**Практическая работа №14**

**Линейный и квадратичный дискриминантный анализ**

# Применение дискриминантного анализа для прогноза продуктивности локальных структур

***Цель работы.*** Освоение методов построения дискриминантной функции для классификации геологических объектов, оценки информативности геологических показателей; использование методических приемов для классификации и прогноза геологических характеристик.

***Формируемые компетенции или их части***

ОК-1 Способность к абстрактному мышлению, анализу, синтезу.

ПК-13 Наличие высокой теоретической и математической подготовки, а также подготовки по теоретическим, методическим и алгоритмическим основам создания новейших технологических процессов геологической разведки, позволяющим быстро реализовывать научные достижения, использовать современный аппарат математического моделирования при решении прикладных научных задач.

ПК-15 Способность обрабатывать полученные результаты, анализировать и осмысливать их с учетом имеющегося мирового опыта, представлять результаты работы с обоснованием предложенных решений на высоком научно-техническом и профессиональном уровне.

**Теоретическая часть**

Деление совокупности многомерных объектов на классы - в общем случае довольно сложная задача, которая во многих задачах обычно решается на уровне качественного моделирования. В настоящее время при построении объективных классификаций получили широкое распространение статистические методы. Одним из таких методов классификации является дискриминантный анализ. Математическая модель дискриминантного анализа основана на процедуре подбора так называемой дискриминантной функции (в нашем случае линейной), которая производит оптимальное разделение объектов на классы. Дискриминантная функция подбирается на некотором эталонном массиве исходных объектов, априорно отнесенных к разным классам. Вычисленные на эталонных данных параметры дискриминантной функции используются для прогноза классификации неизвестных объектов. Задача разделения (дискриминации) сводится к подбору параметров дискриминантной функции таким образом, чтобы разным классам соответствовали разные значения функции. На схеме показано схематическое разделение объектов на классы *А* и *В*(рис. 11.1).



? 2

Граничное значение дискриминантной функции *Do=aX+bY*

*Класс А Класс В*

*Рисунок 11.1. Графическая интерпретация дискриминантной функции*

Для априорно заданных объектов вычислена дискриминантная функция *D=aX+b,* позволяющая в любой точке *XQ, Yo* вычислить граничные значения *DQ.* Теперь оказывается возможным отнесение неизученных объектов к классам *А* и *В.* Например объект 1 при *D>DQ*относят к классу *А,* а объект 2, для которого *D<dQ -</d*к классу*В.*

Естественно, что на практике чаще встречаются задачи разделения многомерных объектов, охарактеризованных не двумя, а тремя или более признаками.

В наиболее общем случае геометрическая интерпретация дискри-минативной функции представляет собой гиперплоскость в ^-мерном признаковом пространстве, а каждый объект есть точка этого пространства. Необходимо провести в этом пространстве такую гиперплоскость, которая обеспечивала бы максимальное различие между множествами объектов, принадлежащих к разным классам, и сводила бы к минимуму рассеяние внутри каждого множества. Такую гиперплоскость называют***дискриминантной.***

Аналитически гиперплоскость в ^-мерном пространстве имеет вид:

*D=axXx+a^2+... +акХк.* (11.1)

Задача, следовательно, заключается в отыскании коэффициентов ар*а2,......ак,* которые обеспечивали бы требуемые условия разделения.

Если дискриминантная функция найдена, то множество значений ее состоит из подмножеств, каждое из которых принадлежит одному из классов. Существует такое граничное значение, которое делит область значений дискриминантной функции на два подмножества. Отнесение неизученного объекта к тому или иному классу производится также, как и в рассмотренном выше простейшем случае двухпризнакового объекта. Значения показателей подставляются как аргументы в построенную на эталонной выборке дискриминантную функцию. Если значение дискриминантной функции изучаемого объекта больше граничного значения *Do,* то функции изучаемого объекта относят к классу *А,* в противном случае - к классу *В.*

Представим, что исходные геологические данные позволяют выделить в многомерной совокупности два класса *А и В,* каждый из *i*объектов которых охарактеризован *j* значениями признаков. Представим эти исходные данные в матричной форме:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| *А =* | * 41
* 41
 | * 4г
* 4г
 | * 4з
* 4з
 | * 4\*
* 4\*
 | (И-2) |
| 4^ |  | 4v,3 | 4^ |
|  | **41** | 4г | 4з | 4\* |  |
| *В =* | **41** | 4г | 4з | 4\* | (11.3) |
|  | *BN^X* | *?^N22* | 4г,з | *^N2k* |  |

где *Nx —* число объектов, входящих в класс *A; N2 —* число объектов, входящих в класс *В’, к-* число признаков, характеризующих каждый объект.

Следующий этап построения дискриминантной функции - составление матриц центрированных сумм квадратов и смешанных произведений:

***S.=***

Ш-4-р)2 2(4;-4-Р)(4;-4-Р)

**j=i /=1**

* 2(4; \_4-Р)(4; -4-р) 2(4> - 4-р)
* 7=1 **7=i**
* 2(4у **\_4ir2-p)(4/-4-р)**2(4v \_4г,-р)(4/ \_4-р)
* 2(4> 4-p)(4v2J 4v2-p) **7=1**
* 2(4;\_4-P)(4v2; \_4v2-P)
* (Н.4)
* \* **1**
* 2(Ау-Д-?)
* 7=1

**Ш-Д-р)(А>-А-р)**

j-i

***Х-А. N1J Nl~?J* м 1-Р/**

**7=1**

Ш-4-р)(Д7-Д-р) **7-1**

**7-1**

Y[bn,-bn*А(в,,-вг****"21 "2~v/ \*1* 2-Р 7**

* 7=1
* 2(^7 **Д-р)(Д»У**Д^-р) **7=1**
* 1L(b1.-b1 (bn .-bn J
* 2/ **2—P *J fif2J* Aj-P *J***

***y(BNi-BN*** J2

**Z^ *1\*21* JVj-P / 7-1**

(И.5)

Вычисляем выборочную матрицу:

* 1 Nx+N2-2
* (Sa+Sb).
* (П.6)

Обращение этой матрицы позволяет вычислить коэффициенты дискриминантной функции:

*С = М1.* (11.7)

Для обращения матрицы М найдем ее детерминант и убедимся, что:

А *- DetM*ф 0 .

