Московский государственный университет
имени М. В. Ломоносова
Факультет вычислительной математики и кибернетики

Реферат по теме

«Байесовские методы классификации»

Выполнил:
студент 316 группы
Плеханов А. Д.

Москва
2021

Содержание

Введение

Постановка задачи классификации

Пусть X — множество описаний объектов, Y — конечное множество номеров (имён, меток) классов. Существует неизвестная целевая зависимость — отображение $y^* : X \to Y$, значения которой известны только на объектах конечной обучающей выборки $X^l = \{(x_1, y_1), \ldots, (x_l, y_l)\}$.

Требуется построить алгоритм $a : X \to Y$, способный классифицировать произвольный объект $x \in X$.

Вероятностная постановка задачи классификации

Пусть $X \subset \mathbb{R}^n$, $Y \subset \mathbb{Z}$.

При вероятностной трактовке задачи классификации предполагается, что на множестве $X \times Y$ задано некоторое вероятностное пространство, а пара (x, y) порождается распределением с плотностью вероятности $p(x, y)$.

По формуле Байеса можем представить $p(x, y)$ в следующем виде:

$$p(x, y) = P(y)p(x|y) = p(x)P(y|x), \tag{1}$$

где $P(y)$ - априорная вероятность класса y, $p(x|y)$ - функция правдоподобия класса y, $p(x)$ - плотность распределения объектов всех классов, $P(y|x)$ - условная вероятность того, что объект x принадлежит классу y.

Тогда процесс генерации нового объекта x можно представить в виде двух этапов:

1. Генерируется метка класса y из распределения $P(y)$;
2. Из распределения $p(x|y)$ генерируется сам объект x

Препдположим, что для всех классов $y \in Y$ известны плотности распределния $p(x|y)$

Байесовский классификатор и его оптимальность

Рассмотрим произвольный алгоритм $a : X \to Y$. Он разбивает множество X на непересекающиеся области $A_y = \{x \in X | a(x) = y\}, y \in Y$. Вероятность того, что появится объект класса y и алгоритм a отнесёт его к классу s, равна $P(y)P(A_s|y)$. Каждой паре $(y, s) \in Y \times Y$ поставим в соответствие величину потери λ_{ys} при отнесении объекта класса y к классу s. Обычно полагают $\lambda_{yy} = 0$, и $\lambda_{ys} > 0$ при $y \neq s$. Соотношения потерь на разных классах, как правило, известны заранее.

Функционал среднего риска:

$$R(a) = \sum_{y \in Y} \sum_{s \in Y} \lambda_{ys} P(y) P(A_s|y), \tag{2}$$

где $P(A_s|y) = \int_{A_s} p(x|y)\, dx$ - вероятность того, что алгоритм отнесет объект класса y к классу s.

В таком случае будет верна следующая теорема:

Теорема
Если известны априорные вероятности $P(y)$ и плотности распределения каждлого класса $p(x|y)$, то минимум среднего риска $R(a)$ достигается алгоритмом

$$a(x) = \arg\min_{s \in Y} \sum_{y \in Y} \lambda_{ys} P(y) P(x|y) \tag{3}$$

В случае, если $\lambda_{yy} = 0$ и $\lambda_{ys} \equiv \lambda_y$, т.е. потери зависят лишь от класса, на котором совершили ошибку, данный алгоритм примет вид

$$a(x) = \arg\max_{s \in Y} \lambda_y P(y) p(x|y) \tag{4}$$

Таким образом, байесовские классификаторы, в основе которых лежит формула (4), теоретически оптимальны в смысле минимизации функционала среднего риска, что является несомненным плюсом такого подхода.

Bayesian Model Averaging Naive Bayes

В предыдущей главе было показано, что при известных априорном распределении классов $P(y)$ и апостериорном распределении объектов $p(x|y)$ возможно в явном виде выписать оптимальный в смысле функционала среднего риска алгоритм классификации (4). На практике эти распределения не известны, но их можно оценить по выборке $X^l = \{(x_i, y_i)\}_{i=1}^l$. В таком случае теоретическая оптимальность классификатора теряется.

В зависимости от способа восстановления плотности распределения $p(x|y)$ и предположений, накладываемых на свойства этого распределения, существуют различные байесовские классификаторы, некоторые из которых будут рассмотрены в данном реферате.

При параметрическом восстановлении плотности распределения $p(x|y)$ возникают такие классификаторы, как, например, Naive Bayes (речь о котором пойдет ниже), или квадратичный дискриминант Фишера, в котором предоплагается гауссовость распределения каждого класса. Параметры распределений оцениваются с помощью метода максимального правдоподобия.

