ECS: Experiment Conduction System

Руководство по запуску обучения моделей

Разработка и поддержка:
Константин Ломотин (ke.lomotin@gmail.com)
Екатерина Козлова (hse.kozlovaes@gmail.com)

Оглавление

Запуск программы

Для запуска эксперимента необходимо создать его описание, заполнив поля в файле `settings.ini` в каталоге эксперимента.

Каталог эксперимента – папка, в которой находится файл настроек `settings.ini`, и куда будут сохранены отчеты и созданные модели.

Затем запустить эксперимент следующей командой:

```
python ecs.py <полный_путь_к_папке_эксперимента>
```

Пример:

```
python ecs.py C:\\Users\\username\\exp_folder
```

Примечание. В качестве разделителя в пути можно использовать либо символ «/», либо двойной обратный слэш («\\»), либо заключать путь в двойные кавычки.

Описание состоит из обязательных секций `TrainingData`, `Experiment`, `Preprocessing`, `WordEmbedding` и `Classification`, а также секций гиперпараметров моделей.

Далее в рамке будут приведены примеры заполненных полей.

Добавление датасетов

Чтобы добавить новые данные для обучения. необходимо:

1. Создать папку в любом доступном для чтения месте
2. Перенести туда обучающий файл с расширением CSV в том же формате, который обычно используется при передаче датасета для экспериментов.
3. [Опционально] Перенести в папку тестовый файл в том же формате.

Установка обученных моделей

Чтобы установить в АТС обученный классификатор и созданную модель Word2Vec, необходимо:

1. Скопировать классификатор (файл с расширением .plk) в папку ATC/analyzer/modules/classifier/
2. В файле ATC/analyzer/modules/classifier/config.ini записать имя скопированного файла в поле, соответствующее рубрикатору и языку модели.

   Старые файлы удалять не нужно. Это позволит использовать команду

```
python ATC.py restore classifier
```

   чтобы восстановить файл конфигурации.
3. Скопировать модель Word2Vec (файл с расширением .model) в папку ATC/analyzer/modules/word_embedding/
4. В файле ATC/analyzer/modules/word_embedding/config.ini записать имя скопированного файла в поле, соответствующее языку модели.

5. Из файла ECS/reports/<experiment_title>/settings.ini скопировать значение поля `pooling` из секции `Preprocessing` в поле `convolution` в файле …/word_embedding/config.ini

Важно! Все классификаторы в АТС обучены с векторами, полученными при помощи установленной модели Word2Vec. Если планируется создать и использовать другую модель, нужно также создать и установить классификаторы для всех рубрикаторов и языков. Иначе возникнут конфликты между старыми классификаторами и новой моделью Word2Vec. Это может привести к падению качества или падению программы.

Кэширование

Обработка текстов может занимать длительное время, поэтому ECS имеет механизм кэширования данных.

Весь цикл обработки данных состоит из шагов препроцессинга и создания векторов. Предобработанные тексты и векторы сохраняются в папке в каталоге с датасетом, и, если процесс был прерван, он может использовать результаты предыдущей работы.

Таким образом, кэширование имеет две цели:

1. Избежать повторной обработки в случае прерывания процесса.
2. Избежать некоторых шагов обработки, если уже существуют данные, обработанные с такими же настройками (например, запуск обучения другого классификатора на том же датасете).

Примечание. На данный момент невозможно продолжить прерванный процесс предобработки или создания векторов, т.е. если предобработка не была завершена, процесс начнется с начала, а если не была завершена векторизация, будут использованы предобработанные тексты, чтобы начать ее заново.

Секция `TrainingData`

В этой секции необходимо задать следующие настройки:

1. Полный путь к обучающему файлу в формате CSV.

```
dataset = C:\Users\VINITI\my_dataset\training.csv
```

2. Опция `test_percent`: доля выборки, используемой для валидации моделей.
   Опция `test_file`: название файла в каталоге выбранного датасета, который будет использоваться для валидации.

