**Лабораторная работа 7.1. Бустинг**

Для решения первого задания урока сначала подготовим данные. Выкачаем набор данных, в котором будет прогнозировать, является ли письмо спамом.

Сделаем это следующим кодом:

df = pd.read\_csv('https://raw.githubusercontent.com/vincentarelbundock/Rdatasets/master/csv/DAAG/spam7.csv')

Из столбца *yesno* сделайте столбец *spam*, где 1 соответствует значению *y* (является спамом), а 0 – *n* (не является спамом). Удалите первый столбец (с индексом) и столбец *"yesno*".

Комбинации признаков должны иметь вид **"признак1\_признак2"**.

У Вас остается шесть столбцов-признаков и один столбец *spam*, который является нашей целевой переменной. Разделим их на *x* и *у*.

Теперь мы хотим создать новые признаки путем попарного перемножения уже имеющихся друг на друга.

**Задание 7.4.1**

Сколько получится признаков после преобразования (включая те шесть, которые были в исходных данных)?

* 15
* 21
* 20
* 14

**Задание 7.4.2**

Разделим получившися данные на обучающую и тестовую выборки в соотношение 80:20. Обучите модель *GradientBoostingClassifier* (из библиотеки *sklearn)* с параметрами по умолчанию (на момент создания урока это learning\_rate=0.1, n\_estimators=100,max\_depth=3, min\_samples\_split=2, min\_samples\_leaf=1, subsample=1,max\_features=None).

Во всех методах, где присутствует случайность, укажите random\_state=42.

Это будет наш *baseline*.

Укажите точность ***accuracy*** на тестовой выборке (параметр score), округлив до третьего знака после точки-разделителя.

**Задание 7.4.3**

Воспользуйтесь методом *feature\_importances\_*. С его помощью вы можете оценить вклад каждого признака в обучение модели. Постройте столбчатую диаграмму, где по горизонтали указаны названия признаков, а по вертикали их важность. В ответе укажите третий по важности признак.

**Задание 7.4.4**

Теперь перейдем к подбору параметров в модели. Для этого в библиотеке *scikit-learn* есть метод *GridSearchCV*. Ему на вход подается модель, список параметров и способ оценивания. При запуске метода *fit* модель обучается со всеми возможными комбинациями параметров, лучшей комбинацией параметров становятся та, при которой значение метрики максимально.

Укажите в качестве *estimator* градиетный бустинг с параметрами по дефолту и random\_state=42. В параметре *scoring* укажите 'accuracy', n\_jobs=-1, cv=5.

Сетку параметров задайте такой:

param\_grid = {'learning\_rate':[0.00001, 0.0001, 0.001, 0.01, 0.1, 1],

'n\_estimators':[100, 250, 500, 750, 1000, 1250, 1500, 1750]}

В ответе укажите точность, округленную до Трёх знаков после точки разделителя, для лучших параметров на кросс-валидации. Для этого посмотрите в [документации](https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.GridSearchCV.html#sklearn.model_selection.GridSearchCV) *sklearn*, какой атрибут GridSearchCV за это отвечает.

**Задание 7.4.5**

Подбор параметров — незаменимый инструмент при настройке моделей. Однако, если данных очень много, а модель сложная, то иногда имеет смысл ограничить количество параметров, которые вы хотите перебрать. Если модель с одним набором параметров обучается 30 минут, а вы хотите перебрать 10 таких наборов, то стоит приготовиться к пятичасовому ожиданию. Также стоит учитывать, что размер некоторых параметров влияет на время, потраченное на обучение моделей. Например, менее глубокие деревья учатся быстрее, чем глубокие.

При использовании *GridSearchCV* перебираются все возможные способы комбинации параметров. Сколько разных моделей (моделей градиентного бустинга с разными наборами параметров) обучилось в прошлом задании?

**Задание 7.4.6**

Теперь воспользуемся *GridSearchCV* для подбора максимальной глубины (max\_depth). В качестве *estimator* используйте градиентный бустинг с раннеее подобранными параметрами и random\_state=42. Переберите все значения max\_depth от 5 до 15 включительно. В ответ запишите значение параметра, при котором точность максимальна.

**Задание 7.4.7**

Может показаться, что, научившись запускать готовые фукнции из библиотек и настраивать параметры, мы абсолютно готовы к работе. Однако на поле боя нас могут подстерегать еще опасности — *legacy code*, то есть код, написанный кем-то до вас. Может быть, его можно было бы и улучшить, но на нём уже столько всего надстроено, что переписывать всё практически невозможно.

А ещё часто будет возникать необходимость просто понять код коллеги или сделать *code review* (грубо говоря, это анализ кода на предмет ошибок и расхождения в стиле). Во всех этих ситуациях необходим навык быcтрого чтения кода.

Поэтому в качестве примера в этом задании мы предлагаем изучить код для реализации *AdaBoost* с нуля. Вам необходимо запустить эту функцию, но с дописанными строчками (вычисление ошибки и веса алгоритма) на всех признаках, получившихся после всех преобразований в предыдущем датасете (до разделения выборки на обучающую и тестовую). Параметр *learning\_rate* возьмите равным 0.001, М = 10. В ответ запишите точность, округленную до трёх знаков после точки-разделителя.

**За основу нужно взять следующий код:**

defAdaBoost\_scratch(X,y, M=10, learning\_rate =1):

# инициалиазция служебных переменных

N = len(y)

estimator\_list, y\_predict\_list, estimator\_error\_list, estimator\_weight\_list, sample\_weight\_list = [], [],[],[],[]

# инициализация весов

sample\_weight = np.ones(N) / N

sample\_weight\_list.append(sample\_weight.copy())

# цикл по длине М

for m in range(M):

# обучим базовую модель и получим предсказание

estimator = DecisionTreeClassifier(max\_depth = 1, max\_leaf\_nodes=2)

estimator.fit(X, y, sample\_weight=sample\_weight)

y\_predict = estimator.predict(X)

# Маска для ошибок классификации

incorrect = (y\_predict != y)

# Оцениваем ошибку

#estimator\_error = TODO

# Вычисляем вес нового алгоритма

#estimator\_weight = TODO

# Получаем новые веса объектов

sample\_weight \*= np.exp(estimator\_weight \* incorrect \* ((sample\_weight > 0) | (estimator\_weight < 0)))

# Сохраяем результаты данной итерации

estimator\_list.append(estimator)

y\_predict\_list.append(y\_predict.copy())

estimator\_error\_list.append(estimator\_error.copy())

estimator\_weight\_list.append(estimator\_weight.copy())

sample\_weight\_list.append(sample\_weight.copy())

# Для удобства переведем в numpy.array

estimator\_list = np.asarray(estimator\_list)

y\_predict\_list = np.asarray(y\_predict\_list)

estimator\_error\_list = np.asarray(estimator\_error\_list)

estimator\_weight\_list = np.asarray(estimator\_weight\_list)

sample\_weight\_list = np.asarray(sample\_weight\_list)

# Получим предсказания

preds = (np.array([np.sign((y\_predict\_list[:,point] \* estimator\_weight\_list).sum()) for point in range(N)]))

print('Accuracy = ', (preds == y).sum() / N)

return estimator\_list, estimator\_weight\_list, sample\_weight\_list

estimator\_list, estimator\_weight\_list, sample\_weight\_list = AdaBoost\_scratch(X, y, M=10, learning\_rate=0.001)