Наивный байесовский классификатор

Наивный байесовский классификатор исходит из "наивного"предположения о том, что признаки $f_j : X \to D_j$ - независимые случайные величины с плотностью распределения $p_j(z|y)$, $y \in Y$, $j = 1, ..., n$. Тогда функции правдоподобия классов $y \in Y$ для объекта $x = (x^{(1)}, ..., x^{(n)})$ представимы в виде

$$p(x|y) = p(x^{(1)}|y) \cdot ... \cdot p(x^{(n)}|y)$$

В таком случае алгоритм классификации (4) можно переписать в виде

$$a(x) = \arg\max_{y \in Y} \left[\lambda_y P(y) \prod_{j=1}^{n} p(x^{(j)}|y) \right], \tag{5}$$

С учетом монотонности натурального логарифма, получим:

$$a(x) = \arg\max_{y \in Y} \left[\ln(\lambda_y P(y)) + \sum_{i=1}^{n} \ln p_j(x|y) \right], \tag{6}$$

Обучение метода классификации сводится к оценке n одномерных плотностей $p_j(z|y)$ по выборке $x^{(j)} = (x_1^{(j)}, ..., x_l^{(j)})$.

Несмотря на наивность предположения о независимости признаков, наивный байесовский классификатор часто используется в задачах классификации текстовых документов.

Метод BMA-NB

В качестве основного метода, рассматриваемого в данном реферате, был выбран BMA-NB классификатор - комбинация обычного наивного байесовского классификатора и байесовского усреднения моделей - подхода, призванного учесть неопределенность при выборе модели, наилучшим образом описывающей данные.

Байесовское усреднение моделей

На практике очень часто бывает, что несколько моделей обеспечивают адекватное описание распределения, генерирующие наблюдаемые данные. В таких случаях обычно выбирают одну модель в соответствии с некоторым критерием, которая впоследствие считается истинной. Проблема в том, что такой подход может приводить к слишком самоуверенным выводам, а значит, и к принятию более рискованных решений. Байесовское усреднение моделей (Bayesian Model Averaging - BMA) представляет собой расширение байесовского вывода через задание априорного распределения для рассматриваемых моделей.

Далее будут использоваться следующие обозначения:

- $\mathbf{x}$ - признаковое описание объекта, для которого необходимо построить прогноз
- $D = \{(x_i, y_i)\}_{i=1}^{N}$ - наблюдаемая выборка
- $m \in \{1, ..., M\}$ - модель из заранее определенного набора

Совместное распределение класса y и признакового описания $\mathbf{x}$ при условии наблюдаемой выборки и модели m:

$$P(y, \mathbf{x}|m, D) = P(\mathbf{x}|m, y, D)P(y|D)$$

Совместное распределение класса y и признакового описания $\mathbf{x}$ при условии наблюдаемой выборки:

$$P(y, \mathbf{x}|D) = \sum_m P(\mathbf{x}|m, y, D)P(y|D)P(m|D) \tag{7}$$

По теореме Байеса для правдоподобия данных D для модели m верно:

$$P(m|D) \propto P(D|m)P(m) = P(m)\prod_{i=1}^{N} P(y_i, \mathbf{x_i}|m) = P(m)\prod_{i=1}^{N} P(y_i)P(\mathbf{x_i}|m, y_i) \tag{8}$$

Комбинируя формулы (7) и (8), получим:

$$P(y, \mathbf{x}|D) \propto \sum_m P(m)\prod_{i=1}^{N+1} P(y_i)P(\mathbf{x_i}|m, y_i) \tag{9}$$

В данной формуле произведение из $N+1$ множителей, т.к. оно учитывает помимо наблюдаемой выборки сам объект $\mathbf{x}$, для которого строится прогноз.

BMA в приложении к NB

Рассмотрим задачу классификации для набора данных с количеством признаков K, и предположим независимость признаков. Несложно подсчитать, что количество потенциальных наивных байесовских классификаторов равно 2^K. т.е. сложность перебора всех моделей экспоненциальна по размерности признакового описания. Далее будет показано, что BMA позволит учитывать "голоса"всего экспоненциального числа моделей в одном алгоритме, сложность обучения которого будет линейна относительно K.

Будем описывать каждую модель m бинарным вектором $\mathbf{f} = (f_1, ..., f_K)$. Если $f_j = 1$, то модель учитывает j-ый признак при построении прогноза, иначе - не учитывает.

