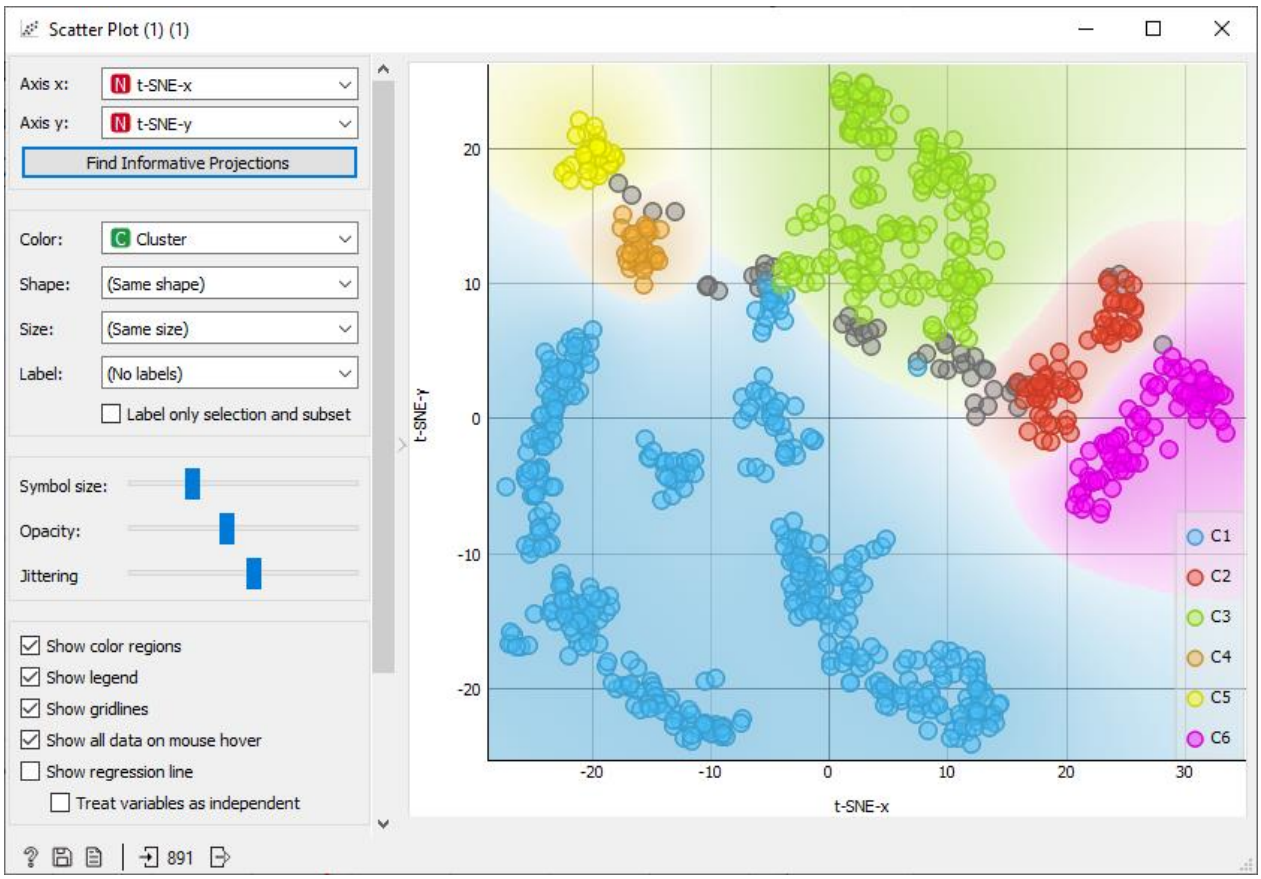


Рис. 63. Результаты кластеризации методом DBSCAN

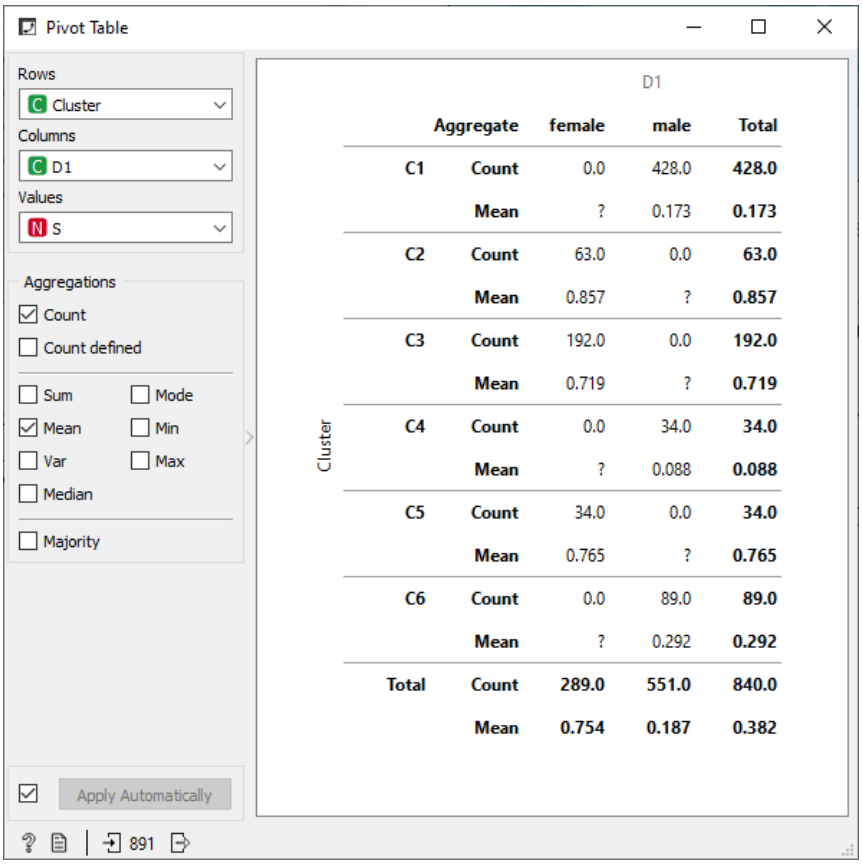


Рис. 64. Доли и числа выживших в катастрофе мужчин и женщин в кластерах

7.4. Выбросы

Выбросы – значения, не укладывающиеся в общую картину зависимостей, – встречаются достаточно часто. Наличие выбросов может сильно исказить зависимости, которые, как правило, носят усредненный характер. Поэтому в ряде случаев выбросы следует изучать отдельно или вообще исключать из исходных данных.

Самый простой способ выделения выбросов – это выделение определенной доли минимальных и / или максимальных значений. Методика выделения выбросов по долям не является универсальной. Поиск выбросов производился для значений одного поля, хотя следует находить выбросы для комбинаций полей.

Виджет «Outliers» (рис. 66) реализует четыре метода определения выбросов по совокупности значений переменных:

– One-class SVM – изменяет признаковое пространство и проводит разделяющую гиперплоскость так, чтобы наблюдения лежали как можно дальше от начала координат;

– Covariance estimator – находит отклонения – точки из области с низкой плотностью вероятности, предполагая многомерное распределение Гаусса;

– Local Outlier Factor – алгоритм сравнивает плотность в окрестности точки с плотностями соседних точек, отклонение свидетельствует о наличии выброса;

– Isolation Forest – при случайном построении дерева выбросы раньше попадают в листья

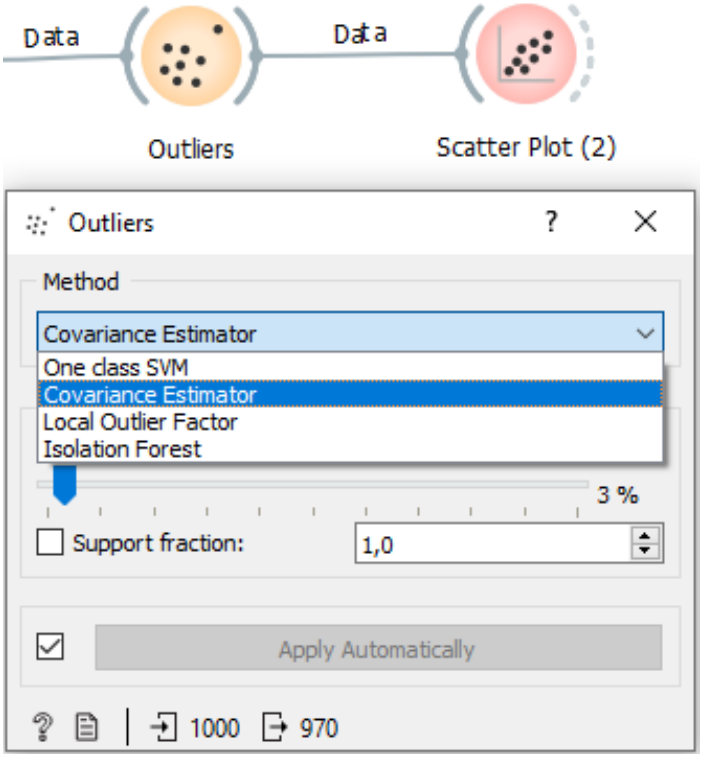


Рис. 65. Виджет «Outliers» определения выбросов

Результатом определения выбросов являются специальные метки объектов, которые можно наблюдать на диаграмме разброса (рис. 67). На рисунке

видно, что выбросы достаточно хорошо определяются большим расстоянием Махаланобиса.

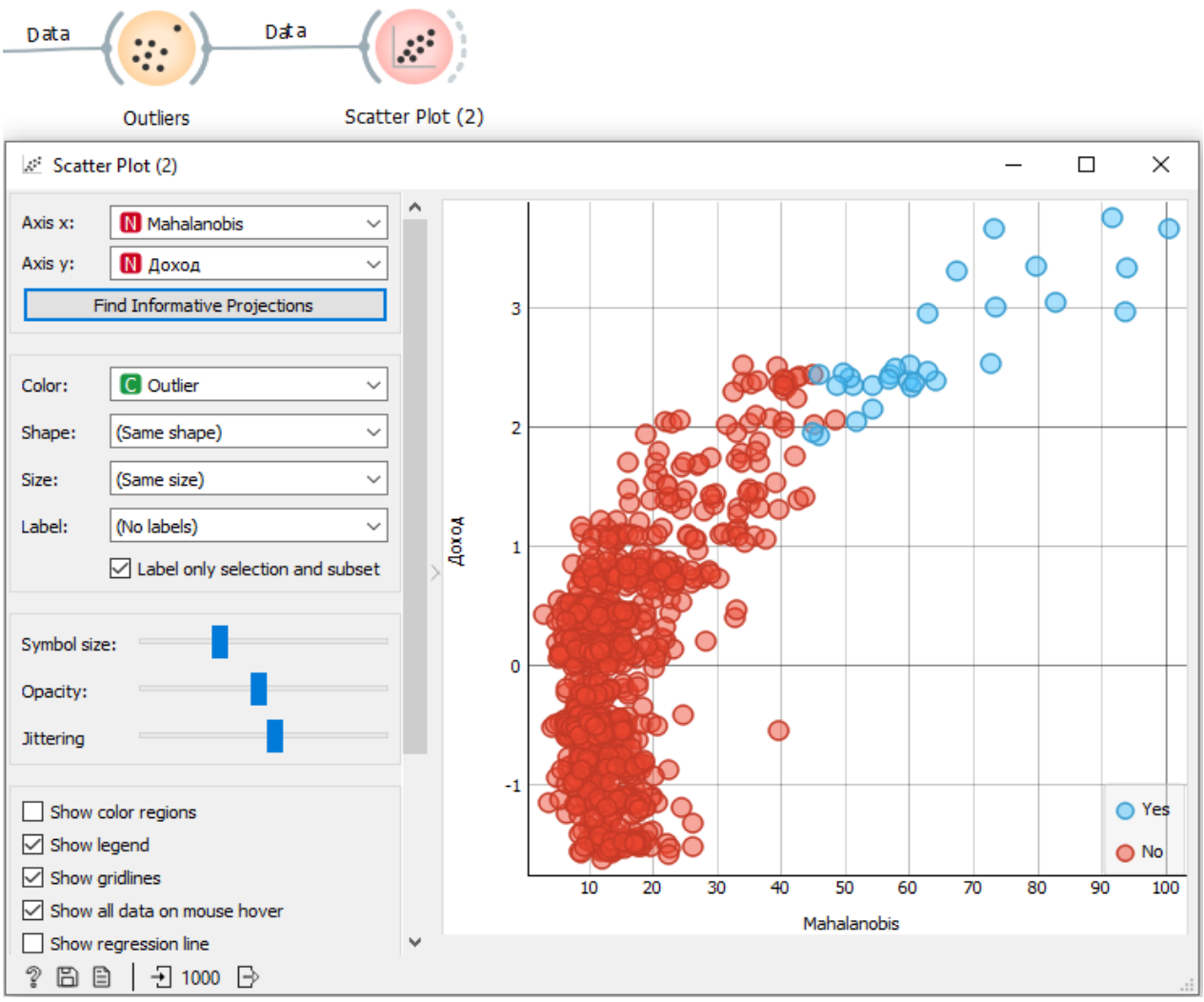


Рис. 66. Расположение выбросов

7.5. Решение задач кластеризации с использованием модулей библиотеки Scikit-Learn на языке Python

Для настройки моделей кластеризации практически всегда требуется предварительная обработка данных. Для исключения влияния шкал переменных выполняют преобразование к одной шкале или к шкалам, близким по размаху значений. Пустые значения часто интерпретируются как нули и тоже могут сильно повлиять на расстояния между объектами. Следующий фрагмент программы строит таблицу с исходными числовыми полями для кластеризации:

```
# Выбор данных для кластеризации (масштабированные числовые поля и перекодированные категориальные)
df_model = df_Titanic[['Survived', 'Pclass', 'StandardScaled_Age', 'SibSp', 'Parch', 'RobustScaled_Fare', 'Sex_id', 'Embarked_id']].
```

Сложным является вопрос выбора метрик расстояния, так как разные метрики могут приводить к разным разбиениям на кластеры. В выборе показателей

качества кластеризации еще больше неопределенности[12]. Каждая метрика лучше работает для одних конфигураций кластеров (например, выпуклых) и хуже для других (например, со сложными, изогнутыми границами).

Метрика «Силуэт» (Silhouette) позволяет оценить качество кластеризации, используя только результат кластеризации, без привлечения других данных о распределении объектов по кластерам. Сначала силуэт определяется отдельно для каждого кластера. Обозначим через a – среднее расстояние от данного объекта до объектов из того же кластера, через b – среднее расстояние от данного объекта до объектов из ближайшего кластера (отличного от того, в котором лежит сам объект). Тогда силуэтом данного объекта называется величина:

$$s = \frac{b-a}{max(a,b)}.$$

Силуэтом выборки называется средняя величина силуэта объектов данной выборки. Таким образом, силуэт показывает, насколько среднее расстояние до объектов своего кластера отличается от среднего расстояния до объектов других кластеров. Данная величина лежит в диапазоне [−1, 1]. Чем больше силуэт, тем более четко выделены кластеры. Значения близкие к –1 свидетельствуют о плохой кластеризации (выпуклой), близкие к 0 – о перекрывающихся кластерах, близкие к 1 говорят о хорошем выделении кластеров. Данная мера лучше подходит для выпуклых кластеров.

Индекс Calinski–Harabasz (CH) или Variance Ratio Criterion (VRC) by Calinski and Harabasz является широко применяемым критерием для n наблюдений и выделенных k-кластерах:

$$CH(k) = \frac{B(k)}{k-1} \Big/ \frac{W(k)}{n-k},$$

где $B(k)$ – сумма расстояний центроидов кластеров от центра всех точек («разделимость»); $W(k)$ – сумма расстоянии от точек кластеров до их центроидов («компактность»), т. е. индекс находит отношение между «разделимостью» и «компактностью». Максимальное значение индекса соответствует лучшей кластеризации.

Ни одна из метрик не дает совершенно точных результатов, позволяющих утверждать, что найдено наилучшее распределение по кластерам. Например, приведенные метрики плохо оценивают кластеры со сложной геометрией.

Для проведения кластеризации создается соответствующий программый объекта и выполняется метод fit (табл. 10).

Таблица 10

Модели и объекты, применяемые для кластеризации

Название модели	Объект, выполняющий кластеризацию и основные параметры
Кластеризация методом k-средних	KMeans(n_clusters=)
Алгомеративная кластеризация	AgglomerativeClustering(n_clusters=)
Кластеризация DBSCAN	DBSCAN(eps=, min_samples=)
Кластеризация с использованием смеси распределений Гаусса	GaussianMixture(n_components=)
Birch-кластеризация	Birch(branching_factor=50, n_clusters=, threshold=,compute_labels=True)

[12] URL: http://neerc.ifmo.ru/wiki/index.php?title= Оценка качества в задаче кластеризации.

Основным параметром большинства моделей является n_clusters – количество кластеров. Результат кластеризации отражает свойство labels_ – набор номеров кластеров, которые можно использовать для изучения свойств полученных кластеров. Более подробную информацию можно найти в документации[13].

Среди методов кластеризации выделяется иерархическая кластеризация, она требует предварительного вычисления попарных расстояний между точками. Для ее выполнения импортируются соответствующие компоненты:

```
from scipy.cluster.hierarchy import linkage, dendrogram
from scipy.cluster.hierarchy import fcluster.
```

Извлекаем измерения как массив NumPy – исходные данные для вычисления расстояний между объектами:

```
samples = df_model.values.
```

Реализация иерархической кластеризации выполняется при помощи функции linkage

```
mergings = linkage(samples, method='complete').
```

По результатам построения дерева определяются кластеры

```
groups = fcluster(mergings, 5, criterion='maxclust').
```

Для визуального контроля можно построить дендрограмму (рис. 68)

```
dendrogram(mergings,)
    truncate_mode='level', p=5,
    labels=groups,
    leaf_rotation=200,
    leaf_font_size=6,
    )
plt.savefig(workdir+"дендрограмма.png").
```

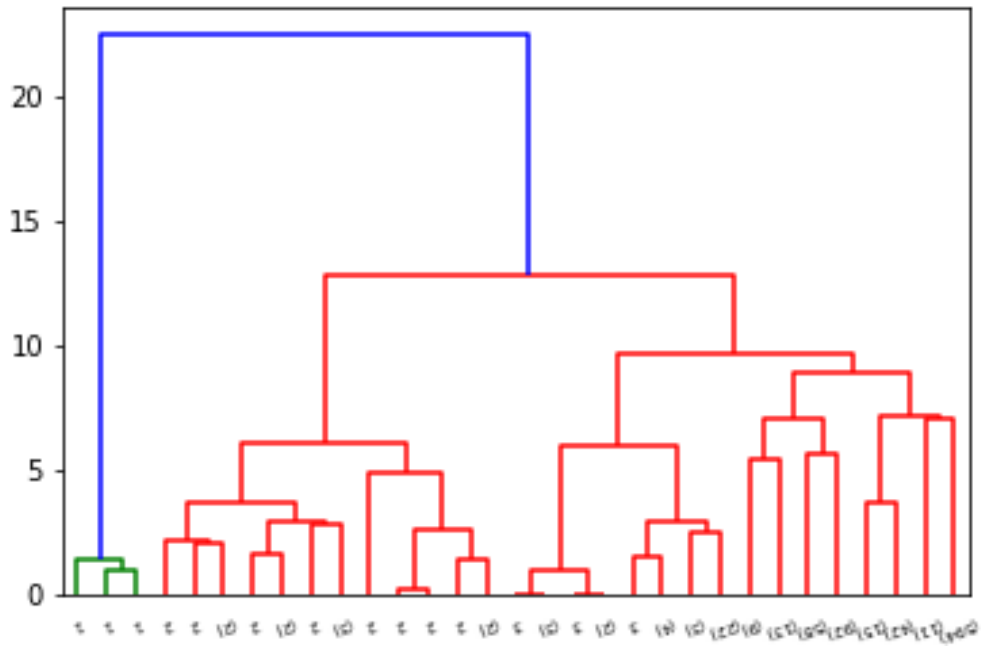


Рис. 67. Дендрограмма иерархической кластеризации

[13] URL: https://scikit-learn.org/stable/modules/clustering.html.