Необходимо указать только одну из этих опций. Если указаны обе, тестирование будет проводиться при помощи указанного файла (приоритет опции `test_file`).

```
test_percent = 30
test_file = C:\Users\VINITI\my_dataset\validation.csv
```

Секция `Experiment`

1. Название эксперимента. Не должно включать в себя символы [\, *, ", ?, /, :, >, <, |]. Отчет по результатам эксперимента сохраняется в папке с назначенным именем в каталоге эксперимента, указанном при запуске программы.
   Если название не указано, в качестве имени назначается временна́я метка.

```
experiment_title = experiment1
```

```
experiment_title =
```

2. Идентификатор рубрикатора. Допустимо несколько значений из списка [`subj, ipv, rgnti`].
   Для каждого рубрикатора будет выбрана и сохранена лучшая модель.

```
rubricator = subj, rgnti
```

3. Бинарная классификация - 1 против всех (1 vs all). Допустимо одно значение из списка `[true, false]`.

```
binary = 0
```

4. Количество потоков исполнения процесса. Оптимальное количество потоков зависит от количества физических ядер.

```
threads = 2
```

5. Количество фолдов кросс-валидации.

При кросс-валидации обучающая выборка делится на `n_folds` частей (фолдов). На каждой итерации в обучении участвуют `(n_folds - 1)` частей, а оставшаяся часть используется для контроля качества.

Таким образом, вся обучающая выборка принимает участие в обучении. В результате получается оценка эффективности выбранной модели с наиболее равномерным использованием имеющихся данных.

Обычное значение: 3. Поле не может быть пустым.

```
n_folds = 3
```

Секции рубрикаторов

Для каждого значения поля `rubricator` доступны одноименные секции со следующими опциями:

- `min_training_rubric` – минимальное количество текстов для рубрики в **обучающей** выборке. Все рубрики, содержащие строго меньшее количество текстов, не будут участвовать в обучении и тестировании.
- `min_validation_rubric` – минимальное количество текстов для рубрики в **тестовой** выборке. Все рубрики, содержащие строго меньшее количество текстов, не будут участвовать в тестировании.

Если порог не требуется, поле можно оставить пустым, заполнить значением 0 или 1.

```
[rgnti]
 min_training_rubric = 25
 min_validation_rubric =
```

Также из тестовой выборки будут удалены рубрики, которых не было в обучающей.

Секция `Preprocessing`

1. Псевдонимы для колонок таблицы.

   Эти параметры задают названия колонок, которые могут отличаться в разных датасетах. Например, в датасетах с русскими текстами колонка идентификаторов обычно называется *id_publ*, а в датасетах с английскими – *bo_id*. Чтобы точно задать положение всех полей данных, необходимо задать следующие опции:

   - id – колонка идентификаторов текстов;
   - title – колонка названий текстов;

- text – колонка тела аннотации;
- keywords – колонка ключевых слов;
- subj – колонка кодов SUBJ;
- ipv – колонка кодов IPV;
- rgnti – колонка кодов RGNTI;
- correct – колонка признака корректности.

```
id = id_publ
title = title
text = ref_txt
keywords = kw_list
subj = SUBJ
ipv = IPV
rgnti = RGNTI
correct = eor
```

2. Удаление стоп-слов. Допустимо одно значение из списка: [`true, false`].

```
remove_stopwords = true
```

3. Тип нормализации слов. Для русского и английского языков доступны разные наборы алгоритмов. Допустимо одно значение из списка.
   Для русского языка:
   - `snowball`
   - `pymystem`
   - `no`

   Для английского языка:
   - `snowball`
   - `wordnet`
   - `porter`
   - `lancaster`
   - `textblob`

Лемматизация – приведение слова в нормальную форму. Реализуется при помощи набора эвристик и словарей. Медленная операция.

Стемминг – удаление частей слова, различающих его разные формы (суффиксов, окончаний и т.д.). Практически не требует словарей, разработаны алгоритмы стемминга. При стемминге теряется значительная часть информации. Работает достаточно быстро, но качество классификации в итоге часто оказывается ниже.

```
normalization = snowball
```

Snowball – алгоритм стемминга, улучшенный стеммер Портера (иногда его называют Porter2). Работает для русского и английского языка.
Pymystem – алгоритм лемматизации, разработанный в Яндекс. Достаточно медленный, но качественный. Только для русского языка.
Wordnet – лемматизатор, основанный на крупной базе английских слов WordNet. Только для английского языка.
Porter – классический алгоритм стемминга. Только для английского языка.
Lancaster – алгоритм стемминга. Более агрессивен, чем стеммеры Портера. Только для английского языка.
Textblob – еще один инструмент, реализующий лемматизацию. Только для английского языка.

Для обратной совместимости значения **lemmatization** и **stemming** также могут быть указаны. В этом случае для стемминга будет использован стеммер **Snowball**, для лемматизации русского текста – **Pymystem**, для лемматизации английского текста – **Wordnet**.