Предполагая $P(m) = \prod_{k=1}^{K} P(f_k)$, тогда условная вероятность класса y для объекта $\mathbf{x}$ и наблюдаемой выборки D имеет вид:

$$\begin{aligned} P(y|\mathbf{x}, D) &= \sum_{\mathbf{f}} \left[\prod_{k=1}^{K} P(f_k)\right] \prod_{i=1}^{N+1} P(y_i) \prod_{k=1}^{K} P(\mathbf{x}_{i,k}|f_k, y_i) = \\ &= \sum_{f_1} ... \sum_{f_K} \left[\prod_{i=1}^{N+1} P(y_i)\right] \prod_{k=1}^{K} P(f_k) \prod_{i=1}^{N+1} P(\mathbf{x}_{i,k}|f_k, y_i) = \\ &= \left[\prod_{i=1}^{N+1} P(y_i)\right] \prod_{k=1}^{K} \sum_{f_k} P(f_k) \prod_{i=1}^{N+1} P(\mathbf{x}_{i,k}|f_k, y_i) \end{aligned} \tag{10}$$

Определим априорную вероятность вероятность $P(f_k)$:

$$P(f_k) \propto \begin{cases} \frac{1}{\beta}, & \text{if } \mathrm{f}_k = 1 \\ 1, & \text{if } \mathrm{f}_k = 0 \end{cases},$$

где β - некоторый положительный гиперпараметр. Можно положить его больше 1, тогда "вес"модели, учитывающей меньшее количество признаков, будет

выше, что соответствует принципу бритвы Оккама.

Вероятность $P(\mathbf{x}_{i,k}|f_k, y_i)$ логично определить так:

$$P(\mathbf{x}_{i,k}|f_k, y_i) = \begin{cases} P(\mathbf{x}_{i,k}|y_i), & \text{if } \mathrm{f}_k = 1 \\ P(\mathbf{x}_{i,k}), & \text{if } \mathrm{f}_k = 0 \end{cases}$$

Таким образом, финальная формула для условной вероятности класса имеет вид:

$$P(y|\mathbf{x}, D) = \left[\prod_{i=1}^{N+1} P(y_i)\right] \prod_{k=1}^{K} \left[\prod_{i=1}^{N+1} P(\mathbf{x}_{i,k}) + \frac{1}{\beta} \prod_{i=1}^{N+1} P(\mathbf{x}_{i,k}|y_i)\right]$$

Сам алгоритм имеет вид:

$$a(x) = \arg\max_{y \in Y} P(y) \cdot \prod_{k=1}^{K} \left(A_k \cdot P(x_k) + \frac{B_k}{\beta} \cdot P(x_k|y)\right), \tag{11}$$

где значения констант $A_k = \prod_{i=1}^{N} P(\mathbf{x}_{i,k})$, $B_k = \prod_{i=1}^{N} P(\mathbf{x}_{i,k}|y_i)$ вычисляются непосредственно из обучающей выборки, а параметры распределений признаков оцениваются методом максимального правдоподобия.

Таким образом, обучение алгоритма заключается в оценке параметров $K \cdot |Y|$ одномерных распределений $P(x_k|y)$ ($|Y|$ - количество классов). Для построения прогноза необходимо вычислить вероятности $P(x_k|y)$ (напрямую) и $P(x_k)$ (вычисляется с помощью формулы полной вероятности).

Реализация и тестирование метода

Метод классификации BMA-NB реализовывался на языке Python 3 с использованием библиотеки numpy. Листинг кода приведен в приложении. В качестве базовой модели был выбран гауссовский вариант наивного байесовского классификатора (предполагается нормальное распрделение каждого из признаков), что ограничивает область применения получившегося метода (в общем случае BMA-NB не делает предположений о виде признаков, можно реализовать и для категориальных).

Также стоит отметить, что данная реализация метода не позволяет применять его на выборках большого размера, поскольку возникает проблема зануления коэффициентов A_k, B_k. Как было показано выше, данные коэффициенты по сути являются произведением N множителей (N - размер выборки) с значенями из отрезка $[0, 1]$. Значения этих коэффициентов хранятся в типе $np.float64$, но при слишком большом размере выборки данного типа не хватает для хранения очень маленьких чисел с плавающей точкой, что приводит к занулению коэффициентов. Данная проблема также не всегда решается нормировкой, поскольку отношение между коэффициентами A_k настолько велико, что неизбежно один из них становится равным 0 или $+inf$.

Тестирование метода проводилось как на синтетических данных, так и на реальных - на встроенном в sklearn наборе данных "Ирисы Фишера".

Датасет Fisher Iris

Набор данных ирисы Фишера включает в себя информацию о 3 видах ирисов - setosa, versicolor и virginica. У каждого объекта 4 числовых признака. Распределение классов в признаковом пространстве представлено на рис. 1. На рис. 2 представлены точности классификаторов BMA-NB, GaussianNB и SVC (с rbf-ядром) по кросс-валидации на 5 фолдах.