Можно сохранить номера кластеров в исходной таблице

```
df_Titanic['hierarchy'] = groups
```

и вычислить метрики кластеризации

```
silhouette_score=metrics.silhouette_score(df_model, groups, metric='euclidean')
calinski_harabaz_score=metrics.calinski_harabaz_score(df_model, groups).
```

Модели кластеризации (кроме иерархической) используют метод fit(<таблица наблюдений>) для выделения кластеров. В приведенном ниже фрагменте программы выполняется кластеризация методом k-средних. В цикле перебираются значения количества кластеров для поиска наилучшего по двум критериям качества кластеризации.

```
from  sklearn.cluster  import KMeans
silhouette_best=[] # наилучшие параметры для метрики силуэта
calinski_best=[] # наилучшие параметры для метрики Calinski–Harabasz
for n_clusters in [3,4,5,7,10, 15]: # перебор значений параметра "количество кластеров"
    # создание объекта – модель кластеризации
    kmeans_model = KMeans( n_clusters=n_clusters)
    # обучение модели
    kmeans_model.fit(df_model)
    # сохранение лучших показателей кластеризации в списках
    if len(Counter(kmeans_model.labels_))>1:
        silhouette_score=metrics.silhouette_score(df_model, kmeans_model.labels_,
        metric='euclidean')
        calinski_harabaz_score=metrics.calinski_harabaz_score(df_model,
        kmeans_model.labels_)
        metrics_list.extend([[n_clusters, silhouette_score, calinski_harabaz_score]])
        if len(silhouette_best)==0:
            silhouette_best=[n_clusters,silhouette_score]
            calinski_best=[n_clusters,calinski_harabaz_score]
        else:
            if silhouette_best[1]<silhouette_score:
                silhouette_best=[n_clusters,silhouette_score]
            if calinski_best[1]<calinski_harabaz_score:
                calinski_best=[n_clusters,calinski_harabaz_score].
```

Аналогично можно выполнить кластеризацию с мопощью других моделей. Полный текст программы, включающий кластеризацию разными методами приведен в Приложении 1.

Различные методы дают разные, но похожие результаты. Из табл. 11 видно, что 656 пассажиров отнесены к кластеру 0 методом k-средних, и к кластеру 1 методом Birch, т. е. эти два кластера, полученные разными методами, практически совпадают. Полностью совпадают кластер 1 k-средних и кластер 0 метода Birch. Хуже всего определен кластер 3 k-средних – ему соответствует кластер 1 (19 совпадений) и кластер 3 (37 совпадений) метода Birch.

Таблица 11

Кластеры количества пассажиров,
полученные двумя разными методами

Метод k-средних	Метод Birch				По строке
	0	1	2	3	
0		656	24		680
1	20				20
2		8	127		135
3		19		37	56
По столбцу	20	683	151	37	891

Полезность кластеров можно определить, изучая их поведение и значения целевых показателей в среднем по кластеру. Например, можно определить долю выживших в каждом кластере. Табл. 12 демонстрирует повышение доли выживших (до 0,7) в кластерах 1 и 2. Принадлежность пассажира одному из этих кластеров позволяет сделать соответствующий вывод в отношении шансов его выживания.

В табл. 12 содержатся показатели качества кластеризации. По индексу «Calinski–Harabasz» лучшим является метод k-средних, а по индексу силуэта – метод DBSCAN, хотя он является одновременно самым плохим по индексу «Calinski–Harabasz» и выделил с указанными параметрами всего один кластер и «шум».

Таблица 12

Показатели качества кластеризации разными методами

Метод	Индекс «Calinski – Harabasz»	Индекс силуэта
K-средние (n_clusters=4)	377,3938	0,414221
Агломеративная кластеризация (n_clusters=4)	362,9538	0,411661
DBSCAN(eps=4.0, min_samples=4)	171,1201	0,809059
Смесь гауссовских распределений (n_components=2)	301,9441	0,375247
Birch (branching_factor=50, n_clusters=4, threshold=0.1)	362,0616	0,398479

8. Анализ временных рядов

8.1. Временные ряды

Исходными данными для анализа временных рядов являются упорядоченные по времени данные. В простейшем случае это может быть один ряд $y_1, \dots, y_n, \dots$ наблюдений некоторой величины в разные моменты времени t. Модели рядов основаны на предположении, что все наблюдения сформировались под действием одних и тех же факторов, которые сохранятся такими же в прогнозируемом периоде.

Один из подходов к изучению временного ряда заключается в разделении его на случайную и детерминированную составляющие. Чаще всего используют две такие модели:

– $y_t = d_t + \varepsilon_t$ – аддитивная модель;

– $y_t = d_t \cdot \varepsilon_t$ – мультипликативная модель.

На самом деле между этими моделями нет существенной разницы: выполнив логарифмическое преобразование мультипликативной модели, получаем аддитивную модель.

Под случайной составляющей подразумевают стационарный случайный ряд, все компоненты которого распределены по одинаковому вероятностному закону. В самом простом случае случайную составляющую образуют независимые и одинаково распределенные случайные величины. В этом случае в аддитивной модели случайная составляющая часто интерпретируется как ошибка измерения. Независимые случайные величины образуют последовательность, в которой значения ряда изменяются скачками. Если наблюдаются плавные изменения, то необходимо применять для случайной составляющей более сложные стационарные модели. Стационарные случайные последовательности определяют для каждого участка последовательности одинаковое распределение вероятности – между элементами может существовать статистическая зависимость, определяющая плавность изменения значений и не изменяющаяся при сдвиге вдоль ряда.

Как и любое другое исследование изучение временных рядов начинается с получения данных. Можно использовать любой виджет, возвращающий таблицу с рядами. Виджеты для работы с рядами сосредоточены на панели «Time Series».

Для начала можно просто построить графики. Виджет «Line Chart» строит такие графики. На рис. 69 представлен помесячный ряд производства вин в Австралии.

Детерминированную составляющую делят на тренд, сезонную составляющую и циклическую составляющую:

$$d_t = r_t + s_t + c_t.$$

Тренд r_t – медленно меняющаяся составляющая ряда, которая описывает влияние на временной ряд долговременно действующих факторов, вызывающих плавные и длительные изменения ряда. Сезонная составляющая s_t временного ряда описывает изменения его значений в пределах некоторого периода и представляет собой последовательность повторяющихся циклов одинаковой протяженности. Циклическая составляющая c_t временного ряда состоит из интервалов подъема и спада, которые имеют различную протяженность, а также различную амплитуду расположенных в них значений. Наличие в рядах данных циклических компонент связано с тем, что в пределах интервалов более глобальных изменений (чем, например, сезонных) могут наблюдаться не имеющие периодичности временные подъемы и спады, которые, в отличие от случайной компоненты, не вызваны действием случайных факторов, а являются особенностями наблюдаемого процесса. Циклическая составляющая в Orange может быть представлена в составе тренда.

На рис. 70 ряд производства вин в Австралии разделен виджетом «Seasonal Adjustment» на периоды по 12 месяцев (параметр Season period) с использованием аддитивной модели. Модель достаточно явно выделила циклическую составляющую и тренд. Об этом свидетельствует меньший по амплитуде ряд остатков (residual).

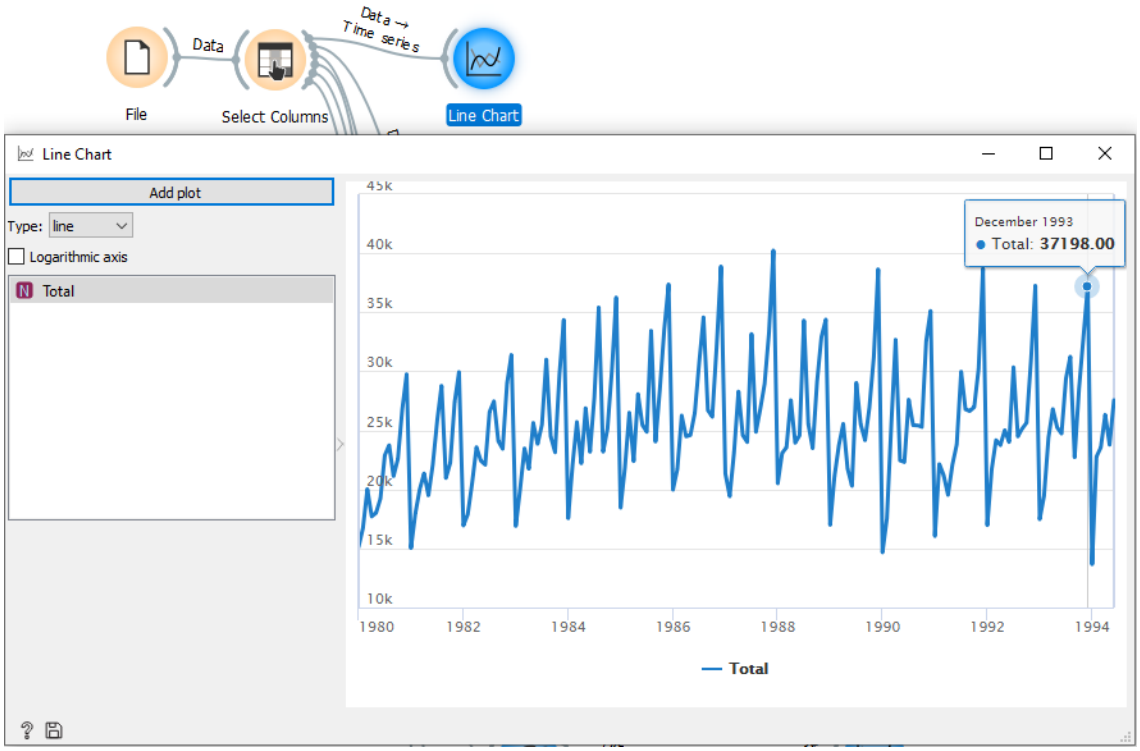


Рис. 68. График ряда, построенный виджетом «Line Chart»

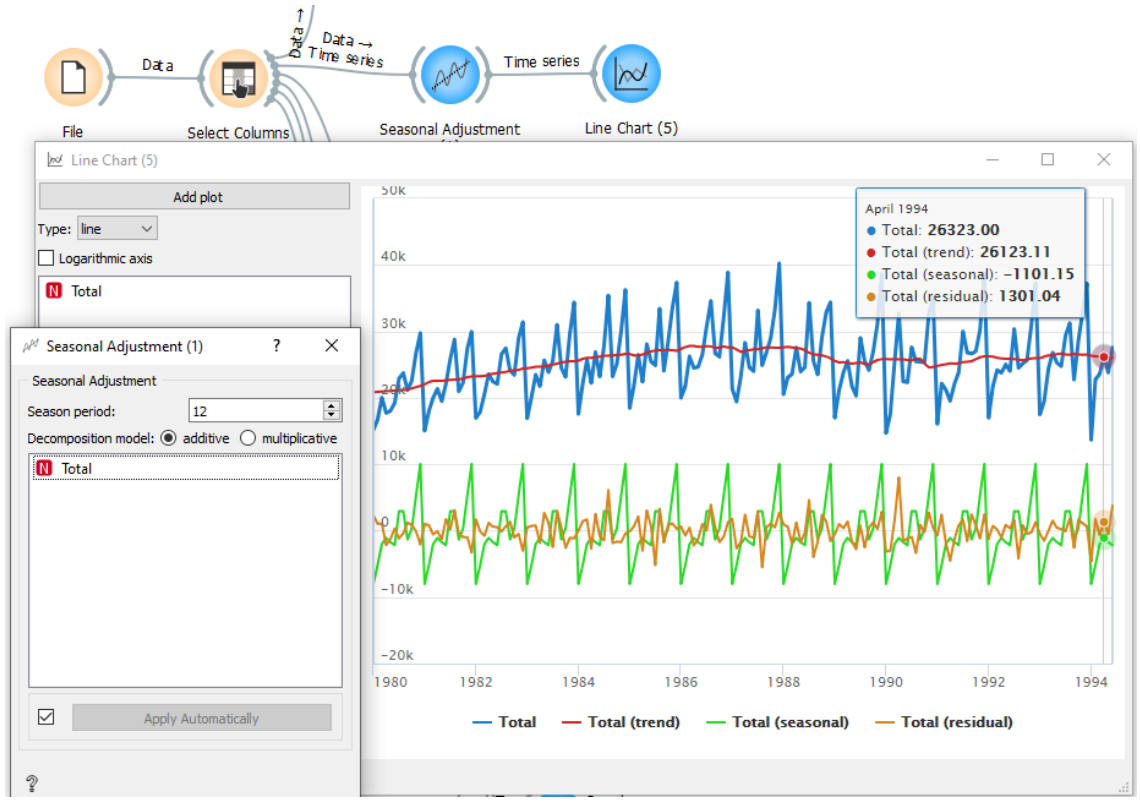


Рис. 69. Выбор периода виджетом «Seasonal Adjustment» и построение соответствующих графиков виджетом «Line Chart»

8.2. Преобразования рядов

Инструментом сглаживания случайных колебаний и выделения на этой основе тренда или других закономерностей является применение скользящего

окна. Окно определяется количеством значений ряда. По значениям, попавшим в окно, вычисляется сводное значение, как правило, среднее, после чего окно смещается на одну позицию ряда, и процедура повторяется. В результате получается новый ряд агрегированных значений, в котором случайные колебания сглаживаются. Полученный ряд дает более точное представление о тренде. Виджет «Moving Transform» (рис. 71) реализует алгоритм скользящего окна и позволяет устанавливать размер окна и функцию агрегирования (см. выпадающий список). На рисунке видно, что сглаженные скользящим усреднением колебания дают достаточно точное представление о тренде.

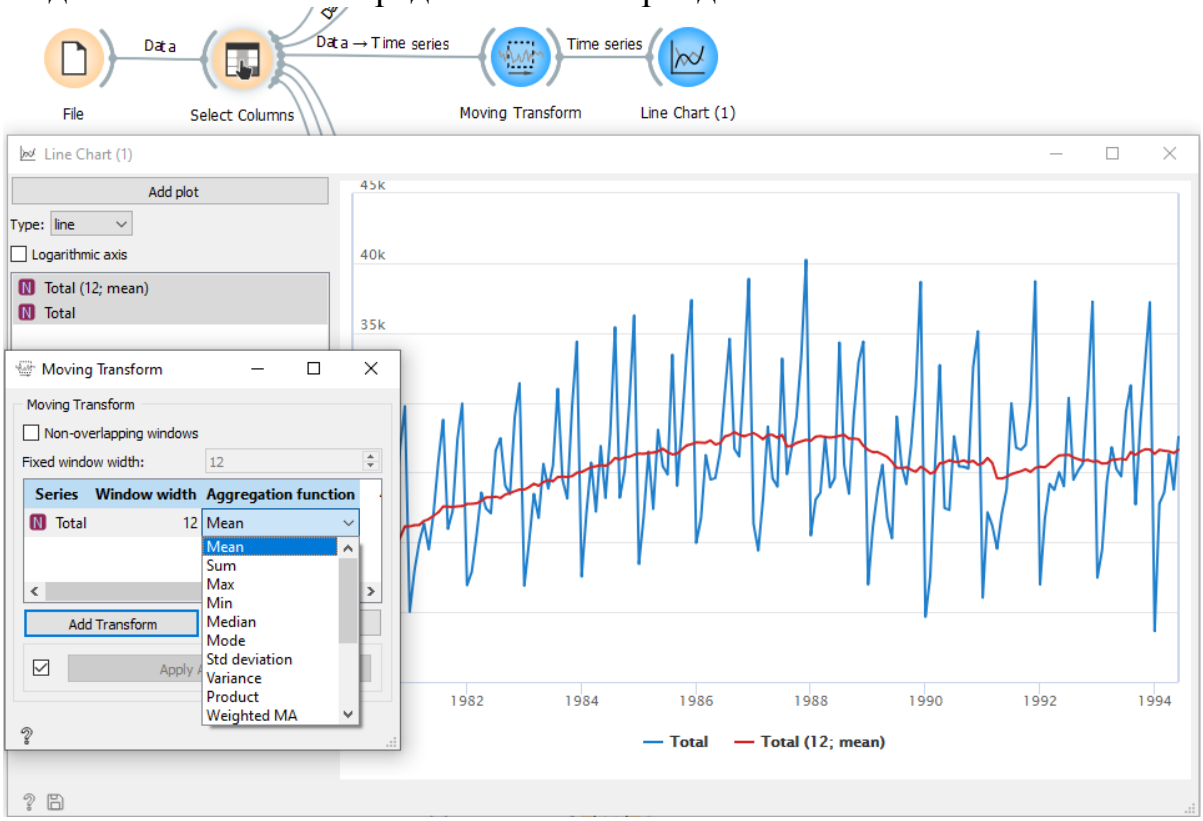


Рис. 70. Сглаживание ряда виджетом «Moving Transform» (скользящее окно)

Скользящее окно сглаживает ряд, но не приводит к получению стационарного ряда. Временной ряд называется стационарным, если его статистические свойства (математическое ожидание, дисперсия, зависимости элементов ряда) одинаковы на всем протяжении ряда. В противном случае ряд называется нестационарным. Применение к нестационарным рядам различных методов анализа, в том числе статистических, затруднено. Поэтому, прежде чем приступать к построению модели ряда, его стараются свести к стационарному. Достаточно часто стационарный ряд образуют приращения ряда (разности первого порядка):

$$\nabla^1(y_t) = y_t - y_{t-1},$$

или приращения приращений (разности второго порядка) –

$$\nabla^2(y_t) = \nabla^1(y_t) - \nabla^1(y_{t-1}) = \nabla^1\left(\nabla^1(y_t)\right) = y_t - 2y_{t-1} + y_{t-2}.$$

В Orange разностные ряды строятся виджетом «Difference» (рис. 72). Его параметрами являются вид разности (параметр Compute), порядок разности (параметр Differencing order) и сдвиг второго компонента разности относительно первого (параметр Shift). Рисунок демонстрирует, что колебания происходят на

уровне оси (исключен тренд), увеличился размах колебаний. Таким образом, разностный ряд может быть с большим основанием отнесен к стационарному.

Рис. 71. Ряды разностей первого и второго порядка,
полученные применение виджета «Difference»

8.3. Характеристики рядов

Виджет «Periodogram» предназначен для определения периодов колебаний. Входной ряд представляется суммой гармонических колебаний, для каждой гармоники вычисляется ее амплитуда. В результате получается график (рис. 73) по которому можно определить период колебания с наибольшей амплитудой.

Исследование корреляционных зависимостей между компонентами ряда выполняется с помощью автокорреляционной функции:

$$r(\tau) = r[y_t, y_{t+\tau}],$$

которая показывает как меняется коэффициент корреляции при увеличении τ – смещения между компонентами ряда. Автокорреляционную функцию строит виджет «Correlogram» (рис. 74). График демонстрирует достаточно сильную положительную зависимость для смещения 12 (месяцев) $r(\tau) = 0{,}86$. Слабое затухание автокорреляционной функции свидетельствует о нестационарности ряда.

