


Лекция 7:

Линейные модели регрессии, регуляризация, преобразование пространства признаков

Методы регуляризации (штраф за сложность)

- Методы выбора подмножества используют МНК для линейной модели, которая содержит подмножество предикторов.
- В качестве альтернативы, можно построить модель потенциально содержащую все предикторы с использованием методики, которая *ограничивает или регуляризует* оценки коэффициентов:

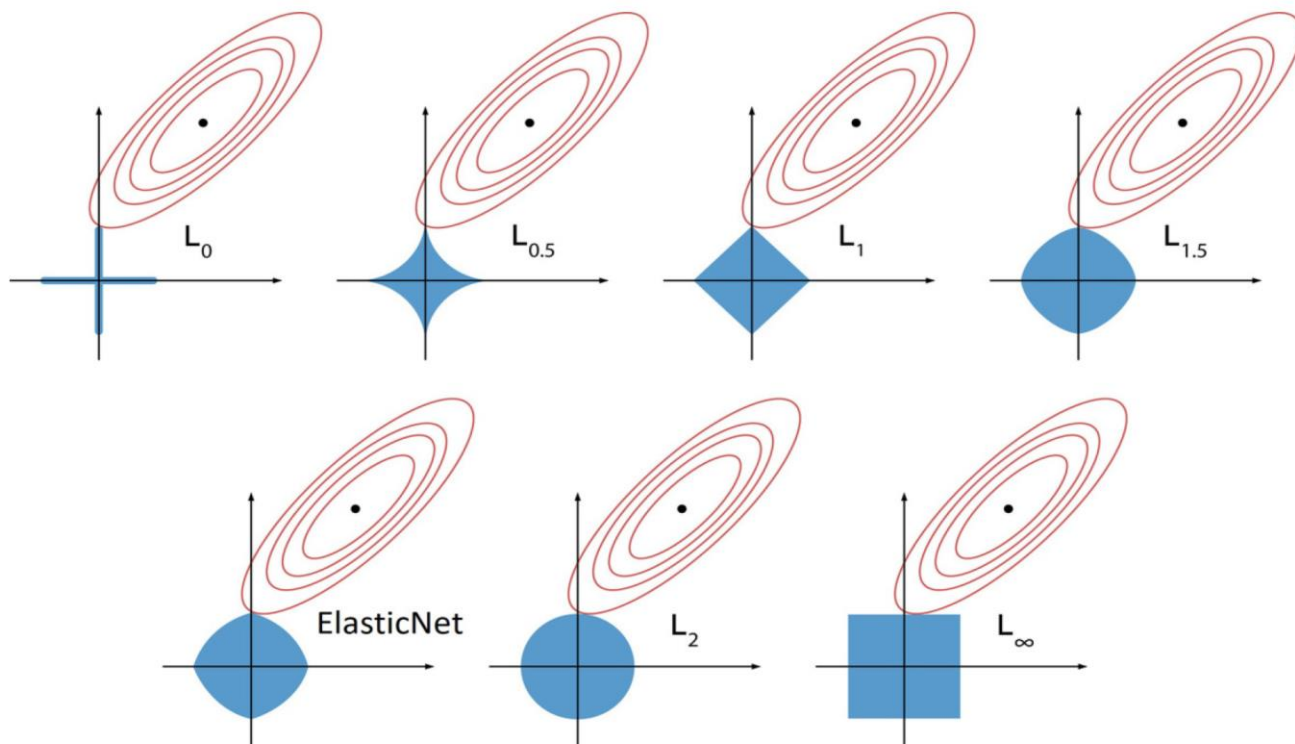
$$\min_w \left[\sum_{i=1}^l \left(y_i - w_0 - \sum_{j=1}^p x_{ij} w_j \right)^2 + \gamma R(\mathbf{w}) \right]$$

Штраф за сложность 

- Может быть не сразу понятно, почему ограничение на абсолютные значения коэффициентов может улучшить модель, но оказывается:
 - оно может значительно уменьшить дисперсию модели
 - оно уменьшает (и даже устраняет) влияние мультиколлинеарности, т.к. не дает неограниченно расти коэффициентам при зависимых переменных

Регуляризация L_p

$$\min_w [RSS(w) + \gamma L_p(w)] \Leftrightarrow \begin{cases} \min_w [RSS(w)] \\ L_p(w) \leq C \end{cases}$$



Масштабирование предикторов

- Оценки коэффициентов стандартным МНК являются масштабируемой - умножение *предиктора* на константу просто приводит к масштабированию оценок коэффициентов МНК.
- Оценки коэффициентов с регуляризацией наоборот - *не масштабируемые*, т.е. могут *существенно* измениться при умножении заданного предиктора на константу.
- Поэтому лучше всего применять регуляризацию:
 - либо после нормирования признакового пространства, например:
$$x'_i = \frac{x_i - E(x_i)}{SE(x_i)}$$
 - либо в штрафе (регуляризаторе) нормировать сами коэффициенты, например, делить их на соответствующие коэффициенты модели МНК w_{OLS} без регуляризации:

$$R(w) = L_p(|w|/|w_{OLS}|)$$

Гребневая регрессия (L_2)

- Целевая функция:

$$\min_w \left[\sum_{i=1}^l \left(y_i - w_0 - \sum_{j=1}^p x_{ij} w_j \right)^2 + \gamma \|w\|^2 \right]$$

- Гребневая регрессия (как и МНК) стремится найти коэффициенты регрессионного уравнения, которые минимизируют RSS, но, второе слагаемое (штраф), мал при коэффициентах, близких к нулю
 - Метапараметр регуляризации $\gamma \geq 0$ управляет относительным влиянием этих двух слагаемых на оценки коэффициентов.
- Можно показать, что:

$$\nabla_w [RSS(w) + \gamma w^T w] = -2X^T(y - Xw) - 2\gamma w = 0 \Rightarrow$$
$$w_\gamma = (\underbrace{X^T X + \gamma I}_{\text{Никогда не вырождена}})^{-1} X^T y$$

Никогда не вырождена

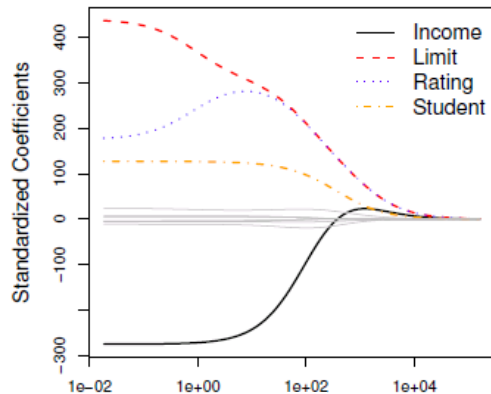
- Аналогично через SVD разложение получаем:

$$w_\gamma = \sum_i \frac{\sqrt{\lambda_i}}{(\lambda_i + \gamma)} u_i (v_i^T y)$$

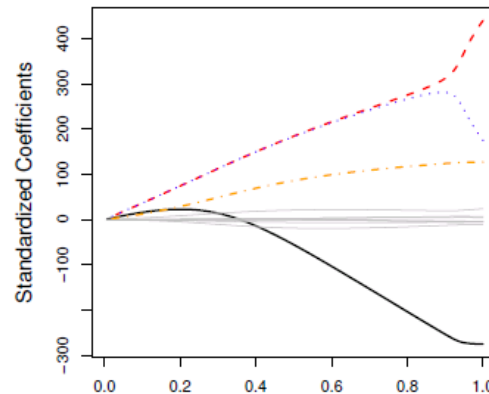
Влияние параметра регуляризации в L_2

- По SVD разложению также можно показать:

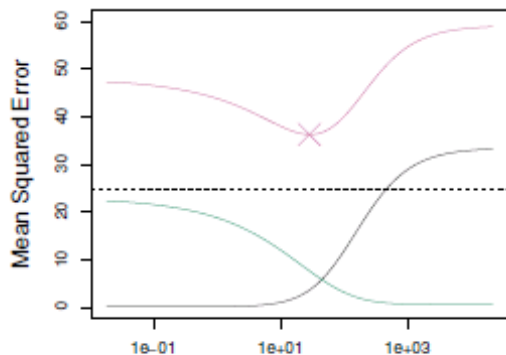
$$\|w_\gamma\|^2 = \sum_i \frac{\lambda_i}{(\lambda_i + \gamma)^2} (v_i^T y)^2 \leq \|w_{OLS}\|^2 = \sum_i \frac{1}{\lambda_i} (v_i^T y)^2$$



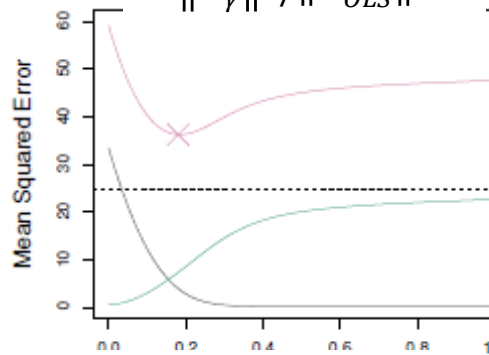
γ



$\|w_\gamma\|^2 / \|w_{OLS}\|^2$



γ



$\|w_\gamma\|^2 / \|w_{OLS}\|^2$

Сокращение «эффективной размерности»:

$$\text{tr}[(X^T X + \gamma I)^{-1} X^T] \leq \text{tr}[(X^T X)^{-1} X^T]$$

- Трассы коэффициентов в зависимости от параметра регуляризации
- MSE декомпозиция в зависимости от параметра регуляризации:

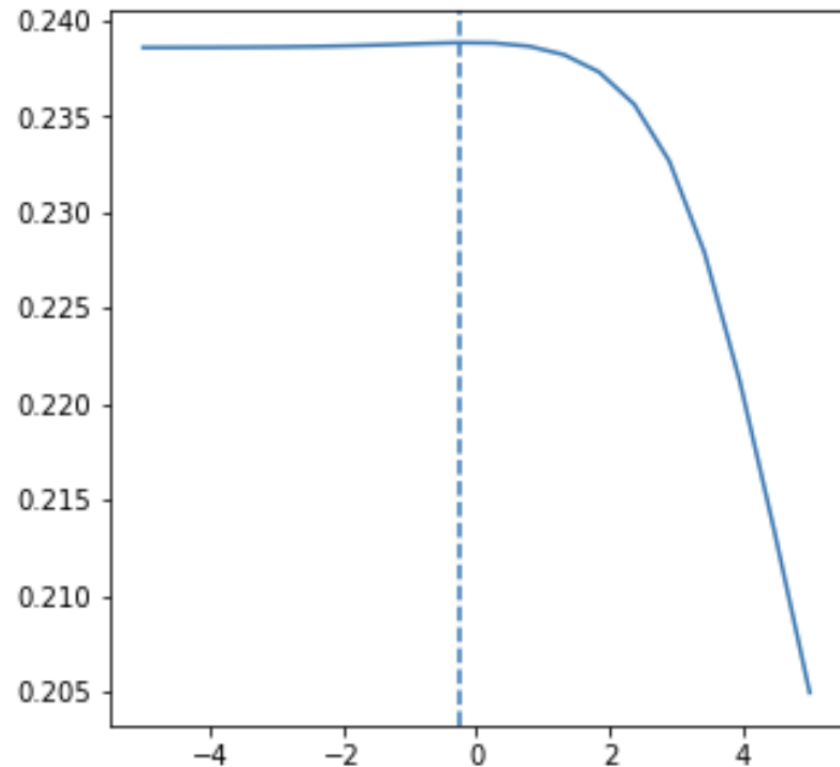
- MSE на тестовой выборке
- Дисперсия модели
- Смещение модели

Пример Ridge

```
from sklearn.linear_model import Ridge

degree = np.linspace(-5, 5, 20)
alphas = np.exp(degree)
scores = []
for alpha in alphas:
    ridge = Ridge(alpha=alpha, fit_intercept=False)
    score = cross_val_score(ridge, X, y, cv=5,
                           scoring="r2")
    scores.append(np.mean(score))

plt.figure(figsize=(5, 5))
plt.plot(degree, scores)
best_score = np.argmax(scores)
plt.axvline(x=degree[best_score], linestyle="--")
plt.show()
```



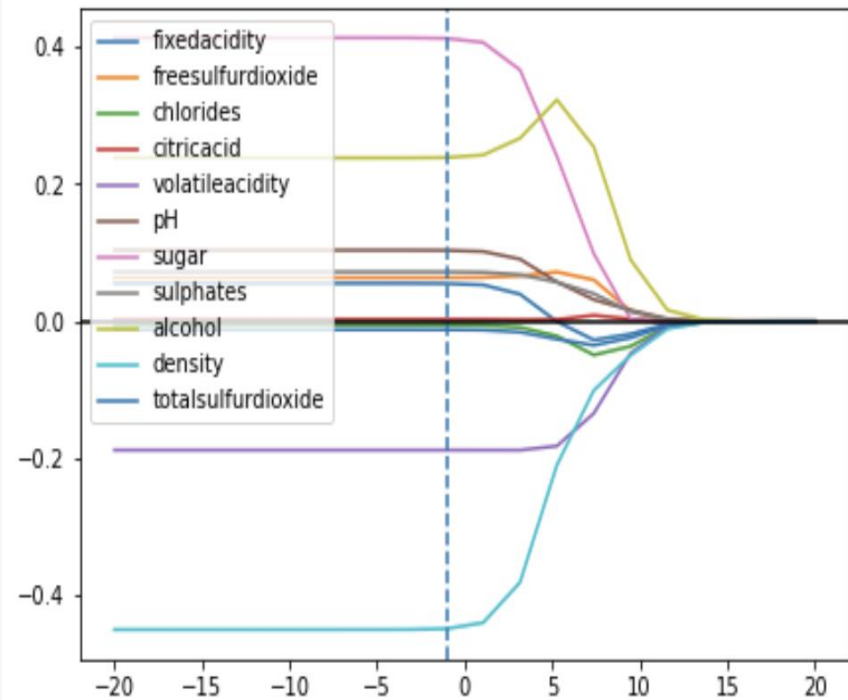
Ошибки перекрестной проверки, которые являются результатом применения ridge регрессии для различных значений параметра регуляризации (шкала логарифмическая, варьируется степень экспоненты).