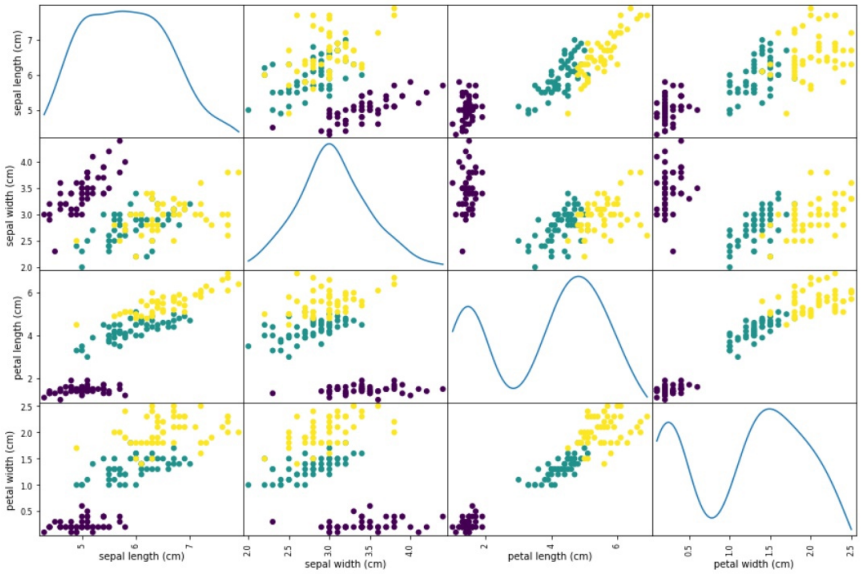

Рис. 1: Ирисы Фишера

```
GaussianNB. Тосность по кросс-валидации на 5 фолдах: 0.9533333333333334
BMA-NB. Тосность по кросс-валидации на 5 фолдах: 0.9533333333333334
SVC. Тосность по кросс-валидации на 5 фолдах: 0.9666666666666666
```

Рис. 2: Точности BMA-NB, GaussianNB, SVM на ирисах Фишера

Как видно из рис. 2, все алгоритмы показали точность выше 0.95, что подтверждает работоспособность реализованного метода.

Синтетические данные

Приведем несколько примеров работы метода на синтетических данных, сгенерированных с помощью функции $make\_classification$ из модуля $sklearn.datasets$.

Сравнение качества работы алгоритма проводилось с методом $GaussianNB$ из модуля $sklearn.naive_bayes$ (Гауссовский наивный байесовский классификатор) и методом SVC из модуля $sklearn.svm$ (SVM-классификатор с RBF-ядром).

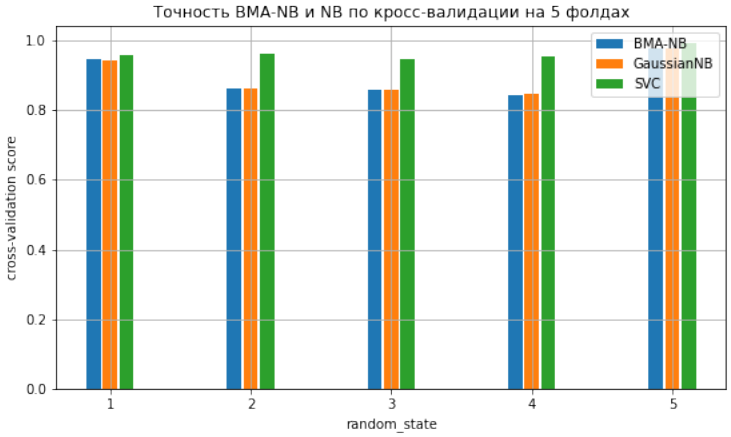

Рис. 3: Результаты тестирования

```
X, y = make_classification(n_samples=1500, n_features=5,
                           n_informative=5, n_redundant=0, n_repeated=0,
                           n_clusters_per_class=1, n_classes=2,
                           random_state=state)
```

Рис. 4: Генерация выборки

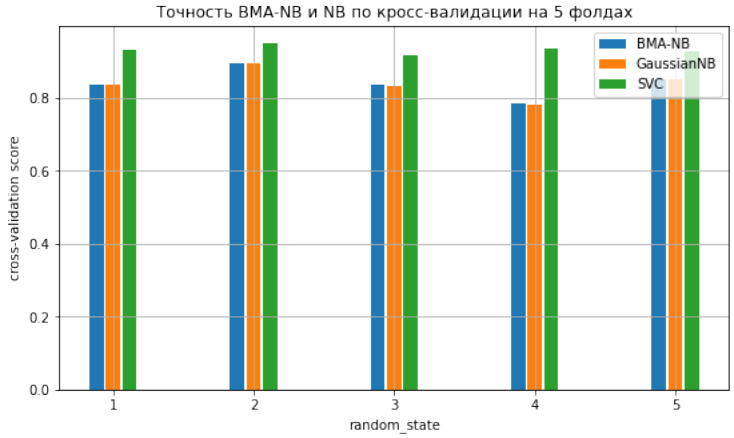

Рис. 5: Результаты тестирования

```
X, y = make_classification(n_samples=3000, n_features=20,
                           n_informative=5, n_redundant=5, n_repeated=5,
                           n_clusters_per_class=1, n_classes=3,
                           random_state=state)
```

