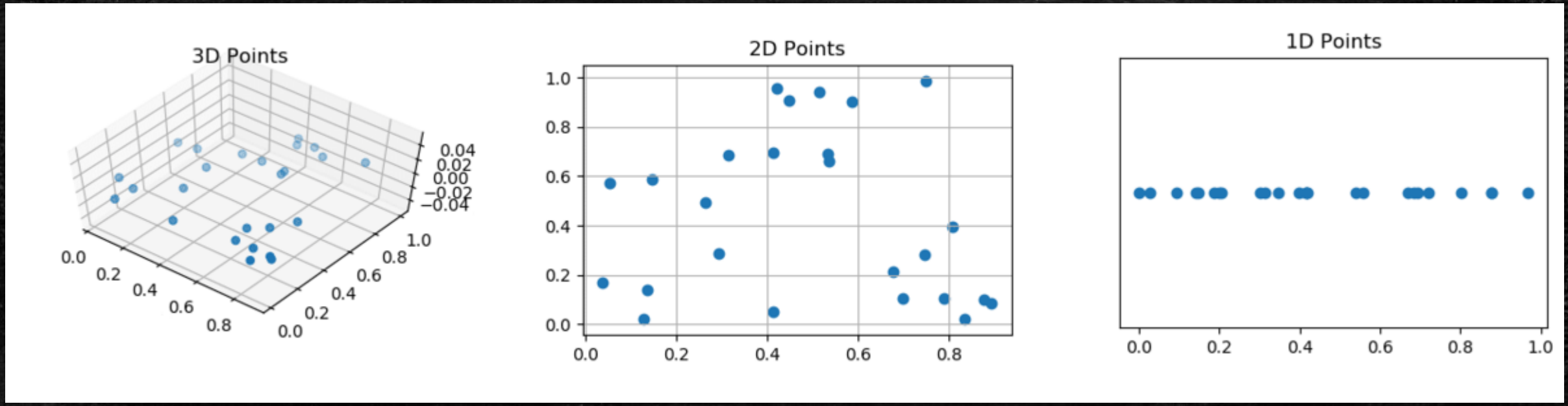
Метод главных компонент

(Principal component analysis)

Стешенко Артем Александрович
@steshenkotema

Применение

- Сокращение признаков для увеличения скорости обучения
- Визуализация
- Компрессия видео и фото



Кратко о машинном обучении

label

features

	Wine	Alcohol	Malic.acid	Ash	Acl	Mg	Phenols	Flavanoids	Nonflavanoid.phenols	Proanth	Color.int	Hue	OD	Proline
0	1	14.23	1.71	2.43	15.6	127	2.80	3.06	0.28	2.29	5.64	1.04	3.92	1065
1	1	13.20	1.78	2.14	11.2	100	2.65	2.76	0.26	1.28	4.38	1.05	3.40	1050
2	1	13.16	2.36	2.67	18.6	101	2.80	3.24	0.30	2.81	5.68	1.03	3.17	1185
3	1	14.37	1.95	2.50	16.8	113	3.85	3.49	0.24	2.18	7.80	0.86	3.45	1480
4	1	13.24	2.59	2.87	21.0	118	2.80	2.69	0.39	1.82	4.32	1.04	2.93	735
5	1	14.20	1.76	2.45	15.2	112	3.27	3.39	0.34	1.97	6.75	1.05	2.85	1450
6	1	14.39	1.87	2.45	14.6	96	2.50	2.52	0.30	1.98	5.25	1.02	3.58	1290
7	1	14.06	2.15	2.61	17.6	121	2.60	2.51	0.31	1.25	5.05	1.06	3.58	1295
8	1	14.83	1.64	2.17	14.0	97	2.80	2.98	0.29	1.98	5.20	1.08	2.85	1045
9	1	13.86	1.35	2.27	16.0	98	2.98	3.15	0.22	1.85	7.22	1.01	3.55	1045

objects

$Y = f(X) + Error : Error \to min$

X – матрица objects × features

В машинном обучении часто встречается задача уменьшения размерности признакового пространства