4. Разделитель ключевых слов. В различных датасетах ключевые слова бывают разделены разными символами. Для корректной обработки необходимо указать этот символ в поле `kw_delim`.

```
kw_delim = \
```

5. Язык текста. Допустимо одно значение из списка [`ru, en, auto`].

```
language = auto
```

6. Количество текстов, которые будут обрабатываться одновременно.
   Библиотеки для нормализации и движок регулярных выражений Python лучше оптимизированы для обработки одного большого текста, чем для обработки множества маленьких.
   Поэтому `batch_size` текстов сначала объединяются в один, обрабатываются за раз, а затем разбиваются обратно. Чем это значение больше, тем больше оперативной памяти будет занято при предобработке. Чем оно меньше, тем дольше будет предобработка.
   Если указанный размер батча больше, чем размер датасета, весь датасет будет предобработан за раз.

Допустимые значения – целые числа.

На практике предобработка 50.000 текстов за раз занимает порядка 20 минут и занимает не слишком много памяти.

```
batch_size = 50000
```

Секция `WordEmbedding`

1. Если есть подходящая обученная модель Word2Vec, то ее можно использовать для создания векторов. Все модели W2V сохраняются в указанной при запуске папке.

   Чтобы использовать существующую модель, в поле `use_model` необходимо указать полный путь к файлу модели.

   Если не удастся загрузить файл модели, или поле будет пустым, будет создана новая модель.

   Допустимые значения – полный путь к файлу модели Word2Vec, пустое поле.

```
use_model = C:\Users\username\Desktop\w2v_model.model
```

```
use_model =
```

2. Размерность векторов. Это значение соответствует количеству семантических признаков, выделяемых моделью. Слишком большое значение приводит к сильному падению производительности. Слишком маленькое – к падению качества классификации.

   В результате серии экспериментов было установлено, что при размерности, большей 50, качество практически не увеличивается, зато значительно падает скорость работы.

   Допустимое значение – целое положительное число.

```
vector_dim = 50
```

3. Принцип word embedding предназначен для отображения слов в векторное пространство признаков. Текст, таким образом, кодируется матрицей, где количество строк равно количеству слов в тексте, а число столбцов – размерности векторов.

Текущие поддерживаемые модели могут работать только с векторами, поэтому к матрице текста применяется операция пулинга по строкам в пределах каждого столбца.

Можно выбрать один из типов пулинга:

- `max` (максимум по строкам);
- `mean` (среднее значение столбца);
- `sum` (сумма столбца).

Серия экспериментов показала, что векторы, созданные при помощи sum pooling, позволяют достичь наиболее высокого качества классификации при использовании основных классификаторов.

```
pooling = sum
```

4. Размер контекстного окна. Количество слов, которые составляют контекст при создании модели Word2Vec.
   Допустимое значение – целое положительное число.

```
window = 5
```

Секция `Classification`

1. Модели для перебора. Допустимо несколько значений из списка
   - `logistic_regression`
   - `svm`
   - `perceptron`
   - `knn`
   - `decision_tree`
   - `random_forest`
   - `adaboost`

Для каждой из перечисленных моделей будет выбран и сохранен вариант, для которого сочетание гиперпараметров дало наиболее высокое качество классификации.

Для каждой модели в соответствующей секции необходимо указать гиперпараметры для перебора.

```
models = logistic_regression, perceptron
```

Гиперпараметры моделей

Для выбора наиболее эффективной модели необходимо задать список моделей в поле models секции Experiment, а также создать секции с гиперпараметрами для каждого выбранного алгоритма. Названия секций должны совпадать с названиями выбранных моделей.

Используемый фреймворк *scikit-learn* позволяет настраивать модели при помощи большого числа гиперпараметров. Многие из них имеют значение по умолчанию, которое мало влияет на качество классификации. Поэтому обычно в переборе такие гиперпараметры не участвуют. Ниже будет приведено описание только тех гиперпараметров, которые используются чаще всего. **Полужирным** шрифтом будет выделено значение по умолчанию.

Для каждого гиперпараметра может быть задано несколько значений через запятую. Например:

```
activation = logistic, relu
```

```
kernel = linear, poly, rbf, sigmoid
```

Каждое добавленное значение увеличивает количество комбинаций гиперпараметров и увеличивает время работы программы, так как для каждой комбинации будет обучена и протестирована модель.