Как известно, обратная матрица имеет вид:

М’1

|  |  |  |
| --- | --- | --- |
| мп | М21 |  |
| А | А |  |
| М12 | **^22** | *Мк2* |
| А | А | А |
|  | **^2^** |  |
| А | А | А |

(И.8)

где *М.. —* алгебраическое дополнение элементов матрицы М.

Коэффициенты дискриминантной функции вычисляют по формуле:

*к*

*а=УспАХ^с*„-ЛА

(П-9)

*Р PJ***7 ср***J***ср) 5**

**7=1**

где *ср.* - элементы обратной матрицы М'1, *Х.ср* - средние значения соответствующего признака.

Отсюда дискриминантная функция имеет вид:

»=?>л. (и.ю)

**/=1**

где *X* — текущее значение р-го признака.

Граничное значение дискриминантной функции, при котором происходит разделение на классы, вычисляем по формуле:

* (H-и)
* 2 **7=1**

При прогнозе, если фактические значения дискриминантной функции больше граничного значения Z)Q, т. е. *D > DQ,* этот объект относится к классу *А* и, наоборот, если *D < DQ,* то объект принадлежит к классу *В.*

Абсолютная величина отклонения *D* от граничного значения служит мерой надежности прогноза:

»=|Д-Д|, (11.12)

**Оборудование и материалы**

Работа выполняется в специализированном компьютерном классе.

Программное обеспечение - *Microsoft Excel.*

Исходные данные - совокупности данных эталонных и прогнозных объектов *(Приложение И).*

**Указания по технике безопасности**

К выполнению лабораторных работ допускаются студенты, ознакомившиеся с правилами работы в лаборатории, прошедшие инструктаж безопасности.

**Указания по порядку выполнения работы**

Рассмотрим методику использования дискриминантного анализа на примере.

Изучение и сопоставление большого числа образцов пород показывает, что в общем породы с низким содержанием нерастворимого остатка, высоким содержанием кальцита и высокой пористостью обладают повышенной пластичностью и образуют практически непроницаемые покрышки - экраны для залежей углеводородов. И наоборот, породы с низкой пористостью, повышенной доломитостью и высоким содержанием нерастворимого остатка склонны к трещино-образованию и часто могут служить коллекторами жидкостей и газов. Однако сложно провести точную границу между покрышками и коллекторами и уверенно отнести образцы пород к какому-либо из этих классов.

*Таблица 11.2*

|  |  |
| --- | --- |
| **Признак** | **Номера образцов** |
| **покрышки** | **коллекторы** | **прогнозные образцы** |
| **а** | **b** | **с** | **I** | **II** | **ш** | **1** | **2** | **3** | **4** | **5** | **6** |
| Л] - пористость, % | 26,12 | 8,04 | 9,99 | 1,42 | 2,80 | 1,73 | 6,50 | 1037 | 9,39 | 0,66 | 3,90 | 4,96 |
| *Х2 -* содержание нерастворимого остатка, % | 3,34 | 3,82 | 5,63 | 22,19 | 1433 | 14,19 | 5,48 | 2,57 | 7,63 | 5,56 | 21,97 | 14,61 |
| *Х} -* содержание карбонатов, % | 96,53 | 92,83 | 90,12 | 70,03 | 71,18 | 68,10 | 90,42 | 95,61 | 8731 | 90,02 | 72,87 | 64,58 |

В табл. 11.2 приведены данные по пористости, содержанию карбонатов и нерастворимого остатка в 12 образцах пород, для которых известно, что три образца отобраны из пород-коллекторов, три - из пород-покрышек, а остальные шесть к какому-либо из этих классов не отнесены.

Используя эти данные, построим линейную дискриминантную функцию, разделяющую покрышки и коллекторы, а также осуществим прогноз для остальных шести образцов.

Прежде всего построим матрицы класса А (покрышки):

|  |  |  |  |
| --- | --- | --- | --- |
|  | 26,12 | 3,34 | 96,53 |
| *А =* | 8,04 | 3,82 | 92,83 |
| и класса В коллекторы: | 9,99 | 5,63 | 90,12 |
|  | 1,42 | 22,19 | 70,03 |
| *В =* | 2,80 | 14,23 | 71,18 |
|  | 1,73 | 14,19 | 68,1 |

Расчет матрицы сумм для покрышек SA приведен в табл. 11.3.

*Таблица 11.3*

**Расчет матрицы SA**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Номер образца** | ***Ху*** | **Х,-Х lj lq>** | ***Ху*** | *ХА,* | *X3j* | **Х„-Х, *У Зср*** |
| 1 | 2 | 3 | 4 | 5 | 6 | 7 |
| 1 | 26,12 | 11,41 | 3,34 | -0,92 | 96,53 | 3,37 |
| 2 | 8,04 | -6,67 | 3,82 | -0,44 | 92,83 | -0,33 |
| 3 | 9,99 | -4,72 | 5,63 | 1,37 | 90,12 | -3,04 |
| Е | 44,15 |  | 12,79 |  | 279,48 |  |

*Продолжение таблицы 11.3*

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Номер образца** |  | *(brV* | *(XyV* |  | та\* |  |
| 1 | 8 | *9* | 10 | 11 | 12 | 13 |
| 1 | 130,19 | 0,85 | 11,36 | -10,50 | 38,45 | -3,10 |
| 2 | 44,49 | 0,19 | 0,11 | 2,93 | 2,20 | 0,15 |
| 3 | 22,28 | 1,88 | 9,24 | -46,47 | 14,35 | -4,16 |
| Е | 196,96 | 2,92 | 20,71 | -14,04 | 55,00 | -7Д1 |

В ней использованы значения средних:

^]ср=26,12+8,04+9,99=44,15:3=14,71; ^=4,26; ^=93,16.