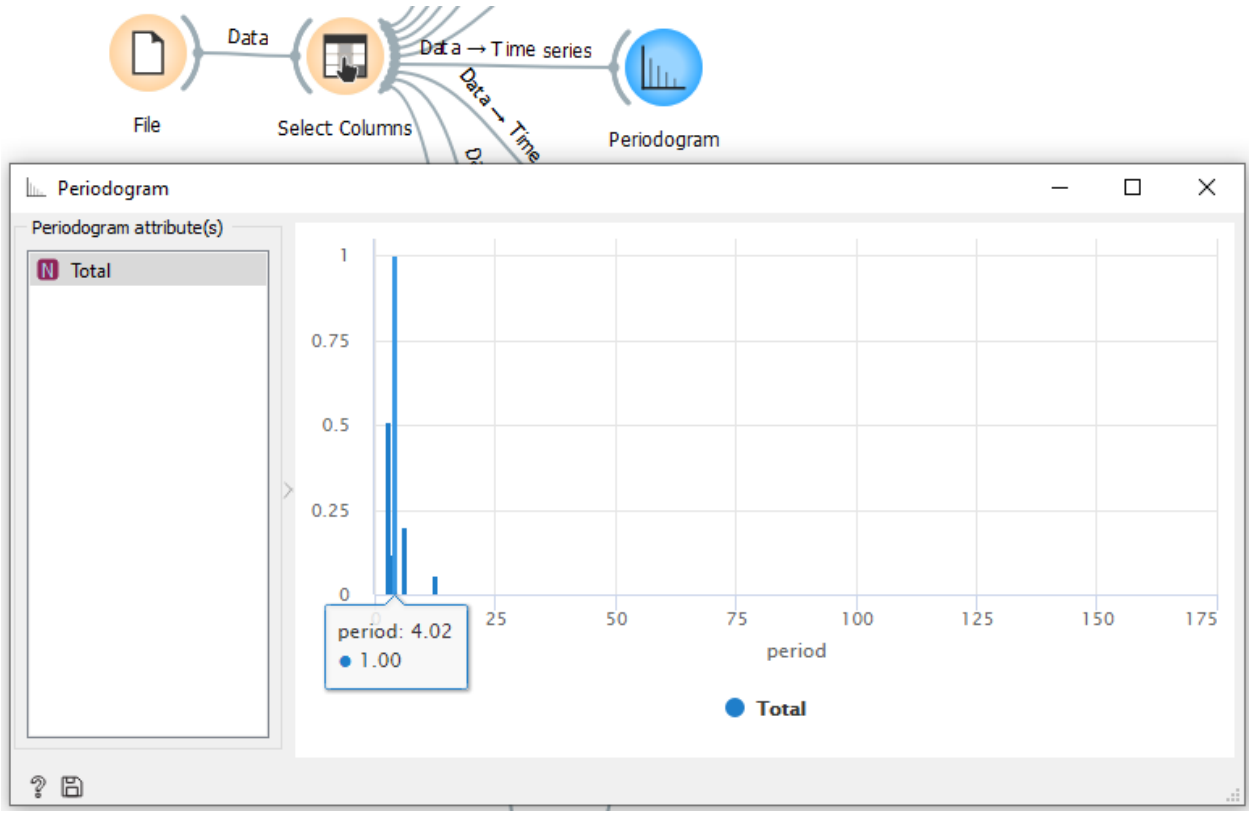


Рис. 72. Амплитуды гармонических колебаний,
вычисленные виджетом «Periodogram»

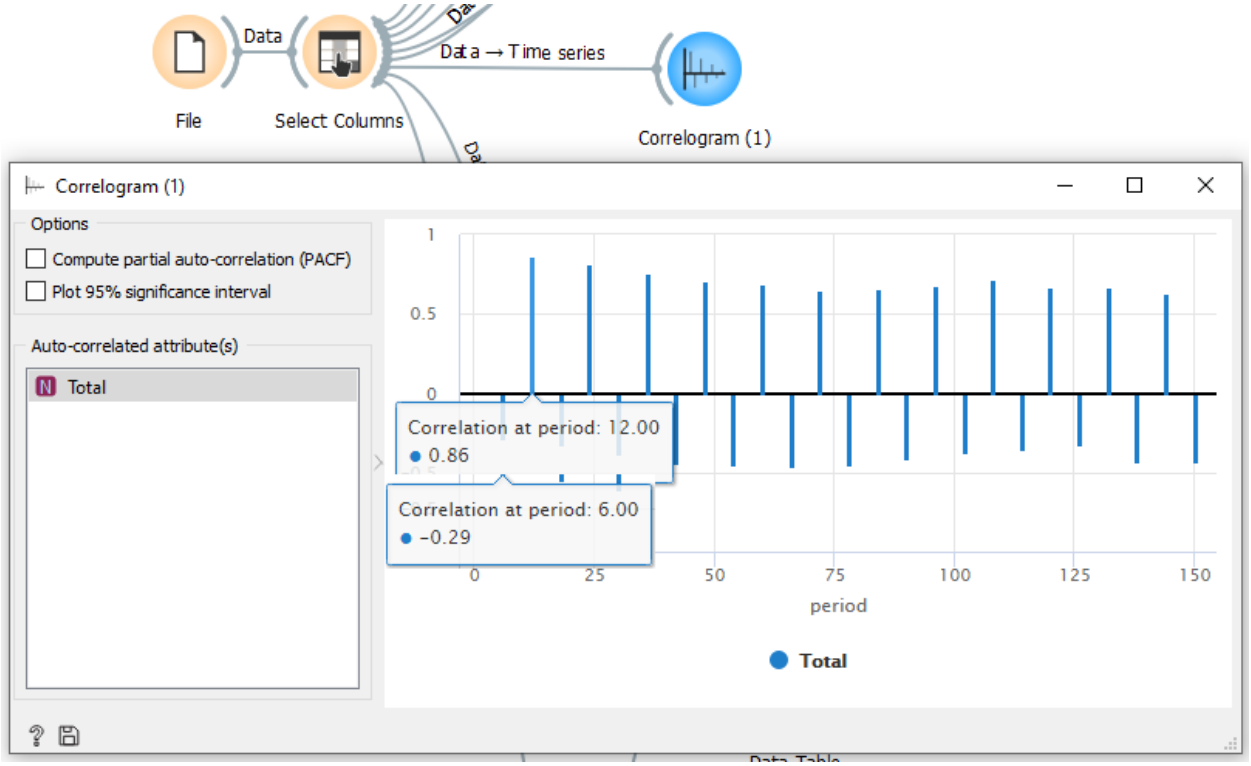


Рис. 73. Автокорреляционная функция, построенная виджетом «Correlogram»

8.4. Модели рядов

Модель авторегрессии появилась как обобщение правила, в котором новое значение вычислялось как предыдущее, увеличенное на некоторую случайную

величину ε_t. В общем случае учитывается p предыдущих значений с соответствующими весовыми коэффициентами:

$$y_t = a_1 y_{t-1} + \ldots a_p y_{t-p} + \varepsilon_t.$$

Такую модель авторегрессии порядка p обозначают как *AR(p)*.

Модель скользящего среднего является обобщением идеи сглаживания значений ряда усреднением q подряд идущих значений. В результате такого сглаживания случайные отклонения усредняются и полученный ряд точнее описывает неслучайные закономерности. Модель скользящего среднего *MA(q)* описывается уравнением:

$$y_t = \varepsilon_t - b_1 \varepsilon_{t-1} - \ldots - b_q \varepsilon_{t-q}.$$

Объединение этих моделей называют моделью авторегрессии и скользящего среднего *ARMA (p, q)*:

$$y_t = a_1 y_{t-1} + \ldots a_p y_{t-p} + \varepsilon_t - b_1 \varepsilon_{t-1} - \ldots - b_q \varepsilon_{t-q}.$$

Эта модель при соблюдении определенных ограничений на коэффициенты описывает стационарные случайные процессы.

Для приведения нестационарного процесса к стационарному применяют вычисление разности Δ^d порядка d:

$$\nabla^1(y_t) = y_t - y_{t-1},$$
$$\nabla^2(y_t) = \nabla^1\left(\nabla^1(y_t)\right) = y_t - 2y_{t-1} + y_{t-2},$$

до тех пор, пока не получат стационарный процесс, который моделируют с помощью модели *ARMA (p, q)*. Полученная модель получила название модель авторегрессии и интегрированного скользящего среднего *ARIMA(p, d, q)* – autoregressive integrated moving average.

В Orange для моделирования и прогнозирования ряда предусмотрен виджет «ARIMA Model» (рис. 75). Параметрами виджета являются параметры p, q модели, доверительная вероятность (Confidence interval) и количество прогнозируемых значений ряда (Forecast steps ahead). Для визуализации прогноза можно применить Line Chart. Прогноз на графике изображен пунктирной линией.

Кроме того, можно получить прогноз в виде таблицы, используя Data Table. В таблице (рис. 76) будет представлен прогноз (forecast), нижняя граница доверительного интервала (low) и его верхняя граница (high). Для приведенного примера разброс получается достаточно большим.

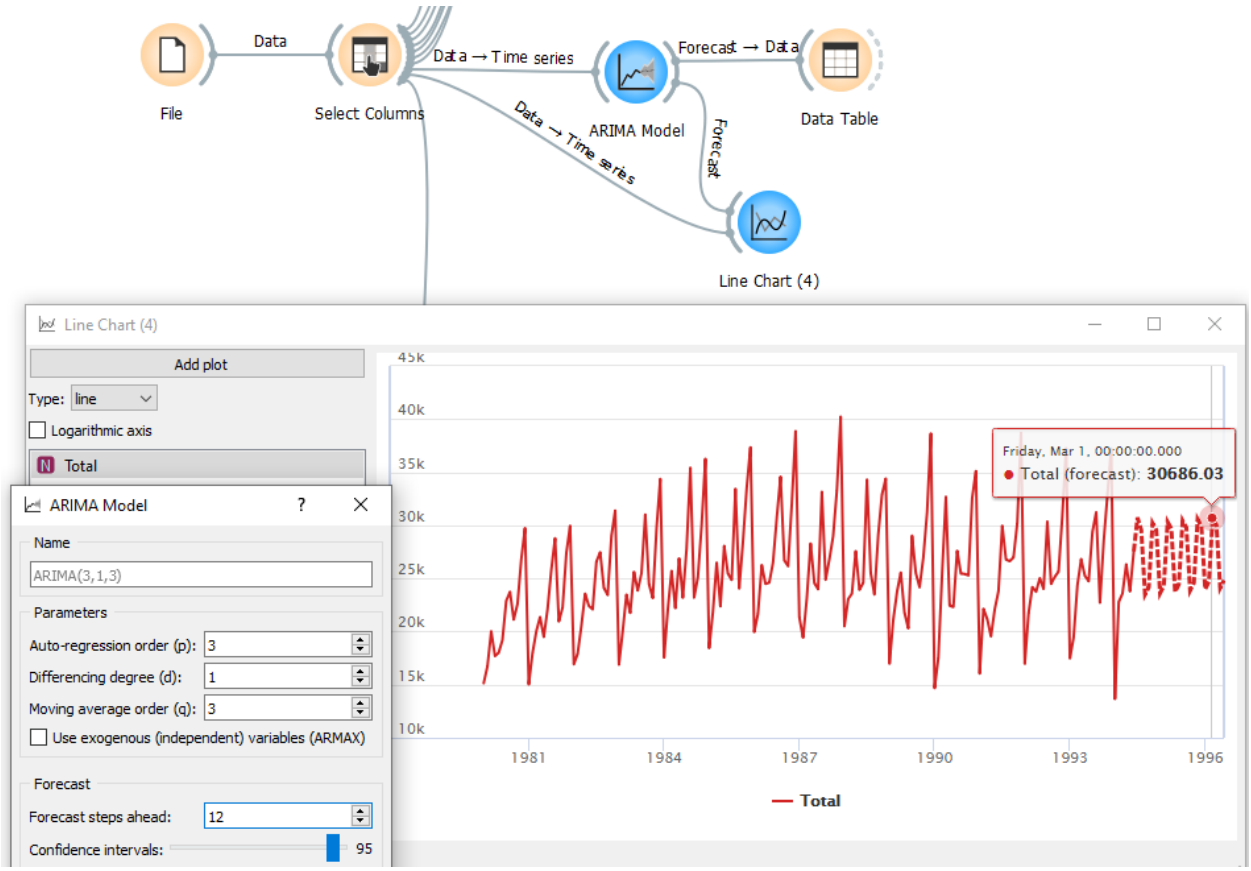
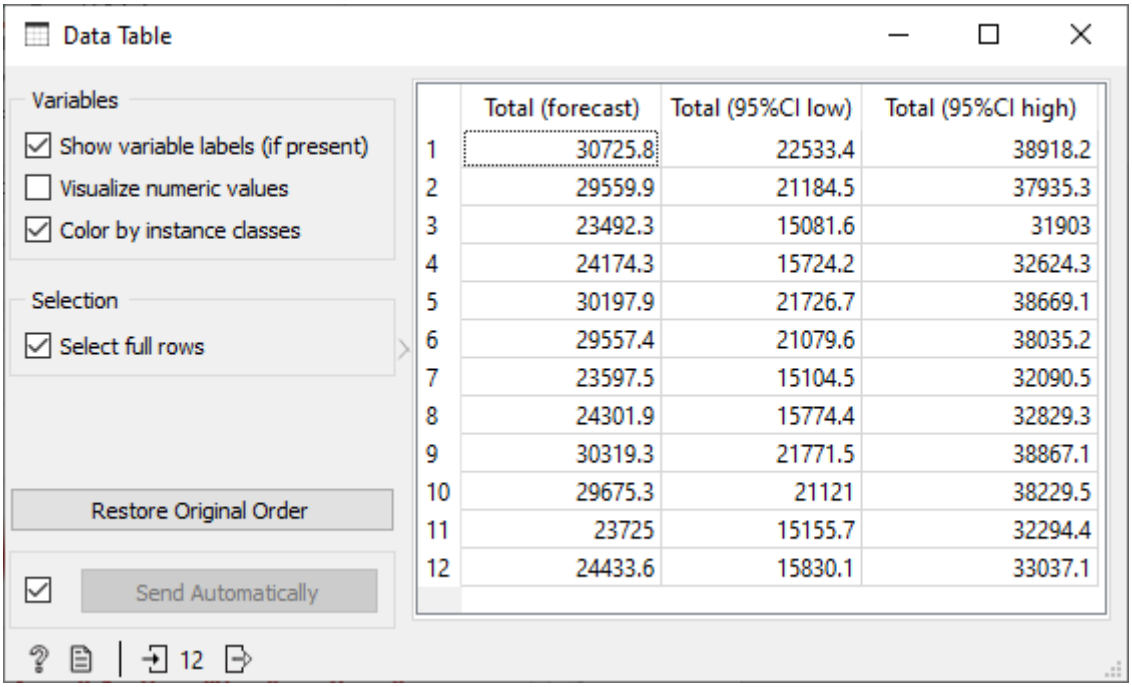


Рис. 74. Моделирование и прогнозирование ряда виджетом «ARIMA Model»



	Total (forecast)	Total (95%CI low)	Total (95%CI high)
1	30725.8	22533.4	38918.2
2	29559.9	21184.5	37935.3
3	23492.3	15081.6	31903
4	24174.3	15724.2	32624.3
5	30197.9	21726.7	38669.1
6	29557.4	21079.6	38035.2
7	23597.5	15104.5	32090.5
8	24301.9	15774.4	32829.3
9	30319.3	21771.5	38867.1
10	29675.3	21121	38229.5
11	23725	15155.7	32294.4
12	24433.6	15830.1	33037.1

Рис. 75. Табличное представление прогноза

Специальный виджет «Model Evaluation» позволяет сравнить качество моделирования (рис. 77). На вход виджета передаются модели и исходный ряд. Сравнение моделей основано на сопоставлении следующих характеристик:

– RMSE (root mean squared error) – среднеквадратическая ошибка;

– MAE median absolute error – медиана абсолютной ошибки;

– MAPE mean absolute percent error – доля MAE;

– POCID prediction of change in direction – предсказание изменения направления;

– R^2 coefficient of determination – коэффициент детерминации (доля дисперсии, объясняемой моделью);

– Akaike information criterion (AIC) – критерий не только вознаграждает за качество приближения, но и штрафует за использование излишнего количества параметров модели (считается, что наилучшей будет модель с наименьшим значением критерия AIC, в общем случае AIC: $AIC = 2k - 2\ln(L)$, где k – число параметров в модели, L – максимальное значение функции правдоподобия);

– BIC Bayesian information criterion – информационный критерий на основе баейсовского подхода, лучшая модель дает меньшее значение BIC.

На рис. 77 демонстрируется примерно одинаковое качество моделей.

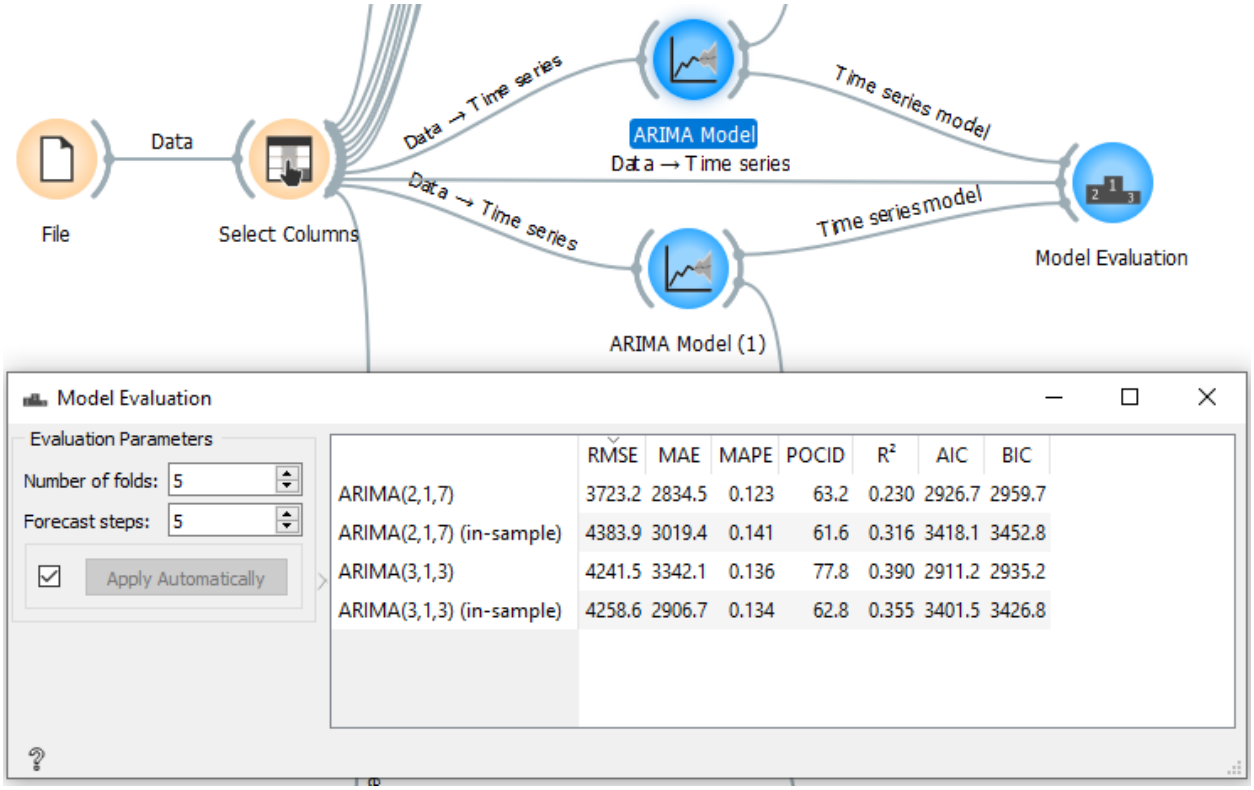


Рис. 76. Сравнение моделей с помощью виджета «Model Evaluation»

9. Поиск ассоциаций

Алгоритм взаимосвязей или ассоциативных правил (AssociationRules) позволяет выявить события, которые часто происходят совместно. Базовым понятием в теории ассоциативных правил является транзакция – некоторое множество событий, происходящих совместно. Типичная транзакция – приобретение клиентом товара в супермаркете (табл. 13), когда проводится группировка товаров, наиболее часто встречающихся в одном заказе (транзакции). После этого, на основе выявленных закономерностей, становится возможной выдача рекоменда-

ций по объединению товаров в наборы. На основе ассоциаций можно также прогнозировать или предлагать товары, которые обычно дополняют уже выбранные покупателем.