Пример Ridge

```
from sklearn.linear_model import Ridge
from sklearn.preprocessing import StandardScaler

scaler = StandardScaler()
s_X = scaler.fit_transform(X)
coefs = []
for alpha in alphas:
    ridge = Ridge(alpha=alpha)
    ridge.fit(s_X, y)
    coef = ridge.coef_.reshape(-1, 1)
    coefs.append(coef.reshape(1, -1))

plt.figure(figsize=(7, 5))
plt.plot(degree, np.vstack(coefs))
plt.legend(X.columns, loc='upper left')
plt.axvline(x=degree[best_score], linestyle="--")
plt.axhline(y=0, color="black")
plt.show()
```



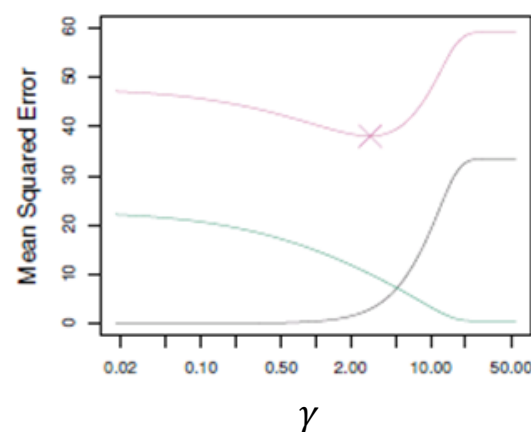
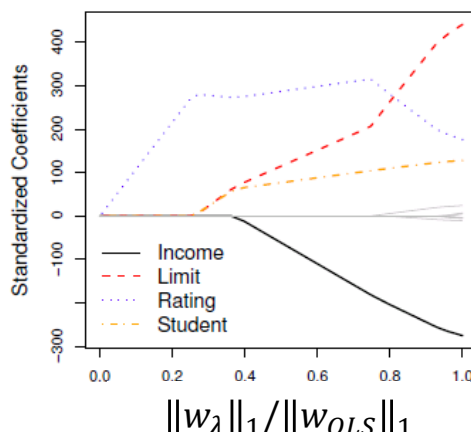
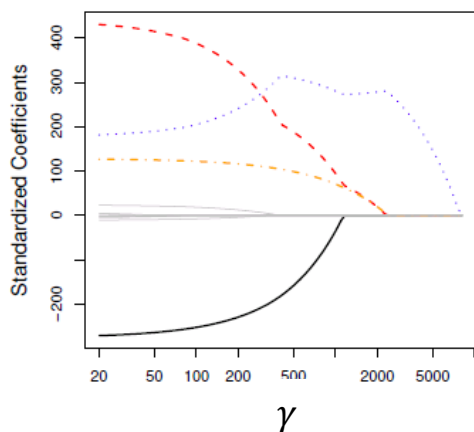
Оценки коэффициентов в зависимости от параметра регуляризации. Вертикальная пунктирная линия обозначает лучшее значение параметра регуляризации, полученное в результате перекрестной проверки.

Метод LASSO (L_1)

- Гребневая регрессия имеет важный недостаток - включает все предикторы, не осуществляет отбор
- LASSO (Least Absolute Shrinkage and Selection Operator) - преодолевает этот недостаток за счет регуляризации L_1
- Целевая функция:

$$\min_w \left[\sum_{i=1}^l \left(y_i - w_0 - \sum_{j=1}^p x_{ij} w_j \right)^2 + \gamma \sum_{j=1}^p |w_j| \right]$$

- Обнуление «не важных» коэффициентов: MSE разложение:



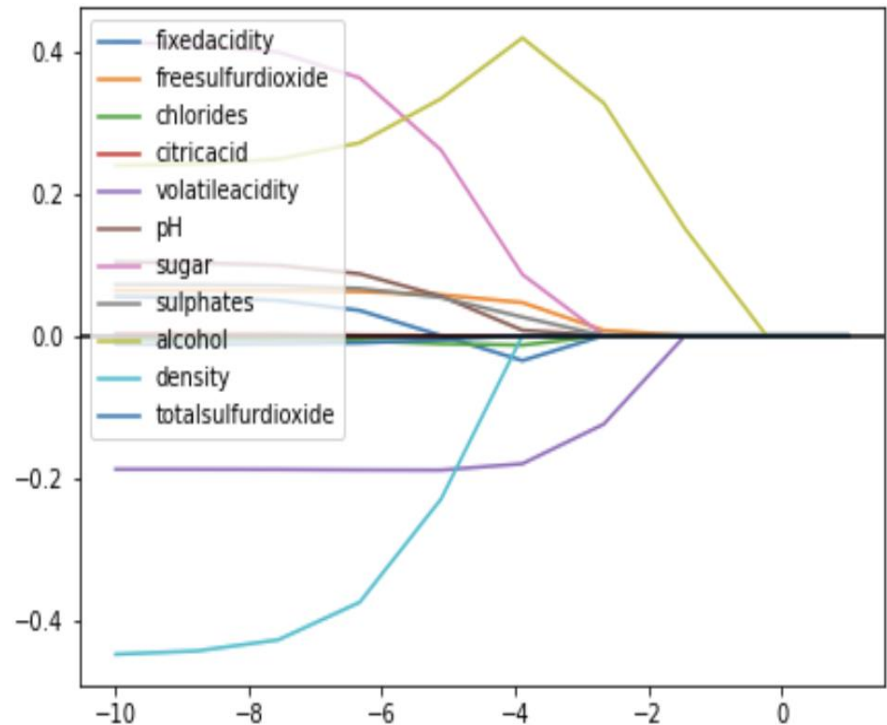
Пример LASSO

```
from sklearn.linear_model import Lasso
from sklearn.preprocessing import StandardScaler

scaler = StandardScaler()
s_X = scaler.fit_transform(X)

coefs = []
degree = np.linspace(-10, 1, 10)
alphas = np.exp(degree)
for alpha in alphas:
    lasso = Lasso(alpha=alpha)
    lasso.fit(s_X, y)
    coef = lasso.coef_.reshape(-1, 1)
    coefs.append(coef.reshape(1, -1))

plt.figure(figsize=(7, 5))
plt.plot(degree, np.vstack(coefs))
plt.legend(X.columns, loc='upper left')
plt.axhline(y=0, color='black')
plt.show()
```



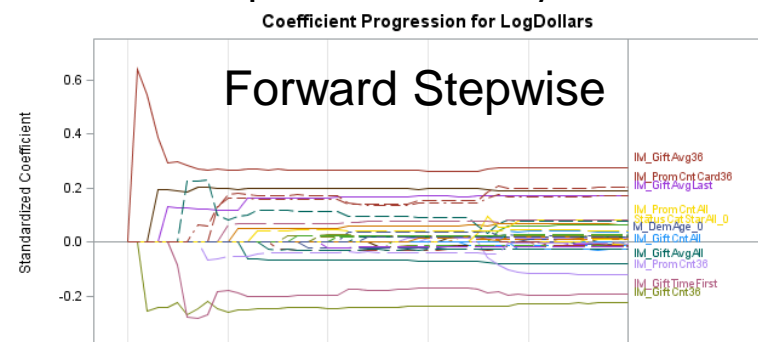
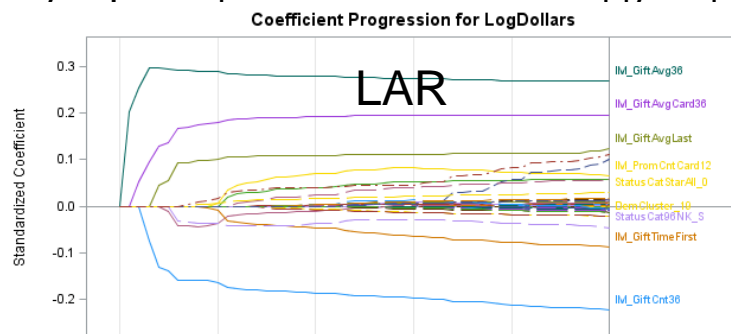
Оценки коэффициентов в зависимости от параметра регуляризации.

Поиск решения и выбор параметра регуляризации

- Параметр регуляризации для регрессий с L_1 и L_2 можно выбирать **по сетке перекрестной проверки** или на основе *информационных критериев* AIC или BIC
- Для линейной регрессии с L_2 есть точное решение в *матричном виде* через разложения
- Для линейной регрессии с L_1 :
 - точного решение в матричном виде нет, задача сводится к задаче условной оптимизации (квадратичного программирования), но при наличии корреляций признаков решение не единственное
 - *ElasticNet* технически сводится к той же постановке задачи, что и L_1
 - *используются численные методы* (часто - покоординатный спуск, он сходится быстрее чем стандартный градиентный и требует меньше вычислений чем стандартный ньютоновский)
 - Относительно недавно удалось найти новый эффективный метод одновременно для поиска решения L_1 и перебора параметра регуляризации на основе регрессии наименьшего угла – **без сетки**

Метод наименьшего угла (least angle regression)

- *Прямой пошаговый* метод с *заглядыванием* на шаг вперед - на каждом шаге выбирается предиктор-кандидат на следующий шаг
- *Нежадный* пересчет коэффициентов - на каждом шаге, не полностью минимизируется целевая функция, остается часть вариации неописанной, так, чтобы ее можно было описать при добавлении следующего предиктора,
- Коэффициенты уже добавленных переменных не пересчитываются заново на каждом шаге, а меняются *пропорционально* (биссектрисе угла между ними), что не дает неограниченно расти коэффициентам при коррелированных предикторах
- После добавления всех предикторов модель эквивалента МНК
- Модели на каждом шаге позволяют найти ограничение на параметр регуляризации LASSO и на следующих шагах это ограничение не уменьшается



Метод наименьшего угла (алгоритм)

- Шаг 1. Стандартизация данных (включая отклик) :

$$\sum_i y_i = 0, \sum_i x_{ij} = 0, \sum_i x_{ij}^2 = 1, w_j = 0, i = \overline{1, l}, j = \overline{1, p}$$

- Шаг 2. Находим x_{j_1} наиболее коррелирующий с y :

$$j_1 = \operatorname{argmax}_j |\langle x_j, y \rangle|$$

- Шаг 3. Делаем максимальный шаг (находим коэффициент) в направлении x_{j_1} , пока не найдется другой x_{j_2} - кандидат, имеющий такую же корреляцию с еще не описанным остатком
- Пока не добавим все p предикторов повторяем Шаг (k):
Добавляем кандидата x_{j_k} и ищем следующее направление пересчета коэффициентов – биссектриса угла между всеми добавленными $\{x_{j_1}, x_{j_2}, \dots, x_{j_k}\}$, пока не найдется $x_{j_{k+1}}$, также коррелирующий с остатками модели от $\{x_{j_1}, x_{j_2}, \dots, x_{j_k}\}$

