

Урок 8. Снижение размерности данных

Большая размерность данных (под ней понимается размерность пространства признаков, то есть их количество) может серьезно усложнить задачу анализа таких данных и даже стать причиной некорректной работы некоторых алгоритмов. Кроме того, часто в исходных данных могут присутствовать лишние признаки, никак не связанные с целевой переменной. Поэтому часто встает задача понижения количества признаков, оставляя при этом самые значимые (наиболее сильно влияющие на значение целевого параметра) с отсечением менее значимых (наиболее слабо коррелирующих со значением целевого параметра) или с формированием новых признаков на основе старых. То есть ставится задача перехода от пространства большей размерности к пространству меньшей размерности с сохранением максимального количества полезной информации.

Алгоритмы снижения размерности

Алгоритмы снижения размерности пространства признаков делятся на две группы - отбор признаков (то есть отбрасывание наименее важных признаков) и понижение размерности путем формирования новых признаков на основе старых.

Отбор признаков

Самым простым и примитивным методом отбора является *одномерный отбор признаков*. Он заключается в оценке предсказательной силы каждого признака, то есть его информативности - насколько он коррелирует с целевой переменной. Затем отбираются либо заданное количество k признаков, либо те признаки, информативность которых выше некоторого порога.

Оценка предсказательной силы признака (или степени связи этого признака и целевой переменной) может проводиться разными методами, например:

- в случае регрессии - *корреляция*

$$R_j = \frac{\sum_{i=1}^l (x_{ij} - \bar{x}_j)(y_i - \bar{y})}{\sqrt{\sum_{i=1}^l (x_{ij} - \bar{x}_j)^2 \sum_{i=1}^l (y_i - \bar{y})^2}},$$

где \bar{x}_j и \bar{y} - среднее значение j -го признака и целевой переменной, соответственно. Чем больше по модулю корреляция (± 1), тем информативнее признак. Следует заметить, что этот метод учитывает только линейную связь между признаком и целевой переменной.

- в случае задачи классификации - *взаимная информация (mutual information)*, моделирующая корреляцию между признаками и классами. Она использует в расчете вероятность того, что одновременно значение j -го признака x_{ij} равно числу v и значение целевой переменной $y_i = k$, или, другими словами, долю таких объектов от общего количества объектов в выборке $P(x = v, y = k)$. Тогда взаимная информация будет находиться как

$$MI_j = \sum_{v \in X} \sum_{k \in Y} P(x = v, y = k) \log \frac{P(x = v, y = k)}{P(x = v)P(y = k)}.$$

Здесь $P(x = v)$ и $P(y = k)$ - доли объектов, на которых значение признака равно v и значение целевой переменной равно k , соответственно. Если признак и целевая переменная независимы, то

взаимная информация обращается в ноль. В отличие от предыдущего метода, этот метод позволяет находить произвольные зависимости (в т.ч. нелинейные) в пространстве произвольной размерности.

Такие методы позволяют оценить важность исключительно каждого признака отдельно, без учета влияния комбинаций признаков на целевую переменную, поэтому они и называются одномерными. На практике зачастую признаки влияют именно в совокупности, и по отдельности могут ошибочно быть расценены как некоррелирующие с целевой переменной, поэтому одномерные методы отбора не являются оптимальным методом в большинстве случаев.

Отдельной группой методов можно назвать так называемые *переборные методы*, которые дискретно оценивают качество модели, обученной на различных подмножествах признаков. При этом происходит полный перебор всех возможных вариантов. Обычно такие алгоритмы делятся на *жадные (greedy)* и *нежадные (non-greedy)*. Полный список их можно найти в дополнительных материалах.

Жадность алгоритмов заключается в том, что если один из признаков включен в подмножество (или исключен в случае исключающего метода), в следующих итерациях поиска он уже не учитывается, так что алгоритм работает на меньшем объеме данных. Известные алгоритмы этого типа - *жадное включение* и *жадное исключение*. В случае жадного включения на первой итерации аналогично одномерному отбору признаков находится признак, обладающий наибольшей предсказательной силой и добавляется в формирующееся подмножество $\{i_1\}$. Далее происходит перебор оставшихся признаков с попеременным добавлением каждого из них в подмножество к первому и оценкой качества получаемой модели, обученной на подмножестве из этих двух признаков $\{i_1, i_2\}$. В итоге в подмножестве остается тот признак, при добавлении которого получается наилучшее качество. Далее эта процедура повторяется до момента, пока ошибка получаемой модели уменьшается. На каждой итерации в подмножество добавляется один признак, максимально улучшающий работу модели. Если на какой-то итерации при добавлении признаков ошибка не уменьшается, процесс останавливается.