Таким образом, матрица сумм центрированных квадратов и смешанных произведений для класса *А* имеет вид

|  |  |  |
| --- | --- | --- |
| 196,96 | -14,04 | 55,00 |
| -14,04 | 2,92 | -7,11 |
| 55,00 | -7,11 | 20,71 |

Расчет аналогичной матрицы для коллекторов приведен в таблице 11.4.

*Таблица 11.4*

**Расчет матрицы SB**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Номер образца** | ***ХУ*** | **Х.-Х 1J *1ср*** | *Х„****У*** | **X-X,*****У 2ср*** | ***ХУ*** | **X -X, 3/ *Зср*** |
| 1 | 2 | *3* | 4 | 5 | 6 | 7 |
| 1 | 1,42 | *-0,56* | 22,19 | 5,32 | 60,03 | 0,26 |

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Номер образца** |  | **Х.-Х 1J *1ср*** | **X,.** | **х,-х,*****1ср*** |  | **Х.-Х, *3j Зср*** |
| 2 | 2,80 | 0,82 | 14,23 | -2,64 | 71,18 | 1,41 |
| 3 | 1,73 | -0,25 | 14,19 | -2,68 | 68,10 | -1,67 |
| Z | 5,95 |  | 20,61 |  | 209,31 |  |

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Номер образца** |  | ***W*** | ***(\*зГХ3"Г*** | Ч<, | Ч-Х> | Ч-^О |
| 1 | 8 | 9 | 10 | И | 12 | 13 |
| 1 | 0,31 | 28,30 | 0,07 | -2,98 | -0,15 | 1,38 |
| 2 | 0,67 | 6,97 | 1,99 | -2,16 | 1,16 | 3,72 |
| 3 | 0,06 | 7,18 | 2,79 | 0,67 | 0,42 | 4,48 |
| Е | 1,04 | 42,45 | 4,85 | -4,47 | 1,43 | 2,14 |

Использованы значения средних JT]cp=l,98; Jf2cp=16,87; =69,77.

Матрица сумм центрированных квадратов и смешанных произведений для класса коллекторов В имеет вид:

|  |  |  |
| --- | --- | --- |
| 1,04 | -4,47 | 1,43 |
| -4,47 | 42,45 | 2,14 |
| 1,43 | 2,14 | 4,85 |

Имея эти данные, построим выборочную матрицу (11.6):

* 1
* 3 + 3-2

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 198,00 -18,51 | 56,43 |  | 49,50 -4,63 | 14,11 |
| -18,51 45,37 | -4,97 | = | -4,63 11,34 | -1,24 |
| 56,43 —4,97 | 25,56 |  | 14,11 -1,24 | 6,39 |

Вычислим обратную матрицу М\*1. Это можно сделать вручную, методом алгебраических дополнений или при помощи функции МОБР *Microsoft Excel.*

Обратная матрица имеет вид:

*М~'*

* 0,05551
* 0,00946
* 0,12073
* 0,00946 0,12073
* 0,09173 -0,00309 -0,00309 0,42251

Отсюда по формуле (10.9)

вычислим коэффициенты дискрими

нантной функции:

*а* =0,05551 (14,71 -1,98)+0,00946(4,26-16,87)+ +0,12073(93,16-69,77)=3,41123 ~ 3,41;

а2=-1,10856 --1,11; я=11,45837~ 11,46.

Таким образом, дискриминантная функция для разделения карбонатных пород на покрышки и коллекторы имеет вид

*D* = 3,41ХГ1,11Х2 +11,46Х3. (11.13)

Граничное значение функции, вычисленное по формуле (11.11) составляет

*D =* [3,41(14,71+1,98)-

- 1,11(4,26+16,87)+11,46(93,16+69,77)]:2 = 950,318.

Полученное значение позволяет осуществить проверку правильности разделения на классы и прогноз неизвестных образцов карбонатных пород по формуле (11.13).

Например, для образца а (см. табл. 11.2)

D=3,41\*26,12-1,11\*3,34+11,46\*96,53=1191,59.

Так как *D} > DQ,* этот образец совершенно справедливо отнесен к классу *А.*

Далее рассчитываем значение дискриминантной функции для прогнозных образцов. Вычисленные значения сравниваем с Do, делаем выводы о принадлежности к тому или иному классу.

Результаты проверки и прогноза заносим в таблицу 11.5.

*Таблица 11.5***Результаты разделения прогнозной совокупности образцов пород на классы с помощью линейной дискриминантной функции**

|  |  |
| --- | --- |
| **Произведение** | **Номера образцов** |
| **покрышки** | **коллекторы** |
| **а** | ъ | с | **I** | **II** | **III** |
| 3,41 *-Х}* | 89,07 | 27,42 | 34,07 | 4,84 | 9,55 | 5,90 |
| -1,11 Л | -3,71 | -4,4 | -6,25 | -24,63 | -15,80 | -15,75 |
| 11,46Х3 | 1106,23 | 1063,83 | 1032,78 | 802,54 | 815,72 | 780,43 |
| *D* | 1191,9 | 1087,01 | 1060,60 | 782,75 | 809,47 | 770,58 |
| Контрольный прогноз | >4 |  | >4 | <4 |  | <4 |
| Вывод о принадлежности к классу | А покрышки | А покрышки | А покрышки | в покрышки | в покрышки | В покрышки |

*Продолжение таблицы 11.5*

|  |
| --- |
| **Номера образцов** |
| **прогнозные образцы** |
| **1** | 2 | 3 | 4 | 5 | 6 |
| 22,17 | 35,02 | 32,02 | 2,25 | 13,30 | 16,9 |
| -6,08 | -2,85 | -8,47 | -6,17 | -24,39 | -16,22 |
| 1036,21 | 1095,69 | 999,43 | 1031,63 | 835,09 | 740,09 |
| 1052,30 | 1127,86 | 1023,45 | 1027,71 | 824,00 | 740,78 |
| >4 | >ч | >4 | <4 | <4 | <4 |
| **А**покрышки | А покрышки | А покрышки | В коллекторы | В коллекторы | В коллекторы |

**Содержание отчета**

Отчет оформляется в соответствии с требованиями, приведенными в приложении А.