Поле гиперпараметра можно удалить или оставить пустым. Например:

```
solver =
```

В таком случае будет использоваться значение по умолчанию.

Важно! Значения некоторых гиперпараметров несовместимы между собой. Например, некоторые алгоритмы обучения не поддерживают определенные виды регуляризации. Такие комбинации гиперпараметров могут вызвать некорректную работу или падение программы.

Некоторые сочетания гиперпараметров могут быть несовместимы, но не вызывать ошибки. В таком случае один из гиперпараметров игнорируется и их не нужно удалять из описания. Далее указано, какие из них не учитываются в сочетании с другими, а какие вызывают ошибку.

Секция `logistic_regression`

Логистическая регрессия – это алгоритм, позволяющий использовать линейную регрессию для классификации. Метод заключается в построении разделяющей гиперплоскости, разделяющей классы. Логистическая функция (сигмоида) позволяет отнести объект к тому или иному классу.

Документация, информация о модели и полный список гиперпараметров доступны по ссылке.

1. `penalty`
   Вид регуляризации. Регуляризация препятствует переобучению модели, накладывая штраф на слишком высокие значения весовых коэффициентов. Вид регуляризации определяет то, как эти штрафы вычисляются. Возможные значения:
   - `l1` (**несовместимо** со значениями `newton-cg`, `sag` и `lbfgs` гиперпараметра `solver`)
   - **`l2`**
2. `C`
   Ослабление регуляризации.
   Чем ниже это значение, тем сильнее будет штраф за высокие значения коэффициентов модели (регуляризация).
   Оптимальное значение может варьироваться до нескольких порядков.
   Значение по умолчанию: **`1.0`**

```
C = 0.001, 0.1, 10
```

3. `fit_intercept`
   Добавлять ли в уравнение линейной модели свободное слагаемой, т.н. *смещение*. Допустимые значения (регистр в гиперпараметрах параметрах имеет значение):
   - **`True`**
   - `False`
4. `random_state`
   Зерно для генерации случайного числа. Служит для инициализации коэффициентов модели. Задание этого числа вручную позволяет сделать эксперимент повторяемым. Допустимые значения: целые числа.
5. `solver`
   Алгоритм оптимизации, применяющийся для обучения. Допустимые значения:
   - `newton-cg`

- `lbfgs`
- **`liblinear`**
- `sag`
- `saga`

Примечание. Алгоритм `liblinear` хорошо подходит для небольших датасетов, тогда как `sag` и `saga` работают быстрее на крупных наборах данных.

6. `max_iter`
   Ограничение на максимальное количество итераций обучения.
   Значение по умолчанию: **`100`**.

Секция `svm`

Метод опорных векторов (Support Vector Machine) – линейный алгоритм, часто применяющийся в классификации. Его сильной стороной является возможность нелинейного преобразования пространства признаков при помощи перехода от скалярных произведений к произвольным ядрам.

Документация, информация о модели и полный список гиперпараметров доступны по ссылке.

1. `C`
   Ослабление регуляризации.
   Чем ниже это значение, тем сильнее будет штраф за высокие значения коэффициентов модели (регуляризация).
   Оптимальное значение может варьироваться до нескольких порядков.
   Значение по умолчанию: **`1.0`**

```
C = 0.001, 0.1, 10
```

2. `kernel`
   Ядро, используемое для преобразования пространства признаков.
   Допустимые значения:
   - `linear`
   - `poly`
   - **`rbf`**
   - `sigmoid`
3. `degree`
   Степень полинома при использовании полиномиального ядра (значение `poly` опции `kernel`). Игнорируется при использовании других ядер.
   Значение по умолчанию: **`3`**.

4. `gamma`
   Коэффициент ядра при значениях `rbf`, `poly` и `sigmoid` опции `kernel`. Игнорируется при использовании линейного ядра (`linear`).
   Значение по умолчанию: **1/n_features**, где n_features – количество признаков (размерность вектора текста).
5. `coef0`
   Свободный член в уравнении гиперплоскости. Игнорируется при использовании ядер `linear` и `rbf`.
   Значение по умолчанию: **`0`**.
6. `max_iter`
   Ограничение на максимальное количество итераций обучения.
   Значение по умолчанию: **`-1`** (без ограничения).
7. `random_state`
   Зерно для генерации случайного числа. Служит для инициализации коэффициентов модели. Задание этого числа вручную позволяет сделать эксперимент повторяемым. Допустимые значения: целые числа.