Плюсом такого алгоритма является относительная быстрота и возможность учета некоторых взаимодействий между признаками (как раз то, чего лишен одномерный отбор). Минусом же можно назвать вероятность застрять в локальном минимуме ошибки, если такой есть. В случае же когда есть единственный глобальный минимум, алгоритм найдет оптимальное решение.

Есть также модификации этого алгоритма с многократным проходом по выборке и поочередным включением/исключением признаков из подмножества для учета совокупного влияния признаков.

Примером нежадного алгоритма может быть простой последовательный полный перебор всех возможных подмножеств признаков. Такой подбор позволяет найти наиболее оптимальное подмножество признаков, но, очевидно, он является достаточно трудоемким (нужно перебрать 2^n вариантов, где n - число признаков), поэтому подходит только для датасетов с небольшим количеством признаков.

Еще одна группа методов отбора признаков - *встроенные в модели*. Они используют эвристики, заложенные в обучающие модели, для оценки важности признаков.

- Например, в случае работы с линейными моделями мы имеем зависимость целевой переменной от взвешенной суммы признаков

$$a(x) = \sum_{i=1}^n w_i x^i.$$

Здесь, если признаки масштабированы, веса будут являться показателями информативности признаков: чем больше вес, тем больший вклад данный признак вносит в значение целевой

переменной. На основе этого показателя можно проводить отбор признаков. Также, вспоминая уроки по линейным моделям, можно упомянуть, что использование L_1 -регуляризации приводит к занулению весов наименее важных признаков, то есть к их отбрасыванию, при этом больший коэффициент регуляризации будет приводить к большему количеству зануленных весов.

- В случае использования решающих деревьев и их композиций, где в каждой вершине происходит разбиение на два поддерева путем сравнения значения одного признака с некоторым значением порога, важность признака можно оценивать по тому, насколько он уменьшает значение критерия информативности, по которому оценивается качество разбиения:

$$Q(X_m, j, t) = H(X_m) - \frac{|X_l|}{|X_m|} H(X_l) - \frac{|X_r|}{|X_m|} H(X_r),$$

где X_m - множество объектов, попавших в вершину на данном шаге, X_l и X_r - множества, попадающие в левое и правое поддерево, соответственно, после разбиения. $H(X)$ - критерий информативности.

Чем сильнее падает критерий информативности при разбиении по данному признаку (то есть чем выше Q), тем этот признак важнее. Таким образом, важность j -го признака можно оценить путем вычисления суммы уменьшений критерия информативности по всем вершинам, в которых делалось разбиение по данному признаку. Чем больше эта сумма, тем важнее данный признак был при построении дерева. В случае композиций деревьев этот показатель суммируется по всем деревьям.

Текст, выделенный полужирным шрифтом### Понижение размерности

Кроме отбора признаков, который не всегда оптимален в плане сохранения максимума полезной информации, существуют еще методы понижения размерности путем формирования новых признаков на основе старых. Новых признаков при использовании такого метода должно быть меньше, чем исходных, при условии сохранения максимально возможного количества информации из исходных признаков. Например, объединение нескольких признаков в линейную комбинацию:

$$z_{ij} = \sum_{k=1}^n w_{jk} x_{ik},$$

где x_{ij} - исходные признаки, z_{ij} - новые признаки.

Простейшим методом такого понижения размерности является метод _случайных проекций_, который заключается в преобразованиях, сохраняющих расстояния и снижающих размерности. Существование таких преобразований доказано для выборок, в которых объектов меньше, чем признаков. Веса при всех признаках в таком методе можно выбирать случайно. При этом не факт, что мы попадем в оптимальное преобразование, но практика показывает, что метод работает, если размерность нового пространства признаков

$$d > \frac{8 \ln l}{\epsilon^2},$$

где l - количество объектов, ϵ - максимальное изменение расстояния между объектами (лемма о малом искажении или лемма Джонсона-Линденштрауса).