**Контрольные вопросы**

* 1. В чем сущность дискриминантного анализа?
* 2. Что такое дискриминантная функция?
* 3. Что такое граничное значение дискриминантной функции?

##  ЛИНЕЙНЫЙ ДИСКРИМИНАНТНЫЙ АНАЛИЗ ФИШЕРА

Это содержимое относится только к Studio (классическая модель). Аналогичные модули перетаскивания были добавлены в конструктор Машинное обучение Azure. Дополнительные сведения см. в статье сравнение двух версий.

## ОБЗОР МОДУЛЯ

В этой статье описывается, как использовать модуль **Discriminant Analysis линейного анализа Фишера** в машинное обучение Azure Studio (классическая модель), чтобы создать новый набор данных компонентов, в котором захватывается сочетание функций, которые лучше разделяются на два или более классов.

Этот метод часто используется для сокращения размерности, поскольку он проецирует набор признаков на меньшее пространство признаков, сохраняя информацию, которая разделяет классы. Это не только сокращает стоимость вычислений для задачи классификации, но также может предотвратить переобучение.

Для создания оценок необходимо указать столбец меток и набор числовых столбцов функций в качестве входных данных. Алгоритм определяет оптимальную комбинацию входных столбцов, которая линейно разделяет каждую группу данных при минимальных расхождениях внутри каждой группы. Модуль возвращает набор данных, содержащий компактные, преобразованные функции, а также преобразование, которое можно сохранить и применить к другому набору данных.

### ДОПОЛНИТЕЛЬНЫЕ СВЕДЕНИЯ О ЛИНЕЙНОМ АНАЛИЗЕ DISCRIMINANTНОСТИ

Линейный discriminantный анализ аналогичен анализу дисперсии (ДВУХФАКТОРНЫЙ) в том, что он работает путем сравнения средств переменных. Как и в случае с ДИСПЕРСИОННЫЙ обоснованной, она основана на следующих допущениях:

* предикторы являются независимыми;
* Стандартные функции плотности вероятности каждого примера распределены
* схожие значения дисперсии среди групп.

Анализ линейного Discriminantа иногда сокращается до LDA, но это легко путают с скрытых Дирихле метода выделения. Методы совершенно различны, поэтому в этой документации мы используем полные имена везде, где это возможно.

## НАСТРОЙКА ЛИНЕЙНОГО АНАЛИЗА DISCRIMINANT

Добавьте входной набор данных и убедитесь, что входные данные соответствуют этим требованиям:

* Данные должны быть как можно более полными. Строки с любыми отсутствующими значениями игнорируются.
* Предполагается, что значения имеют нормальное распределение. Прежде чем использовать **линейный анализ Discriminant для Фишера**, проверьте данные для выбросов или протестируйте распределение.
* Число прогностические факторы должно быть меньше, чем количество примеров.
* Удалите все нечисловые столбцы. Алгоритм проверяет **все** допустимые числовые столбцы, содержащиеся во входных данных, и возвращает ошибку, если включены недопустимые столбцы. Если необходимо исключить какие-либо числовые столбцы, добавьте в модуль набора данных флажок Select Columns in DataSet (выбор столбцов ) перед **анализом линейного Discriminant анализа**, чтобы создать представление, содержащее только те столбцы, которые необходимо проанализировать. Позднее можно повторно присоединить столбцы с помощью инструкции Add Columns. Исходный порядок строк сохраняется.

Подключите входные данные к модулю " **линейный Discriminant Analysis** " (Фишера).

Для **столбца метки класса** щелкните **запустить селектор столбцов** и выберите один столбец меток.

В поле **число средств извлечения** признаков введите нужное число столбцов.

Например, если набор данных содержит восемь числовых столбцов с числовыми характеристиками, можно ввести его, 3 чтобы свернуть в новое, уменьшенное пространство функций только для трех столбцов.

Важно понимать, что выходные столбцы не точно соответствуют входным столбцам, а представляют собой компактное преобразование значений во входных столбцах.

Если в качестве значения для **числа извлечений компонентов** используется 0, а в качестве входных данных используется n столбцов, то возвращаются n средств извлечения, содержащие новые значения, представляющие n-мерный размерность компонента.

### РЕЗУЛЬТАТЫ

Алгоритм определяет сочетание значений во входных столбцах, которые линейно разделяют каждую группу данных, одновременно сокращая расстояния внутри каждой группы и создают два выхода:

**Преобразованные функции**. Набор данных, содержащий указанное число столбцов средства извлечения, именуемых **col1**, **Col2**, **Col3** и т. д. Кроме того, выходные данные включают также переменную класса или метки.

Этот компактный набор значений можно использовать для обучения модели.

**Преобразование "линейное discriminantе" для анализатора Фишера**. Преобразование, которое можно сохранить, а затем применить к набору данных, имеющему ту же схему. Это полезно, если вы анализируете множество наборов данных одного типа и хотите применить одинаковое уменьшение функций к каждому. Набор данных, к которому он применяется, должен иметь одну и ту же схему.

## ПРИМЕРЫ

Примеры выбора компонентов в машинном обучении см. в Коллекция решений ии Azure:

: демонстрируется использование этого модуля для уменьшения размерности.

## ТЕХНИЧЕСКИЕ ПРИМЕЧАНИЯ

В этом разделе содержатся сведения о реализации, советы и ответы на часто задаваемые вопросы.

### сОВЕТЫ ПО ИСПОЛЬЗОВАНИЮ

Этот метод работает только с непрерывными переменными, категориальные или порядковые переменные являются недопустимыми.