Таблица 13

Исходные данные алгоритма взаимосвязей заказов в магазине

Чек	Товар
SO51184	Гель для туалета
SO51184	Сода кальцинированная
SO51184	Чистящий порошок универсальный
SO51184	Микроспрей
SO51188	Средство для чистки плит
SO51200	Дозатор
SO51200	Микроспрей

Следующее важное понятие – предметный набор. Это непустое множество объектов, появившихся в одной транзакции. Первый набор для данных (см. табл. 13) включает четыре товара – гель для туалета, сода кальцинированная, чистящий порошок универсальный, микроспрей.

Поиск ассоциаций – определение частоты (вероятности) предметного набора. Для практического применения нужны предметные наборы, имеющие значительную частоту.

Ассоциативное правило состоит из двух наборов, называемых условием (antecedent) и следствием (consequent), записываемых в виде $X \longrightarrow Y$ (читается «из X следует Y»). Таким образом, ассоциативное правило формулируется в виде: «если *условие*, то *следствие*». Фрагмент списка правил представлен на рис. 78. Первое правило утверждает, что если покупатель приобрел микроспрей и соду кальцинированную, то он купит чистящий порошок универсальный.

Antecedent		Consequent
Микроспрей=1,	Сода кальцинированная=1 →	Чистящий порошок универсальный.
Микроспрей=1,	Средство от накипи=1 →	Чистящий порошок универсальный.
Микроспрей=1,	Мыло кусковое=1 →	Мыло жидкое=1
Микроспрей=1,	Мыло жидкое=1 →	Мыло кусковое=1
Микроспрей=1,	Мыло кусковое=1 →	Средство для мытья посуды=1
Микроспрей=1,	Средство для мытья посуды=1 →	Мыло кусковое=1
Кондиционер для белья=1,	Микроспрей=1 →	Стиральный порошок-автомат=1
Микроспрей=1,	Стиральный порошок-автомат=1 →	Кондиционер для белья=1

Рис. 77. Ассоциативные правила

Для Orange данные с целью поиска ассоциаций должны быть представлены таблицей (табл. 14), в которой транзакции (наборы) представлены строками. Возможные компоненты набора располагаются в заголовках столбцов. Число в ячейке столбца означает, что соответствующий компонент входит в набор указанное количество раз.

Таблица 14

Пример описания набора в Orange

Чек	Антистатик спрей	Бумага туалетная	Гель для туалета	Дозатор	Запасной баллон для освежителя	Зубная паста	Картридж с жидким мылом	Кондиционер для белья	Микроспрей	Мыло жидкое	Мыло кусковое	Освежитель воздуха	Отбеливатель	Пена/соль для ванн	Перчатки резиновые	Перчатки тканевые
SO51184			1						1							
SO51188																
SO51200				1					1							
SO51205			1						1	1	1					
SO51206			1							1	1		1			
SO51215			1													
SO51216			1							1	1		1			
SO51217			1			1										

Для поиска наборов используется виджет «Frequent Itemsets» (рис. 79). Результатом его работы являются наборы, отсортированные в порядке убывания частоты в виде количества и доли – колонки «Support» и «%». Список наборов имеет иерархическую структуру: компонент с наибольшей частотой, подсписок компонентов, с которыми он попадает в один набор и т. д. Виджет может использоваться для фильтрации по частоте (Support) и наименованию компонент. Его можно использовать для изучения наборов и поиска наборов для некоторого компонента.

Виджет «Association Rules» строит список правил (рис. 80). Для каждого правила вычисляются следующие характеристики:

1. Поддержка (support) – это частота (оценка вероятности) выполнения правила

$S(A \rightarrow B)=P\{A \cap B\}=Q(A \cap B) / Q(\Omega)$,

где $Q(A \cap B)$ – количество транзакций, содержащих *A* и *B*; $Q(\Omega)$ – общее количество транзакций.

Поддержку можно измерять и как количество случаев выполнения правила (абсолютная частота), но здесь доля случаев (частота как оценка вероятности) дает больше информации для сравнения правил.

2. Достоверность (confidence) – это условная частота (вероятность) *B* при условии, что *A* выбрано

$C(A \rightarrow B) = P\{B|A\} = Q(A \cap B) / Q(A)$,

где $Q(B)$ – количество транзакций, содержащих *B*.

Единичная достоверность свидетельствует о детерминированности правила: «Если в транзакции есть *A*, то в транзакции присутствует *B*». Если достоверность совпадает (или близка) с частотой *B*, то это свидетельствует о независимости следствия *B* от условия *A*.

3. Покрытие (coverage) – вероятность появления условия-антецедента $P\{A\}=Q(A) / Q(\Omega)$.

4. Strength соотношение вероятностей следствия и условия
$Strg(A \rightarrow B) = Q(B) / Q(A)$.

5. Лифт (lift) – показывает во сколько раз условная частота следствия больше безусловной частоты следствия
$L(A \rightarrow B) = P\{B|A\} / P\{B\}$.

Лифт является обобщенной мерой связи двух предметных наборов: при значениях лифта больше одного связь положительная, при единице она отсутствует, а при значениях меньше одного – отрицательная. Экстремальные значения, свидетельствующие о тесной зависимости, тем не менее, могут оказаться малозначимыми в силу незначительной поддержки (частота применения правила близка к нулю).

6. Левередж (leverage) или рычаг – разность между наблюдаемой частотой, с которой условие и следствие появляются совместно, и произведением частот появления условия и следствия по отдельности
$T(A \rightarrow B) = S(A \rightarrow B) – S(A)S(B)$.

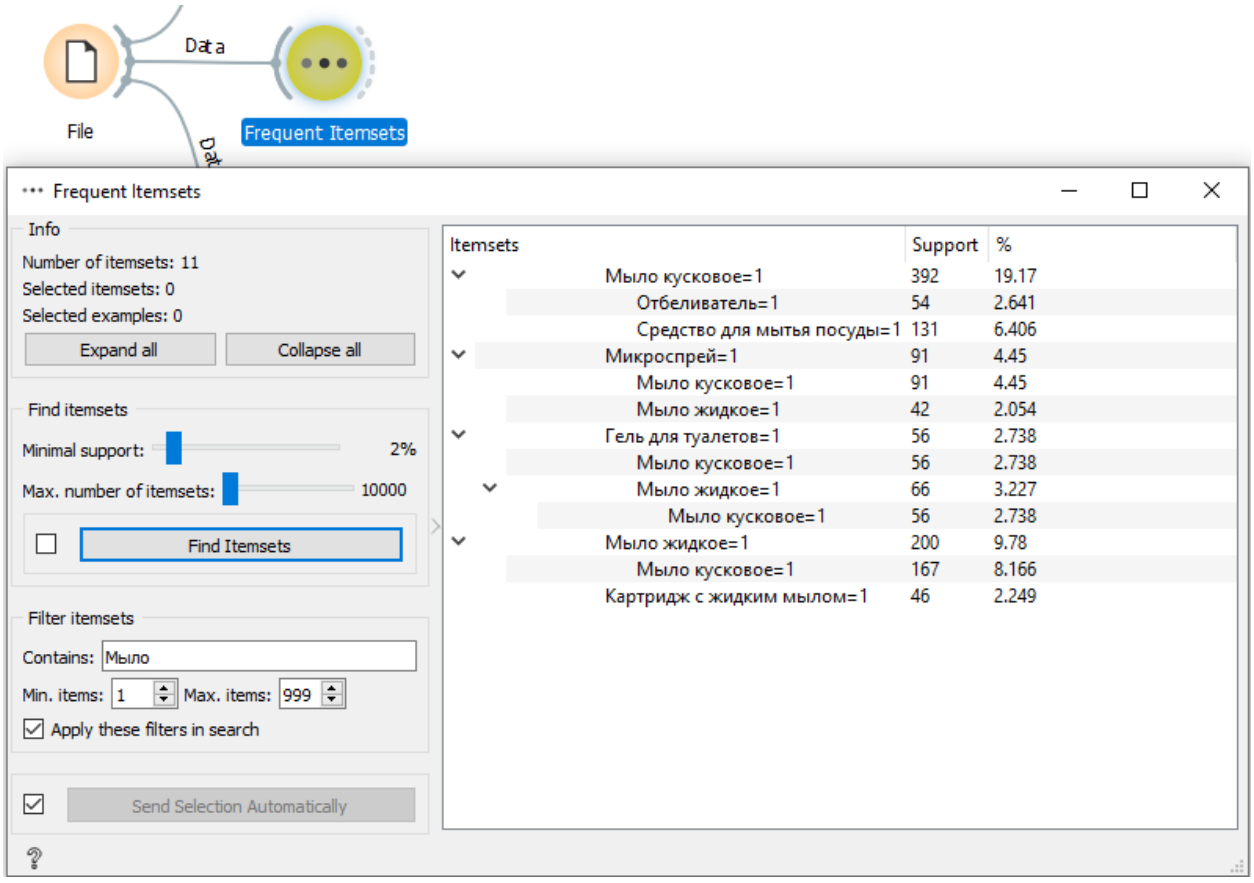


Рис. 78. Формирование списка наборов виджетом «Frequent Itemsets»

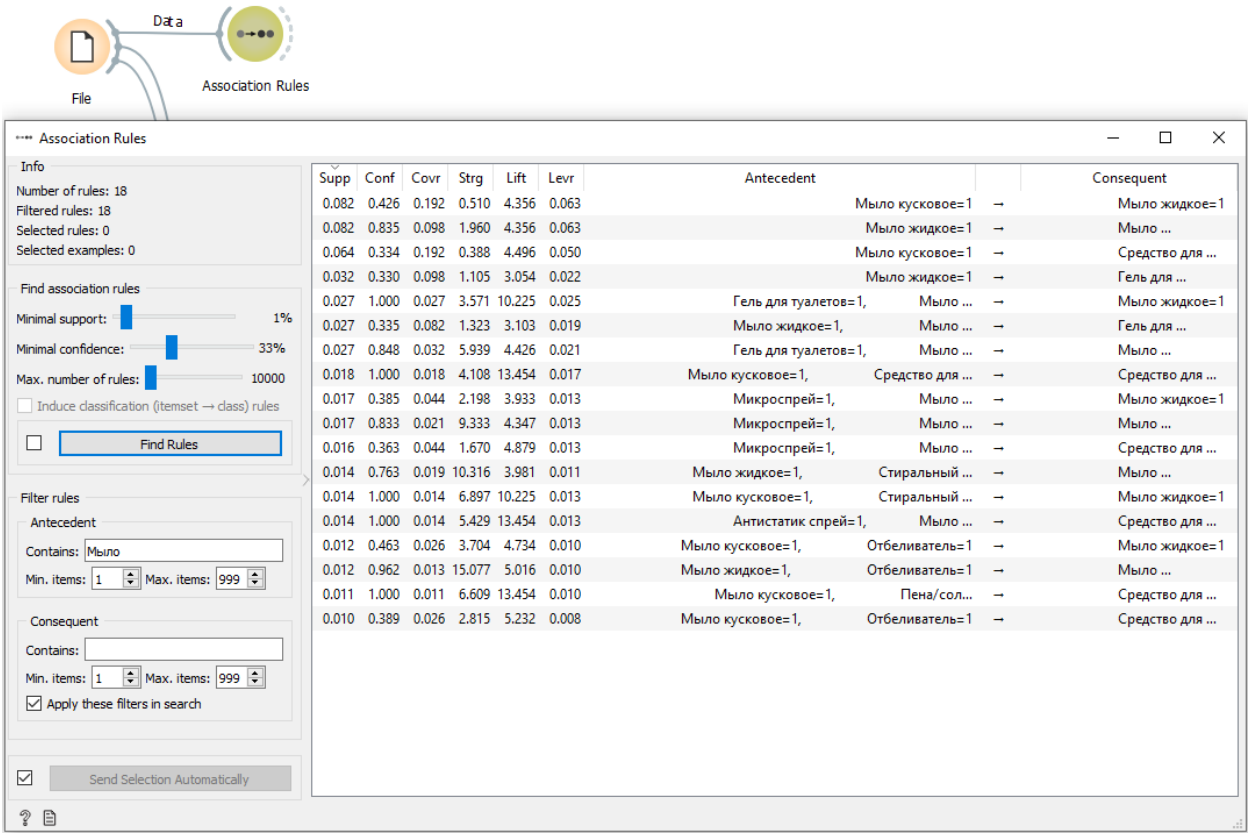


Рис. 79. Формирование списка ассоциативных правил виджетом «Association Rules»

Виджет «Association Rules» позволяет исследовать ассоциативные правила и оценивать частоты (достоверность – confidence) разных компонетов для выбранного условия.

10. Исследование текстов – text mining

10.1. Предварительная обработка и параметризация корпуса текстов

Огромное количество текстов фиксируется в электронных системах. Это – электронные сообщения, отправляемые по почте, sms, twitter, мессенджеры, чаты, отзывы, комментарии, социальные сети и многое др. Извлечение полезных зависимостей из текстовой информации – актуальная задача [5, 6]. Все поисковые системы пытаются решить ее, индексируя просторы интернета и расшифровывая запросы пользователей. Ниже приведены задачи, решаемые на основе text mining:

– обнаружение спама;

– выявление заимствования (плагиата);

– поиск в интернете;

– SEO (Search Engine Optimization) – продвижение сайта для его выхода на первые позиции в поисковых системах;

– машинный перевод (учет контекста);

– выявление мошенничества;

– выявление генерации положительных отзывов.

Обработка текстов на естественном языке – это сложная задача. Значение слов существенно зависит от контекста. В текстах полно идиом, сленга, гипербол, сравнений, сложных конструкций, запутанного синтаксиса и не менее запутанной семантики. Поэтому зачастую применяют самые простые, примитивные подходы, которые, тем не менее, оказываются полезными для решения многих задач.

В Orange виджеты для работы с текстами объединены на вкладке Text Mining. Исследование начинается с загрузки корпуса текстов (виджет «Corpus») (рис. 81). Виджет понимает много форматов: файлы Excel (**.xlsx**), поля, разделенные запятыми (**.csv**) или знаками табуляции (**.tab**). Можно указывать папки с текстами. В примере на рисунке для входных данных используется файл Excel, который содержит короткие сообщения и классы сообщений (позитивный или негативный). Файл взят с сайта https://study.mokoron.com/ и содержит 226 834 сообщения. В параметрах указываются поля с текстами (Used text features) и поля, которые следует исключить из анализа (Ignored text features).

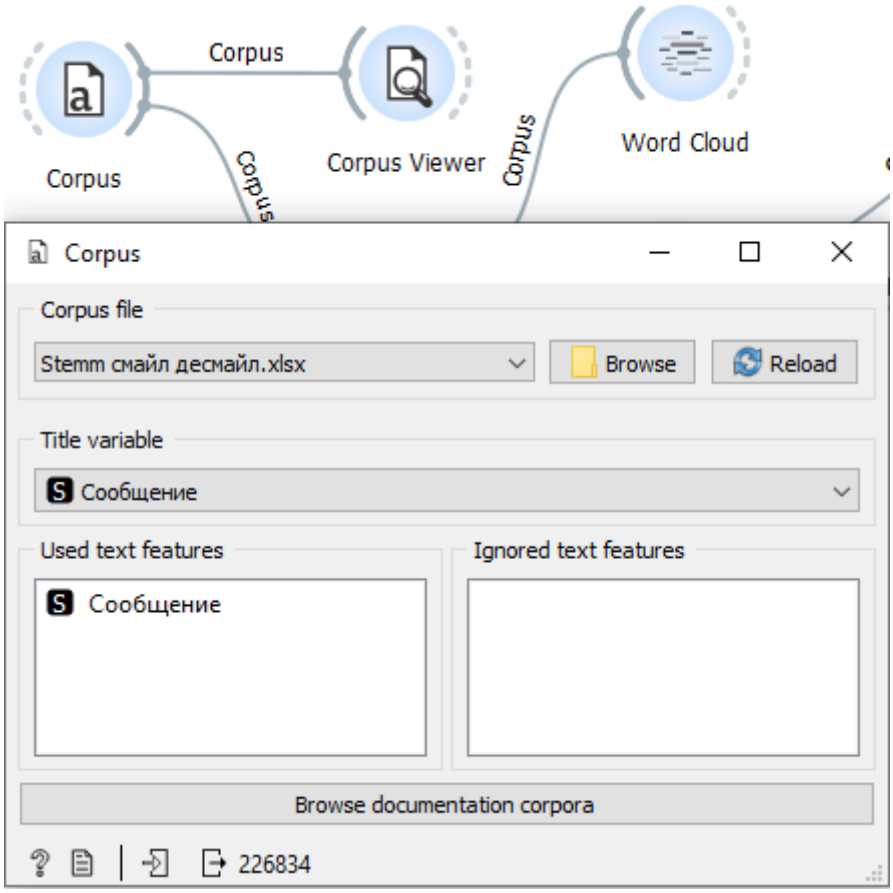


Рис. 80. Загрузка корпуса текстов

Для просмотра загруженных текстов используется виджет «Corpus Viewer» (рис. 82). Для фильтрации текстов можно применять регулярные выражения[14].

Параметризация текстов основана на выделении в них отдельных самостоятельных частей. Чаще всего используют слова и их устойчивые сочетания (n-граммы). Тексты редко снабжают наборами ключевых слов. Это делается

[14] URL: https://ru.wikipedia.org/wiki/Регулярные_выражения.

обычно при оформлении научных публикаций. Автоматически можно попытаться выделить в тексте слова, частота употребления которых максимальна. Однако такому подходу препятствуют такие обстоятельства, как часто употребляемые предлоги, вводные слова, некоторые глаголы. Для исключения влияния таких слов их объединяют в так называемый стоп-лист и исключают из параметров текстов.

Другая большая проблема связана с использованием различных форм одного и того же термина. Особенно это характерно для русского языка с большим количеством падежей, склонений, согласований, суффиксов и приставок. Большое значение имеет часть речи употребления слова. Для уточнения особенностей применения слов используют лексический анализ: выделение частей текста – лексем, и идентификацию типа лексемы. Лексема и ее тип в тексте образуют токен. Кроме лексем выделяют леммы. Лемма – нормальная форма слова (например, форма единственного числа, именительного падежа для существительных). Кроме лемм часто применяют стеммы. Стемма – неизменяемая часть (основа) слова.

