# РАБОТА АНСАМБЛЯМИ МЕТОДОВ

Задание основано на применении пакета Python Scikit-learn для ансамблирования методов на примере градиентного бустинга и случайного леса.

Цели практики:

- освоить работу с градиентным бустингом и подбором его гиперпараметров
- научиться сравнивать разные способы построения ансамблей
- узнать, в каком случае лучше использовать случайный лес, а где градиентный бустинг показывает более высокие результаты
- овладеть применением метрики log-loss

Начало работы

В пакете scikit-learn градиентный бустинг реализован в модуле **ensemble** в виде классов **GradientBoostingClassifier** и **GradientBoostingRegressor**. Основные параметры, которые будут интересовать нас: n**_estimators, learning_rate.**

Иногда может быть полезен параметр **verbose** для отслеживания процесса обучения.

Чтобы была возможность оценить качество построенной композиции на каждой итерации, у класса есть метод **staged_decision_function**. Для заданной выборки он возвращает ответ на каждой итерации.

В этом задании предложено воспользоваться функцией **train_test_split** модуля **cross_validation.** С помощью нее можно разбивать выборки случайным образом. На вход можно передать несколько выборок (с условием, что они имеют одинаковое количество строк).

Пусть, например, имеются данные X и y, где X — это признаковое описание объектов, y — целевое значение. Тогда следующий код будет удобен для разбиения этих данных на обучающее и тестовое множества:

```
X_train, X_test, y_train, y_test =
                    train_test_split(X, y,
                                     test_size=0.33,
                                            random_state=42)
```

При фиксированном параметре **random_state** результат разбиения можно воспроизвести.

Метрика **log-loss** реализована в пакете **metrics.** Данная метрика предназначена для

классификаторов, выдающих оценку принадлежности классу, а не бинарные ответы. Градиентный бустинг, и случайный лес обладают свойствами построения прогнозов, для этого необходимо применять **predict_proba**:

pred = c l f . predict_proba ( X_test )

Метод **predict_proba** возвращает матрицу, i-й столбец которой содержит оценки принадлежности i-му классу.

Для рисования кривых качества на обучении и контроле можно воспользоваться следующим кодом:

```
import matplotlib.pyplot as plt
%matplotlib inline
plt.figure()
plt.plot(test_loss, 'r', linewidth=2)
plt.plot(train_loss, 'g', linewidth=2)
plt.legend(['test', 'train'])
```

**Задание**

В рамках задания используется датасет «gbm-data.csv».

1. Загрузите выборку из файла gbm-data.csv с помощью pandas и преобразуйте ее в массив numpy (параметр ***values*** у датафрейма). В первой колонке файла с данными записано, была или нет реакция.
   Остальные колонки *(d1 - d1776)* содержат различные характеристики молекулы, такие как *размер, форма и т.д.*
   Разбейте выборку на обучающую и тестовую, используя функцию **train_test_split** с параметрами **test_size = 0.8 и random_state = 241.**

2. Обучите **GradientBoostingClassifier** с параметрами **n_estimators=250**, **verbose=True, random_state=241** и для каждого значения **learning_rate** из списка **[1, 0.5, 0.3, 0.2, 0.1]** проделайте следующее:

   - используйте метод **staged_decision_function** для предсказания качества на обучающей и тестовой выборке на каждой итерации.

   - прреобразуйте полученное предсказание по формуле $\frac{1}{1+e^{-y\_pred}}$

   где y_pred — предсказанное значение.

   - Вычислите и постройте график значений log-loss на обучающей и

тестовой выборках, а также найдите минимальное значение метрики и номер итерации, на которой оно достигается.

3. Как можно охарактеризовать график качества на тестовой выборке, начиная с некоторой итерации: переобучение **(overfitting)** или недообучение **(underfitting)?**

   В ответе укажите одно из слов ***overfitting*** либо ***underfitting***.

4. Приведите минимальное значение **log-loss** на тестовой выборке и номер итерации, на котором оно достигается, при **learning_rate = 0.2**.

5. На этих же данных обучите **RandomForestClassifier** с количеством деревьев, равным количеству итераций, на котором достигается наилучшее качество у градиентного бустинга из предыдущего пункта, **random_state=241** и остальными параметрами по умолчанию.

   Какое значение **log-loss** на тесте получается у этого случайного леса? (ВАЖНО: предсказания нужно получать с помощью функции **predict_proba**. В данном случае брать сигмоиду от оценки вероятности класса не нужно).

**Ответ на каждое задание** — текстовый файл, содержащий ответ в первой строчке (#.txt). Обратите внимание, что отправляемые файлы не должны содержать перевод строки в конце.

**Утонения по выполнению задания:**

Если ответом является нецелое число, то целую и дробную часть необходимо разграничивать точкой, например, 0.42.
При необходимости округляйте дробную часть до двух знаков

## ТЕОРЕТИЧЕСКИЕ ОСНОВЫ

Построение ансамблей представляет собой важный метод в области машинного обучения, который объединяет множество слабых алгоритмов в один более мощный. Этот подход широко применяется на практике для решения различных задач.

Например, метод градиентного бустинга, может последовательно создавать ансамбль алгоритмов. Каждый последующий алгоритм выбирается таким образом, чтобы исправлять ошибки, допущенные предыдущими алгоритмами в ансамбле. Обычно в роли базовых алгоритмов используются деревья небольшой глубины, поскольку их легко строить, и они создают нелинейные разделяющие поверхности.

Еще одним методом построения ансамбля является случайный лес. В отличие от градиентного бустинга, здесь заключается построении отдельных деревьях независимо и без ограничений на глубину. Дерево наращивается до тех пор, пока не достигнет наилучшего качества на обучающей выборке.

В данном задании мы решаем задачу классификации. В качестве функции потерь используется log-loss:

$$\text{log-loss} = -\frac{1}{N}\sum_{i=1}^{N}[y_i \log(z_i) + (1 - y_i)\log(1 - z_i)]$$

uде $y_i$ - истинный ответ, $z_i$ - прогноз алгоритма.

Эта функция дифференцируема, что делает ее подходящей для использования в градиентном бустинге. При ее использовании можно также показать, что итоговый алгоритм будет приближать вероятности принадлежности к классам.