При вычислении матрицы преобразования строки с недостающими значениями игнорируются.

Если сохранить преобразование из эксперимента, то преобразования, вычисленные из исходного эксперимента, применяются к каждому новому набору данных и не пересчитываются. Таким образом, если требуется вычислить новый набор функций для каждого набора данных, используйте новый экземпляр **линейного Discriminant анализа Фишера** для каждого набора данных.

 [Введение в анализ данных — Байесовские классификаторы: от LDA и QDA до наивного Байеса (mipt-stats.gitlab.io)](https://mipt-stats.gitlab.io/courses/ad_fivt/bayes_classification.html)

**1. Линейный и квадратичный дискриминантный анализ**

Загрузим датасет Ирисы Фишера — [классический учебный датасет](https://en.wikipedia.org/wiki/Iris_flower_data_set), который встроен в sklearn. Числовые столбцы отвечают за длину и ширину наружной и внутренней доли околоцветника для трех сортов ириса: setosa, virginica, versicolor.

In [2]:

iris = load\_iris()

X = iris.data

y = iris.target

Данные представлены в виде двумерной numpy-матрицы

In [3]:

X[:5]

Out[3]:

array([[5.1, 3.5, 1.4, 0.2],

 [4.9, 3. , 1.4, 0.2],

 [4.7, 3.2, 1.3, 0.2],

 [4.6, 3.1, 1.5, 0.2],

 [5. , 3.6, 1.4, 0.2]])

В данных представлены три класса

In [4]:

np.unique(y)

Out[4]:

array([0, 1, 2])

Имена классов

In [5]:

iris.target\_names

Out[5]:

array(['setosa', 'versicolor', 'virginica'], dtype='<U10')

Имена признаков

In [6]:

iris.feature\_names

Out[6]:

['sepal length (cm)',

 'sepal width (cm)',

 'petal length (cm)',

 'petal width (cm)']

Визуализируем данные в проекции на первый и третий признаки

In [7]:

plt.figure(figsize=(8, 5))

plt.scatter(X[:, 1], X[:, 3], c=y, cmap='Set1')

plt.xlabel(iris.feature\_names[1])

plt.ylabel(iris.feature\_names[3]);

Далее будем работать только с этими двумя признаками *для удобства визуализации*.

In [8]:

feature\_indexes = np.array([1, 3])

X = X[:, feature\_indexes]

**1.1 Линейный дискриминантный анализ**

Обучим модель

In [9]:

model = LinearDiscriminantAnalysis()

model.fit(X, y)

Out[9]:

LinearDiscriminantAnalysis()

Получим предсказания модели с помощью метода predict и посчитаем количество объектов, на которых модель дала неверный результат

In [10]:

(model.predict(X) != y).sum()

Out[10]:

5

Стоит отметить, что для вычисления ошибки мы использовали обучающую выборку.

Сделаем двумерную сетку с помощью функции np.mgrid, которой указываем диапазон и шаг по каждой координате.

In [11]:

grid = np.mgrid[1:5:0.01, -1:3.5:0.01]

grid.shape

Out[11]:

(2, 400, 450)

Растянем эту сетку в таблицу объекты-признаки с помощью объединения двух последних осей методом reshape и последующего транспонирования. Затем построим предсказания класса для всех элементов сетки и обратно преобразуем в формат сетки

In [12]:

\_, h, w = grid.shape

grid\_labels = model.predict(grid.reshape((2, h\*w)).T).reshape(h, w)

Визуализируем предсказания с помощью plt.pcolormesh

In [13]:

plt.figure(figsize=(10, 6))

plt.pcolormesh(grid[0], grid[1], grid\_labels,

 shading='gouraud', cmap='Set1', alpha=0.2)

plt.scatter(X[:, 0], X[:, 1], c=y,

 cmap='Set1', edgecolors='black')

plt.title('Классификация линейным дискриминантным анализом')

plt.xlabel(iris.feature\_names[feature\_indexes[0]])

plt.ylabel(iris.feature\_names[feature\_indexes[1]]);

Визуализируем предсказания вероятностей каждого из классов, которые можно получить с помощью метода predict\_proba у обученной модели.

In [14]:

grid\_labels = model.predict\_proba(grid.reshape((2, h\*w)).T).reshape(h, w, 3)

plt.figure(figsize=(15, 5))

**for** i **in** range(3):

 plt.subplot(1, 3, i+1)

 plt.pcolormesh(grid[0], grid[1], grid\_labels[:, :, i],

 shading='gouraud', cmap='Greens', alpha=0.1)

 plt.scatter(X[:, 0], X[:, 1], c=y,

 cmap='Set1', edgecolors='black')

 plt.title(f'Вероятность класса **{**iris.target\_names[i]**}**')

 plt.xlabel(iris.feature\_names[feature\_indexes[0]])

 plt.ylabel(iris.feature\_names[feature\_indexes[1]])

plt.tight\_layout()

Визуализируем также логарифмы предсказаний вероятностей классов, которые можно получить с помощью метода predict\_log\_proba у обученной модели.