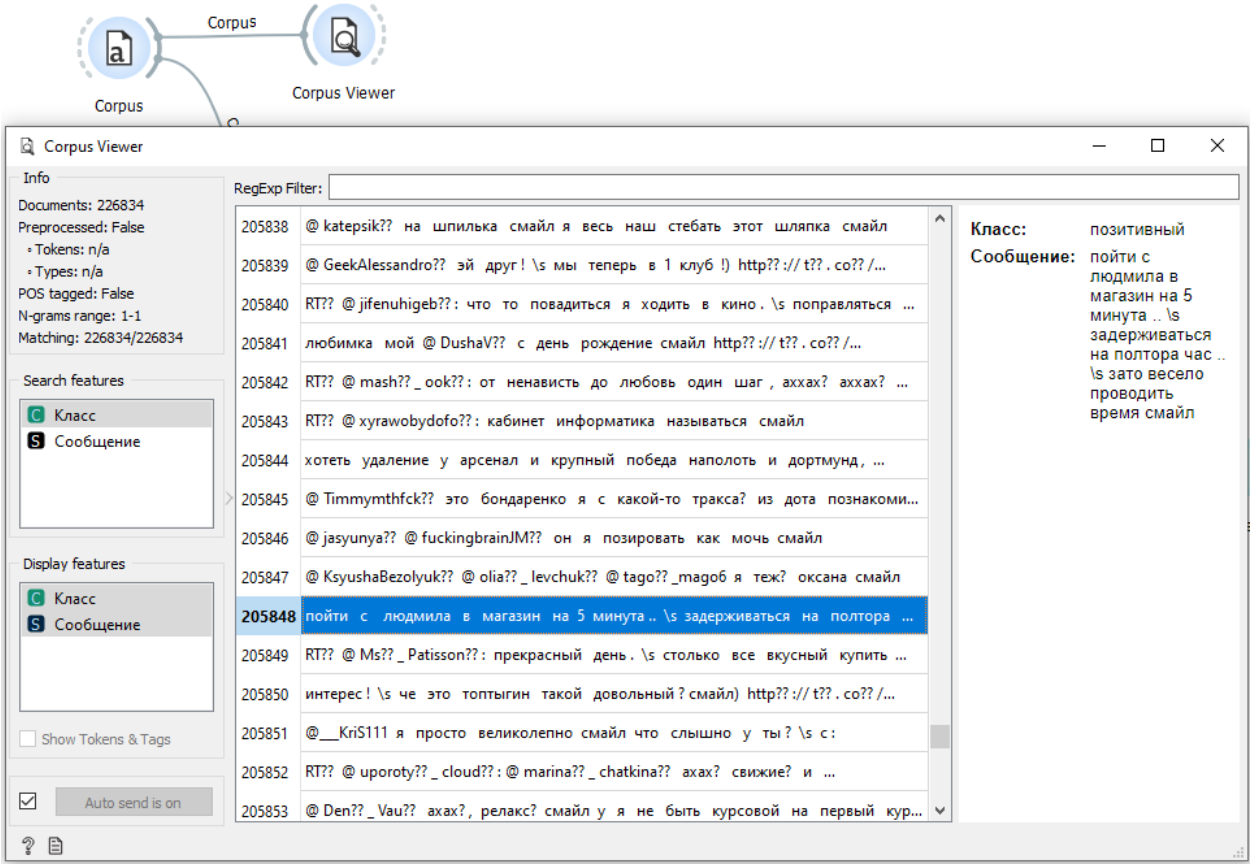


Рис. 81. Просмотр корпуса текстов

Синтаксические конструкции также оказывают значительное влияние на смысл предложения – очевидно, что сравнения, отрицания и другие конструкции, значительно влияют на смысл употребления термина. Поэтому лексический анализ следует дополнять синтаксическим – распознаванием грамматических конструкций языка.

Наибольшую сложность составляет распознавание смысла текста – семантический анализ. Часто для этого строят онтологические модели – точные спецификации некоторой предметной области. Выделяют концепты – базовые понятия некоторой предметной области, строят связи между концептами – определяют соотношения и взаимодействия базовых понятий.

Для текстов на русском языке можно использовать Mystem[15], написанный для «Яндекс» И.В. Сегаловичем и В.А. Титовым. Лемматизация текста выполняется запуском

mystem.exe IN.txt OUT.txt -c -l -s –d.

В результате слова в тексте приводятся к единой форме, и последующая параметризация текста выполняется более эффективно. Ранее (см. рис. 82) представлен корпус текстов уже после лемматизации.

Виджет «Preprocess Text» позволяет выполнить предварительную обработку текста (рис. 83). Слева расположен список преобразований, каждое из которых можно неоднократно добавлять в конвейер преобразований в списке справа. Возможно применять следующие преобразования.

1. Transformation – преобразование текста:

– Lowercase – преобразование прописных букв в строчные;

– Remove accents – исключение ударений и диакритических знаков;

– Parse html – замена гиперссылок их обозначением;

– Remove – исключение URL.

2. Tokenization – деление текста на слова и n-граммы:

– Word & Punctuation – слова и пунктуация;

– Whitespace – деление пробелами;

– Sentence – деление на предложения;

– Regexp – регулярные выражения;

– Tweet – учет смайликов.

3. Normalization – преобразование слов к леммам и стеммам с использованием следующих программ:

– Porter Stemmer;

– Snowball Stemmer;

– WordNet Lemmatizer.

4. Filtering – удаление или оставление слов:

– Stopwords – исключение слов из стоп-списка;

– Lexicon – оставляет слова из списка терминов;

– Regexp – удаляет слова, соответствующие регулярным выражениям;

– Document frequency – оставляет токены удовлетворяющие ограничениям частоты (DF = (3, 5) токен должнен появиться три раза или чаще не более чем в пяти документах);

– Most frequent tokens – остаются часто встречающиеся токены;

– N-grams Range – выбираются сочетания не более чем из n-слов (n-граммы).

[15] Segalovich I. A fast morphological algorithm with unknown word guessing induced by a dictionary for a web search engine. 2003. URL: http://cache-novosibrt03.cdn.yandex.net/download.yandex.ru/company/iseg-las-vegas.pdf, 10/07/2018.

5. POS Tagger – определение части речи и грамматических характеристик слов в тексте с применением одной из следующих моделей:

– Averaged Perceptron Tagger;

– Treebank POS Tagger (MaxEnt);

– Stanford POS Tagger.

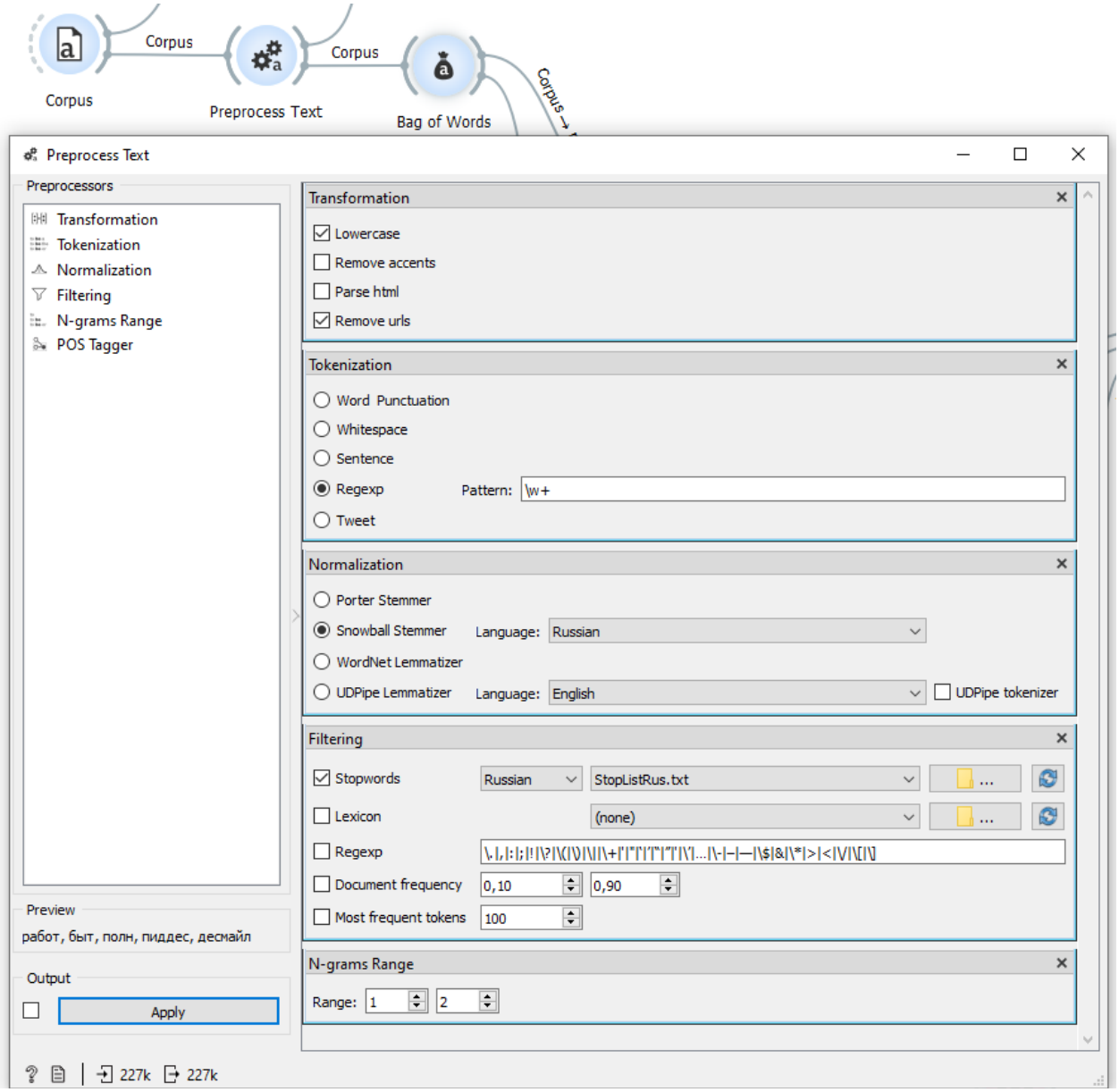


Рис. 82. Предварительная обработка корпуса текстов

Следует отметить что большая часть трансформаций ориентирована на англоязычные тексты. Поэтому рекомендуется использовать стеммеры, разработанные специально для русского языка.

Представление о словаре корпуса текстов дает виджет «World Cloud», который определяет частоты (количество употреблений) слов и представляет результат в виде таблицы и облака (рис. 84). В приведенном примере чаще всего употребляются «смайл» и «десмайл», которые появились в результате замены

смайликов на перечисленные слова в корпусе текстов. Анализ частоты слов может улучшить параметризацию – в стоп-лист можно было бы включить «s», «http» и другие «слова», не несущие смысловой и эмоциональной нагрузки.

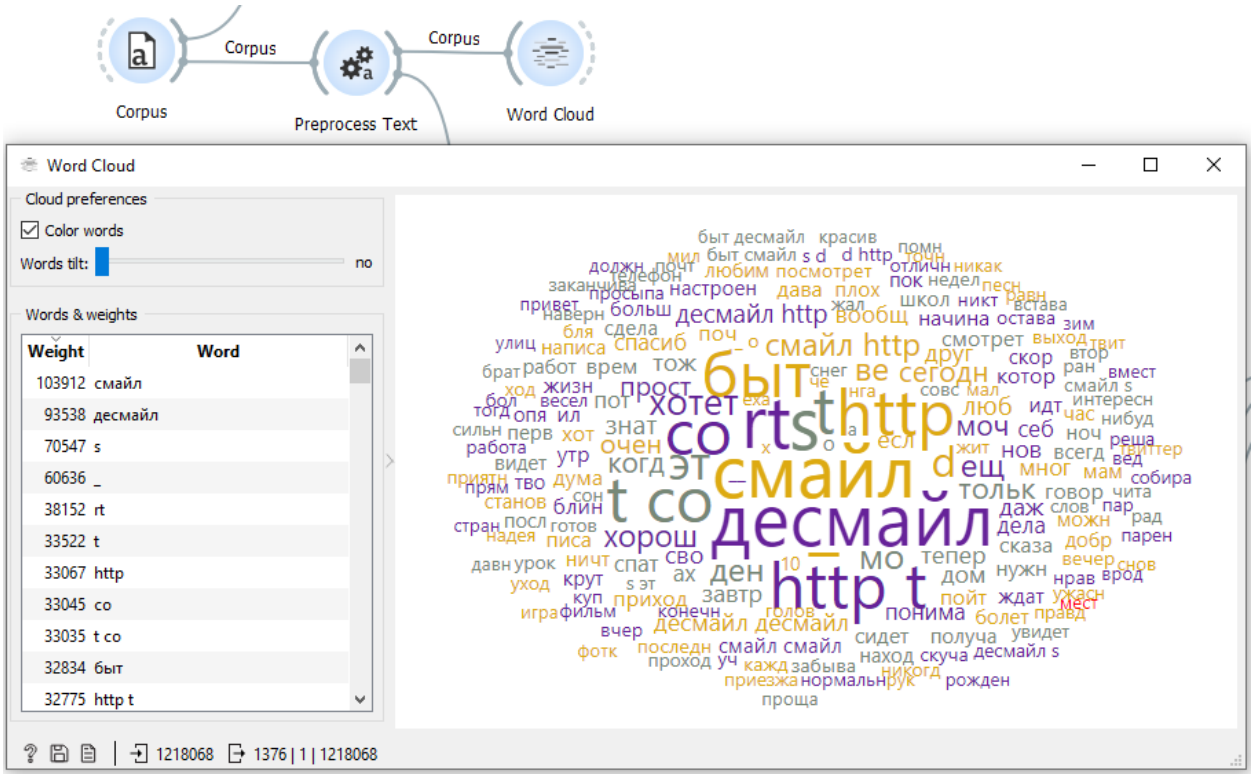


Рис. 83. Статистика употребления слов в корпусе текстов

Простейшая параметризация текста заключается в определении частотной характеристики каждого слова, входящего в текст, которая может быть одной из следующих:

1. Абсолютная частота (count) n_{td} – количество употреблений слова t в тексте d.

2. Относительная частота –

$$tf(t,d) = \frac{n_{td}}{\sum_v n_{vd}}.$$

3. TF-IDF характеристика (tf – term frequency, idf – inverse document frequency) – оценка важности терма в документе, являющегося частью корпуса документов. Вес некоторого терма пропорционален частоте употребления этого терма в документе и обратно пропорционален частоте употребления терма во всех документах корпуса –

$$tf - idf(t,d,D) = tf(t,d) \times idf(t,D),$$

где $idf(t,D) = \log\left(\frac{|D|}{|\{d:t\in d\}|}\right)$; $|D|$ – количество документов в корпусе D; $|\{d: t \in d\}|$ – количество документов, использующих слово t.

TF-IDF характеристика позволяет сравнивать тексты существенно разные по количеству слов.

В рассматриваемом примере все тексты короткие и можно использовать просто количество употреблений слова в тексте. В результате параметризации

получается огромная матрица (рис. 86), в которой количество строк равно количеству документов в корпусе, а количество столбцов – количеству слов в корпусе. Такая матрица получается существенно разреженной и сохраняется в специальном формате.

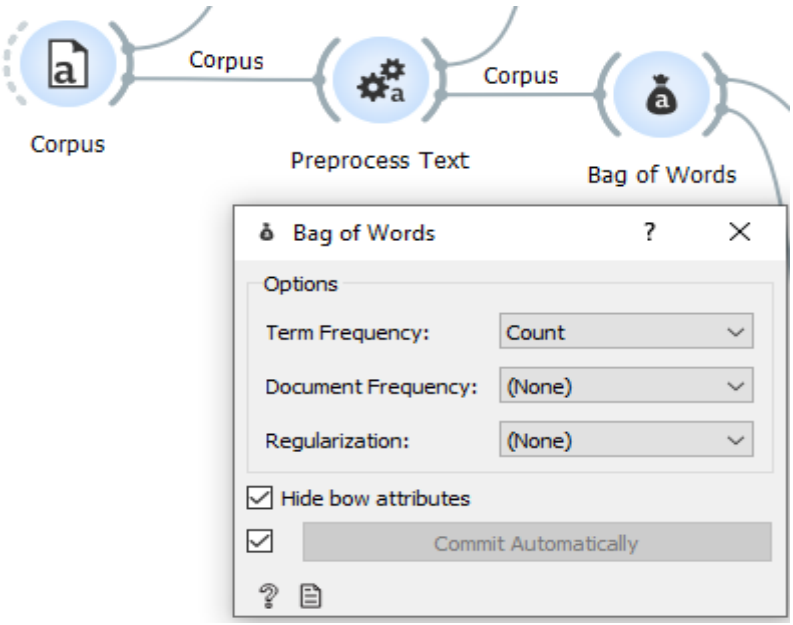


Рис. 84. Параметризация текстов в корпусе

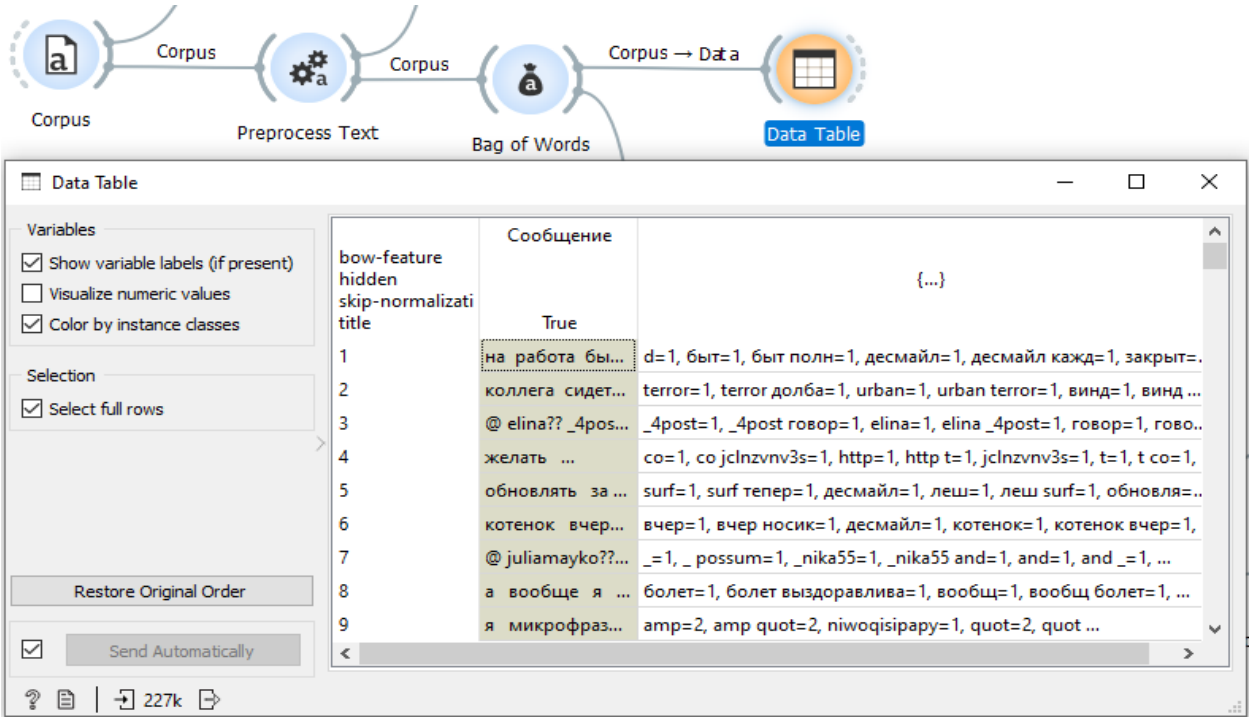


Рис. 85. Результат параметризации – статистика употребления слов в корпусе текстов

Обработка большого количества данных требует большого количества времени. Для снижения сложности настройки моделей применяют сэмплинг – выбор некоторого небольшого количества данных для настройки моделей. На рис. 87 определен выбор 500 данных для дальнейшей обработки.