Метод наименьшего угла (основной шаг)

- k - номер итерации,

r_k - регрессионные остатки ($r_0 = y$)

E_k - матрица единичных векторов координат уже добавленных переменных

$Z_k = XE_k$ - ограниченная матрица данных (по добавленным переменным)

$H_k = Z_k(Z_k^T Z_k)^{-1}Z_k^T$ аналогично ограниченная матрица проекции

- направление пересчета $u_k = H_k r_{k-1}$,

- пересчет коэффициентов $w^{(k)} = w^{(k-1)} + \mu u_k$

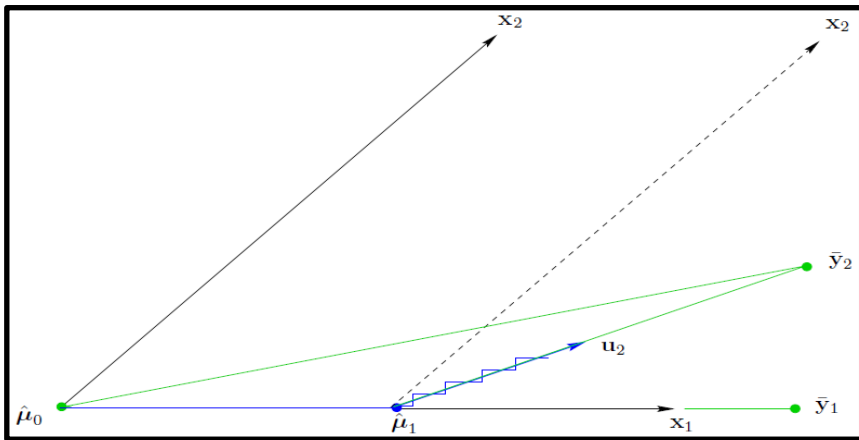
- для еще не выбранного x_j находим:

$$\mu_{k,j}^+ = \frac{\langle r_{k-1}, x_{jk} \rangle - \langle r_{k-1}, x_j \rangle}{\langle r_{k-1}, x_{jk} \rangle - \langle Xu_k, H_k x_j \rangle}, \mu_{k,j}^- = \frac{\langle r_{k-1}, x_{jk} \rangle + \langle r_{k-1}, x_j \rangle}{\langle r_{k-1}, x_{jk} \rangle + \langle Xu_k, H_k x_j \rangle}$$

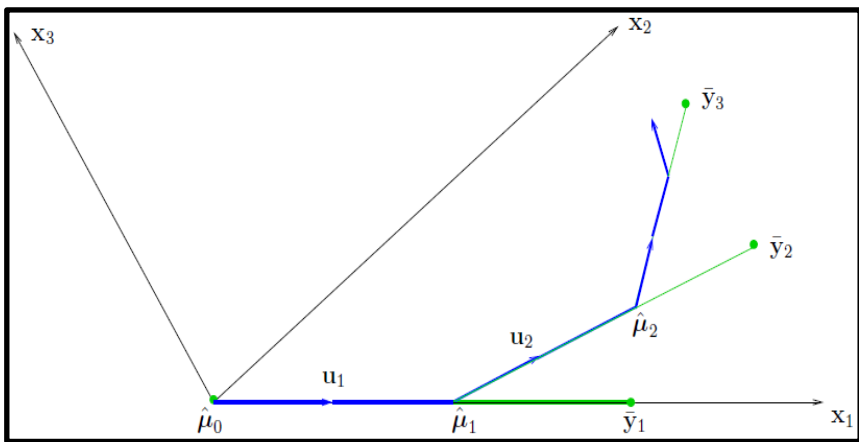
- выбираем такой x_j и $\mu_k = \min_j \{\mu \in [0,1]: (\mu = \mu_{k,j}^+) \vee (\mu = \mu_{k,j}^-)\}$

- пересчет остатков: $\langle r_k(\mu_k), x_j \rangle = \langle r_{k-1}, x_j \rangle - \mu_k \langle r_{k-1}, H_k x_j \rangle$

2D и 3D примеры



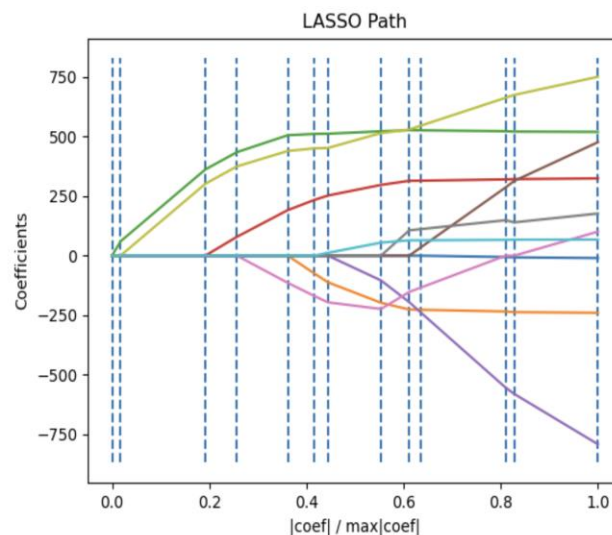
- j – номер шага
- x_j - переменные
- u_j - направление пересчета коэффициентов
- μ_j - длина шага
- \bar{y}_j - проекция отклика (с учетом текущей проекционной матрицы)



Поиск решения для LASSO на основе LAR

- при переборе кандидатов на добавление и расчете длины шага легко учесть условие $\sum |w^{(k)}| \leq C$
- процесс добавления переменных на каждом шаге k можно рассматривать при условии $\sum |w^{(k)}| \leq C_k$, и получить невозрастающую последовательность параметров регуляризации $\{C_0 = \infty, C_1, C_2, \dots, C_k, \dots, C_p = 0 | C_k \geq C_{k+1}\}$

Таким образом мы сразу получаем все варианты для процедуры перекрёстной проверки или расчета информационных критериев, не нужно делать свою сетку



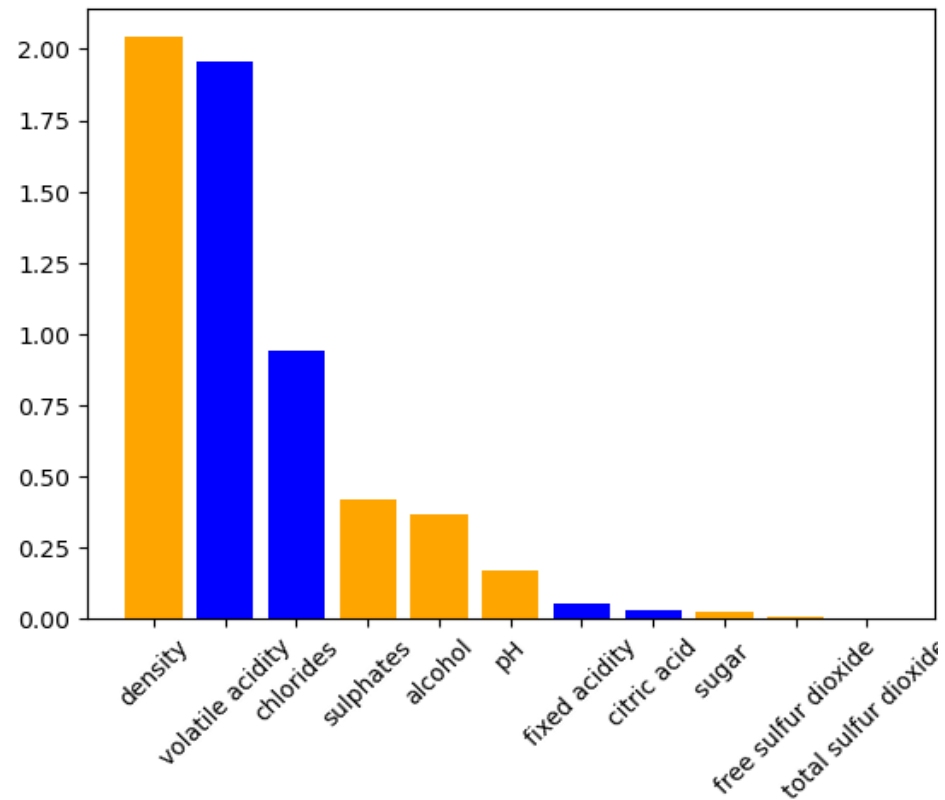
Пример LAR

```
from sklearn.linear_model import LassoLarsIC

lars = LassoLarsIC(criterion="aic",
                  normalize=True,
                  fit_intercept=False)

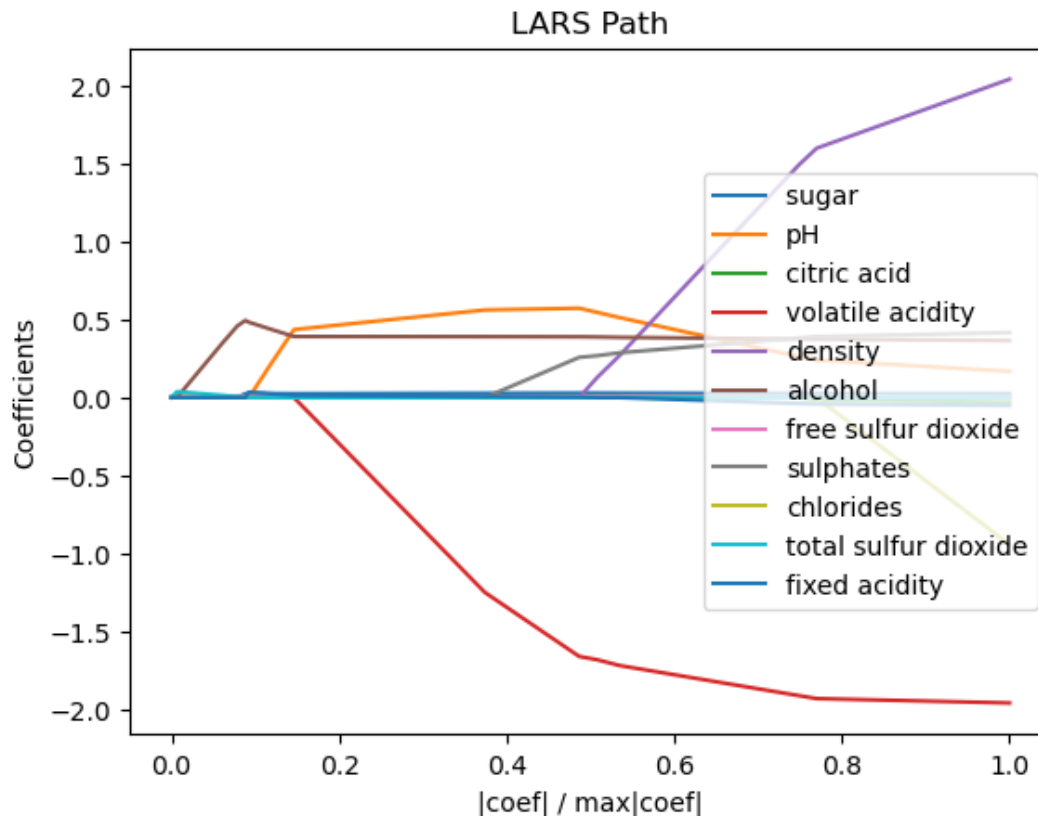
lars.fit(X, y)
```

```
coef = lars.coef_
colors = {True:"orange", False:"blue"}
sign = pd.Series(coef > 0).map(colors)
coef = abs(coef)
ind = np.argsort(coef)[::-1]
plt.bar(lars.feature_names_in_[ind],
        coef[ind], color=sign[ind])
plt.xticks(rotation=45)
plt.show()
```



Пример трассы коэффициентов регрессии наименьшего угла

```
from sklearn import linear_model
alphas, _, coefs = linear_model.lars_path(
    X.to_numpy(), y.to_numpy(), method="lasso")
xx = np.sum(np.abs(coefs.T), axis=1)
xx /= xx[-1]
```



