Машинное обучение, DS-поток

Домашнее задание 2

Правила:

- Дедлайн **28 февраля 10:00**. После дедлайна работы не принимаются кроме случаев наличия уважительной причины.
- Выполненную работу нужно отправить на почту `mipt.stats@yandex.ru` , указав тему письма `"[ml] Фамилия Имя - задание 2"` . Квадратные скобки обязательны. Если письмо дошло, придет ответ от автоответчика.
- Прислать нужно ноутбук и его pdf-версию (без архивов). Названия файлов должны быть такими: `2.N.ipynb` и `2.N.pdf` , где `N` - ваш номер из таблицы с оценками.
- Решения, размещенные на каких-либо интернет-ресурсах не принимаются. Кроме того, публикация решения в открытом доступе может быть приравнена к предоставлении возможности списать.
- Для выполнения задания используйте этот ноутбук в качестве основы, ничего не удаляя из него.
- Никакой код из данного задания при проверке запускаться не будет.

Баллы за задание:

- Задача 1 - 5 баллов
- Задача 2 - 15 баллов

In [ ]:

```
%matplotlib inline

import numpy as np
import pandas as pd
import seaborn as sns
import scipy.stats as sps
import matplotlib.pyplot as plt

from sklearn.metrics import accuracy_score
from sklearn.datasets import load_breast_cancer
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import train_test_split

import warnings

sns.set_style("dark")
sns.set(font_scale=1.4)
warnings.filterwarnings('ignore')
```

Задача 1

Рассмотрим метод *логистической регрессии*. Пусть $x_i \in \mathbb{R}^d, Y_i \sim Bern(\mu_\theta(x_i))$.

Мы предполагаем, что $\mu_\theta(x_i) = P_\theta(Y_i = 1) = \sigma(x_i^T\theta) = \frac{e^{x_i^T\theta}}{1+e^{x_i^T\theta}}$.

Регуляризацию в методе логистической регрессии можно задать с помощью введения априорного распределения на θ. Будем считать, что априорное распределения $\mathcal{N}(0, \alpha^{-1} I_d)$. В данном случае распределения не являются сопряженными, поэтому простым путем найти апостериорное распределение не получится. Однако, можно найти моду этого распределения. Выпишите соответствующую задачу оптимизации.

Для данной задачи:

1. Получите формулу градиентного спуска.
2. Получите формулу метода IRLS.

Задача 2

1.

Реализуйте логистическую регрессию с регуляризацией для трех вариантов поиска оценки параметров:

- обычный градиентный спуск;
- стохастический mini-batch градиентный спуск, размер батча 5-10;
- IRLS.

Для измерения времени работы **каждого** шага используйте

```
from time import time
```

Замечание. Для чистоты эксперимента время шага внутри цикла `for` нужно замерять от конца предыдущего шага до конца текущего, а не от начала текущего шага.

In [ ]:

```python
class LogisticRegression():
    '''
    Модель логистической регрессии. Имеет следующие гиперпараметры:

    * alpha: параметр регуляризации.
            Если равно 0, то регуляризация не происходит.
    * lr: константа, на которую домножаем градиент при обучении
    * eps: ограничение на норму невязки в случае
         если используется критерий criterion='eps'
    * max_iter: ограничение на кол-во итераций в случае
              если используется критерий criterion='max_iter'
    * method: если равно 'gd', то используется обычный градиентный спуск,
             если равно 'sgd', то используется стохастический
                  градиентный спуск,
             если равно 'irls', то используется метод IRLS.
    * criterion: если равно 'eps', то используем ограничение
                  на норму невязки,
                если равно 'max_iter', то используем ограничение
                  на количество итераций
    * fit_intercept: указывает, следует ли добавить константу в признаки
    * save_history: указывает, следует ли сохранять историю обучения
    '''


    def __init__(self, alpha=0, lr=0.5, eps=1e-3, max_iter=1e5,
                 method='gd', criterion='max_iter',
                 fit_intercept=True, save_history=True):
        ''' Создает модель и инициализирует параметры '''

        assert criterion in ['max_iter', 'eps'], 'выбран неправильный критерий
        assert method in ['gd', 'sgd', 'irls'], 'выбран неправильный метод'

        self.alpha = alpha
        self.lr = lr
        self.eps = eps
        self.max_iter = max_iter
        self.criterion = criterion
        self.method = method
        self.fit_intercept = fit_intercept
        self.save_history = save_history
        self.history = []  # для хранения истории обучения


    @staticmethod
    def _sigmoid(x):
        return 1 / (1 + np.exp(-x))


    def _compute_loss(self, X, y):
        return <...>


    def _add_intercept(self, X):
        '''
        Добавляем свободный коэфициент к нашей модели.
        Это происходит путем добавления вектора из 1 к исходной матрице.
        '''

        X_copy = np.full((X.shape[0], X.shape[1] + 1), fill_value=1)
```

```
        X_copy[:, :-1] = X

        return X_copy


    def fit(self, X, Y):
        '''
        Обучает модель логистической регресии с помощью выбранного метода,
        пока не выполнится критерий остновки self.criterion.
        Также, в случае self.save_history=True, добавляет в self.history
        текущее значение оптимизируемого функционала
        и время обновления коэффициентов.
        '''

        assert X.shape[0] == Y.shape[0]

        if self.fit_intercept:  # добавляем свободный коэфициент
            X_copy = self._add_intercept(X)
        else:
            X_copy = X.copy()

        <...>

        self.coef_ = <...>  # коэффициенты модели
        self.intercept_ = <...>  # свободный коэффициент
        self.n_iter_ = <...>  # произведенное число итераций

        return self


    def predict(self, X):
        '''
        Применяет обученную модель к данным
        и возвращает точечное предсказание (оценку класса).
        '''

        if self.fit_intercept:
            X_copy = self._add_intercept(X)
        else:
            X_copy = X.copy()

        assert X_copy.shape[1] == self.weights.shape[0]

        <...>

        return predictions  # shape = (n_test,)


    def predict_proba(self, X):
        ''' Применяет обученную модель к данным
        и возвращает предсказание вероятности классов 0 и 1. '''

        if self.fit_intercept:
            X_copy = self._add_intercept(X)
        else:
            X_copy = X.copy()

        assert X_copy.shape[1] == self.weights.shape[0]

        <...>