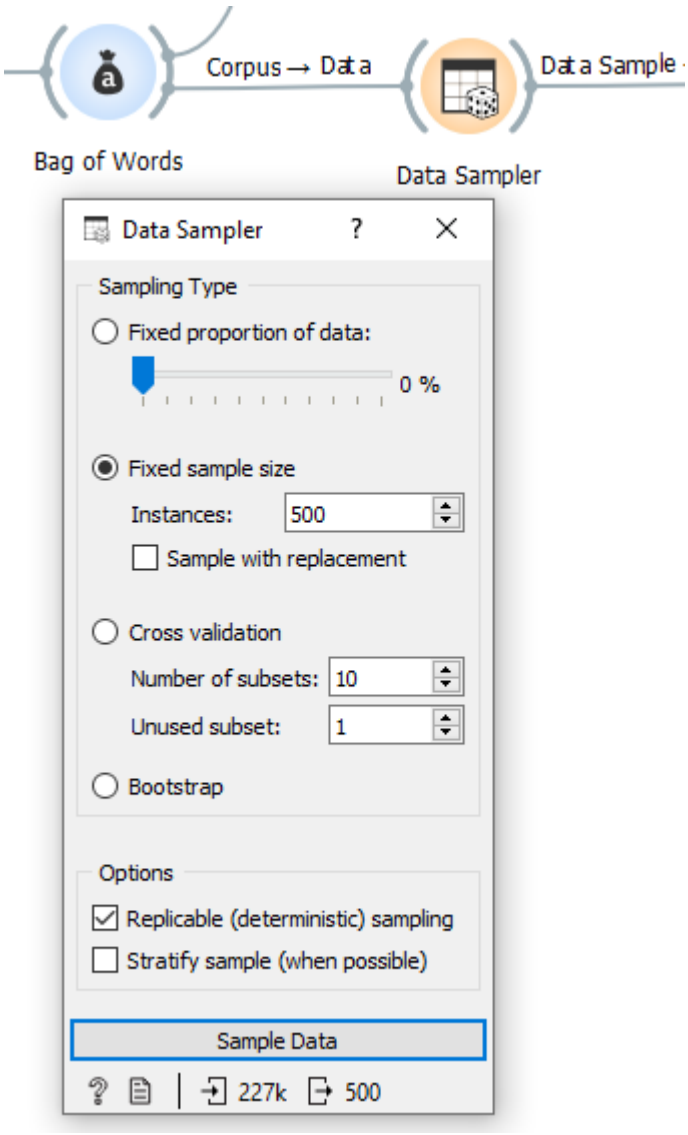


Рис. 86. Сэмплинг – случайный выбор образцов из корпуса текстов

10.2. Классификация текстов

Параметрическое описание текстов можно применить для настройки классификаторов. Выбраны простые классификаторы (рис. 88): наивный байесовский алгоритм и логистическая регрессия, которые, тем не менее, дают в общем хорошие результаты с средней ошибкой, равной 0,879 и 0,922 (рис. 89), это подтверждается матрицей ошибок (рис. 90) и ROC-кривыми (рис. 91).

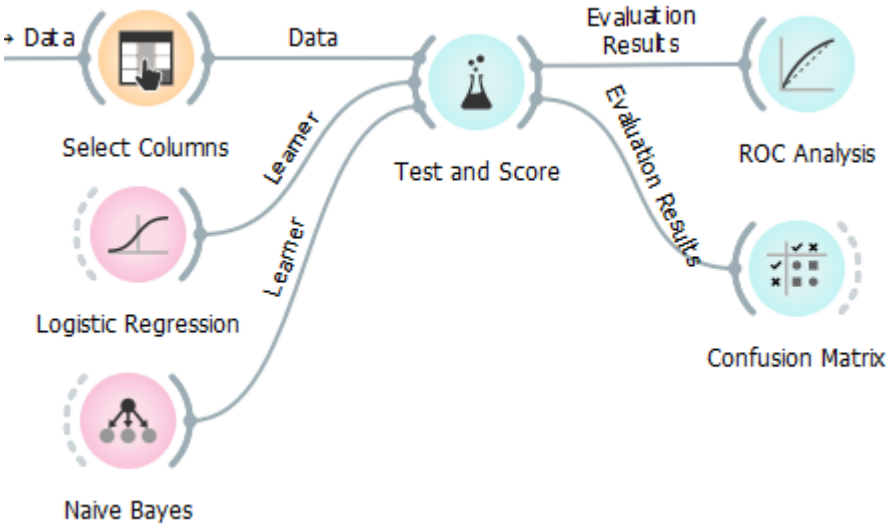


Рис. 87. Настройка классификаторов

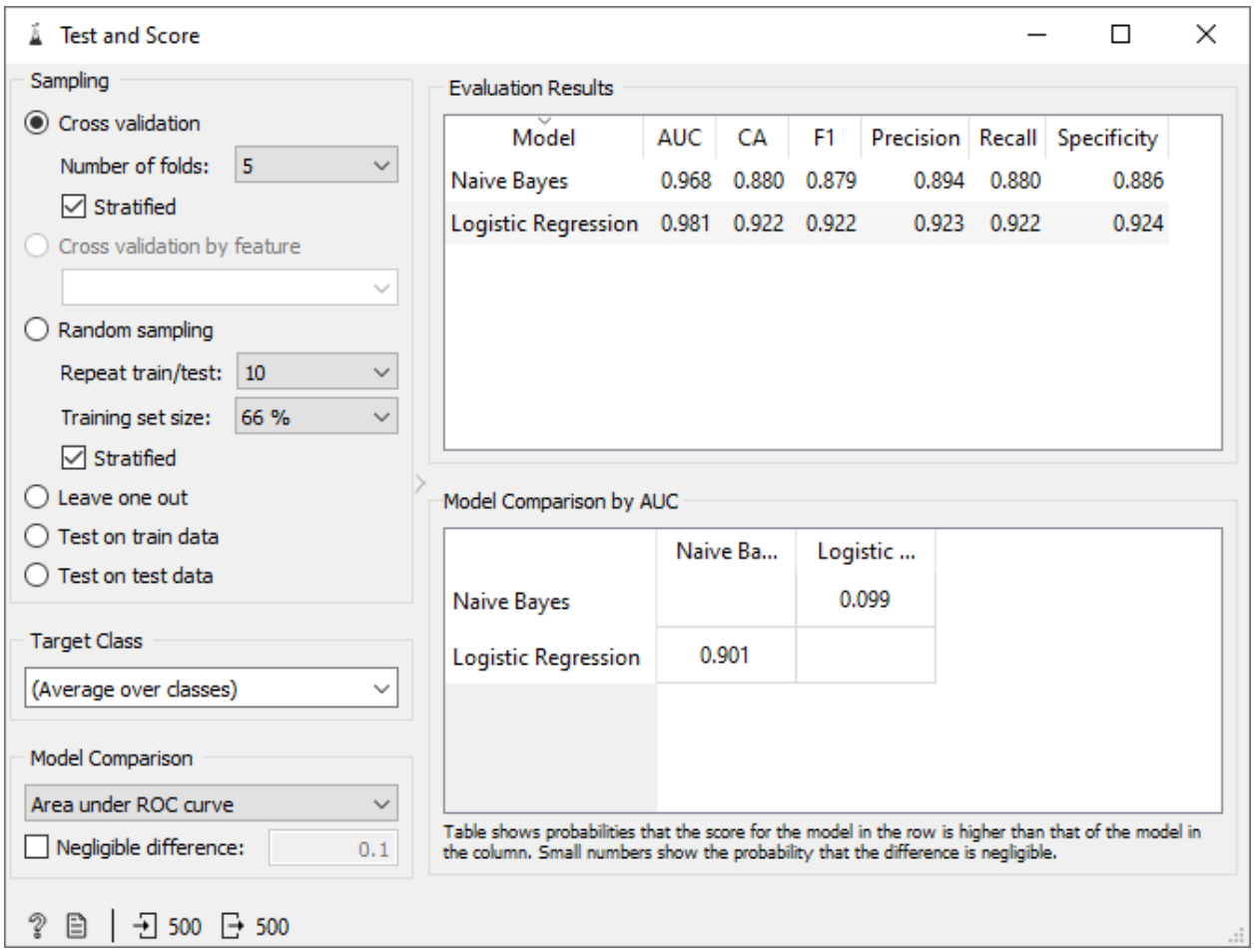


Рис. 88. Параметры и характеристики настройки классификаторов

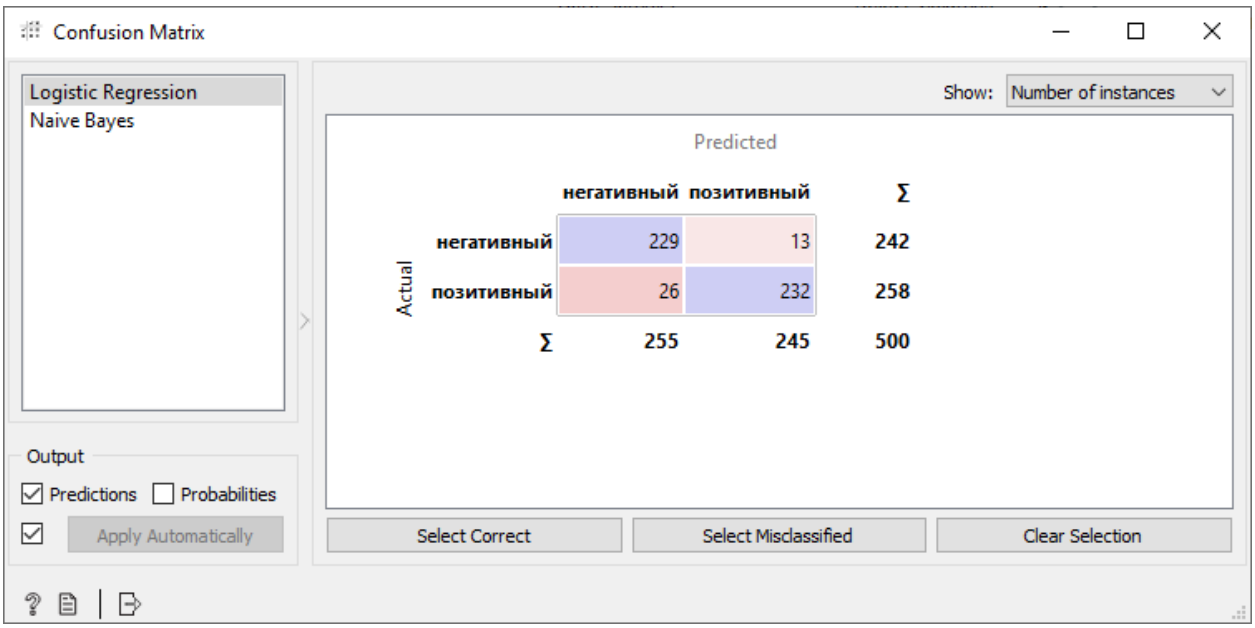


Рис. 89. Матрица ошибок классификации текстов
с помощью логистической регрессии

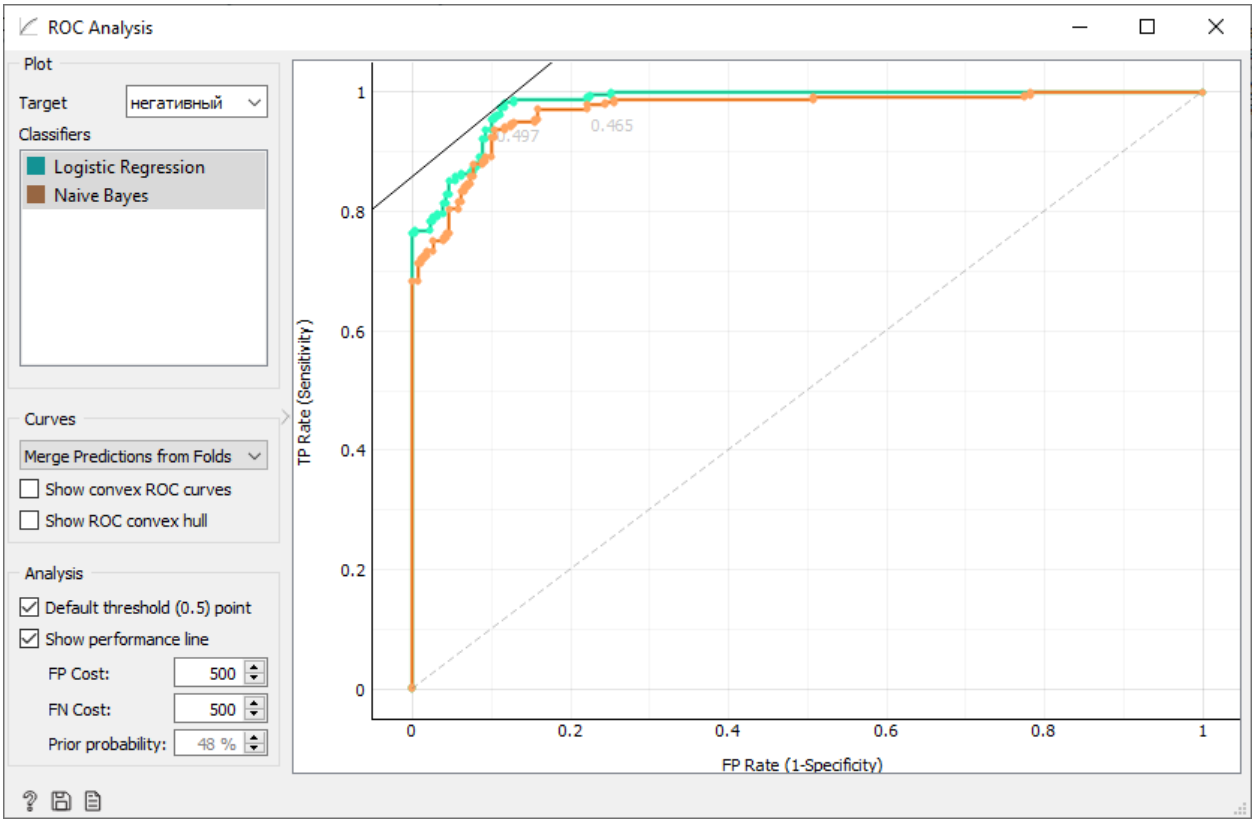


Рис. 90. ROC-кривые классификации текстов

10.3. Кластеризация текстов

Кластеризацию текстов можно применять, если в корпусе текстов нет классификационных признаков. К предварительно обработанным текстам, прошедшим параметризацию по методике «мешок слов», можно применять известные методы кластеризации. На рис. 92 представлена последовательность виджетов для настройки модели кластеризации с использованием алгоритма k-средних.

Предварительно с помощью сэмплинга выбирается ограниченное количество текстов, для которых выполняется вычисление частотных характеристик слов, и затем по этим параметрам выполняется кластеризация. Результаты кластеризации записываются в Excel-файл. Для обработки был выбран корпус текстов с позитивными и негативными сообщениями.

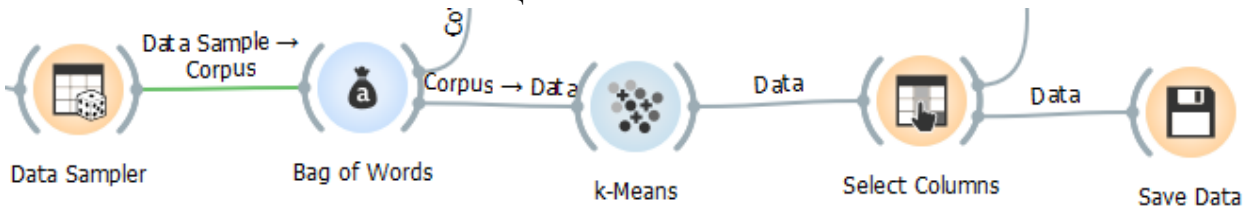


Рис. 91. Кластеризация текстов с использованием алгоритма k-средних

В табл. 15 приведены данные о количестве документов позитивного и негативного классов в каждом кластере. Очевидно, что такое распределение кластеров можно использовать для прогнозирования класса документа: нужно определить частоты слов документа и по этим характеристикам отнести его к одному из кластеров. Кластер С2 свидетельствует о позитивном классе, кластер С6 – о негативном.

Таблица 15

Распределение классов документов по кластерам

Кластер	Класс		Общий итог
	Негативный	Позитивный	
С1	10	42	52
С2	–	209	209
С3	3	–	3
С4	5	6	11
С5	–	1	1
С6	224	–	224
Общий итог	242	258	500

10.4. Решение задач классификации текстов с использованием модулей библиотеки Scikit-Learn на языке Python

Предварительная обработка текстов заключается главным образом в лемматизации текста – приведении слов к словарной форме. Для текстов на русском языке можно воспользоваться программой MyStem[16], разработанной для Yandex. Эта программа позволяет выполнить предварительную лемматизацию вне среды Python.

Соответствующие средства можно подключить и в среде Python
pip install pymystem3.
После этого можно импортировать стеммер
from pymystem3 import Mystem
создавать объект – стеммизатор

[16] URL: https://yandex.ru/dev/mystem.

m = Mystem()
и преобразовывать текст
text = "Красивая мама красиво мыла раму"
m.lemmatize(text)
в список лемм ['красивый', ' ', 'мама', ' ', 'красиво', ' ', 'мыть', ' ', 'рама', '\n'] или в текст из лемм
lemmas = ''.join(m.lemmatize(text)).

Рассмотрим классификацию текстов на примере обращений в контакт-центр. Первоначально загружаются данные из таблицы, содержащей обращения
df_texts = pd.read_excel(WorkDir+'Запросы+НомерУслуги.xlsx', 'Sheet1')
и стоп-список из таблицы Excel
StopList = pd.read_excel(WorkDir+'StopListRus.xlsx')['Термин'].values.tolist().

Можно выполнить лемматизацию текста обращения
df['Лемм_описание'] = df_texts['КРАТКОЕ_ОПИСАНИЕ'].apply(lambda x: ''.join(m.lemmatize(x)[0:-1])).
Однако, такой вызов стеммера является медленным (около одной секунды на запись. Если записей много, то проще воспользоваться программой MyStem для обработки файла целиком.

Затем выполняется импорт «мешка слов» с абсолютными частотами
from sklearn.feature_extraction.text import CountVectorizer[17]
и создание объекта для построения «мешка слов»
CV=CountVectorizer(lowercase=True, ngram_range=(1,MaxNgram), stop_words=StopList)
этот объект применяется для построения матрицы M тексты-слова
M=CV.fit_transform(df_texts['КРАТКОЕ_ОПИСАНИЕ']).

В полученной матрице частот каждому тексту соответствует строка, содержащая абсолютные частоты (количество включений в текст) слов. Кроме матрицы строится словарь термов – слов и словосочетаний (табл. 16). Словарь терминов является переменной типа dict и содержит для каждого терма (слова и словосочетания) его индекс в матрице «мешка слов».

Таблица 16

Фрагмент словаря термов

Термин	Индекс
шрифт	308099
шрифт печать	308133
электронный	309938
электронный почта	309993
эпс	310568
эпс электронный	310972
этран	311262

[17] URL: https://scikit-learn.org/stable/modules/generated/sklearn.feature_extraction.text.CountVectorizer.html#sklearn.feature_extraction.text.CountVectorizer.

Матрица частот может использоваться для обучения классификатора. В данном случае можно применить мультиклассовый байесовский классификатор[18]
from sklearn.naive_bayes import MultinomialNB.

После импорта выполняется создание объекта – мультиклассового байесовского классификатора – и его настройка
clf = MultinomialNB().fit(M, df_texts['НОМЕР_УСЛУГИ']).

После настройки классификатор можно применять для вычисления (предсказания) класса (в примерах вычисление производится по обучающей выборке)
PredictedClass = clf.predict(M)
а также для вычисления вероятностей классов
proba = clf.predict_proba(M).

Вычисление точности предсказания
clf_score = clf.score(M, df_texts['НОМЕР_УСЛУГИ'])
позволяет оценить долю совпадения предсказанных классов с наблюдаемыми. Для данного примера она составила 0,79. Более подробную информацию дает отчет о классификации
Rep=classification_report(df_texts['НОМЕР_УСЛУГИ'],PredictedClass).

Отчет предоставляет оценки точности предсказания каждого класса (табл. 17). Колонки в таблице содержат значения следующих показателей:

– precision – доля правильно положительно классифицированных наблюдений среди всех наблюдений положительного класса;

– recall – доля правильно положительно классифицированных наблюдений среди всех положительно классифицированных наблюдений;

– f1-score – взвешенное гармоническое среднее показателей precision и recall;

– support – количество наблюдений каждого класса.

Таблица 17

Фрагмент отчета оценки точности предсказания классов

№	precision	recall	f1-score	support
1	0,99	0,66	0,79	274
2	0,92	1,00	0,96	16 600
3	0,95	0,18	0,31	765
4	1,00	0,04	0,08	47
5	0,00	0,00	0,00	30
6	0,87	0,88	0,87	6 362
7	1,00	0,13	0,23	305
8	1,00	0,15	0,27	182
9	0,00	0,00	0,00	28
…				
88	0,00	0,00	0,00	1
89	0,64	0,84	0,73	10 514
90	0,00	0,00	0,00	563
accuracy			0,79	106 836

[18] URL: https://scikit-learn.org/stable/modules/generated/sklearn.naive_bayes.MultinomialNB.html?highlight=multinomialnb #sklearn.naive_bayes.MultinomialNB.