Регрессии на основе преобразования пространства признаков

- Рассмотренные методы (отбор, регуляризация и их комбинация как в LASSO/LARS) были связаны с построением модели линейной регрессии в *исходном пространстве* признаков
- Основные проблемы: **шум, зависимости, большая размерность** и как результат: переобучение, нестабильность,...
- Можно попробовать перейти из исходного пространства признаков X в *новое пространство* G размерности меньше исходного $m \ll p$, чтобы избавиться от основных проблем

$$X_{l \times p} = \begin{pmatrix} x_{11}, x_{12}, \dots, x_{1p} \\ \dots \\ x_{l1}, x_{l2}, \dots, x_{lp} \end{pmatrix} \Rightarrow G_{l \times m} = \begin{pmatrix} g_1(x_1) & g_2(x_1) & \dots & g_m(x_1) \\ \dots & \dots & \dots & \dots \\ g_1(x_l) & g_2(x_l) & \dots & g_m(x_l) \end{pmatrix}$$

и максимально сохранить информацию об исходном пространстве:

$$\|X - g^{-1}(G)\| \rightarrow \min_g$$

Линейное преобразования пространства признаков

- Линейная функция $g_i: X \rightarrow G$, $g_i(x) = \sum_j u_{ij}x_j$
- Матрица линейного преобразования:

$$U_{p \times m} = \begin{pmatrix} u_{11}, u_{12}, \dots, u_{1m} \\ \dots \\ u_{p1}, u_{p2}, \dots, u_{pm} \end{pmatrix}$$

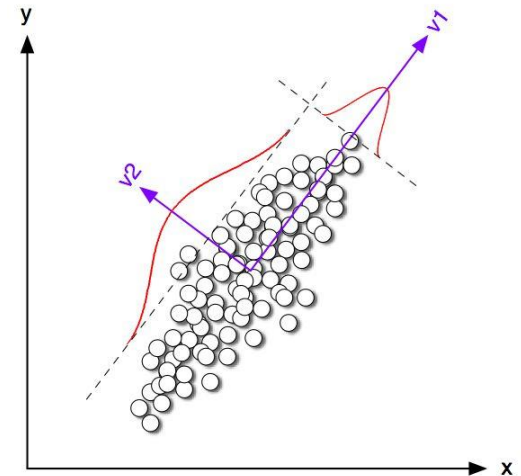
- Хотим, чтобы:
 - $X \approx GU^T \Rightarrow \|X - GU^T\| \rightarrow \min_U$
 - G – ортогональна, $GG^T = \Delta$ – диагональная матрица
- Такое преобразование существует, оно находится с помощью метода главных компонент PCA (principal component analysis)

Общая идея PCA

- Строится новый базис (линейное преобразование исходного пространства) такой, что:
 - Центр координат совпадает с мат. ожиданием наблюдений
 - Первый вектор направлен таким образом, что дисперсия вдоль него была максимальной
 - Каждый последующий вектор ортогонален предыдущим и направлен по направлению максимальной дисперсии
 - Последние компоненты – не очень важны

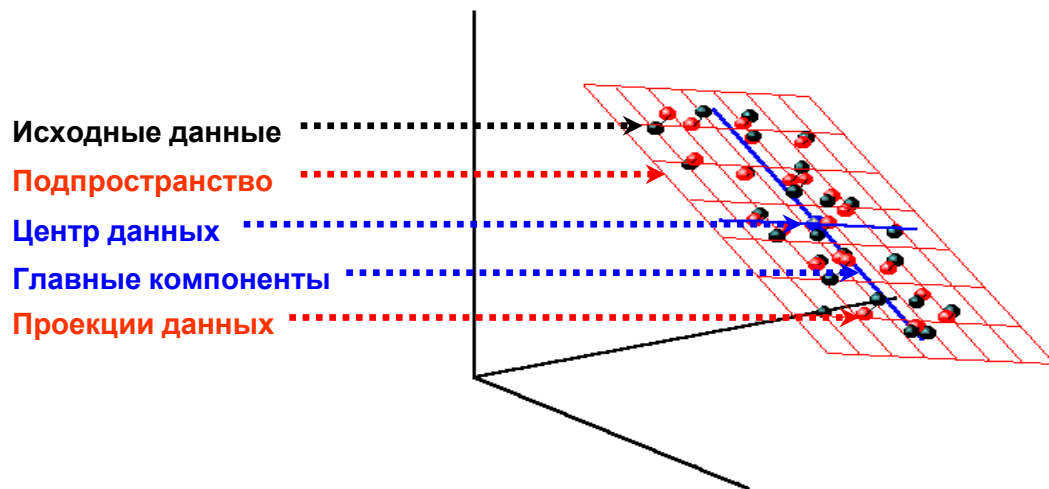
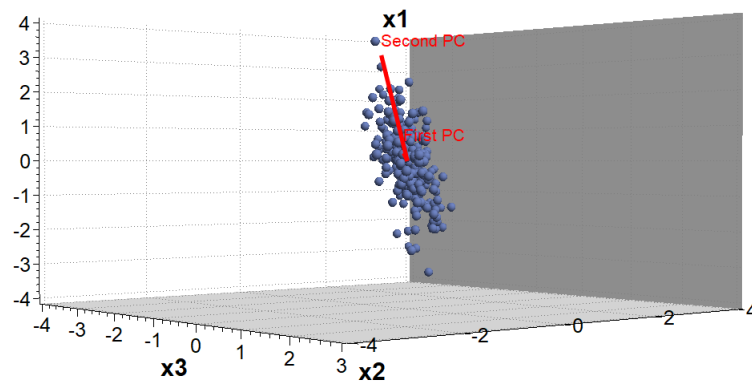
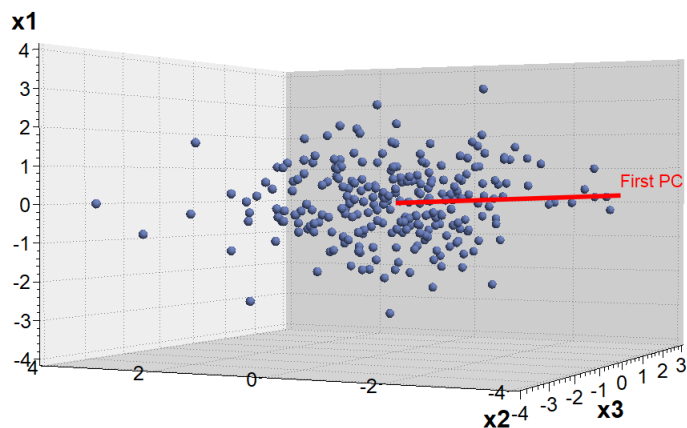
- Формально:

$$v = \underset{\|v\|=1}{\operatorname{argmax}} E \left\{ \left[v^T \left(x - \sum_{i=1}^{k-1} v_i v_i^T x \right) \right]^2 \right\}$$



- Решение - собственные значения ковариационной матрицы
- Можем найти с помощью SVD разложения

Геометрическая интерпретация



Собственные значения и вектора ковариационной (корреляционной) матрицы

- Рассчитаем ковариационную (или корреляционную) матрицу:

- $cov(x_i, x_j) = E[(x_i - E(x_i))(x_j - E(x_j))]$

- Ковариация = 0 – независимы

- Ковариация > 0 – вместе растут и убывают

- Ковариация < 0 – «противофаза»

$$C_p = \begin{pmatrix} cov(x_1, x_1) & \cdots & cov(x_1, x_p) \\ \vdots & \ddots & \vdots \\ cov(x_p, x_1) & \cdots & cov(x_m, x_p) \end{pmatrix}$$

- Проблема с.зн.:

$$Cv = \lambda v$$

решение: поиск корней $\det(C - \lambda I) = 0$, матрица положительно определенная – есть вещественные корни

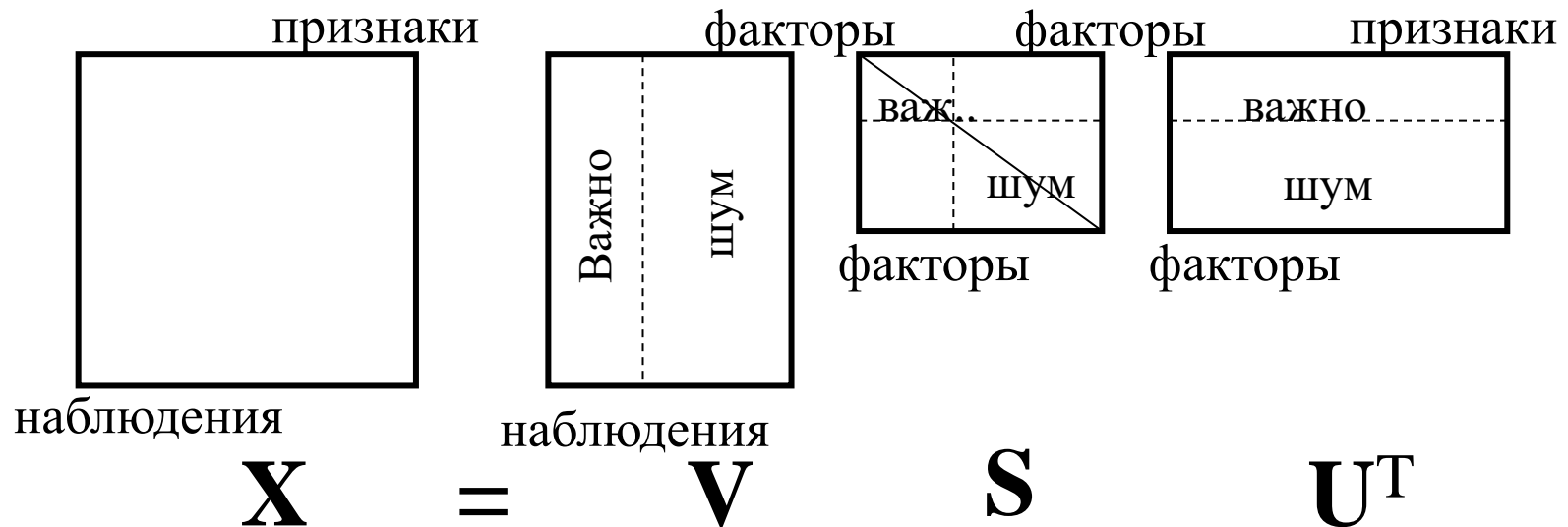
- Результат:

- λ – дисперсии, с.в. – главные компоненты

$$g_1 = u_{11}x_1 + \cdots + u_{1p}x_p, \dots, g_m = u_{m1}x_1 + \cdots + u_{mp}x_p, \forall i \sum_{j=1}^p u_{ij}^2 = 1$$

Сингулярное разложение в PCA

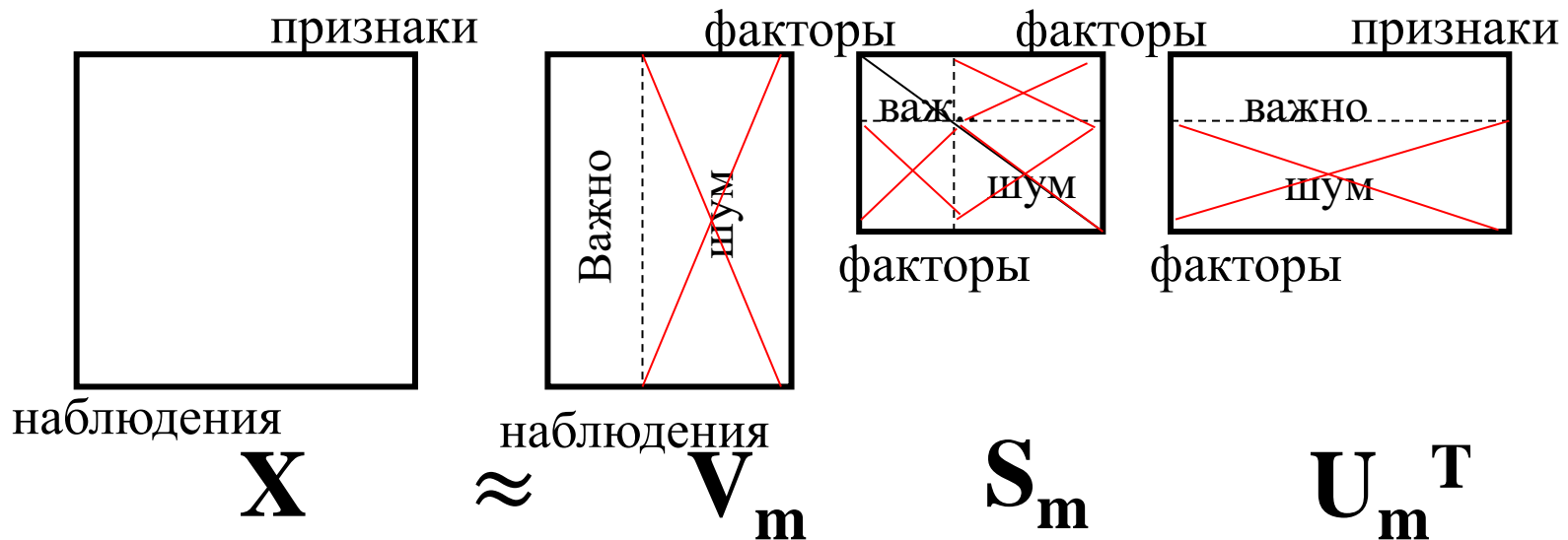
- Рассмотрим SVD, но отбросим наименее значимые гл. компоненты:



- Число факторов m = число признаков p
- Орт. матрица $U^T U = I$ правых сингулярных векторов, с.в. $X^T X$
- Орт. матрица $V V^T = I$ левых сингулярных векторов, с.в. $X X^T$
- $S = \text{diag}(\sqrt{\lambda_1}, \dots, \sqrt{\lambda_p})$ – с.зн. $X X^T$ и $X^T X$

Сингулярное разложение в PCA

- Рассмотрим SVD, но отбросим наименее значимые гл. компоненты:



- Число факторов $m \ll$ числа признаков p
- Орт. матрица $U_m^T U_m = I$ для первых m с.в. $X^T X$
- Орт. матрица $V_m V_m^T = I$ для первых m с.в. XX^T
- $S_m = \text{diag}(\sqrt{\lambda_1}, \dots, \sqrt{\lambda_m}, \dots, \sqrt{\lambda_p})$ – первых m с.зн. XX^T и $X^T X$

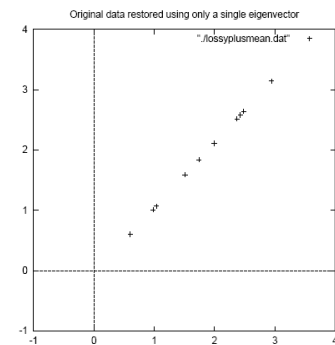
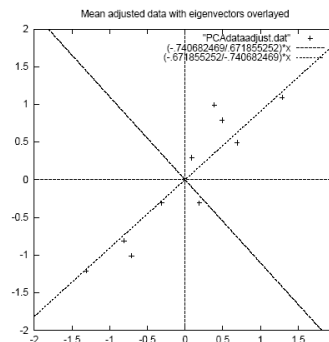
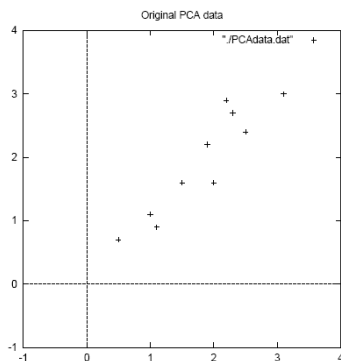
SVD разложение и обратная проекция

- SVD разложение матрицы: $X_{l \times p} = V_{l \times l} S_{l \times l} U_{p \times p}^T$
- SVD приближение (метод главных компонент):
 - отбрасываются с.в., соответствующие наименьшим с.з.
 - остается m -я часть главных с.в., которые характеризуют основные зависимости:

$$\min_{V, S, U} \|X_{l \times p} - V_{l \times m} S_{m \times m} U_{m \times p}^T\|$$

- можем построить приближенную проекцию исходной матрицы:

$$\tilde{X} = U U^T X$$



Робастные методы главных компонент

- RPCA раскладывает исходную матрицу как сумму двух:.

$$X = L + S$$

Для PCA разложения

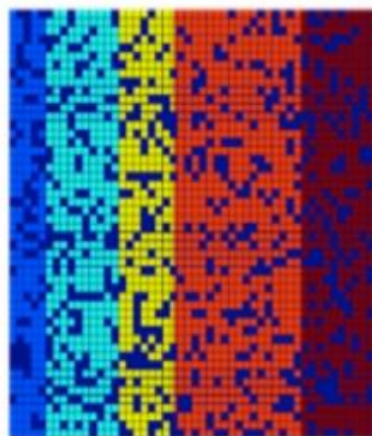
Разреженная матрица «шума»

- Разные алгоритмы решения, есть покоординатный спуск, есть через задачу условной оптимизации:

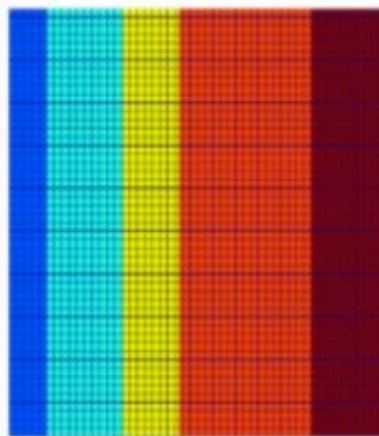
$$\min\{\|L\|_* + \gamma\|S\|_1\},$$

$$X = L + S, L = V_m S_m U_m^T, \|L\|_* = \text{tr}(S_m)$$

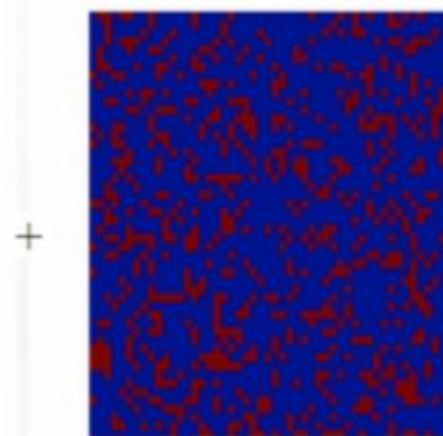
γ - параметр регуляризации, m – число компонент



Matrix of corrupted observations



Underlying low-rank matrix



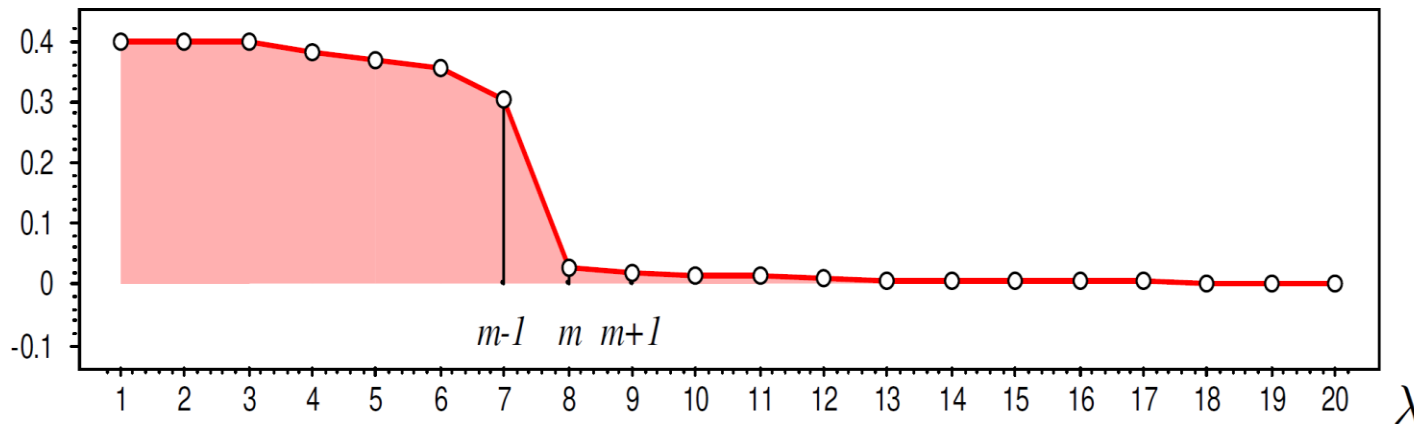
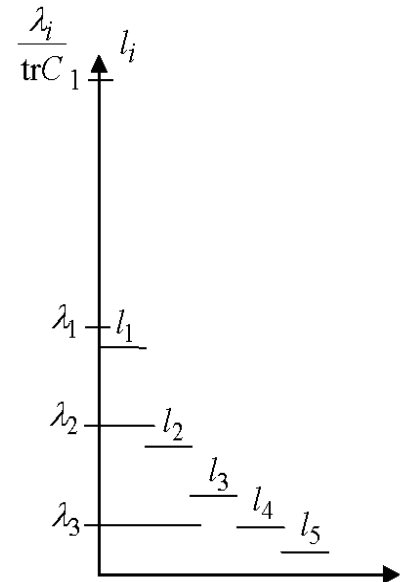
Sparse error matrix

Отбор числа главных компонент

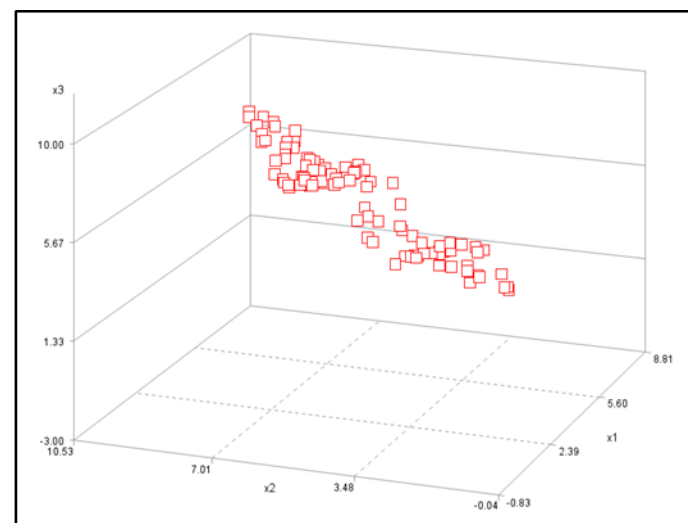
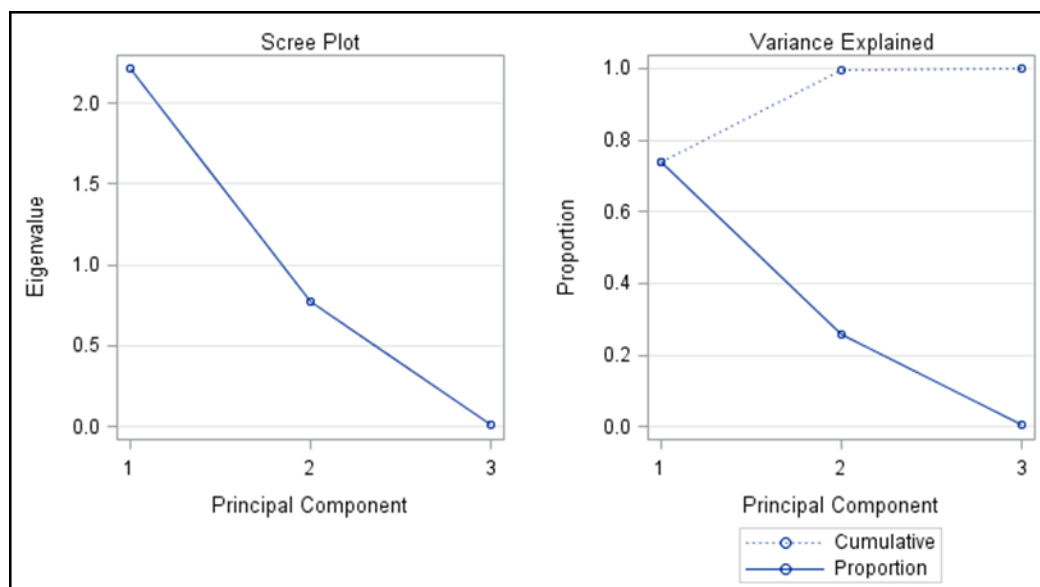
- Пропорция описанной вариации пространства признаков
- Метод Кайзера: оставляем с.в. у которых с.зн. больше среднего (или больше 1 для разложения корреляционной матрицы)
- Правило «сломанной стрости»

$$l_i = E(L_i) = \frac{1}{p} \sum_{j=i}^p \frac{1}{j}, \lambda_i \text{ оставляем если } \frac{\lambda_i}{\text{tr}(C)} > l_i$$

- По графикам – «скачки» и «плато» с.зн.