Секция `perceptron`

Простая многослойная нейронная сеть, в которой сигнал распространяется от входов к выходам.

Документация, информация о модели и полный список гиперпараметров доступны по ссылке.

1. `hidden_layer_sizes`
   Количество нейронов в скрытых слоях нейросети.
   Если xn - количество нейронов в n-м скрытом слое, то запись `[x1, x2], [y1, y2, y3]` означает, что сначала будет обучена нейросеть с двумя скрытыми слоями с `x1` и `x2` нейронами, а затем модель с тремя слоями с `y1`, `y2` и `y3` нейронами.
   Описание каждой нейросети обязательно заключать в квадратные скобки:

```
hidden_layer_sizes = [10, 15, 1], [6]
```

2. `activation`
   Функция активации нейронов. На данный момент возможно использование одной функции активации для всех нейронов. Список допустимых значений:
   - `identity`

- `logistic`
- `tanh`
- **`relu`**

3. `solver`
   Алгоритм оптимизации, применяющийся для обучения. Допустимые значения:
   - `lbfgs`
   - `sgd`
   - **`adam`**

   Примечание. Для больших датасетов алгоритм `adam` работает быстрее на обучении и валидации на больших датасетах (тысячи и десятки тысяч записей). Алгоритм `lbfgs` сходится и работает быстрее на маленьких наборах данных.

4. `alpha`
   Коэффициент регуляризации. Регуляризация препятствует переобучению модели, накладывая штраф на слишком высокие значения весовых коэффициентов.
   Значение по умолчанию: **`0.0001`**
5. `learning_rate`
   Коэффициент скорости обучения. Слишком большое значение может привести к тому, что алгоритм обучения никогда не сойдется. Слишком маленький коэффициент приведет к слишком долгой сходимости. Используется только при алгоритме оптимизации `sgd`.
   Допустимые значения:
   - **`constant`**
   - `invscaling`
   - `adaptive`
6. `learning_rate_init`
   Начальное значение гиперпараметра `learning_rate`. При постоянной скорости обучения остается неизменным, а при `invscaling` и `adaptive` уменьшается со временем.
   Значение по умолчанию: **`0.001`**
7. `power_t`
   Значение по умолчанию: **`0.5`**. При использовании `invscaling` `learning_rate` скорость обучения меняется по формуле:

$$\text{effective\_learning\_rate} = \text{learning\_rate\_init} / \text{pow}(t, \text{power\_t})$$

где t – номер итерации,
effective_learning_rate – скорость обучения на шаге t,

learning_rate_init – начальная скорость обучения.

Гиперпараметр игнорируется при использовании других видов скорости обучения.

8. `max_iter`
   Ограничение на максимальное количество итераций обучения.
   Значение по умолчанию: **`200`**.
9. `random_state`
   Зерно для генерации случайного числа. Служит для инициализации коэффициентов модели. Задание этого числа вручную позволяет сделать эксперимент повторяемым. Допустимые значения: целые числа.

Секция `knn`

Метрический алгоритм, суть которого заключается в том, что новый объект присваивается тому классу, который является наиболее распространённым среди `n_neighbors` соседей данного элемента, классы которых уже известны.

Документация, информация о модели и полный список гиперпараметров доступны по ссылке.

1. `n_neighbors`
   Количество ближайших соседей объекта. Оптимальное значение зависит от данных. Большое количество соседей часто помогает подавить шумы, но делает границу между классами менее отчетливой.
   Значение по умолчанию: **5**.
2. `weights`
   Весовая функция, используемая в предсказании. Допустимые значения:
   - **`uniform`** (все соседи оцениваются одинаково)
   - `distance` (соседи поблизости вносят больший вклад, чем дальние)
3. `p`
   Степень метрики Минковского. При $p = 1$ используется Манхэттенское расстояние, при $p = 2$ используется Евклидово расстояние, при произвольном p используется метрика Минковского в общем виде.
   Значение по умолчанию: **2**.

Секция `decision_tree`

Алгоритм поддержки принятия решений, выявляющий закономерности в признаках объектов обучающей выборки.

Документация, информация о модели и полный список гиперпараметров доступны по ссылке.