Рис. 6: Генерация выборки

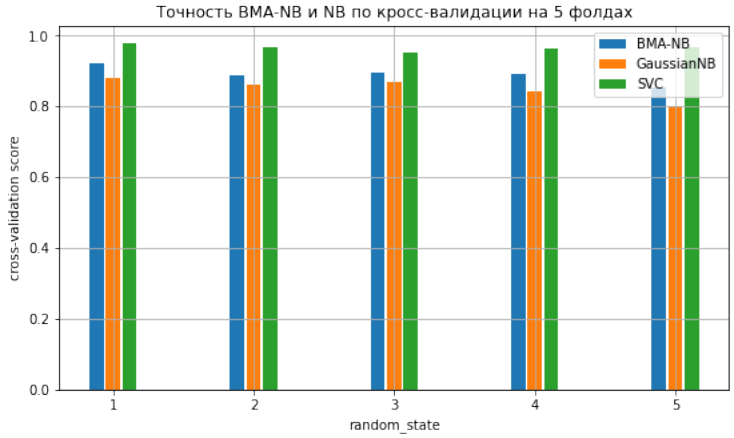

Рис. 7: Результаты тестирования

```
X, y = make_classification(n_samples=1500, n_features=500,
                           n_informative=30, n_redundant=3,
                           n_clusters_per_class=1, n_classes=2,
                           random_state=state)
```

Рис. 8: Генерация выборки

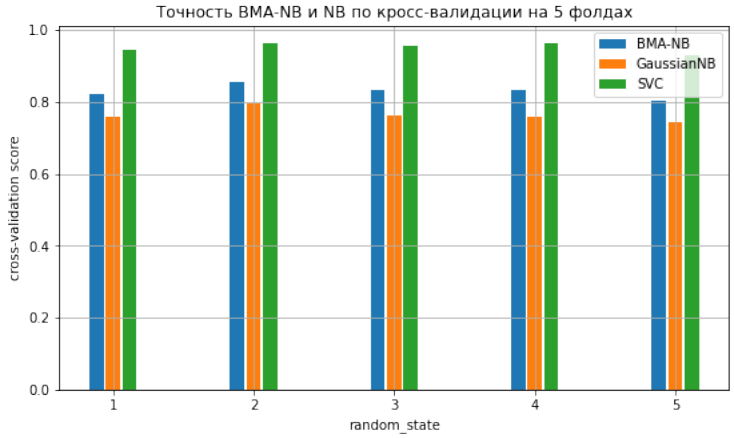

Рис. 9: Результаты тестирования

```
X, y = make_classification(n_samples=2000, n_features=500,
                           n_informative=30, n_redundant=3,
                           n_clusters_per_class=1, n_classes=3,
                           random_state=state)
```

Рис. 10: Генерация выборки

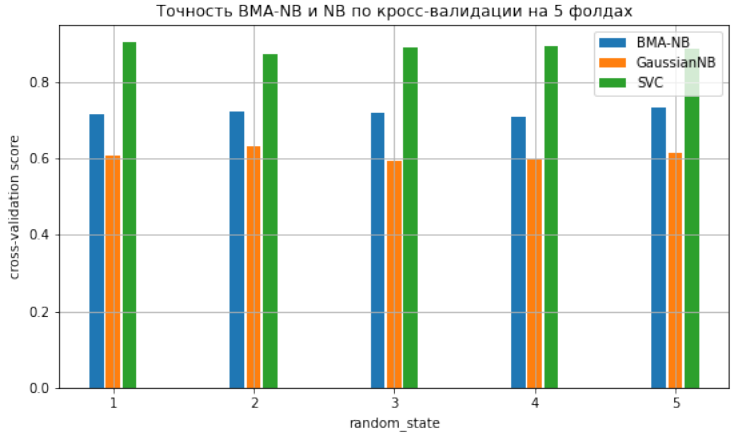

Рис. 11: Результаты тестирования

```
X, y = make_classification(n_samples=2000, n_features=500,
                           n_informative=30, n_redundant=3,
                           n_clusters_per_class=1, n_classes=5,
                           random_state=state)
```

Рис. 12: Генерация выборки

В первых двух тестах методы BMA-NB и GaussianNB стабильно показывают одинаковую точность (она варьирует в зависимости от *random_state*, но в среднем около 0.8). В то же время метод SVM в большинстве случаев заметно

превосходит оба статистических алгоритма. Это объясняется тем фактом, что метод SVM с RBF-ядром вообще является довольно сильным методом, он не делает наивных предположений о данных.