Метод главных компонент (PCA)

Одним из наиболее известных и широко применяемых методов понижения размерности является *метод*

главных компонент (*principal component analysis, PCA*). Он заключается в приближении матрицы признаков матрицей меньшего ранга - так называемом низкоранговом приближении.

Запишем показанную ранее формулу линейного преобразования признаков в матричном виде:

$$Z = XW^T,$$

где X - матрица "объекты-признаки", где по строкам отложены объекты, а по столбцам - значения признаков, Z - матрица новых признаков, W^T - транспонированная матрица весов. Приближение заключается формировании новой матрицы признаков $\tilde{X} = ZW \approx X$ с возможностью восстановления старых признаков по новым с максимальным уровнем точности, или, если говорить иначе, чтобы их различие было минимальным:

$$\|ZW - X\|^2 \rightarrow \min_{Z, W}.$$

При этом метод главных компонент предполагает, что матрица весов должна быть ортогональной, то есть произведение WW^T должно равняться единичной матрице. Восстановленная матрица ZW может иметь ранг меньший, чем исходная X , поэтому приближение будет называться низкоранговым.

Геометрически метод можно представить как проецирование признаков на гиперплоскость с максимизацией дисперсии получаемой выборки.

Если ранг матрицы исходных признаков $rank(X) \geq d$, где d - число новых признаков, то минимум функционала различия, описанного выше, достигается тогда, когда в качестве строк матрицы W используются собственные векторы матрицы $X^T X$, соответствующие максимальным собственным значениям $\lambda_1, \dots, \lambda_d$. Максимальные собственные значения и называются главными компонентами, от чего пошло название метода. Первая главная компонента соответствует максимальному собственному значению и т.д.

Некоторые полезные свойства метода:

- Матрица Z при этом будет такой, что $Z^T Z = \Lambda = diag(\lambda_1, \dots, \lambda_d)$.

- Минимизированный функционал ошибки будет равен

$$\|ZW - X\|^2 = \|X\|^2 - tr\Lambda,$$

где $tr\Lambda$, - след матрицы Λ , то есть сумма всех собственных значений $\lambda_1, \dots, \lambda_d$, а $\|X\|^2$ - сумма всех собственных значений исходной матрицы $\lambda_1, \dots, \lambda_n$, таким образом

$$\|ZW - X\|^2 = \sum_{j=d+1}^n \lambda_j,$$

то есть значение функционала ошибки будет равно сумме собственных значений, которые не были взяты в получаемое разложение. Поэтому логично брать в разложение максимальные собственные значения, оставляя минимальные.

- Матрица $X^T X$ - матрица ковариации, то есть матрица, которая характеризует дисперсию выборки. Дисперсия выборки после проецирования будет равна собственному значению λ , поэтому логично, что первым берется собственный вектор, соответствующий максимальному собственному значению - нам нужно сохранить максимум дисперсии.

Таким образом, для реализации метода главных компонент нужно :

- найти собственные значения матрицы $X^T X$;

- отобрать d максимальных;
- составить матрицу W^T , столбцы которой будут являться собственными векторами, соответствующими отобранному собственным значениям, расположенным в порядке убывания;
- получить новую матрицу "объекты-признаки", умножив исходную матрицу X на матрицу весов W :

$$Z = XW.$$

PCA и SVD

Сформулировав принцип реализации метода главных компонент, нельзя не заметить его родство с сингулярным разложением матриц (SVD). Вспомним, что сингулярное разложение матрицы - это разложение вида

$$X = UDV^T,$$

где столбцы ортогональной матрицы U - это собственные векторы матрицы XX^T , столбцы ортогональной матрицы V - собственные векторы матрицы $X^T X$, а на главной диагонали диагональной матрицы D расположены собственные значения матриц XX^T и $X^T X$ (они равны и также называются сингулярными числами матрицы X).

Если число новых признаков d равно старому числу признаков n , то можно приравнять разложения

$$X = ZW = UDV^T.$$

При этом матрицы W и V^T состоят из собственных векторов матрицы $X^T X$, то есть они равны при $Z = UD$.