In [15]:

grid\_labels = model.predict\_log\_proba(grid.reshape((2, h\*w)).T).reshape(h, w, 3)

plt.figure(figsize=(15, 5))

**for** i **in** range(3):

 plt.subplot(1, 3, i+1)

 plt.pcolormesh(grid[0], grid[1], grid\_labels[:, :, i],

 shading='gouraud', cmap='Greens', alpha=0.1)

 plt.scatter(X[:, 0], X[:, 1], c=y,

 cmap='Set1', edgecolors='black')

 plt.title(f'Логарифм вер-ти класса **{**iris.target\_names[i]**}**')

 plt.xlabel(iris.feature\_names[feature\_indexes[0]])

 plt.ylabel(iris.feature\_names[feature\_indexes[1]])

plt.tight\_layout()

**1.2 Квадратичный дискриминантный анализ**

Давайте теперь посмотрим на то, что получится с помощью квадратичного дискриминантного анализа. Обучим модель и посчитаем количество ошибок

In [16]:

model = QuadraticDiscriminantAnalysis()

model.fit(X, y)

(model.predict(X) != y).sum()

Out[16]:

7

Создадим сетку и получим по ней предсказания

In [17]:

grid = np.mgrid[-1:7:0.01, -3:5.5:0.01]

\_, h, w = grid.shape

grid\_labels = model.predict(grid.reshape((2, h\*w)).T).reshape(h, w)

Посмотрим на предсказания классов

In [18]:

plt.figure(figsize=(10, 6))

plt.pcolormesh(grid[0], grid[1], grid\_labels,

 shading='gouraud', cmap='Set1', alpha=0.2)

plt.scatter(X[:, 0], X[:, 1], c=y,

 cmap='Set1', edgecolors='black')

plt.title('Классификация квадратичным дискриминантным анализом')

plt.xlabel(iris.feature\_names[feature\_indexes[0]])

plt.ylabel(iris.feature\_names[feature\_indexes[1]]);

Несмотря на то, что модель более мощная с точки зрения теории, судя по картинке, модель на всей плоскости работает несколько странно.

Посмотрим также на вероятности

In [19]:

grid\_labels = model.predict\_proba(grid.reshape((2, h\*w)).T).reshape(h, w, 3)

plt.figure(figsize=(15, 5))

**for** i **in** range(3):

 plt.subplot(1, 3, i+1)

 plt.pcolormesh(grid[0], grid[1], grid\_labels[:, :, i],

 shading='gouraud', cmap='Greens', alpha=0.1)

 plt.scatter(X[:, 0], X[:, 1], c=y,

 cmap='Set1', edgecolors='black')

 plt.title(f'Вероятность класса **{**iris.target\_names[i]**}**')

 plt.xlabel(iris.feature\_names[feature\_indexes[0]])

 plt.ylabel(iris.feature\_names[feature\_indexes[1]])

plt.tight\_layout()

**Выводы.**

* Модель линейного дискриминантного анализа описывает более простые зависимости, что на практике скорее оказывается ее преимуществом. В частности, более стабильное предсказание, меньшее количество параметров, что напрямую влияет на требуемый объем обучающей выборки.
* Стоит отметить также квадратичное число параметров по числу признаков, которые нужно оценить при построении классификатора. Для этого требуется достаточно большая выборка.
* Помимо получения классификатора в процессе обучения модель полностью "познает" природу данных, включая распределение в пространстве признаков.

**2. Наивный байесовский классификатор для детекции спама**

Разберем теперь наивный байесовский классификатор. Несмотря на его простоту в некоторых задачах он работает даже лучше других, более сложных моделей. В любом случае, наивный байесовский классификатор содержит в себе важные теоретические идеи, поэтому с ним в любом случае полезно ознакомиться.

Что такое **спам**? Посмотрим [Википедию](https://ru.wikipedia.org/wiki/SPAM).

SPAM — торговая марка консервированного мяса, производимого американской компанией Hormel Foods Corporation. SPAM появился в 1936 году. Аббревиатура от **S**houlder of **P**ork and h**AM** — «свиные лопатки и окорока», a по другим данным, от англ. **SP**iced h**AM** — «ветчина со специями».

К 1936 году компания Hormel Foods начала терять свои позиции на рынке консервов. Как маркетинговый шаг, она переименовала свои консервы в SPAM. Выход нового бренда на рынок сопровождался бурной рекламной кампанией; по радио проигрывали песни наподобие этой:

Spam, Spam, Spam, Spam

Hormel’s new miracle meat in a can

Tastes fine, saves time

If you want something grand, ask for SPAM.

Впрочем, основной рекламой SPAM’у стала Вторая мировая война. SPAM в больших количествах выдавался солдатам. Великобритания зависела от привозного мяса, поэтому во время войны население получало по карточкам около 2,5 кг мяса в месяц на человека (из них 500 г — мясо первого сорта), в то время как SPAM продавался относительно свободно и поэтому стал основным заменителем мяса. Немало SPAM’а потреблялось беднотой и в послевоенные годы; неудивительно, что за это время SPAM изрядно надоел британцам. Всё это было обыграно в знаменитом скетче «SPAM» (1969) комик-группы «Монти Пайтон».

Данные для решения задачи детекции спама можно приготовить следующим образом:

1. Взять **набор размеченных текстовых сообщений**, часть которых размечена как спам, а остальные — как не спам;
2. Зафиксировать **словарь**, например, взяв все слова, встречающиеся в наборе текстовых сообщений;
3. **Преобразовать текстовые данные в целочисленные**, посчитав для каждого слова из словаря, встречается ли оно в данном сообщении.

Таким образом, каждому предложению соответствует вектор из нулей и единиц длины, равной мощности словаря. На полученных данных уже стандартным образом можно обучить наивный байесовский классификатор.

*Замечание.* При реализации вероятностных моделей надо помнить один очень важный на практике момент: произведение вероятностей большого количества чисел может очень быстро сравняться с нулем при вычислении на компьютере, так как компьютеру может не хватить вычислительной точности. Поэтому при реализации стоит использовать **логарифмы вероятностей**.

Применим наивный байесовский классификатор к конкретному [датасету](https://archive.ics.uci.edu/ml/datasets/SMS%2BSpam%2BCollection%22%20%5Ct%20%22_blank).

**2.1 Чтение данных**

Считаем данные

In [20]:

data = pd.read\_csv('./SMSSpamCollection', sep='**\t**', header=**None**)

data.columns = ['label', 'sms']

data.head()

Out[20]:

|  | **label** | **sms** |
| --- | --- | --- |
| **0** | ham | Go until jurong point, crazy.. Available only ... |
| **1** | ham | Ok lar... Joking wif u oni... |
| **2** | spam | Free entry in 2 a wkly comp to win FA Cup fina... |
| **3** | ham | U dun say so early hor... U c already then say... |
| **4** | ham | Nah I don't think he goes to usf, he lives aro... |

В датасете метки бывают 2 видов:

* ham — означает, что сообщение **не является спамом**,
* spam — означает, что сообщение **является спамом**.