В третьем, четвертом и пятом тестах, при уменьшении доли информативных признаков, алгоритм BMA-NB в среднем превосходит GaussianNB на 0.05, но его точность всё еще меньше SVC. Также можно заметить, что при увеличении количества классов отличие между точностями BMA-NB и GaussianNB увеличиается примерно в два раза.

Список литературы

1. К. В. Воронцов - "Байесовская классификация. Непараметрические методы."

2. Tiago M. Fragoso, Francisco Louzada Neto - "Bayesian model averaging: A systematic review and conceptual classification"

3. Ga Wu, Scott Sanner, and Rodrigo F.S.C. Oliveira - "Bayesian Model Averaging Naive Bayes (BMA-NB): Averaging over an Exponential Number of Feature Models in Linear Time"

4. Документация библиотеки scikit-learn

In [ ]:

```
import numpy as np
import sklearn as sk
import matplotlib.pyplot as plt

%matplotlib inline
```

In [ ]:

```
from sklearn.model_selection import cross_val_score
from sklearn.metrics import accuracy_score
from sklearn.naive_bayes import GaussianNB
from sklearn.svm import SVC
from sklearn.datasets import make_blobs, make_classification
```

In [ ]:

```
def gaussian_kernel(x, mu=0, sigma=1):
    return 1 / (((2 * np.pi) ** 0.5) * sigma) * np.exp(-(x - mu) ** 2 / (2 * sigma.astype(np.float64) ** 2))

class BMANBClassifier:
    """
    BMA-NB классификатор. Предполагается, что распределение всех признаков нормальное
    """
    def __init__(self, beta=1, coef=2):
        self.beta = beta

        # константа нормализации
        self.coef = coef

        # количество классов
        self.m = None

        # массив меток классов размерности (m, ) (m - число классов)
        self.labels = None

        # массив априорных вероятностей классов размерности (m, )
        self.cls_probs = None

        # размерность признакового пространства
        self.k = None

        # массив средних размерности (m, k) (k - число признаков)
        self.mus = None

        # массив стандартных отклонений размерности (m, k)
        self.sigmas = None

        # массив величин A_k размерности (1, k)
        self.a = None

        # массив величин B_k размерности (1, k)
        self.b = None

    def fit(self, X, y, coef=None):
        """
        X - признаковое описание объектов: np.array размерности (n, k)
        (n - количество объектов,  k - размерность признакового пространства)
        y - метки классов: np.array размерности (n, )

        Вычисляем self.cls_probs, self.mus, self.sigmas, self.a, self.b
        returns: self
        """
        if coef != None:
            self.coef = coef
        # размер обучающей выборки
        n = y.shape[0]

        # размерность признакового пространства
        self.k = X.shape[1]

        # массив меток классов
        self.labels = np.unique(y)

        # количество классов
        self.m = self.labels.shape[0]

        # вычисление априорных вероятностей классов, параметров распределения для каждого класса и каждого приз
        self.cls_probs = np.zeros(self.m)
        self.mus = np.zeros((self.m, self.k))
        self.sigmas = np.zeros((self.m, self.k))
        for i, label in enumerate(self.labels):
            self.cls_probs[i] = (y == label).sum() / n

            # признаковое описание объектов с меткой label
```

```
            X_cls = X[y == label]

            # вычисление мат.ожиданий и стандартных отклонений сразу по всем признакам
            self.mus[i] = X_cls.mean(axis=0)
            self.sigmas[i] = (((X_cls - self.mus[i]) ** 2).sum(axis=0) / X_cls.shape[0]) ** 0.5

        # занумеруем классы для удобства дальнйших вычислений
        y_nums = np.zeros(n).astype(int)
        for i, label in enumerate(self.labels):
            y_nums[np.where(y == label)] = i

        # вычисление коэффициентов A_k и B_k
        self.a = np.zeros(self.k).astype(np.float64)
        self.b = np.zeros(self.k).astype(np.float64)
        # рассматриваем j-й признак
        for j in range(self.k):
            arr = np.tile(X[:, j].reshape((-1,)), (self.m, 1))
            # вычисляем P(x_{ij} | y_j) для всех y_q, q = 1,...,m
            for q, label in enumerate(self.labels):
                arr[q] = gaussian_kernel(arr[q], self.mus[q, j], self.sigmas[q, j])

            vec_a = (self.coef * np.transpose(arr) * self.cls_probs).sum(axis=1)
            vec_b = (self.coef * gaussian_kernel(X[:, j], self.mus[:, j][y_nums], self.sigmas[:, j][y_nums]))

            norm_coef = 10 ** np.median(np.log10(np.hstack([vec_a, vec_b]).flatten()))
            vec_a /= norm_coef
            vec_b /= norm_coef
            self.a[j] = vec_a.prod()
            self.b[j] = vec_b.prod()

        return self


    def predict(self, X=None):
        """
        X - признаковое описание объектов: np.array размерности (n, k)
        returns: np.array размерности (n,) - метки классов для соответствующих объектов из X
        """
        n = X.shape[0]

        # Вычисление P(x_{id} | y_j) и P(x_{id}) для всех i = 1, ..., n, d = 1, ..., k, j = 1, ..., m
        prob_arr = np.zeros((n, self.m + 1, self.k))

        for i in range(n):
            for j in range(self.m):
                prob_arr[i][j] = gaussian_kernel(X[i], self.mus[j], self.sigmas[j])
            prob_arr[i][self.m] = (prob_arr[i][:-1] * self.cls_probs[:, None]).sum()

        # строим прогноз...
        result = np.zeros(n)
        for i in range(n):
            l = np.log10(self.a * np.tile(prob_arr[i][self.m], (self.m, 1)) + self.b / self.beta * prob_arr[i][
            result[i] = self.labels[np.argmax(l.sum(axis=1) + np.log10(self.cls_probs))]


        return result
```