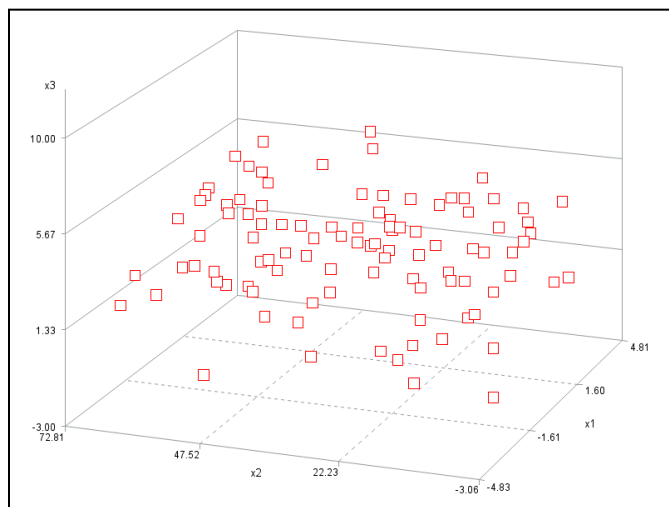
Пример 1



Eigenvalues of the Correlation Matrix				
	Eigenvalue	Difference	Proportion	Cumulative
1	2.21358154	1.44403082	0.7379	0.7379
2	0.76955072	0.75268297	0.2565	0.9944
3	0.01686775		0.0056	1.0000

Eigenvectors			
	Prin1	Prin2	Prin3
x1	0.650940	0.263685	0.711862
x2	0.645235	0.301851	-.701825
x3	-.399937	0.916164	0.026348

Пример 2

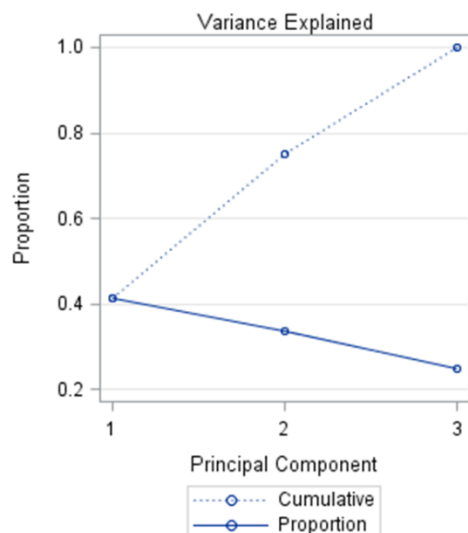
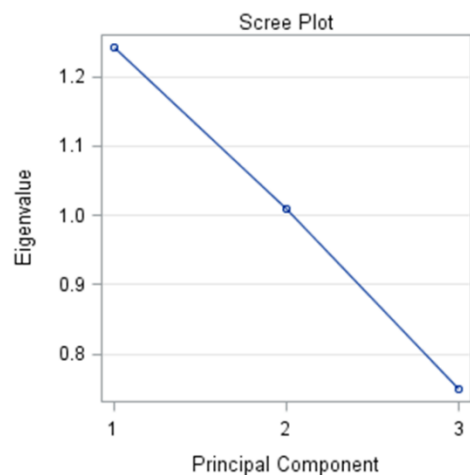


Eigenvalues of the Correlation Matrix

	Eigenvalue	Difference	Proportion	Cumulative
1	1.24205697	0.23274599	0.4140	0.4140
2	1.00931098	0.26067894	0.3364	0.7505
3	0.74863205		0.2495	1.0000

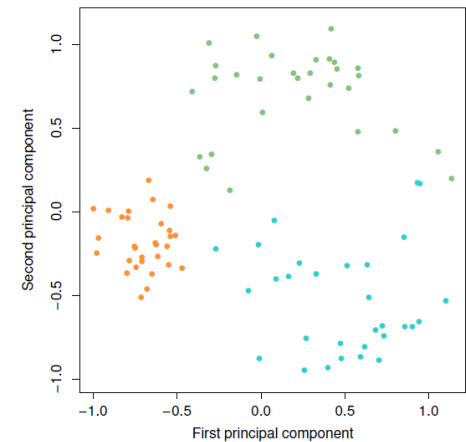
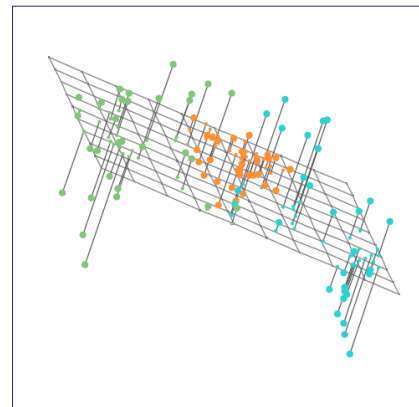
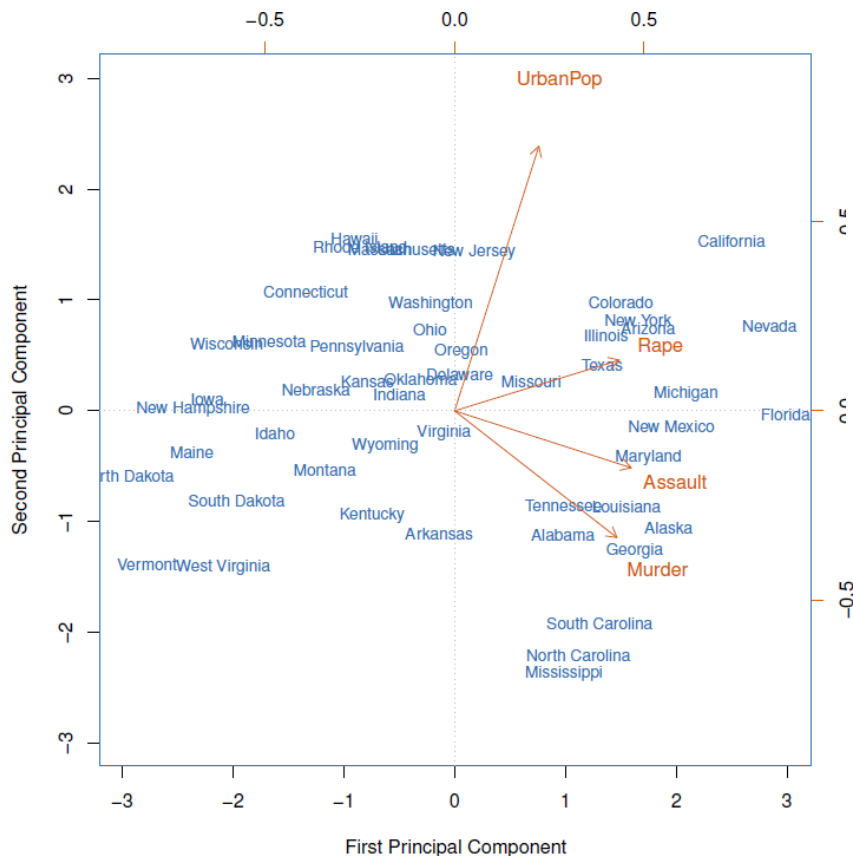
Eigenvectors

	Prin1	Prin2	Prin3
x1	0.704619	-.156546	0.692102
x2	0.708942	0.113759	-.696032
x3	0.030228	0.981097	0.191139



Визуализация

- Одно из важнейших применений – проекция множества наблюдений на пары главных компонент



	PC1	PC2
Murder	0.5358995	-0.4181809
Assault	0.5831836	-0.1879856
UrbanPop	0.2781909	0.8728062
Rape	0.5434321	0.1673186

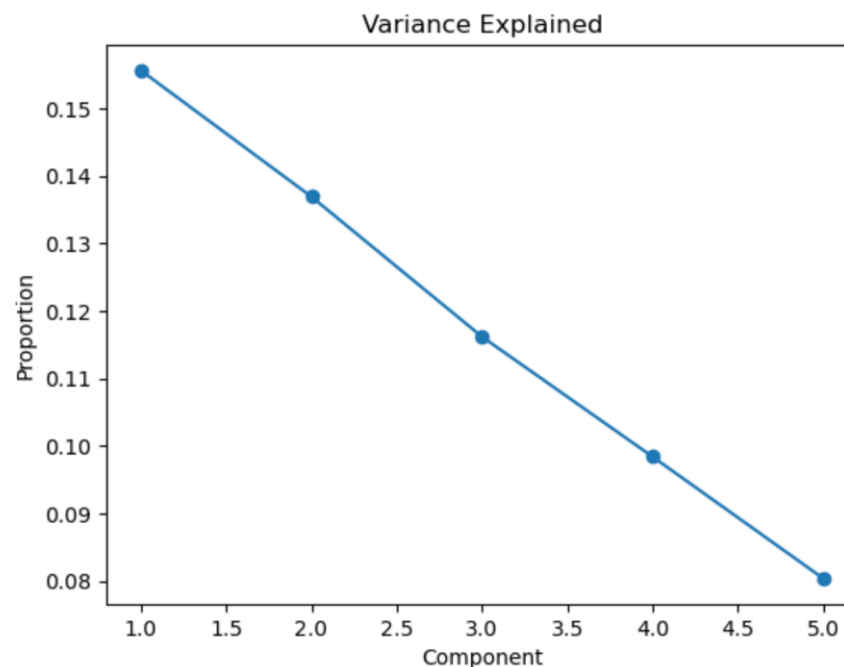
- Дана матрица клиент-продукт:

Пример использования (Python)

```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.decomposition import PCA
```

```
N = 5
pca = PCA(n_components=N)
features = pca.fit_transform(assoc)
```

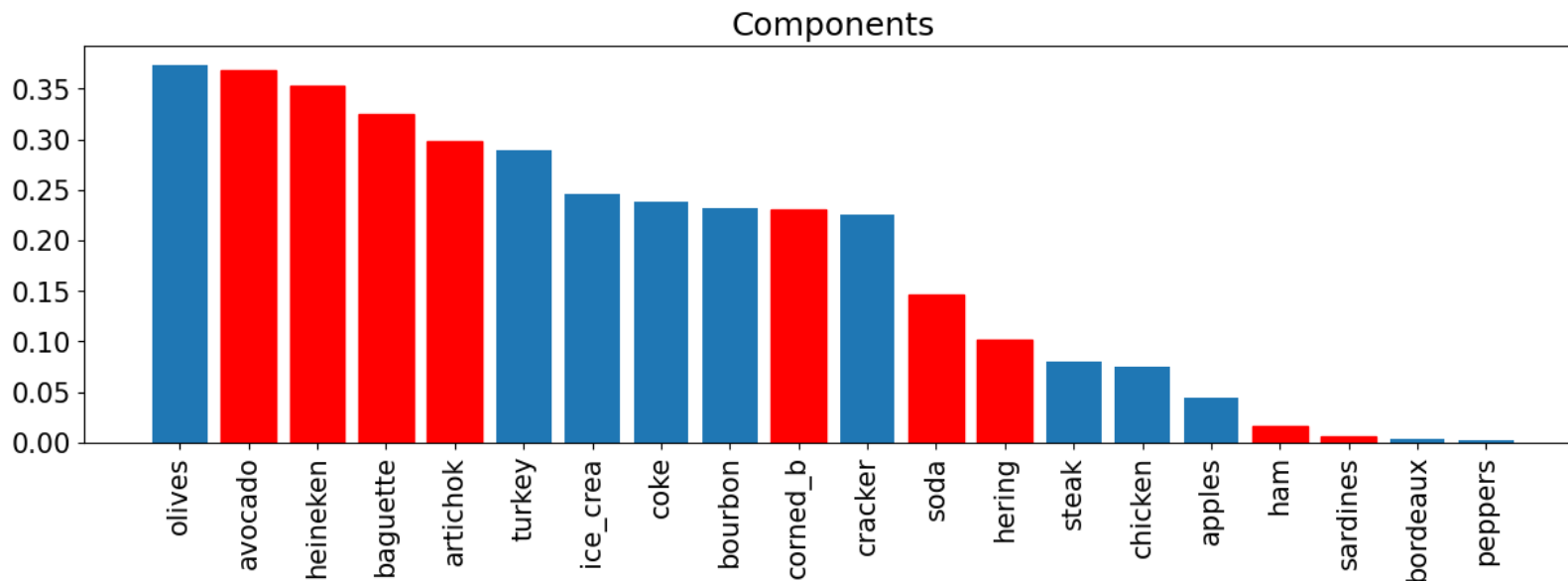
```
plt.plot(range(1, N+1), pca.explained_variance_ratio_)
plt.scatter(range(1, N+1), pca.explained_variance_ratio_)
plt.xlabel("Component")
plt.ylabel("Proportion")
plt.title("Variance Explained")
pass
```