Посмотрим на нормальное сообщение

In [21]:

data['sms'][0]

Out[21]:

'Go until jurong point, crazy.. Available only in bugis n great world la e buffet... Cine there got amore wat...'

Посмотрим на спам

In [22]:

data['sms'][2]

Out[22]:

"Free entry in 2 a wkly comp to win FA Cup final tkts 21st May 2005. Text FA to 87121 to receive entry question(std txt rate)T&C's apply 08452810075over18's"

Разделим данные на обучающую и тестовую часть

In [23]:

data\_train, data\_test = train\_test\_split(data, test\_size=0.2, random\_state=21)

Помним, что мы можем делать аналитику только на обучающей части. На тестовой только посчитаем качество в самом конце.

Посчитаем количество сообщений каждого класса

In [24]:

counts = pd.value\_counts(data\_train['label'], sort=**True**)

counts

Out[24]:

ham 3865

spam 592

Name: label, dtype: int64

Видим, что в данных есть сильный дисбаланс классов — спама сильно меньше, чем нормальных сообщений.

In [25]:

counts.plot(kind='bar');

Посмотрим на топ-70 слов в каждом классе

In [26]:

*# метки сетки по оси икс*

labels = [1e-3, 2e-3, 5e-3, 1e-2, 2e-2, 5e-2]

plt.figure(figsize=(16, 18))

*# для каждого класса*

**for** i, label **in** enumerate(['ham', 'spam']):

 *# извлечем все сообщения и склеим их*

 all\_words = ' '.join(

 data\_train[data\_train['label'] == label]['sms']

 *# приведем к нижнему регистру, удалим точки, разделим на слова*

 ).lower().replace('.', '').split()

 *# общее количество слов в этом классе*

 words\_count = len(all\_words)

 *# посчитаем встречаемость каждого слова и вернем топ-70*

 counts = Counter(all\_words).most\_common(70)

 counts = pd.DataFrame(counts, columns=['word', 'count'])

 plt.subplot(1, 2, i+1)

 sns.barplot(y=counts['word'], x=counts['count']/words\_count)

 plt.xscale('log')

 plt.xlabel('Частота слов')

 plt.xticks(labels, labels)

plt.tight\_layout()

Интересно посмотреть также на длину сообщений. Тут видим, что спамные сообщения чаще более длинные. Видимо, длина сообщений — хороший признак. Однако, мы будем далее смотреть только на текстовые признаки.

In [27]:

plt.figure(figsize=(8, 4))

sns.histplot(x=data\_train['sms'].apply(len),

 hue=data\_train['label'], stat='density',

 kde=**True**, common\_norm=**False**,

 kde\_kws=dict(gridsize=1000))

plt.xlim((0, 250))

plt.xlabel('Длина сообщения');

**2.2 Предобработка данных.**

Очевидно, что сразу в таком виде нельзя передавать данные наивному байесовскому классификатору. Их надо привести к численному виду.

Столбец label привести к численному виду можно очень просто.

In [28]:

data\_train['label'] = (data\_train['label'] == 'spam').astype(int)

data\_train.head()

Out[28]:

|  | **label** | **sms** |
| --- | --- | --- |
| **1199** | 0 | Al he does is moan at me if n e thin goes wron... |
| **3777** | 0 | Ok lor. Msg me b4 u call. |
| **3599** | 0 | Aight, we'll head out in a few |
| **1859** | 0 | Sir, i am waiting for your call. |
| **3341** | 0 | Like I made him throw up when we were smoking ... |

Для преобразования текстовых сообщений воспользуемся CountVectorizer, работающему по принципу мешка слов (*bag of words*). Он имеет следующие гиперпараметры (т.е. те, которые задаются пользователем):

* max\_df — максимальная доля сообщений, в которых может встречатся слово из словаря, то есть в словарь не включаются слишком **частые** слова, что помогает бороться со стоп-словами;
* min\_df — минимальная доля сообщений, в которых может встречатся слово из словаря, то есть в словарь не включаются слишком **редкие** слова;
* max\_features — максимальное возможное количество выбранных слов, они выбираются среди наиболее частых;
* stop\_words — можно просто взять и задать стоп-слова, которые не будут добавлены в словарь ни при каких обстоятельствах.

Мешок слов — специальное представление текстового сообщения в виде целочисленного вектора. Значение вектора на позиции jj имеет смысл количества раз, сколько слово jj встретилось в данном тексте. Порядок слов в тексте и грамматика при этом не учитываются. Соответствующим словарем называется все множество рассматриваемых слов.

Построим векторное представление для наших сообщений. Для этого мы объявляет объект класса CountVectorizer, указываем значения гиперпараметров. Далее применяем к обучающим данным функцию fit\_transform, которая последовательно выполняет следующие функции:

* fit — обучение модели, в данном случае подсчет частот слов и определение словаря;
* transform — по существующему словарю преобразует сообщения в векторы.

In [29]:

vectorizer = CountVectorizer(min\_df=0.01, max\_df=0.05)

vec\_data\_train = vectorizer.fit\_transform(data\_train['sms']).toarray()

Напечатаем весь мешок слов и их количество.