1. `criterion`
   Критерий меры качества разбиения образцов по ветвям. Допустимые значения:
   - **`gini`**
   - `entropy`
2. `splitter`
   Стратегия выбора разбиения. Допустимые значения:
   - **`best`** (лучшее разбиение)
   - `random` (лучшее случайное разбиение)
3. `max_depth`
   Максимальная глубина дерева.
   Значение по умолчанию: **None** (неограниченно).
   Примечание. Парсинг значения None пока не реализован, поэтому оно не может быть указано. Вместо этого допустимо оставить поле пустым.
4. `min_samples_split`
   Минимальное количество образцов, необходимое для разбиения узла.
   Значение по умолчанию: **2**.
5. `min_samples_leaf`
   Минимальное количество образцов для формирования листа дерева.
   Значение по умолчанию: **1**.
6. `max_features`
   Количество признаков, рассматриваемых при каждом разбиении.
   Значение по умолчанию: **None** (все признаки).
   Примечание. Парсинг значения None пока не реализован, поэтому оно не может быть указано. Вместо этого допустимо оставить поле пустым.
   Допустимые значения:
   - целое число признаков
   - число с плавающей точкой (доля признаков от всех)
   - `auto` (в этой реализации эквивалентно sqrt)
   - `sqrt` (квадратный корень от всех признаков)
   - `log2` (логарифм по основанию 2 от всех признаков)
7. `random_state`

Зерно для генерации случайного числа. Служит для инициализации коэффициентов модели. Задание этого числа вручную позволяет сделать эксперимент повторяемым. Допустимые значения: целые числа.

Секция `random_forest`

Классификатор «Случайный лес» основан на бэггинге (обучении нескольких моделей на разных подвыборках с последующим голосованием), однако в дополнение к нему случайным образом выбирает подмножество признаков в каждом узле, с целью сделать деревья более независимыми;

Документация, информация о модели и полный список гиперпараметров доступны по ссылке.

1. `n_estimators`
   Количество деревьев решений в составе модели.
   Значение по умолчанию: **`10`**.
2. `criterion`
   Критерий меры качества разбиения образцов по ветвям. Допустимые значения:
   - **`gini`**
   - `entropy`
3. `max_depth`
   Максимальная глубина дерева.
   Значение по умолчанию: **None** (неограниченно).
   Примечание. Парсинг значения None пока не реализован, поэтому оно не может быть указано. Вместо этого допустимо оставить поле пустым.
4. `min_samples_split`
   Минимальное количество образцов, необходимое для разбиения узла.
   Значение по умолчанию: **2**.
5. `min_samples_leaf`
   Минимальное количество образцов для формирования листа дерева.
   Значение по умолчанию: **1**.
6. `max_features`
   Количество признаков, рассматриваемых при каждом разбиении.
   Значение по умолчанию: **None** (все признаки).
   Примечание. Парсинг значения None пока не реализован, поэтому оно не может быть указано. Вместо этого допустимо оставить поле пустым.
   Допустимые значения:
   - целое число признаков

- число с плавающей точкой (доля признаков от всех)
- **auto** (в этой реализации эквивалентно sqrt)
- sqrt (квадратный корень от всех признаков)
- log2 (логарифм по основанию 2 от всех признаков)

7. random_state
   Зерно для генерации случайного числа. Служит для инициализации коэффициентов модели. Задание этого числа вручную позволяет сделать эксперимент повторяемым. Допустимые значения: целые числа.

Секция adaboost

Алгоритм адаптивного бустинга. Идея бустинга заключается в построении каскада моделей, где каждая последующая модель обучается на ошибках предыдущей.

Документация, информация о модели и полный список гиперпараметров доступны по ссылке.

Эта реализация ансамблевого алгоритма AdaBoost способна работать с различными классификаторами, но в ECS на текущий момент реализован адаптивный бустинг на базе деревьев решений.

1. n_estimators
   Максимальное количество моделей, объединяемых в бустинг. Обучение прекращается, как только завершается обучение этого количества моделей. Обучение может завершиться раньше, если ошибка станет убывать слишком медленно.
   Значение по умолчанию: **50**.
2. learning_rate
   Вклад каждой модели в составе бустинга. Для оптимального результата нужно найти компромисс между количеством моделей и долей вклада каждой из них.
   Значение по умолчанию: **1**.
3. algorithm
   Алгоритм обучения бустинга. Допустимые значения:
   - **SAMME.R** (как правило, сходится быстрее и достигает более низкой ошибки)
   - SAMME
4. random_state
   Зерно для генерации случайного числа. Служит для инициализации коэффициентов модели. Задание этого числа

вручную позволяет сделать эксперимент повторяемым. Допустимые значения: целые числа.