Отчет демонстрирует существенную неоднородность точности предсказания. Для классов с небольшим количеством наблюдений алгоритм выдает значения 0 или 1. Полный текст программы классификации текстов представлен в Приложении 2.

Приведенные примеры демонстрируют, что даже простые частотные методы дают достаточно надежные предсказания классов. Такие текстовые классификаторы можно применять для автоматического определения причин обращений при условии высокой вероятности определения класса. Обращения с низкой вероятностью можно по-прежнему обрабатывать вручную. По предварительным расчетам доля таких обращений не превысит 20 %.

Заключение

Зависимости, обнаруженные в данных, могут найти эффективное применение для решения разнообразных задач. В этом случае аналитические технологии могут быть встроены в применяемые информационные технологии поддержки принятия решений. Прежде чем выполнять и внедрять достаточно трудоемкую разработку аналитических технологий, целесообразно выполнить предварительное обнаружение и исследование зависимостей. Для этих целей применение Orange является хорошим решением. Orange предоставляет достаточно представительный набор методов, моделей и алгоритмов с удобной технологией построения вычислительного эксперимента, основанной на визуальном проектировании процессов обработки данных.

В случае положительного решения о внедрении опробованных методов выделения зависимостей возможно применение модулей Python[19], лежащих в основе виджетов для построения программного кода и встраивания его в информационную систему. Кроме этого, можно применять модули scikit-learn[20] и многие др.

[19] URL: https://orange-data-mining-library.readthedocs.io/en/latest/#tutorial.
[20] URL: https://scikit-learn.org/stable/user_guide.html.

Список рекомендуемой литературы

1. Алгоритмы интеллектуального анализа данных (службы Analysis Services – интеллектуальный анализ данных) // Основные сведения об Analysis Services. – URL: https://msdn.microsoft.com/ru-ru/library/bb522607(v=sql.120).aspx (дата обращения: 30.08.2021).

2. Платформа Loginom. – URL: https://basegroup.ru/deductor/description (дата обращения: 06.12.2020).

3. Orange. Documentation. – URL: https://orange.biolab.si/docs/ (дата обращения: 06.12.2020).

4. Паклин Н.Б. Бизнес-аналитика: от данных к знаниям : учеб. пособие / Н.Б. Паклин, В.И. Орешков. – Санкт-Петербург : Питер, 2013. – 701 с.

5. Анализ данных и процессов: учеб. пособие / А.А. Барсегян, М.С. Куприянов, И.И. Холод, М.Д. Тесс, С.И. Елизаров. – Изд. 3-е, перераб. и доп. – Санкт-Петербург : БХВ-Петербург, 2009. – 512 с.

6. Автоматическая обработка текстов на естественном языке и анализ данных : учеб. пособие / Е.И. Большакова, К.В. Воронцов, Н.Э. Ефремова, Э.С. Клышинский, Н.В. Лукашевич, А.С. Сапин. – Москва : Изд-во НИУ ВШЭ, 2017. – 269 с.

Приложение 1

Текст программы классификации и кластеризации
данных о пассажирах Титаника

```
# -*- coding: utf-8 -*-
# Импорт модулей
import numpy as np
import pandas as pd
import matplotlib.pyplot as pp

# Загруззка Excel таблицы в DataFrame
workdir='C:\\Users\\user\\D\\Титаник\\'  # каталог исходных и выходных файлов
df_Titanic=pd.read_excel(workdir+'Titanic.xlsx')
# Создание списка категориальных переменных (колонок)
categorical_columns = [c for c in df_Titanic.columns if df_Titanic[c].dtype.name ==
    'object']
# Создание списка числовых переменных (колонок)
numerical_columns = [c for c in df_Titanic.columns if df_Titanic[c].dtype.name !=
    'object']

# Замена пропусков
df_Titanic['Age'] = df_Titanic['Age'].fillna(df_Titanic['Age'].mean()) # замена про-
    пусков возраста на средний возраст
df_Titanic['Embarked']          =          df_Titanic['Embarked'].fillna(df_Titanic['Em-
    barked'].mode()[0]) # замена на наиболее вероятный пункт посадки

# Квантование возраста
df_Titanic['Age_class'] = [0 if df_Titanic['Age'].iloc[i]<16 else
    1 if df_Titanic['Age'].iloc[i]<20 else
    2 if df_Titanic['Age'].iloc[i]<35 else
    3 if df_Titanic['Age'].iloc[i]<55 else
    4 if df_Titanic['Age'].iloc[i]<75 else
    5
    for i in range(0, len(df_Titanic))]

# Замена категориальных значений

from sklearn.preprocessing import LabelEncoder # импорт перекодировщика

label = LabelEncoder() # создание перекодировщика

# Перекодировка пола
label.fit(df_Titanic['Sex'].drop_duplicates()) #задаем список значений для кодиро-
    вания
```

```
df_Titanic['Sex'+'_id'] = label.transform(df_Titanic['Sex']) #заменяем значения из списка кодами закодированных элементов

# Перекодировка порта посадки
field='Embarked'
label = LabelEncoder()
label.fit(df_Titanic[field].drop_duplicates()) #задаем список значений для кодирования
df_Titanic[field+'_id'] = label.transform(df_Titanic[field]) #заменяем значения из списка кодами закодированных элементов

# Масштабирование полей
from sklearn.preprocessing  import MinMaxScaler # импорт МИН-МАКС преобразователя
df_Titanic['MinMaxScaled_Age'] = MinMaxScaler().fit_transform(df_Titanic[['Age']]) # МИН-МАКС преобразование
# df_Titanic['MinMaxScaled_Age'].hist()
from sklearn.preprocessing  import StandardScaler # импорт преобразователя к стандартному нормальному распределению
df_Titanic['StandardScaled_Age'] = StandardScaler().fit_transform(df_Titanic[['Age']]) # преобразование к стандартному нормальному распределению
# df_Titanic['StandardScaled_Age'].hist()
from sklearn.preprocessing import RobustScaler # импорт робастного преобразователя
df_Titanic['RobustScaled_Age'] = RobustScaler().fit_transform(df_Titanic[['Age']]) # робастное преобразование
# df_Titanic['RobustScaled_Age'].hist()
df_Titanic['RobustScaled_Fare'] = RobustScaler().fit_transform(df_Titanic[['Fare']]) # робастное преобразование платы за проезд

# КЛАСТЕРИЗАЦИЯ
import pylab as pl # Это для графиков
# разделение выборки на тестовую и обучающую
from sklearn.model_selection import train_test_split
# Метод главных компонент
from  sklearn.decomposition  import PCA
# Генератор случайных чисел
from scipy.stats import randint

# Алгоритмы кластеризации
from sklearn.cluster import DBSCAN
from  sklearn.cluster  import KMeans
from  sklearn.cluster  import AgglomerativeClustering
```

```
# Метрики качества моделирования
from sklearn import metrics
from sklearn.metrics import pairwise_distances
from sklearn.metrics import davies_bouldin_score

from collections import Counter

# Выбор данных для кластеризации (масштабированные сисловые поля и пере-
      кодированные категориальные)
df_model = df_Titanic[['Survived',
'Pclass',
'StandardScaled_Age',
'SibSp',
'Parch',
'RobustScaled_Fare',
'Sex_id',
'Embarked_id']]

# попарные графики рассеивания
#   pd.plotting.scatter_matrix(df_model[['StandardScaled_Age',  'RobustScaled_Fare',
      'Sex_id']], alpha=0.7, figsize=(14,8))
# df_model.corr()  # вычисление коэффициентов корреляции

# Понижение размерности с методом t-SNE
# (t-distributed stochastic neighbor embedding)
# Импорт библиотек
from sklearn import datasets
from sklearn.manifold import TSNE
import matplotlib.pyplot as plt

# Определяем модель и параметры: количество осей, скорость обучения, количе-
      ство итераций
model = TSNE(n_components=2,learning_rate=10,n_iter=1000)
# Обучаем модель
transformed = model.fit_transform(df_model)
# Представляем результат в двумерных координатах
x_axis = transformed[:, 0]
y_axis = transformed[:, 1]

claster_rezult = [] # таблица сравнения моделей кластеризации

#иерархическая кластеризация ( hierarchical clustering )
# Импортируем библиотеки
from scipy.cluster.hierarchy import linkage, dendrogram
from scipy.cluster.hierarchy import fcluster
```

```
# Извлекаем измерения как массив NumPy
samples = df_model.values

# Реализация иерархической кластеризации при помощи функции linkage
mergings = linkage(samples, method='complete')
# Определение кластеров
groups = fcluster(mergings, 5, criterion='maxclust')
# Строим дендрограмму, указав параметры удобные для отображения
dendrogram(mergings,
        truncate_mode='level', p=5,
        labels=groups,
        leaf_rotation=200,
        leaf_font_size=6,
        )
plt.savefig(workdir+"дендрограмма.png")

# вычисление показателей качества кластеризации
silhouette_score=metrics.silhouette_score(df_model,  groups, metric='euclidean')
calinski_harabaz_score=metrics.calinski_harabaz_score(df_model,  groups)
# изображение кластеров на точечной диаграмме
plt.clf() # очистка графика
plt.scatter(x_axis, y_axis, c=groups)
plt.savefig(workdir+"Иерархическая кластеризация.png")
# сохранение номеров кластеров
df_Titanic['hierarchy'] = groups
# df_Titanic[['hierarchy']].hist()
# сохранение итогов кластеризации
claster_rezult.extend(['hierarchy', 5, silhouette_score, 5, calinski_harabaz_score])

# Кластеризация методом k-средних
metrics_list = []
silhouette_best=[]
calinski_best=[]
for eps in [0.1]: # перебор значений параметра "точность"
    for n_clusters in [3,4,5,7,10, 15]: # перебор значений параметра "количество кла-
        стеров"
        # создание объекта - модель кластеризации
        kmeans_model = KMeans( n_clusters=n_clusters)
        # обучение модели
        kmeans_model.fit(df_model)
        # сохранение показателей кластеризации в списках
        if len(Counter(kmeans_model.labels_))>1:
            silhouette_score=metrics.silhouette_score(df_model,   kmeans_model.labels_,
metric='euclidean')
```

```
        calinski_harabaz_score=metrics.calinski_harabaz_score(df_model,
    kmeans_model.labels_)
        metrics_list.extend([[n_clusters, silhouette_score, calinski_harabaz_score]])
        if len(silhouette_best)==0:
            silhouette_best=[n_clusters,silhouette_score]
            calinski_best=[n_clusters,calinski_harabaz_score]
        else:
            if silhouette_best[1]<silhouette_score:
                silhouette_best=[n_clusters,silhouette_score]
            if calinski_best[1]<calinski_harabaz_score:
                calinski_best=[n_clusters,calinski_harabaz_score]
# модель с наилучшими параметрами
kmeans = KMeans(n_clusters=calinski_best[0])
# обучение модели
kmeans.fit(df_model)
# сохранение номеров кластеров
df_Titanic['Kmeans_best']=kmeans.labels_
# df_Titanic[['Kmeans_best']].hist()
# сохранение итогов кластеризации
claster_rezult.extend(['KMeans', silhouette_best[0], silhouette_best[1],ca-
    linski_best[0],calinski_best[1]])
# изображение кластеров на точечной диаграмме
plt.clf() # очистка графика
plt.scatter(x_axis, y_axis, c=kmeans.labels_)
plt.savefig(workdir+"Кластеризация k-средних.png")

from sklearn.cluster import AffinityPropagation
af_n = []
for x in [-55,-58,-59,-61]:
    af = AffinityPropagation(preference=x).fit(df_model)
    cluster_centers_indices = af.cluster_centers_indices_
    af_n.extend([[x,len(cluster_centers_indices)]])
af = AffinityPropagation(preference=-60).fit(df_model)
labels = af.labels_
silhouette_score=metrics.silhouette_score(df_model,  labels, metric='euclidean')
calinski_harabaz_score=metrics.calinski_harabaz_score(df_model,  labels)
plt.clf() # очистка графика
plt.scatter(x_axis, y_axis, c=af.labels_)#df_global['DBSCAN']) #
plt.savefig(workdir+"AffinityPropagation.png")

# Алгомеративная кластеризация ( agglomerative clustering)
metrics_list = []
silhouette_best=[]
calinski_best=[]
for n_clusters in [3,4,5,7,10, 15]:
```

```
    agg = AgglomerativeClustering( n_clusters=n_clusters)
    agg.fit(df_model)
    if len(Counter(kmeans_model.labels_))>1:
        silhouette_score=metrics.silhouette_score(df_model,  agg.labels_, metric='eu-
        clidean')
        calinski_harabaz_score=metrics.calinski_harabaz_score(df_model,  agg.labels_)
        metrics_list.extend([[n_clusters, silhouette_score, calinski_harabaz_score]])
        if len(silhouette_best)==0:
            silhouette_best=[n_clusters,silhouette_score]
            calinski_best=[n_clusters,calinski_harabaz_score]
        else:
            if silhouette_best[1]<silhouette_score:
                silhouette_best=[n_clusters,silhouette_score]
            if calinski_best[1]<calinski_harabaz_score:
                calinski_best=[n_clusters,calinski_harabaz_score]
agg = AgglomerativeClustering(n_clusters=calinski_best[0])
agg.fit(df_model)
df_Titanic['AgglomerativeClustering']=agg.labels_
df_Titanic[['AgglomerativeClustering']].hist()
claster_rezult.extend(['AgglomerativeClustering',        silhouette_best[0],        silhou-
        ette_best[1],calinski_best[0],calinski_best[1]])
plt.clf() # очистка графика
plt.scatter(x_axis, y_axis, c=agg.labels_)#df_global['DBSCAN']) #
plt.savefig(workdir+"Алгомеративная кластеризация.png")

# Кластеризация DBSCAN
metrics_list = []
silhouette_best=[]
calinski_best=[]
for eps in [  3.0, 4.0, 5.0, 8.0, 10.0]:
    for min_samples in [4,5,6,7]: #[3,4,5,7,10,15,20, 25]:
        dbscan = DBSCAN( eps=eps, min_samples=min_samples)
        dbscan.fit(df_model)
        if len(Counter(dbscan.labels_))>1:
            silhouette_score=metrics.silhouette_score(df_model, dbscan.labels_, met-
        ric='euclidean')
            calinski_harabaz_score=metrics.calinski_harabaz_score(df_model, dbscan.la-
        bels_)
            metrics_list.extend([[eps,min_samples, silhouette_score, calinski_har-
        abaz_score]])
            if len(silhouette_best)==0:
                silhouette_best=[eps,min_samples,silhouette_score]
                calinski_best=[eps,min_samples,calinski_harabaz_score]
            else:
                if silhouette_best[2]<silhouette_score:
```

```
            silhouette_best=[eps,min_samples,silhouette_score]
        if calinski_best[2]<calinski_harabaz_score:
            calinski_best=[eps,min_samples,calinski_harabaz_score]

# Определяем модель с лучшими параметрами
dbscan = DBSCAN( eps=silhouette_best[0], min_samples=silhouette_best[1])
dbscan = DBSCAN( eps=calinski_best[0], min_samples=calinski_best[1])
dbscan = DBSCAN( eps=1.1, min_samples=2)
# Обучаем
dbscan.fit(df_model)
df_Titanic['DBSCAN'] = dbscan.labels_
silhouette_score=metrics.silhouette_score(df_model,  dbscan.labels_, metric='euclide-
    an')
calinski_harabaz_score=metrics.calinski_harabaz_score(df_model,  dbscan.labels_)
#df_Titanic[['DBSCAN']].hist()
claster_rezult.extend(['DBSCAN', 4, silhouette_score, 4,calinski_harabaz_score])
plt.clf() # очистка графика
plt.scatter(x_axis, y_axis, c=dbscan.labels_)
plt.savefig(workdir+"DBSCAN кластеризация.png")

# Кластеризация с использование смеси распределений Гаусса
from sklearn import mixture
metrics_list = []
silhouette_best=[]
calinski_best=[]
for n_clusters in [2,3,4,5,7,10, 15]:
    gmm = mixture.GaussianMixture(n_components=n_clusters)
    gmm.fit(df_model)
    gmm_labels = gmm.predict(df_model)
    if len(Counter(gmm_labels))>1:
        silhouette_score=metrics.silhouette_score(df_model,  gmm_labels,  metric='eu-
        clidean')
        calinski_harabaz_score=metrics.calinski_harabaz_score(df_model,  gmm_la-
        bels)
        metrics_list.extend([[n_clusters, silhouette_score, calinski_harabaz_score]])
        if len(silhouette_best)==0:
            silhouette_best=[n_clusters,silhouette_score]
            calinski_best=[n_clusters,calinski_harabaz_score]
        else:
            if silhouette_best[1]<silhouette_score:
                silhouette_best=[n_clusters,silhouette_score]
            if calinski_best[1]<calinski_harabaz_score:
                calinski_best=[n_clusters,calinski_harabaz_score]
gmm = mixture.GaussianMixture(n_components=2)
gmm.fit(df_model)
```

```
gmm_labels = gmm.predict(df_model)
silhouette_score=metrics.silhouette_score(df_model, gmm_labels, metric='euclidean')
calinski_harabaz_score=metrics.calinski_harabaz_score(df_model,  gmm_labels)
df_Titanic['GMM'] = gmm_labels
#df_Titanic[['GMM']].hist()
claster_rezult.extend(['GaussianMixture', silhouette_best[0], silhouette_best[1],ca-
      linski_best[0],calinski_best[1]])
plt.clf() # очистка графика
plt.scatter(x_axis, y_axis, c=gmm_labels)
plt.savefig(workdir+"GMM кластеризация.png")

# Birch   кластеризация
from sklearn.cluster import Birch
metrics_list = []
silhouette_best=[]
calinski_best=[]
for threshold in [  0.1, 0.2, 0.3, 0.5, 0.8]:
   for n_clusters in [4,6,10,15]:
      brc = Birch(branching_factor=50, n_clusters=n_clusters, threshold=thresh-
      old,compute_labels=True)
      brc.fit(df_model)
      if len(Counter(brc.labels_))>1:
         silhouette_score=metrics.silhouette_score(df_model,  brc.labels_, metric='eu-
      clidean')
         calinski_harabaz_score=metrics.calinski_harabaz_score(df_model,  brc.la-
      bels_)
         metrics_list.extend([[threshold,n_clusters, silhouette_score, calinski_har-
      abaz_score]])
         if len(silhouette_best)==0:
            silhouette_best=[threshold,n_clusters,silhouette_score]
            calinski_best=[threshold,n_clusters,calinski_harabaz_score]
         else:
            if silhouette_best[2]<silhouette_score:
               silhouette_best=[threshold,n_clusters,silhouette_score]
            if calinski_best[2]<calinski_harabaz_score:
               calinski_best=[threshold,n_clusters,calinski_harabaz_score]

brc = Birch(branching_factor=50, n_clusters=calinski_best[1], threshold=ca-
      linski_best[0],compute_labels=True)
brc.fit(df_model)
silhouette_score=metrics.silhouette_score(df_model, brc.labels_, metric='euclidean')
calinski_harabaz_score=metrics.calinski_harabaz_score(df_model,  brc.labels_)
df_Titanic['Birch'] = brc.labels_
#df_Titanic[['Birch']].hist()
```

```
claster_rezult.extend(['Birch', silhouette_best[1], silhouette_best[2],ca-
    linski_best[1],calinski_best[2]])
plt.clf() # очистка графика
plt.scatter(x_axis, y_axis, c=brc.labels_)
plt.savefig(workdir+"Birch кластеризация.png")

# КЛАССИФИКАЦИЯ

#   модуль для кросс-валидации
from sklearn.model_selection import cross_val_score
#   модуль поиска параметров классификатора
from sklearn.model_selection import RandomizedSearchCV
#   модуль для выполнения ROC-анализа характеристик классификатора
from sklearn.metrics import roc_curve, auc, roc_auc_score

#   модули алгоритмов опорных векторов
from sklearn import svm
#   алгоритм k-ближайших соседей
from sklearn.neighbors import KNeighborsClassifier
#   метод случайного леса
from sklearn.ensemble import RandomForestClassifier
#   логистическая регрессия
from sklearn.linear_model import LogisticRegression
#   адаптивнй бустинг
from sklearn.ensemble import AdaBoostClassifier
#   градиентный бустинг
from sklearn.ensemble import GradientBoostingClassifier
#   машина опорных векторов
from sklearn.svm import SVC
#   нейронная сеть
from sklearn.neural_network import MLPClassifier
#   дерево решений
from sklearn.tree import DecisionTreeClassifier

result_list=[]

import pylab as pl
#import plotly.graph_objs as go
# Импорт функции подготовки данных для настройки и тестирования классифи-
    каторов
from sklearn.model_selection import train_test_split
targetfield='Survived'
quota=25
qParts=5
n_estimators=100
```

```
#
X_train, X_test, y_train, y_test = train_test_split(df_model.drop([targetfield], axis=1),
        df_model[targetfield], test_size=float(quota)/100.0)

# kNN – kk ближайших соседей
# Создание объекта – классификатора
KNN=KNeighborsClassifier(n_neighbors=5)
# Для подбора параметров создается словарь по числу параметров. Для данного
    алгоритма – это n_neighbors – количество соседей. Для этого параметра бу-
    дут выполняться три варианта настройки: [ 3,5,7].
Params = {'n_neighbors': [ 3,5,7]}
# Выполнение настройки классификатора для всех комбинаций параметров и
    определение наилучшей по критерию 'roc_auc' – площадь под ROC-кривой.
    Применяется кросс-валидация с разбиением на qParts частей.
gridSearch = RandomizedSearchCV(estimator=KNN, param_distributions=Params,
    n_iter=5,
                scoring='roc_auc', cv=qParts, verbose=2).fit(X_train, y_train)
# Наилучшие параметры
Best_Params = gridSearch.best_params_
# Наилучшее значение критерия качества классификатора
best_roc_auc=gridSearch.best_score_
# Создание объекта с наилучшими параметрами
KNN = KNeighborsClassifier(n_neighbors = gridSearch.best_params_['n_neigh-
    bors'])
# Обучение классификатора
KNN.fit(X_train,y_train)
# Запись в таблицу колонки с прогнозом класса
df_Titanic['KNN'] = KNN.predict(df_model.drop([targetfield], axis=1))
# Запись в таблицу колонки с вероятностью класса
df_Titanic['p(KNN)'] = KNN.predict_proba(df_model.drop([targetfield], axis=1))[:,1]
# Вычисление прогнозов класса по тестовой выборке
test_labels = KNN.predict(X_test)
# Вычисление вероятностей классов по тестовой выборке
test_proba = KNN.predict_proba(X_test)
# Вычисление характеристики точности классификации по тестовой выборке
KNN_score = KNN.score(X_test, y_test)
# Вычисление площади под ROC-кривой по тестовой выборке
auc_score = roc_auc_score(y_test, test_proba[:,1], average='macro', sam-
    ple_weight=None)
# Запоминание точек ROC-кривой
fprKNN, tprKNN, thresholdKNN = roc_curve(y_test, test_proba[:,1])

result_list.extend(['KNN',Best_Params,auc_score]) #test_score = LR.score(X_test,
    y_test)
```

```
#Logistic – логистическая регрессия
LR=LogisticRegression()
Params = {'C': [ 0.5, 0.8, 1.0]}
gridSearch = RandomizedSearchCV(estimator=LR, param_distributions=Params,
    n_iter=5,
            scoring='roc_auc', cv=qParts, verbose=2).fit(X_train, y_train)
Best_Params = gridSearch.best_params_
best_roc_auc=gridSearch.best_score_

LR.fit(X_train,y_train)
df_Titanic['LR'] = LR.predict(df_model.drop([targetfield], axis=1))
df_Titanic['p(LR)'] = LR.predict_proba(df_model.drop([targetfield], axis=1))[:,1]
test_labels = LR.predict_proba(np.array(X_test.values))[:,1]
auc_score = roc_auc_score(y_test,test_labels , average='macro', sam-
    ple_weight=None)
#y_pred=LR.predict(np.array(X_test.values))
fprLR, tprLR, thresholdLR = roc_curve(y_test, test_labels)
result_list.extend(['LogisticRegression',str(LR.get_params),auc_score]) #test_score =
    LR.score(X_test, y_test)

# DecisionTreeClassifier
DTree=DecisionTreeClassifier()
Params = {'max_depth': [ 3,4,5,6,9],'min_samples_leaf': [3,5,7] }
gridSearch = RandomizedSearchCV(estimator=DTree, param_distributions=Params,
    n_iter=5,
            scoring='roc_auc', cv=qParts, verbose=2).fit(X_train, y_train)
Best_Params = gridSearch.best_params_
best_roc_auc=gridSearch.best_score_
DTree = DecisionTreeClassifier(max_depth = gridSearch.best_params_['max_depth'],
            min_samples_leaf = gridSearch.best_params_['min_sam-
    ples_leaf'])
DTree_params=str(DTree.get_params)
DTree.fit(X_train,y_train)
df_Titanic['DTree'] = DTree.predict(df_model.drop([targetfield], axis=1))
df_Titanic['p(DTree)'] = DTree.predict_proba(df_model.drop([targetfield],
    axis=1))[:,1]
test_labels = DTree.predict_proba(np.array(X_test.values))[:,1]
auc_score = roc_auc_score(y_test,test_labels , average='macro', sam-
    ple_weight=None)
fprDT, tprDT, thresholdDT = roc_curve(y_test, test_labels)
result_list.extend(['DTree',DTree_params,auc_score]) #test_score =
    DTree.score(X_test, y_test)

#SVC – машина опорных векторов
svm=SVC()
```

```
Params = {'C': [ 1.0, 10.0, 100.0],'kernel': ['linear', 'poly', 'rbf', 'sigmoid' ] }
#Params = {'C': [ 1.0],'kernel': ['linear'] }
gridSearch = RandomizedSearchCV(estimator=svm, param_distributions=Params,
    n_iter=5,
            scoring='roc_auc', cv=qParts, verbose=2).fit(X_train, y_train)
Best_Params = gridSearch.best_params_
best_roc_auc=gridSearch.best_score_
svm = SVC(C = gridSearch.best_params_['C'],
                kernel = gridSearch.best_params_['kernel'], probability=True)
svm_params=str(svm.get_params)
svm.fit(X_train,y_train)
df_Titanic['DTree'] = svm.predict(df_model.drop([targetfield], axis=1))
df_Titanic['p(DTree)'] = svm.predict_proba(df_model.drop([targetfield], axis=1))[:,1]
test_labels = svm.predict_proba(np.array(X_test.values))[:,1]
auc_score = roc_auc_score(y_test,test_labels , average='macro', sam-
    ple_weight=None)
fprSVM, tprSVM, thresholdSVM = roc_curve(y_test, test_labels)
result_list.extend(['svm',svm_params,auc_score]) #test_score = DTree.score(X_test,
    y_test)

# RF – случайный лес
Params = {'n_estimators': [ 40,70,100,200,400]}
RF = RandomForestClassifier(n_estimators = 70) #в параметре передаем кол-во де-
    ревьев
gridSearch = RandomizedSearchCV(estimator=RF, param_distributions=Params,
    n_iter=5,
            scoring='roc_auc', cv=qParts, verbose=2).fit(X_train, y_train)
best_n_estimators=gridSearch.best_params_
best_roc_auc=gridSearch.best_score_
RF = RandomForestClassifier(n_estimators = gridSearch.best_params_['n_estima-
    tors'])
RF.fit(X_train,y_train)
df_Titanic['RF'] = RF.predict(df_model.drop([targetfield], axis=1))
df_Titanic['p(RF)'] = RF.predict_proba(df_model.drop([targetfield], axis=1))[:,1]
test_labels = RF.predict_proba(np.array(X_test.values))[:,1]
auc_score = roc_auc_score(y_test,test_labels , average='macro', sam-
    ple_weight=None)
fprRF, tprRF, threshold = roc_curve(y_test, test_labels)
result_list.extend(['RF',str(RF.get_params),auc_score]) #test_score =
    DTree.score(X_test, y_test)

# Значимость полей
importances = RF.feature_importances_
indices = np.argsort(importances)[::-1]
```

```
importances = [[X_train.columns[indices[i]],importances[indices[i]]] for i in
    range(0,7)]
# Точность кросс-валидации
scores = cross_val_score(RF, X_train, y_train, cv=5)
print("Среднее Кросс-валидации: %0.2f (+/- %0.2f)" % (scores.mean(), scores.std()
    * 2))
#scores = cross_validation.cross_val_score(model_rfc, X_train, y_train)#, cv = qParts,
    n_jobs=1)

# AdaBoost – адаптивный бустинг
Params = {'n_estimators': [ 70,100,200], 'base_estimator':[DTree,LR,svm]}
AB = AdaBoostClassifier(base_estimator=DTree,n_estimators = 400) #в параметре
    передаем кол-во деревьев
gridSearch = RandomizedSearchCV(estimator=AB, param_distributions=Params,
    n_iter=5,
            scoring='roc_auc', cv=qParts, verbose=2).fit(X_train, y_train)
best_n_estimators=gridSearch.best_params_
best_roc_auc=gridSearch.best_score_
AB = AdaBoostClassifier(base_estimator=gridSearch.best_params_['base_estimator'],
             n_estimators = gridSearch.best_params_['n_estimators'])
AB = AdaBoostClassifier(base_estimator=LR,n_estimators = 400) #в параметре пе-
    редаем кол-во деревьев
AB.fit(X_train,y_train)
test_labels = AB.predict_proba(np.array(X_test.values))[:,1]
auc_score = roc_auc_score(y_test,test_labels , average='macro', sam-
    ple_weight=None)
fprAB, tprAB, threshold = roc_curve(y_test, test_labels)

df_Titanic['AB'] = AB.predict(df_model.drop([targetfield], axis=1))
df_Titanic['p(AB)'] = AB.predict_proba(df_model.drop([targetfield], axis=1))[:,1]
result_list.extend(['AB',str(AB.get_params),auc_score])

# GradientBoostingClassifier
Params = {'n_estimators': [ 70,100,200], 'max_depth':[3,4,5,7]}
GB = GradientBoostingClassifier()
gridSearch = RandomizedSearchCV(estimator=GB, param_distributions=Params,
    n_iter=5,
            scoring='roc_auc', cv=qParts, verbose=2).fit(X_train, y_train)
best_n_estimators=gridSearch.best_params_
best_roc_auc=gridSearch.best_score_
GB = GradientBoostingClassifier(max_depth=gridSearch.best_params_['max_depth'],
             n_estimators = gridSearch.best_params_['n_estimators'])
GB.fit(X_train,y_train)
test_labels = GB.predict_proba(np.array(X_test.values))[:,1]
```

```
auc_score = roc_auc_score(y_test,test_labels, average='macro', sample_weight=None)
fprGB, tprGB, thresholdGB = roc_curve(y_test, test_labels)

df_Titanic['GB'] = GB.predict(df_model.drop([targetfield], axis=1))
df_Titanic['p(GB)'] = GB.predict_proba(df_model.drop([targetfield], axis=1))[:,1]
result_list.extend(['GB',str(GB.get_params),auc_score])
# Значимость полей
importances = GB.feature_importances_
indices = np.argsort(importances)[::-1]
importances = [[X_train.columns[indices[i]],importances[indices[i]]] for i in range(0,7)]

# MLPClassifier
#Params = {'n_estimators': [ 70,100,200], 'max_depth':[3,4,5,7]}
MLP = MLPClassifier()
#gridSearch = RandomizedSearchCV(estimator=GB, param_distributions=Params, n_iter=5,
#                scoring='roc_auc', fit_params=None, cv=qParts, verbose=2).fit(X_train, y_train)
#best_n_estimators=gridSearch.best_params_
#best_roc_auc=gridSearch.best_score_
MLP.fit(X_train,y_train)
test_labels = MLP.predict_proba(np.array(X_test.values))[:,1]
auc_score = roc_auc_score(y_test,test_labels, average='macro', sample_weight=None)
fprMLP, tprMLP, thresholdMLP = roc_curve(y_test, test_labels)

df_Titanic['MLP'] = MLP.predict(df_model.drop([targetfield], axis=1))
df_Titanic['p(MLP)'] = MLP.predict_proba(df_model.drop([targetfield], axis=1))[:,1]
result_list.extend(['MLP',str(MLP.get_params),auc_score])

# Построение графиков ROC-кривых
plt.clf() # очистка графика
fig, ax = plt.subplots()
ax.plot(fprKNN, tprKNN)
ax.plot(fprLR, tprLR)
ax.plot(fprDT, tprDT)
ax.plot(fprSVM, tprSVM)
ax.legend(['K-соседей', 'Логистическая регрессия','Дерево решений', 'Машина опорных векторов'], loc='right')
ax.set(xlabel='fpr', ylabel='tpr',title='ROC')
ax.grid()
fig.savefig("KNN LR DT SVM.png")
```

```
# Построение графиков ROC-кривых
fig, ax = plt.subplots()
ax.plot(fprMLP, tprMLP)
ax.plot(fprRF, tprRF)
ax.plot(fprDT, tprDT)
ax.plot(fprAB, tprAB)
ax.plot(fprGB, tprGB)
ax.legend([ 'Нейронная сеть'
        ,'Случайный лес', 'Дерево решений', 'Адаптивный бустинг', 'Градиентный
    бустинг'], loc='right')
ax.set(xlabel='fpr', ylabel='tpr',title='ROC')
ax.grid()
fig.savefig("NN RF DT ADAB GB.png")

# Формирование сводных данных сравнения моделей кластеризации
t=[]
for i in range(0,6):
   t.extend([[claster_rezult[i*5+j] for j in range(0,5)]])
dfClaster = pd.DataFrame(t, columns = ['Модель','Кластеров 1','silhou-
    ette_score','Кластеров 2','calinski_harabaz_score'])

# Формирование сводных данных сравнения моделей классификации
t=[]
for i in range(0,8):
   t.extend([[result_list[i*3+j] for j in range(0,3)]])
dfClass = pd.DataFrame(t, columns = ['Модель','Параметрыs','auc_score'])

writer = pd.ExcelWriter('C:\\Users\\user\\TitanicClastersClasses.xlsx')
dfClaster.to_excel(writer,  sheet_name='Clasters', index=False)
dfClass.to_excel(writer,  sheet_name='Classes', index=False)
df_Titanic.to_excel(writer,  sheet_name='Data', index=False)
df_Titanic.describe().to_excel(writer,  sheet_name='describe', index=False)
df_Titanic[categorical_columns].describe().to_excel(writer,          sheet_name='de-
    scribe_categorial', index=False)
writer.save()
```