Получается, что метод главных компонент - в своем роде "урезанная версия" сингулярного разложения, из которого убрали минимальные собственные значения с соответствующими собственными векторами. Таким образом, для реализации понижения размерности методом главных компонент с помощью SVD нужно:

- найти сингулярное разложение матрицы X ;
- сформировать из столбцов матрицы V , соответствующих d наибольшим сингулярным числам, матрицу весов W ;
- получить новую матрицу "объекты-признаки", умножив исходную матрицу X на матрицу весов W :

$$Z = XW.$$

Для закрепления теории реализуем PCA с помощью Python.

In [1]:

```
1 import numpy as np
2 from sklearn import datasets
3 import matplotlib.pyplot as plt
```

In [2]:

```
1 # Загрузим игрушечный датасет из sklearn
2 iris = datasets.load_iris()
3 X = iris.data
4
5 print(X)
```

```
[6.4 2.7 5.3 1.9]
[6.8 3.  5.5 2.1]
[5.7 2.5 5.  2. ]

[5.8 2.8 5.1 2.4]
[6.4 3.2 5.3 2.3]
[6.5 3.  5.5 1.8]
[7.7 3.8 6.7 2.2]
[7.7 2.6 6.9 2.3]
[6.  2.2 5.  1.5]
[6.9 3.2 5.7 2.3]
[5.6 2.8 4.9 2. ]
[7.7 2.8 6.7 2. ]
[6.3 2.7 4.9 1.8]
[6.7 3.3 5.7 2.1]
[7.2 3.2 6.  1.8]
[6.2 2.8 4.8 1.8]
[6.1 3.  4.9 1.8]
[6.4 2.8 5.6 2.1]
[7.2 3.  5.8 1.6]
```

In [3]:

```
1 # Для начала отмасштабируем выборку
2 X_ = X.astype(float)
3
4 rows, cols = X_.shape
5
6 # центрирование - вычитание из каждого значения среднего по строке
7 means = X_.mean(0)
8 for i in range(rows):
9     for j in range(cols):
10         X_[i, j] -= means[j]
11
12 # деление каждого значения на стандартное отклонение
13 std = np.std(X_, axis=0)
14 for i in range(cols):
15     for j in range(rows):
16         X_[j][i] /= std[i]
```

In [4]:

```
1 # Найдем собственные векторы и собственные значения
2
3 covariance_matrix = X_.T.dot(X_)
4
5 eig_values, eig_vectors = np.linalg.eig(covariance_matrix)
6
7 # сформируем список кортежей (собственное значение, собственный вектор)
8 eig_pairs = [(np.abs(eig_values[i]), eig_vectors[:, i]) for i in range(len(eig_values))]
9
10 # и отсортируем список по убыванию собственных значений
11 eig_pairs.sort(key=lambda x: x[0], reverse=True)
12
13 print('Собственные значения в порядке убывания:')
14 for i in eig_pairs:
15     print(i[0])
```

Собственные значения в порядке убывания:

```
437.77467247979905
137.10457072021052
22.013531335697255
3.107225464292904
```

Оценим долю дисперсии, которая описывается найденными компонентами.

In [5]:

```
1 eig_sum = sum(eig_values)
2 var_exp = [(i / eig_sum) * 100 for i in sorted(eig_values, reverse=True)]
3 cum_var_exp = np.cumsum(var_exp)
4 print(f'Доля дисперсии, описываемая каждой из компонент \n{var_exp}')
5
6 # а теперь оценим кумулятивную (то есть накапливаемую) дисперсию при учетывании каждой
7 print(f'Кумулятивная доля дисперсии по компонентам \n{cum_var_exp}')
```

Доля дисперсии, описываемая каждой из компонент

```
[72.96244541329987, 22.850761786701764, 3.668921889282877, 0.517870910715484
2]
```

Кумулятивная доля дисперсии по компонентам

```
[ 72.96244541  95.8132072  99.48212909 100.          ]
```

Таким образом, первая главная компонента описывает почти 73% информации, а первые две в сумме - 95.8%. В то же время последняя компонента описывает всего 0.5% и может быть отброшена без страха значительных потерь в качестве нашего анализа. Мы отбросим последние две компоненты, оставив первые две.