In [ ]:

```
def kfold_split(num_objects, num_folds):
    indexes = np.arange(num_objects)
    delims = np.arange(0, num_objects // num_folds * num_folds, num_objects // num_folds)[1:]
    folds = np.split(indexes, delims)

    l = []
    for i in range(num_folds):
        l.append((np.delete(indexes, folds[i]), folds[i]))

    return l

def cross_val(model, X, y, cv=5, score_function=accuracy_score, seed=0):
    folds = kfold_split(X.shape[0], cv)
    np.random.seed(seed)
    indexes = np.random.permutation(X.shape[0])
    X, y = X[indexes], y[indexes]
    accuracy_list = []
    for train_fold, test_fold in folds:
        model.fit(X[train_fold], y[train_fold])
        y_pred = model.predict(X[test_fold])
        accuracy_list.append(score_function(y[test_fold], y_pred))

    return np.array(accuracy_list)
```

Test 0

In [ ]:

```
from sklearn.datasets import load_iris

data = load_iris()
X = data["data"]
y = data["target"]

print("GaussianNB. Тосность по кросс-валидации на 5 фолдах:", cross_val_score(GaussianNB(), X, y, cv=5).mean())
print("BMA-NB. Тосность по кросс-валидации на 5 фолдах:", cross_val(BMANBClassifier(coef=1, beta=1), X, y, cv=5
print("SVC. Тосность по кросс-валидации на 5 фолдах:", cross_val_score(SVC(), X, y, cv=5).mean())
```

Test 1

In [ ]:

```
bmanb_list = []
nb_list = []
svc_list = []

for state in range(5):

    X, y = make_classification(n_samples=1500, n_features=5,
                               n_informative=5, n_redundant=0, n_repeated=0,
                               n_clusters_per_class=1, n_classes=2,
                               random_state=state)

    bmanb_list.append(cross_val(BMANBClassifier(), X, y, cv=5, seed=2).mean())
    nb_list.append(cross_val_score(GaussianNB(), X, y, scoring="accuracy", cv=5).mean())
    svc_list.append(cross_val_score(SVC(), X, y, scoring="accuracy", cv=5).mean())
    print(state)
```

In [ ]:

```
# задаем размеры
plt.figure(figsize=(9,5))

# заголовок
plt.title('Точность BMA-NB и NB по кросс-валидации на 5 фолдах')

# ширина столбцов
width = 0.35

# координаты столбцов
ids = np.arange(1, 6)

# рисуем графики
plt.bar(ids - width / 3, bmanb_list, width / 3.5, label="BMA-NB")
plt.bar(ids, nb_list, width / 3.5, label="GaussianNB")
plt.bar(ids + width / 3, svc_list, width / 3.5, label='SVC')

# подписи осей
plt.xlabel('random_state')
plt.ylabel('cross-validation score')

plt.legend()

# сетка графика
plt.grid(True)

plt.show()
```

Test 2

In [ ]:

```
bmanb_list = []
nb_list = []
svc_list = []

for state in range(5):

    X, y = make_classification(n_samples=3000, n_features=20,
                               n_informative=5, n_redundant=5, n_repeated=5,
                               n_clusters_per_class=1, n_classes=3,
                               random_state=state)

    bmanb_list.append(cross_val(BMANBClassifier(), X, y, cv=5, seed=2).mean())
    nb_list.append(cross_val_score(GaussianNB(), X, y, scoring="accuracy", cv=5).mean())
    svc_list.append(cross_val_score(SVC(), X, y, scoring="accuracy", cv=5).mean())
    print(state)
```