Пример использования (Python)

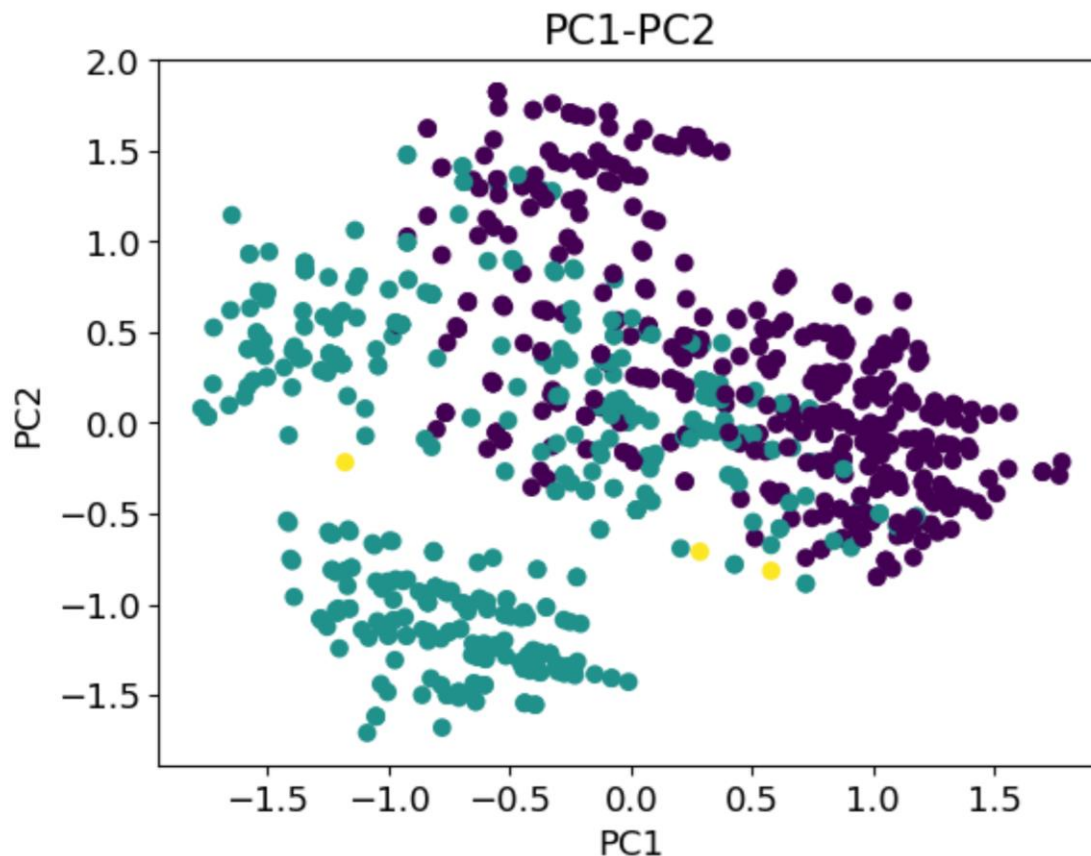
```
components = pca.components_ # [Число компонент, Число переменных]
sample = components[0] # Для примера возьмем первую
order = np.argsort(-abs(sample))
```

```
plt.xticks(rotation='vertical') # Поворачиваем текст на графике
# Нарисуем по модулю:
barplot = plt.bar(assoc.columns[order], abs(sample[order]))
# Отрицательные покрасим синими:
[x.set_color("red") for i, x in enumerate(barplot) if sample[i] > 0]
plt.title("Components")
pass
```



Пример использования (Python)

```
# Расскраска - объем покупки оливок  
plt.scatter(features[:, 0], features[:, 1], c=assoc["olives"])  
plt.title("PC1-PC2")  
plt.xlabel("PC1")  
plt.ylabel("PC2")  
pass
```



Пример использования (Python)

Новые признаки клиентов –
features (матрица V):

```
pd.DataFrame(data=features, index=assoc.index,  
             columns=[f"PC{i}" for i in range(1, N+1)])
```

	PC1	PC2	PC3	PC4	PC5
CUSTOMER					
0.0	-1.405232	-0.748403	-0.232557	-0.591675	0.319550
1.0	0.428760	-0.779644	-0.853852	0.144034	-0.758485
2.0	0.827363	0.242493	0.067664	-0.848863	1.020968
3.0	-1.516972	0.454502	-0.049412	-0.359889	0.325329
4.0	-0.667293	-1.491540	0.803896	-0.082407	-0.167066
...

Новые признаки продуктов –
components (матрица U):

```
pd.DataFrame(data=components, columns=assoc.columns,  
             index=[f"PC{i}" for i in range(1, N+1)])
```

PRODUCT	apples	artichok	avocado	baguette	bordeaux
PC1	-0.043935	0.298770	0.367965	0.324553	0.003791
PC2	-0.132510	-0.029870	-0.050568	-0.041311	0.001540
PC3	0.377463	0.040309	0.270928	0.188769	0.002230
PC4	0.247279	-0.259004	-0.072430	0.098635	-0.030733
PC5	-0.178475	0.313816	0.179802	-0.379810	0.009148

Веса предпочтений
(синг. числа λ) => `pca.singular_values_`
`array([26.34434604, 24.71066022, 22.76228395, 20.94970349, 18.93517467])`

■ Как узнать купит ли клиент продукт?

- Посчитать скалярное произведение $\langle u, v \rangle * \lambda$

Регрессия главных компонент

- Применяем анализ главных компонент (РСА), чтобы построить новое некоррелированное признаковое пространство меньшей размерности и строим в нем линейную регрессию МНК:

$$\min \sum_{i=1}^l \left(y_i - w_0 - \sum_{j=1}^m g_{ij} w_j \right)^2, \text{ где } g_{ij} = \underbrace{u_{j1}x_{i1} + \dots + u_{jp}x_{ip}}_{\text{Похоже на регуляризацию?}}, j = \overline{1, m}$$

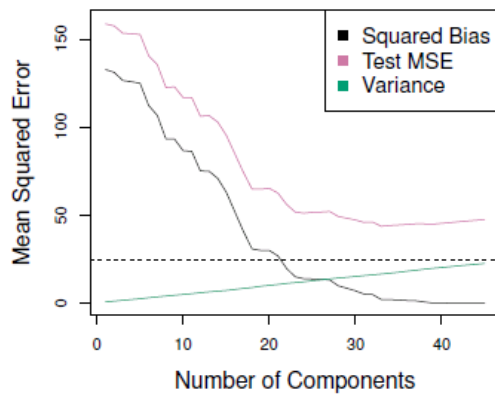
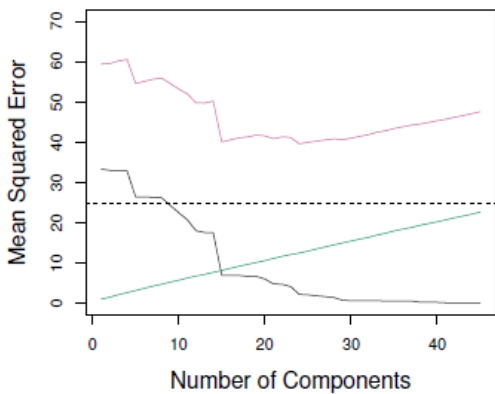
- В матричном виде:

$$\|Gw - y\|^2 \rightarrow \min, G = U^T X$$

- Можно подставить линейные равенства g_{ij} в полученную по МНК линейную регрессию $w = (G^T G)^{-1} G^T y$ и понять, что:
 - Полученная регрессия есть в том числе и линейная регрессия от исходных признаков, с коэффициентами $w_x = Uw$
 - Можно рассматривать регрессию главных компонент как одну из форм регуляризации, где ограничения на параметры модели заданы не через L_p , а равенствами через матрицу U

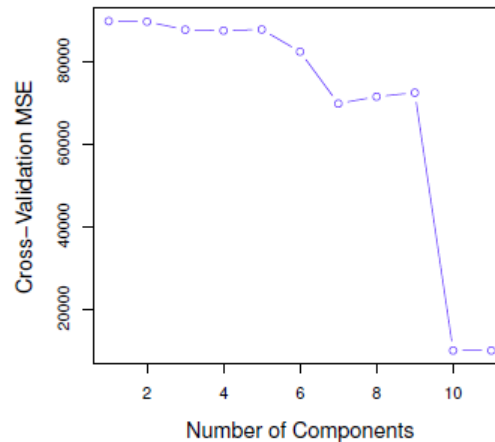
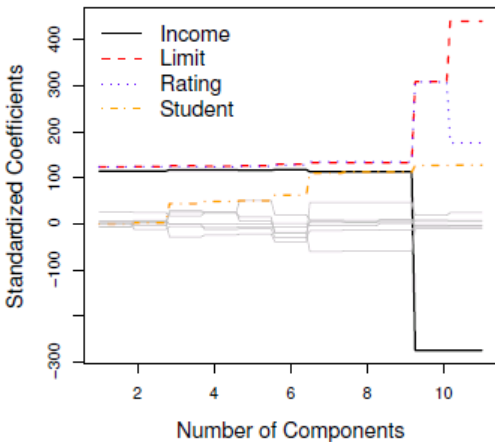
О выборе числа главных компонент

- MSE декомпозиция зависит от числа компонент:



MSE на тестовой выборке
Дисперсия модели
Смещение модели

- Трассы коэффициентов



Применение регрессии главных компонент

```
from sklearn.decomposition import PCA
from sklearn.pipeline import make_pipeline
from sklearn.model_selection import cross_val_predict
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error

PCR = make_pipeline(PCA(n_components=5), LinearRegression())
y_cv = cross_val_predict(PCR, X, y, cv=10)
mean_squared_error(y, y_cv)
```

0.6193829630825505

Поиск количества компонент

```
def optimise_comp_cv(X, y, n_comp):|
    PCR = make_pipeline(PCA(n_components=n_comp),
                        LinearRegression())
    # Расчет ошибки на кросс-валидации
    y_cv = cross_val_predict(PCR, X, y, cv=10)
    mse = mean_squared_error(y, y_cv)
    var = explained_variance_score(y, y_cv)
    bias = mse - var
    return mse, bias, var
```

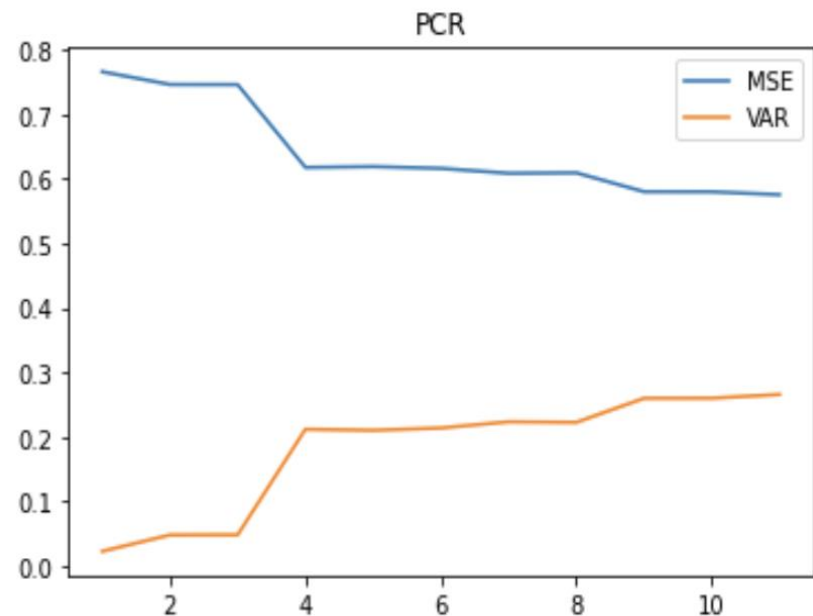
Выбор количества компонент M

```
from sklearn.metrics import mean_squared_error
from sklearn.metrics import explained_variance_score

def optimise_comp_cv(X, y, n_comp):
    PCR = make_pipeline(PCA(n_components=n_comp),
                        LinearRegression())
    # Расчет ошибки на кросс-валидации
    y_cv = cross_val_predict(PCR, X, y, cv=10)
    mse = mean_squared_error(y, y_cv)
    var = explained_variance_score(y, y_cv)
    return mse, var

mse_, var_ = [], []
list_components = list(range(1, X.shape[1]+1))
for n_comp in list_components:
    mse, var = optimise_comp_cv(X, y, n_comp)
    mse_.append(mse)
    var_.append(var)

plt.plot(list_components, np.array(mse_))
plt.plot(list_components, np.array(var_))
plt.xlabel('Количество компонент PCR')
plt.legend(["MSE", "VAR"])
plt.title('PCR')
plt.show()
```