In [6]:

```
1 # Сформируем вектор весов из собственных векторов, соответствующих первым двум главным
2 W = np.hstack((eig_pairs[0][1].reshape(4,1), eig_pairs[1][1].reshape(4,1)))
3
4 print(f'Матрица весов W:\n', W)
```

Матрица весов W:

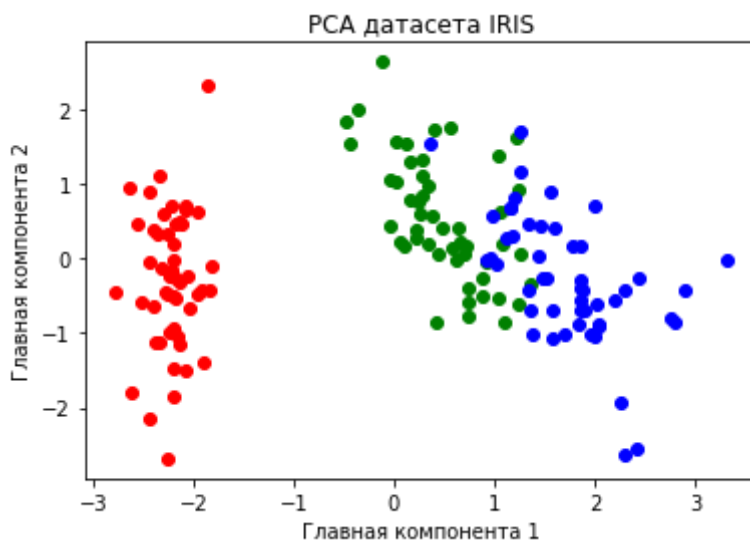
```
[[ 0.52106591 -0.37741762]
 [-0.26934744 -0.92329566]
 [ 0.5804131  -0.02449161]
 [ 0.56485654 -0.06694199]]
```

In [7]:

```
1 # Сформируем новую матрицу "объекты-признаки"
2 Z = X_.dot(W)
```

In [8]:

```
1 plt.figure()
2 y = iris.target
3 for c, i in zip("rgb", [0, 1, 2]):
4     plt.scatter(Z[y==i, 0], Z[y==i, 1], c=c)
5 plt.xlabel('Главная компонента 1')
6 plt.ylabel('Главная компонента 2')
7 plt.title('PCA датасета IRIS')
8 plt.show()
```



Таким образом, мы перешли от четырехмерного пространства признаков к двумерному и при этом классы остались разделимы в пространстве, то есть классификация возможна.

PCA наиболее хорошо работает, когда собственные значения λ на каком-то участке графика распределения убывают скачкообразно (критерий крутого склона), другими словами, если существуют предпосылки к тому, что следует решать задачу в пространстве меньшей размерности. Если же они убывают монотонно, следует рассмотреть вариант использования других методов работы с пространством признаков.

Дополнительные материалы

- ◀ ▶

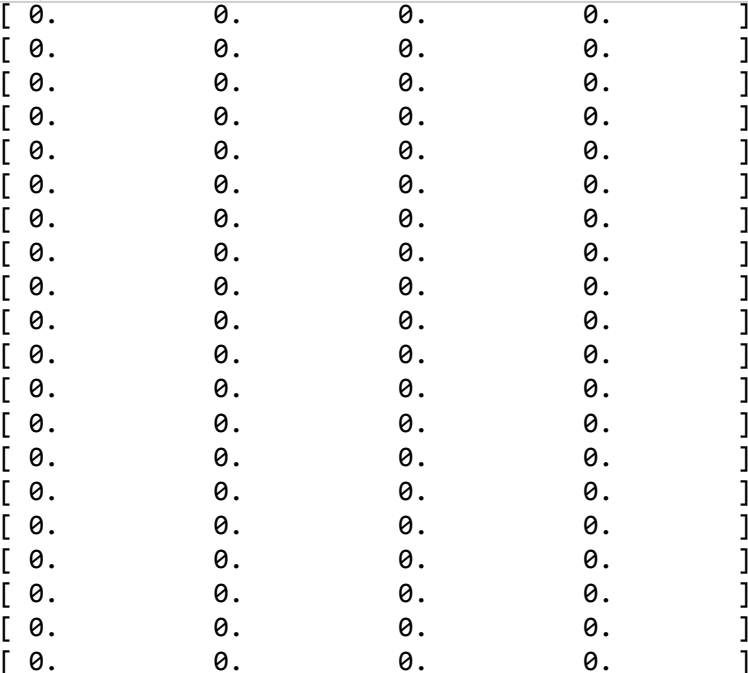
После того, как отработает метод главных компонент, матрица признаков переходит к новому базису. Причем процесс этот не обратим и обратно исходную матрицу не восстановить. Поэтому те компоненты-признаки, которые образуются после применения PCA, не имеют ничего общего с исходными признаками. Следовательно наиболее значимые признаки с помощью PCA подобрать нельзя.