In [ ]:

```
# задаем размеры
plt.figure(figsize=(9,5))

# заголовок
```

```
plt.title('Точность BMA-NB и NB по кросс-валидации на 5 фолдах')

# ширина столбцов
width = 0.35

# координаты столбцов
ids = np.arange(1, 6)

# рисуем графики
plt.bar(ids - width / 3, bmanb_list, width / 3.5, label="BMA-NB")
plt.bar(ids, nb_list, width / 3.5, label="GaussianNB")
plt.bar(ids + width / 3, svc_list, width / 3.5, label='SVC')

# подписи осей
plt.xlabel('random_state')
plt.ylabel('cross-validation score')

plt.legend()

# сетка графика
plt.grid(True)

plt.show()
```

Test 3

In [ ]:

```
bmanb_list = []
nb_list = []
svc_list = []

for state in range(5):

    X, y = make_classification(n_samples=1500, n_features=500,
                               n_informative=30, n_redundant=3,
                               n_clusters_per_class=1, n_classes=2,
                               random_state=state)

    bmanb_list.append(cross_val(BMANBClassifier(), X, y, cv=5, seed=2).mean())
    nb_list.append(cross_val_score(GaussianNB(), X, y, scoring="accuracy", cv=5).mean())
    svc_list.append(cross_val_score(SVC(), X, y, scoring="accuracy", cv=5).mean())
    print(state)
```

In [ ]:

```
# задаем размеры
plt.figure(figsize=(9,5))

# заголовок
plt.title('Точность BMA-NB и NB по кросс-валидации на 5 фолдах')

# ширина столбцов
width = 0.35

# координаты столбцов
ids = np.arange(1, 6)

# рисуем графики
plt.bar(ids - width / 3, bmanb_list, width / 3.5, label="BMA-NB")
plt.bar(ids, nb_list, width / 3.5, label="GaussianNB")
plt.bar(ids + width / 3, svc_list, width / 3.5, label='SVC')

# подписи осей
plt.xlabel('random_state')
plt.ylabel('cross-validation score')

plt.legend()

# сетка графика
plt.grid(True)

plt.show()
```

Test 4

In [ ]:

```
bmanb_list = []
nb_list = []
svc_list = []
```

```
for state in range(5):
    X, y = make_classification(n_samples=2000, n_features=500,
                               n_informative=30, n_redundant=3,
                               n_clusters_per_class=1, n_classes=3,
                               random_state=state)

    bmanb_list.append(cross_val(BMANBClassifier(), X, y, cv=5, seed=2).mean())
    nb_list.append(cross_val_score(GaussianNB(), X, y, scoring="accuracy", cv=5).mean())
    svc_list.append(cross_val_score(SVC(), X, y, scoring="accuracy", cv=5).mean())
    print(state)
```

In [ ]:

```
# задаем размеры
plt.figure(figsize=(9,5))

# заголовок
plt.title('Точность BMA-NB и NB по кросс-валидации на 5 фолдах')

# ширина столбцов
width = 0.35

# координаты столбцов
ids = np.arange(1, 6)

# рисуем графики
plt.bar(ids - width / 3, bmanb_list, width / 3.5, label="BMA-NB")
plt.bar(ids, nb_list, width / 3.5, label="GaussianNB")
plt.bar(ids + width / 3, svc_list, width / 3.5, label='SVC')

# подписи осей
plt.xlabel('random_state')
plt.ylabel('cross-validation score')

plt.legend()

# сетка графика
plt.grid(True)

plt.show()
```

Test 5

In [ ]:

```
bmanb_list = []
nb_list = []
svc_list = []

for state in range(5):

    X, y = make_classification(n_samples=2000, n_features=500,
                               n_informative=30, n_redundant=3,
                               n_clusters_per_class=1, n_classes=5,
                               random_state=state)

    bmanb_list.append(cross_val(BMANBClassifier(), X, y, cv=5, seed=2).mean())
    nb_list.append(cross_val_score(GaussianNB(), X, y, scoring="accuracy", cv=5).mean())
    svc_list.append(cross_val_score(SVC(), X, y, scoring="accuracy", cv=5).mean())
    print(state)
```

In [ ]:

```
# задаем размеры
plt.figure(figsize=(9,5))

# заголовок
plt.title('Точность BMA-NB и NB по кросс-валидации на 5 фолдах')

# ширина столбцов
width = 0.35

# координаты столбцов
ids = np.arange(1, 6)

# рисуем графики
plt.bar(ids - width / 3, bmanb_list, width / 3.5, label="BMA-NB")
plt.bar(ids, nb_list, width / 3.5, label="GaussianNB")
plt.bar(ids + width / 3, svc_list, width / 3.5, label='SVC')

# подписи осей
plt.xlabel('random_state')
plt.ylabel('cross-validation score')

plt.legend()

# сетка графика
plt.grid(True)

plt.show()
```