In [30]:

print(len(vectorizer.get\_feature\_names()))

print(vectorizer.get\_feature\_names())

173

['150p', '16', '50', 'about', 'after', 'again', 'all', 'already', 'also', 'always', 'am', 'amp', 'an', 'any', 'anything', 'around', 'as', 'ask', 'babe', 'back', 'been', 'before', 'buy', 'by', 'cant', 'care', 'cash', 'claim', 'com', 'come', 'contact', 'cos', 'could', 'customer', 'da', 'day', 'dear', 'did', 'doing', 'don', 'dont', 'down', 'even', 'every', 'feel', 'find', 'first', 'free', 'give', 'go', 'going', 'good', 'got', 'great', 'gt', 'gud', 'had', 'happy', 'has', 'he', 'her', 'here', 'hey', 'hi', 'him', 'his', 'home', 'hope', 'im', 'its', 'keep', 'know', 'last', 'late', 'later', 'let', 'life', 'like', 'll', 'lol', 'lor', 'love', 'lt', 'make', 'many', 'meet', 'message', 'min', 'miss', 'mobile', 'more', 'morning', 'msg', 'much', 'need', 'new', 'next', 'nice', 'night', 'number', 'off', 'oh', 'ok', 'one', 'only', 'our', 'out', 'over', 'phone', 'pick', 'place', 'please', 'pls', 'prize', 're', 'really', 'reply', 'right', 'said', 'say', 'see', 'send', 'sent', 'she', 'should', 'some', 'someone', 'something', 'sorry', 'still', 'stop', 'sure', 'take', 'tell', 'text', 'thanks', 'them', 'then', 'there', 'they', 'thing', 'think', 'time', 'today', 'tomorrow', 'tonight', 'too', 'try', 'txt', 'uk', 'urgent', 'us', 've', 'very', 'wait', 'want', 'was', 'wat', 'way', 'week', 'well', 'were', 'when', 'where', 'who', 'why', 'win', 'won', 'work', 'would', 'www', 'yeah', 'yes']

Как видим, сообщения автоматически были порезаны на слова, а слова переведены в нижний регистр.

Посмотрим на преобразованные данные. Каждая строчка таблицы — векторное представление некоторого сообщения.

In [31]:

print(vec\_data\_train[:5])

[[0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 1 0 0 0 0 1 0 0

 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0]

 [0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0

 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0]

 [0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0

 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0]

 [0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0]

 [0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0

 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 1 0 0

 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0

 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 1 1 0 0 0 0 0 0 0 0 0 0]]

Выполним теперь аналогичные преобразования с тестовыми данными

In [32]:

data\_test['label'] = (data\_test['label'] == 'spam').astype(int)

vec\_data\_test = vectorizer.transform(data\_test['sms']).toarray()

**3. Классификатор**

В библиотеке sklearn имеются следующие реализации наивного байесовского классификатора:

1. BernoulliNB — байесовский классификатор для данных, в которых все признаки являются бинарными;
2. MultinomialNB — байесовский классификатор для данных, в которых все признаки являются дискретными;
3. GaussianNB — байесовский классификатор для вещественных данных, каждый из признаков которых имеет нормальное распределение.

Важно отметить про следующий параметр:

* alpha — коэффициент сглаживания Лапласа или Линдсона, при фиксированном значении alpha условные плотности будут записаны следующим образом:

P(Xj=xj|Y=k)=∑ni=1I{Yi=k,Xij=xj}+α∑ni=1I{Yi=k}+αmj,P(Xj=xj|Y=k)=∑i=1nI{Yi=k,Xij=xj}+α∑i=1nI{Yi=k}+αmj,

где mjmj — количество различных значений признака xjxj; при alpha=0 сглаживания не происходит и получаются стандартные формулы для условных вероятностей.

В нашей текущей задаче для признаков, описывающих количество вхождений каждого слова из словаря в сообщение, логично использовать MultinomialNB. Однако после мы сравним точность предсказаний MultinomialNB с точностью предсказаний BernoulliNB для бинарных признаков: каждый признак является индикатором того, присутствует ли данное слово из словаря в сообщении.

Еще раз посмотрим на данные

In [33]:

vec\_data\_train

Out[33]:

array([[0, 0, 0, ..., 0, 0, 0],

 [0, 0, 0, ..., 0, 0, 0],

 [0, 0, 0, ..., 0, 0, 0],

 ...,

 [0, 0, 0, ..., 0, 1, 0],

 [0, 0, 0, ..., 0, 1, 0],

 [0, 0, 0, ..., 0, 0, 0]])

Обучаем модель

In [34]:

multinomial\_nb = MultinomialNB()

multinomial\_nb.fit(vec\_data\_train, data\_train.label)

Out[34]:

MultinomialNB()

Смотрим качество на тестовой выборке. В данном случае мы используем метрику "точность" (accuracy), которая определяется как доля верно классифицированных объектов. Ее реализует функция sklearn.metrics import accuracy\_score.

In [35]:

predictions = multinomial\_nb.predict(vec\_data\_test)

print(f'Точность: **{**accuracy\_score(data\_test.label, predictions) **:**.3**}**')

Точность: 0.974

Результат получился весьма неплохой.

А теперь посмотрим, как с этой же задачей справится наивный байесовский классификатор на бинарных данных. Для этого бинаризуем все данные.

In [36]:

X\_train = (vec\_data\_train > 0).astype(int)

X\_test = (vec\_data\_test > 0).astype(int)

In [37]:

bernoulli\_nb = BernoulliNB()

bernoulli\_nb.fit(X\_train, data\_train.label)

predictions = bernoulli\_nb.predict(X\_test)

print(f'Точность: **{**accuracy\_score(data\_test.label, predictions) **:**.3**}**')

Точность: 0.967

Точность наивного байесовского классификатора на бинарных данных оказалась всего немного меньше, чем на полных данных. Это связано с тем, что преимущественно слова в наших текстах всречаются не более одного раза.

А вообще хорошие ли мы результаты получили по нашей метрике? Мы можем "построить" тривиальную модель — всегда выдавать самый частый класс. В данном случае это класс 0, который соответствует нормальному сообщению. Посчитаем качество такого классификатора.

In [38]:

print(f'Точность: **{**accuracy\_score(data\_test.label, [0]\*len(data\_test)) **:**.3**}**')

Точность: 0.861

Точность оказалась не такой низкой. Поэтому получив точность 97% нельзя сразу радоваться, что мы получили крутой результат. Сначала нужно сравнить с какими-то тривиальными моделями.