```

```
121         return prob_predictions  # shape = (n_test, 2)
```

Рассмотрим игрушечный датасет на 30 признаков `load_breast_cancer` из библиотеки `sklearn`. Это относительно простой для двуклассовой классификации датасет по диагностике рака молочной железы.

Ради интереса можно прочитать описание признаков.

In [ ]:

```
1 dataset = load_breast_cancer()
2 dataset['DESCR'].split('\n')[11:31]
```

Разделим нашу выборку на обучающую и тестовую:

In [ ]:

```
1 X, Y = dataset['data'], dataset['target']
2
3 X_train, X_test, Y_train, Y_test \
4         = train_test_split(X, Y, test_size=0.2, random_state=42)
5 X_train.shape, X_test.shape, Y_train.shape, Y_test.shape
```

При использовании регуляризации данные необходимо нормализовать. Воспользуемся для этого классом `StandardScaler` из библиотеки `sklearn`.

In [ ]:

```
1 scaler = StandardScaler()
```

2. Теперь обучите три модели логистические регрессии без регуляризации с помощью методов

- обычный градиентный спуск;
- стохастический mini-batch градиентный спуск;
- IRLS

Постройте график, на котором нанесите три кривые обучения, каждая из которых отображает зависимость оптимизируемого функционала от номера итерации метода. Функционал должен быть одинаковый для всех моделей, то есть без минусов. Нариуйте также график зависимости этого функционала от времени работы метода.

Для чистоты эксперимента желательно не запускать в момент обучения другие задачи и провести обучение несколько раз, усреднив результаты.

Напоминание: все графики должны быть информативны, с подписанными осями и т.д.

Сделайте выводы. Что будет, если при обучении на очень большой по количеству элементов датасете?

In [ ]:

```
1
```

3. Сравните два реализованных критерия остановки по количеству проведенных итераций: евклидова норма разности текущего и нового векторов весов стала меньше, чем 1e-4 и ограничение на число

итераций (например, 10000). Используйте градиентный спуск.

In [ ]:

```
1
```

4. Рассмотрите как влияет размер шага (`learning rate`) на качество модели. Обучите каждую модель одинаковое число итераций (например, 10000), а затем посчитайте качество. Воспользуйтесь ограничением на число итераций в качестве критерия остановки, так как для больших `learning rate` у вас может не сойтись модель. Используйте стохастический градиентный спуск. Сделайте выводы.

In [ ]:

```
1 lrs = [0.01, 0.1, 0.2, 0.3, 0.5, 0.7, 1, 2, 5, 10]
2
```

Постройте кривые обучения для различных `learning rate` . Не обязательно рассматривать все `learning rate` из предыдущего задания, так как их слишком много, и график будет нагроможден. Возьмите около половины из них. Какой `learning rate` лучше выбрать? Чем плохи маленькие и большие `learning rate` ?

In [ ]:

```
1
```

5. Рассмотрите несколько моделей, в которых установите не менее 5-ти различных коэффициентов регуляризации, а также модель без регуляризатора. Сравните, влияет ли наличие регуляризации на скорость сходимости и качество, сделайте выводы. Под качеством подразумевается значение метрики, рассмотренных на семинаре.

In [ ]:

```
1
```

6. Выберите произвольные два признака, в пространстве которых визуализируйте предсказания вероятностей класса 1 для модели, которая показала наилучшее качество на предыдущем шаге.

In [ ]:

```
1
```