In [9]:

In [10]:

In [11]:

```
1 U, s, W = np.linalg.svd(X)
2 # Транспонируем матрицу W
3 V = W.T
4
5 # s - список диагональных элементов, его нужно привести к виду диагональной матрицы для
6 Sigma = np.zeros_like(X, dtype=float)
7 Sigma[np.diag_indices(min(X.shape))] = s
8 print(f'Матрица Sigma размер:{Sigma.shape} :\n{Sigma}')
9 print(f'Матрица U размер:{U.shape} :\n{U}')
10 print(f'Матрица V размер:{V.shape} :\n{V}')
```



A 16x5 grid of zeros representing matrix U. Each row and column contains 16 and 5 zeros respectively, arranged in a grid format.

In [12]:

```
1 eig_sum = sum(s)
2 var_exp = [(i / eig_sum) * 100 for i in sorted(s, reverse=True)]
3 cum_var_exp = np.cumsum(var_exp)
4 print(f'Доля дисперсии, описываемая каждой из компонент \n{var_exp}')
```

5

```
6 # а теперь оценим кумулятивную (то есть накапливаемую) дисперсию при учетывании каждой
7 print(f'Кумулятивная доля дисперсии по компонентам \n{cum_var_exp}')
```

Доля дисперсии, описываемая каждой из компонент

```
[80.59340691495326, 14.916876798004958, 2.906715976729492, 1.5830003103122972]
```

Кумулятивная доля дисперсии по компонентам

```
[ 80.59340691  95.51028371  98.41699969 100.          ]
```

Видим, что на первые два компонента приходится более 95% дисперсии. Занулим 3 и 4.

In [13]:

```
1 n_elements = 2
2 Sigma = Sigma[:, :n_elements]
3 Sigma
```

Out[13]:

[illegible]

In [14]:

```
1 V = V[:, :n_elements]
2 V
```

Out[14]:

```
array([[ -0.75110816,   0.2841749 ],
       [ -0.38008617,   0.5467445 ],
       [ -0.51300886,  -0.70866455 ],
       [ -0.16790754,  -0.34367081 ]])
```

In [15]:

```
1 # Создадим новую матрицу признаков с 4 признаками, но рангом 2
2 B = U.dot(Sigma.dot(V.T))
3 B
```

```
[6.45170011, 2.72458368, 5.61678761, 1.98815351],
[7.07836761, 3.18949442, 5.71367983, 1.97888643],
[7.18289837, 3.09402129, 6.11748277, 2.15218979],
[8.00424223, 3.68769544, 6.27956573, 2.15587192],
[6.45454552, 2.7121756 , 5.64975615, 2.00278377],
[6.27816553, 2.8511819 , 5.01789057, 1.73268388],
[6.15013521, 2.59050415, 5.36931989, 1.90202165],
[7.47592704, 3.26532505, 6.26604488, 2.19443089],
[6.66025859, 2.90931078, 5.58181665, 1.9547504 ],
[6.5570269 , 2.91407352, 5.38360238, 1.87406997],
[6.12621535, 2.82812087, 4.79350481, 1.64431665],
[6.86962851, 3.11652481, 5.49793987, 1.89921925],
[6.78596906, 2.95263502, 5.7131341 , 2.00335628],
[6.82013719, 3.14944992, 5.33426189, 1.82957415],
[5.92300206, 2.53678163, 5.07704158, 1.78938593],
[6.94688336, 2.99572703, 5.90892758, 2.07806896],
[6.8953799 , 3.00965886, 5.78414682, 2.02613891],
[6.66546178, 2.99769195, 5.39325923, 1.86925995],
[6.13889606, 2.68777816, 5.13096763, 1.7954606 ],
[6.52794178, 2.94674481, 5.25756553, 1.81967836]
```