Приложение 2

Текст программы классификации обращений в контакт-центр

```
# -*- coding: utf-8 -*-
"""
Created on Wed Mar  3 19:41:17 2021

@author: user
"""

import math
import numpy as np
import pandas as pd
import time
start_time = time.time()

# Параметры
MaxNgram=2  # Слов в сочетаниях
Q_features=2**21 # Количество Hash-адресов
# Папка с рабочими файлами
WorkDir='C:\\Users\\user\\D\\Данные для DM\\Text Mining\\КонтактЦентр\\'

df_texts = pd.read_excel(WorkDir+'Запросы+НомерУслуги.xlsx', 'Sheet1')
StopList = pd.read_excel(WorkDir+'StopListRus.xlsx')['Термин'].values.tolist()

"""
from pymystem3 import Mystem
stemator = Mystem()
df['Лемм описание']=df_texts['КРАТКОЕ_ОПИСАНИЕ'].apply(lambda x:
        ".join(m.lemmatize(x)[0:-1]))
start_time = time.time()
print("--- %s seconds ---" % (time.time() - start_time))
"""
"""
StopList = [pd.read_excel(WorkDir+'StopListRus.xlsx')['Термин'][i] for i in
        range(0,pd.read_excel(WorkDir+'StopListRus.xlsx').shape[0])]
df_texts['КРАТКОЕ_ОПИСАНИЕ'] = df_texts['КРАТКОЕ_ОПИСА-
        НИЕ'].fillna("") # замена пропусков
df_texts['Класс'] = df_texts['Класс'].fillna("") # замена пропусков
df_texts['NumClass'] = df_texts['Класс']=='позитивный'
"""
# Импорт "Мешка слов"
from sklearn.feature_extraction.text import CountVectorizer
# Создание объекта для построения мешка слов с абсолютными частотами
```

```
CV=CountVectorizer(lowercase=True, ngram_range=(1,MaxNgram),
        stop_words=StopList)
# Построение матрицы M тексты слова
M=CV.fit_transform(df_texts['Лемм_описание'])
# Запоминание сформированного словаря
V = CV.vocabulary_

# Импорт мультиклассового байесовского классификатора
from sklearn.naive_bayes import MultinomialNB
# Создание объекта - мультиклассового байесовского классификатора - и его
        настройка
clf = MultinomialNB().fit(M, df_texts['НОМЕР_УСЛУГИ'])
# Вычисление (предсказание) класса
PredictedClass = clf.predict(M)
# Вычисление вероятностей классов
proba = clf.predict_proba(M)
# Вычисление точности предсказания
clf_score = clf.score(M, df_texts['НОМЕР_УСЛУГИ'])

# Матрицы ошибок для мультиклассовой классификации
from sklearn.metrics import multilabel_confusion_matrix
class_confusion_matrix = multilabel_confusion_ma-
        trix(df_texts['НОМЕР_УСЛУГИ'],PredictedClass)

# Отчет о точности для мультиклассовой классификации
from sklearn.metrics import classification_report
Rep=classification_report(df_texts['НОМЕР_УСЛУГИ'],PredictedClass)

# Сохранение прогноза класса
df_texts['Предсказанный класс'] = PredictedClass
# Сохранение вероятности класса
df_texts['Макс.вероятность'] =  proba.max(axis=1)
# Сортировка по убыванию вероятности класса
df_sort = df_texts.sort_values(['Макс.вероятность'], ascending=[False])

# Формирование индикатора совпадения класса с прогнозом
df_sort['Совпадение'] = df_sort['Предсказанный класс']==df_sort['НО-
        МЕР_УСЛУГИ']
df_sort['Совпадение'] = df_sort['Совпадение'].apply(lambda x: 1 if x else 0)
# Определение точности соответствующей вероятности класса
df_sort['Точность'] = [ float(df_sort['Совпадение'].iloc[0:i+1].sum())/float(i+1) for i
        in range(0,df_sort.shape[0])]

# Построение графика точности прогноза класса в зависимости от максималь-
        ной вероятности
```

```
import matplotlib.pyplot as plt
plt.style.use('seaborn-whitegrid')
plt.plot(df_sort['Макс.вероятность'], df_sort['Точность'], color='black');

# Сохранение итогов обработки в файл
writer = pd.ExcelWriter(WorkDir+'ЗапросыНомерУслугиПрогноз.xlsx')
df_sort.to_excel(writer,  sheet_name='Прогноз', index=False)
writer.save()

# Вычисление TF-IDF частот
from sklearn.feature_extraction.text import TfidfTransformer
tf_transformer = TfidfTransformer(use_idf=False).fit(M)
M_tf = tf_transformer.transform(M)

# Настрйка мультиклассового байесовского классификатора по  TF-IDF часто-
      там
clf_tf = MultinomialNB().fit(M_tf, df_texts['НОМЕР_УСЛУГИ'])
# Вычисление (предсказание) класса
PredictedClass_tf = clf.predict(M_tf)
# Вычисление вероятностей классов
proba_tf = clf.predict_proba(M_tf)
# Вычисление точности предсказания
clf_score_tf = clf.score(M_tf, df_texts['НОМЕР_УСЛУГИ'])
```