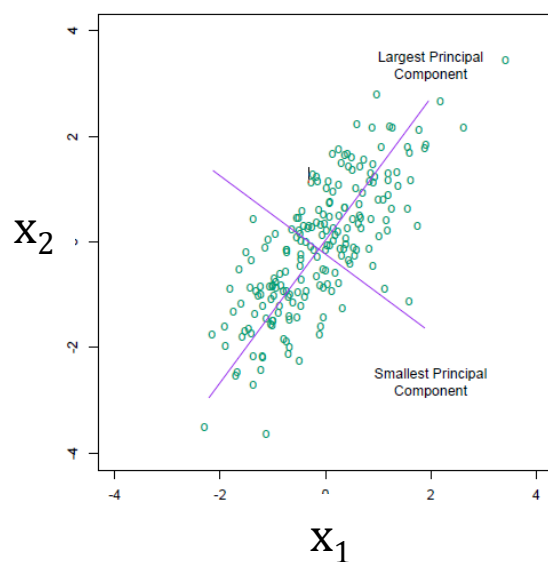


Метод частичных наименьших квадратов (PLS)

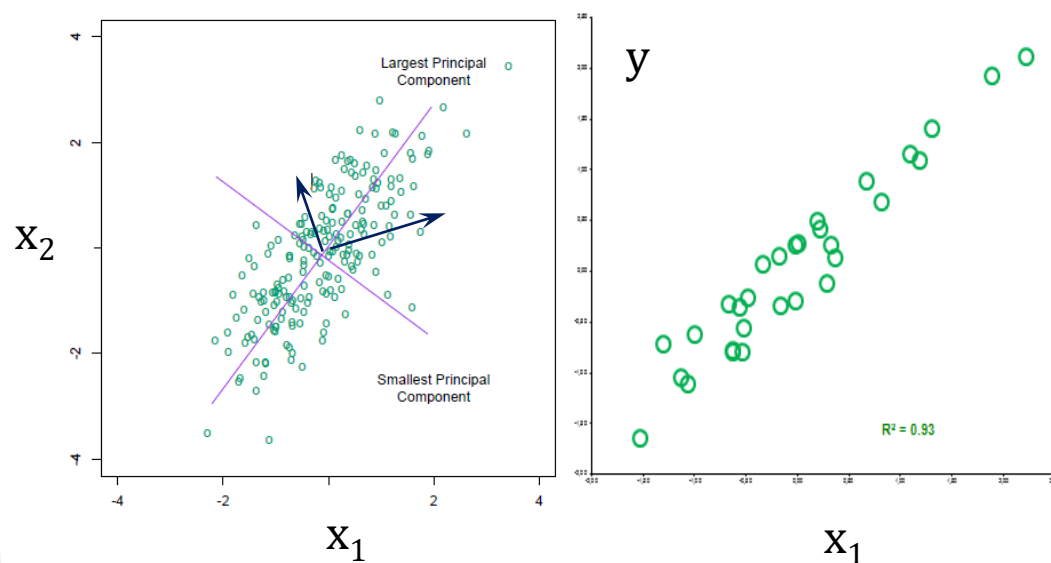
- PCR определяет линейные комбинации, или *направления*, которые наилучшим образом представляют предикторы
- Эти направления определяются *обучением без учителя*, так как отклик не используется при определении направлений главных компонент.
- Следовательно, PCR страдает от потенциально **серьезного недостатка**: нет никакой гарантии, что направления, которые наилучшим образом объясняют предикторы, также будут лучшими направлениями при использовании для прогнозирования отклика.
- PLS работает аналогично PCR, но определяет новый сокращенный набор ортогональных признаков с *учетом отклика*:

$$\max_{||z||=1} \mathbf{Corr}^2(y, Xz) \mathbf{Var}(Xz)$$

Метод частичных наименьших квадратов (PLS)



Метод главных компонент



Метод наименьших частичных
квадратов

Метод частичных наименьших квадратов (алгоритм SIMPLS)

- Стандартизируем X и Y
- Повторяем итерации $i = \overline{1, p}$
 - $c_{ij} = \langle x_j^{(i-1)}, y^{(i-1)} \rangle$ - расчет корреляций
 - $z_i = \sum_{j=1}^p c_{ij} x_j^{(i-1)}$ - расчет главной компоненты
 - $t_i = \frac{\langle z_i, y \rangle}{\langle z_i, z_i \rangle}$ - расчет проекции отклика на гл. компоненту
 - $y^{(i)} = y^{(i-1)} + t_i z_i$ - пересчет отклика
 - $x_j^{(i)} = x_j^{(i-1)} - z_i \frac{\langle z_i, x_j^{(i-1)} \rangle}{\langle z_i, z_i \rangle}$ - проекция данных

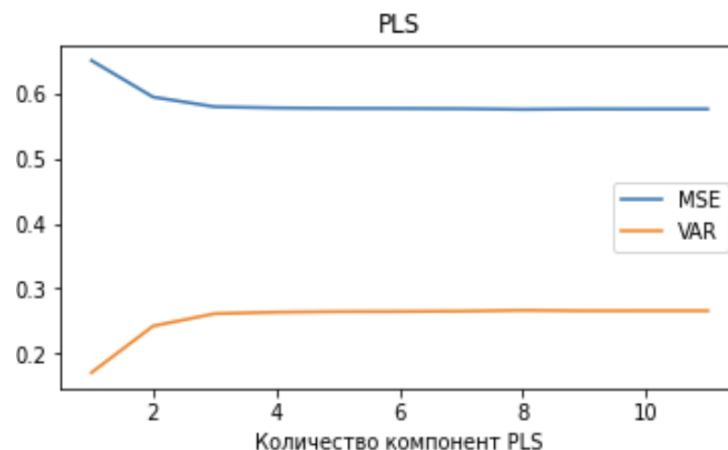
Применение PLS

```
from sklearn.cross_decomposition import PLSRegression
```

```
def optimise_comp_cv(X, y, n_comp):  
    pls = PLSRegression(n_components=n_comp)  
    # Расчет ошибки на кросс-валидации  
    y_cv = cross_val_predict(pls, X, y, cv=10)  
    mse = mean_squared_error(y, y_cv)  
    var = explained_variance_score(y, y_cv)  
    return mse, var
```

```
mse_, var_ = [], []  
list_components = list(range(1, X.shape[1]+1))  
for n_comp in list_components:  
    mse, var = optimise_comp_cv(X, y, n_comp)  
    mse_.append(mse)  
    var_.append(var)
```

```
plt.figure(figsize=(6, 3))  
plt.plot(list_components, np.array(mse_))  
plt.plot(list_components, np.array(var_))  
plt.xlabel('Количество компонент PLS')  
plt.legend(["MSE", "VAR"])  
plt.title('PLS')  
plt.show()
```

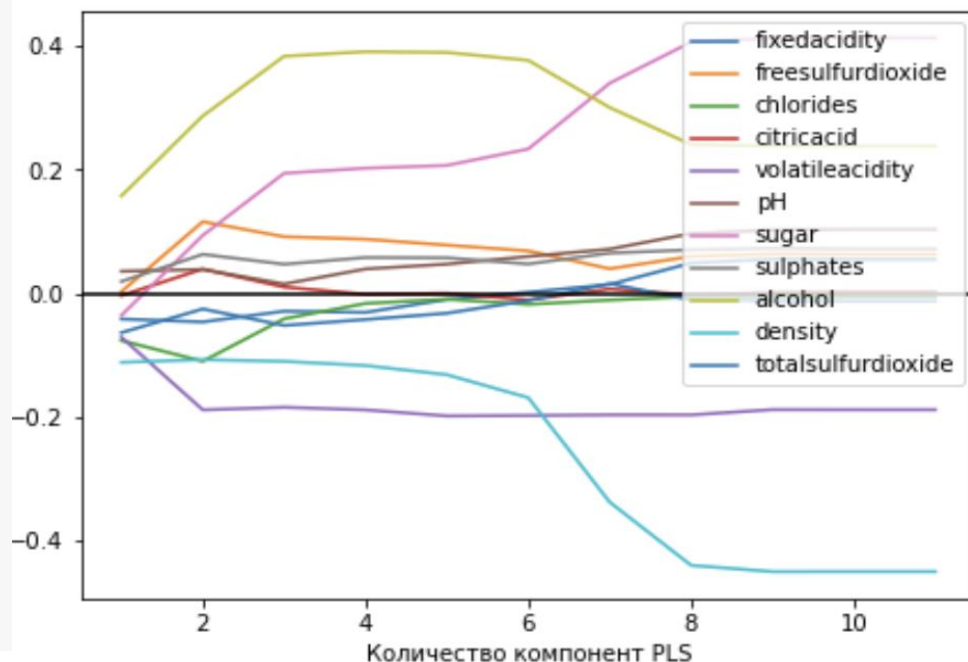


Применение PLS

```
from sklearn.preprocessing import StandardScaler

coefs = []
list_components = list(range(1, X.shape[1]+1))
for n_comp in list_components:
    pls = PLSRegression(n_components=n_comp)
    pls.fit(X, y)
    coef = pls.coef_.reshape(-1, 1)
    coefs.append(coef.reshape(1, -1))

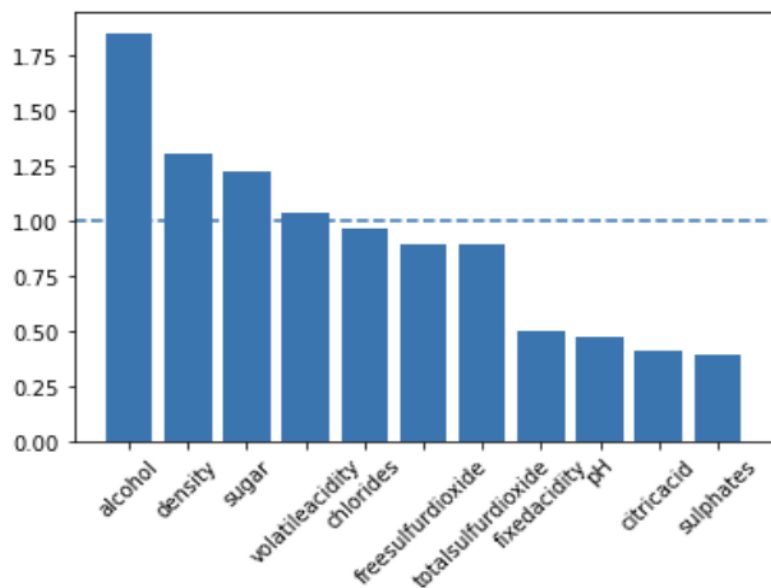
plt.figure(figsize=(7, 5))
plt.plot(list_components, np.vstack(coefs))
plt.legend(X.columns, loc='upper right')
plt.xlabel('Количество компонент PLS')
plt.axhline(y=0, color="black")
plt.show()
```



Оценка важности переменных в PLS

- VIP статистика (Variable importance in projection) оценивает важность каждого предиктора x_j с учетом значений его коэффициентов u_{jk} в проекции на каждую из k главных компонент с учетом пропорции описанной ей вариации отклика SSY_k/SSY_{total} :

$$vip^2(x_j) = \sum_k u_{jk}^2 SSY_k / SSY_{total}$$



```
def vip(model):  
    t = model.x_scores_  
    w = model.x_weights_  
    q = model.y_loadings_  
    p, h = w.shape  
    vips = np.zeros((p,))  
    s = np.diag(t.T @ t @ q.T @ q).reshape(h, -1)  
    total_s = np.sum(s)  
    for i in range(p):  
        weight = np.array([(w[i,j] / np.linalg.norm(w[:,j]))**2  
                           for j in range(h)])  
        vips[i] = np.sqrt(p*(s.T @ weight)/total_s)  
    return vips  
  
pls = PLSRegression(n_components=5)  
pls.fit(X, y)  
  
vips=vip(pls)  
ind = np.argsort(vips)[::-1]  
plt.bar(pls.feature_names_in_[ind], vips[ind])  
plt.xticks(rotation=45)  
plt.axhline(y=1, linestyle="--")  
plt.show()
```