In [16]:

```
1 # Проведем сингулярное разложение матрицы B
2 U, s, W = np.linalg.svd(B)
3 # Транспонируем матрицу W
4 V = W.T
5 print("Элементы диагональной матрицы", s)
```

Элементы диагональной матрицы [9.59599139e+01 1.77610337e+01 2.07617549e-14
2.38003498e-15]

Видим, что они практически равны 0. Создадим матрицу признаков C в новом двухмерном базисе через матрицу перехода

In [17]:

```
1 C = W.dot(B.T).T
2 C
-2.22044605e-16],
[-9.12651817e+00, -3.32247752e-01,  3.08086889e-15,
 0.00000000e+00],
[-7.27560832e+00, -4.60955955e-01,  2.22044605e-15,
 0.00000000e+00],
[-8.55814639e+00, -3.97341697e-01,  2.56739074e-15,
 4.44089210e-16],
[-7.87237747e+00, -4.85081002e-01,  1.77635684e-15,
 0.00000000e+00],
[-8.66605949e+00, -2.86037963e-01,  1.92901251e-15,
 2.22044605e-16],
[-6.45347358e+00,  2.23799845e-02,  1.51267887e-15,
 1.11022302e-16],
[-8.63768432e+00, -2.45515595e-01,  2.37310172e-15,
 4.44089210e-16],
[-7.16780021e+00, -2.91011251e-01,  1.94289029e-15,
 0.00000000e+00],
[-6.47915170e+00, -3.09633237e-01,  1.30451205e-15,
 2.22044605e-16],
[-7.97879519e+00, -1.75031016e-01,  2.08166817e-15,
```

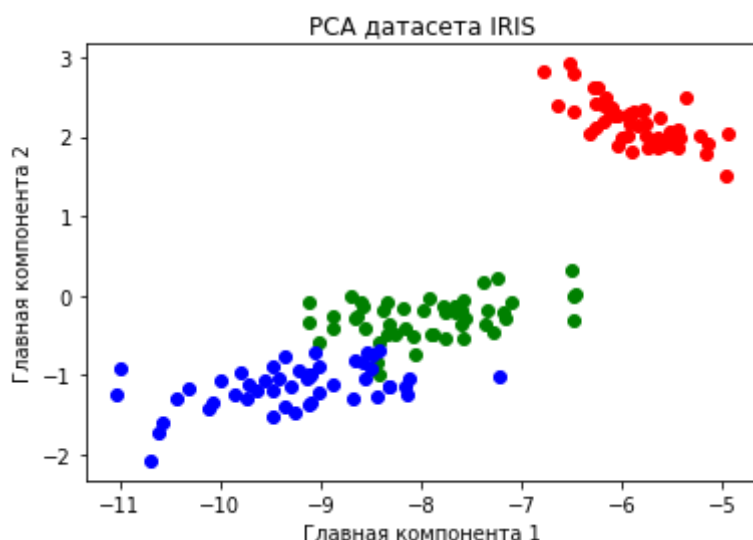
Видим, что последние два столбца признаков стали равны 0. Их можно удалить.

In [18]:

```
1 # Сформируем новую матрицу "объекты-признаки"
2 D = C[:, :2]
```

In [19]:

```
1 plt.figure()
2 y = iris.target
3 for c, i in zip("rgb", [0, 1, 2]):
4     plt.scatter(D[y==i, 0], D[y==i, 1], c=c)
5 plt.xlabel('Главная компонента 1')
6 plt.ylabel('Главная компонента 2')
7 plt.title('PCA датасета IRIS')
8 plt.show()
```



In [25]:

```
1 y_predict_train_PCA = model.predict(X_train_PCA)
2 y_predict_test_PCA = model.predict(X_test_PCA)
3 print("Точность на обучающей выборке", accuracy_score(y_train_PCA, y_predict_train_PCA))
4 print("Точность на тестовой выборке", accuracy_score(y_test_PCA, y_predict_test_PCA))
```

Точность на обучающей выборке 0.9714285714285714

Точность на тестовой выборке 1.0

Вывод: Результаты работы модели после снижения размерности стали лучше