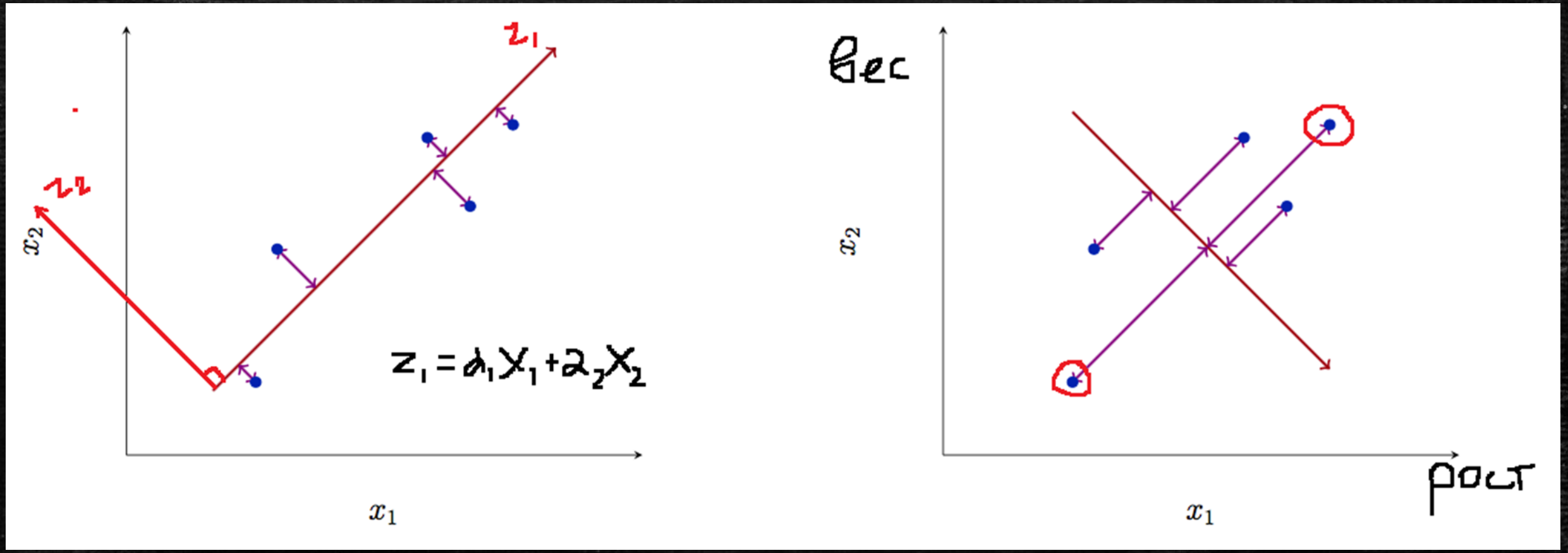
Допустим $X \in R^{n*l}$,где l – количество признаков(столбцы), n – количество объектов(строки), x_i – объект

Пусть данные у нас центрированные(то есть среднее в каждом столбце равно 0)

Задача: необходимо получить матрицу $Z \in R^{n*d}$, такую чтобы потерять как можно меньше информации из матрицы X

Геометрический смысл

Z_i – главная компонента, линейная комбинация $x_1, \ldots, x_n$. Подберем такие параметры $a_1, \ldots, a_n$, чтобы дисперсия выборки была максимальной, так как дисперсия показывает, то как хорошо мы преобразовали информацию, на картинке показаны правильный и неправильный выбор параметров



Будем искать главные компоненты $u_1, u_2, \ldots, u_d$. Они будут задавать новое векторное пространство

1) $(u_i, u_j) = 0$
2) $\|u_i\|^2 = 1$
3) $D_{U_i} \to \max$

Дисперсия проецированной выборки показывает, как много информации нам удалось сохранить после понижения размерности

Проекция объекта на компоненту: (x_i, u_i) (такая формула, потому что главные компоненты нормированы $\|u_i\|^2 = 1$,), проекция всех объектов: $X * u_i$ -

Проекция всех объектов на все компоненты: $X * U_d$

Посчитаем дисперсию проецированной выборки. Мы можем записать ковариационную матрицу. И числа по диагонали будут дисперсиями проекции X на все u_i. То есть след ковариационной матрицы - дисперсия

Ковариационная матрица многомерной случайной величины X, это

$$\Sigma = [\sigma_{ij}],\ \sigma_{ij} = \operatorname{cov}(X_i, X_j) = \mathrm{E}[(X_i - \mathrm{E}[X_i])(X_j - \mathrm{E}[X_j])].$$

X*ui – случайный вектор, где xi, объект, - это случайный вектор, X – множество случайных векторов

Элемент ковариационной матрицы:
$\mathrm{cov}(X*u_i, X*u_j) = E(u_i^T*X^T*X*u_j) - E(X*u_i) - E(X*u_j) = E(u_i^T*X^T*X*u_j)$
$E(X*u_i) = 0,\ E(X*u_j) = 0$, так как X – центрированная матрица

Ковариационная матрица U^TX^TXU

На диагонали ковариационной матрицы находятся дисперсии
$tr(U^TX^TXU) = \Sigma||X*u_i||^2$, **где** $||X*u_i||^2 = u_i^T*X^T*X*u_i$ – норма матрицы из одного элемента – это сам элемент

$D_i = ||X*u_i||^2$

Найдем компоненту u_i. Так как нам нужно максимизировать дисперсию Di при условии $||u_i||^2 = 1$, то мы можем записать это как оптимизационную задачу и решить с помощью функции Лагранжа

$$\begin{cases} ||X*u_i||^2 \rightarrow \max \\ ||u_i||^2 = 1 \end{cases}$$

$L(u_i, \lambda) = ||X*u_i||^2 + \lambda*(||\ u_i\ ||^2 - 1)$ необходимо найти экстремум

$d(L(u_i, \lambda))/d(u_i) = 2*X^T*X*u_i + 2*\lambda*u_i = 0$

u_i – собственный вектор ковариационной матрицы X^T*X, где $X^T*X*u_i = \lambda*u_i$, отсюда посчитаем дисперсию $||X*\ u_i\ ||^2 = u_i^T*X^T*X*u_i = u_i^T*\lambda*u_i \rightarrow \max$

3) условие, которое требует $(u_i, u_j) = 0$ также выполняется, так как мы знаем, что собственные векторы, отвечающие различным собственным значениям, ортогональны

Главная компонента u_i равна собственному вектору, соответствующему собственному значению λi, где $\lambda 1 > \lambda 2 > \lambda 3 > \ldots > \lambda d$

U – матрица поворота, матрица главных компонент . $Z = X*U$

Матрица VSU^T – сингулярное разложение матрицы X, то мы можем представить задачу так:

$Z = XU = VSU^TU = VS = Z$

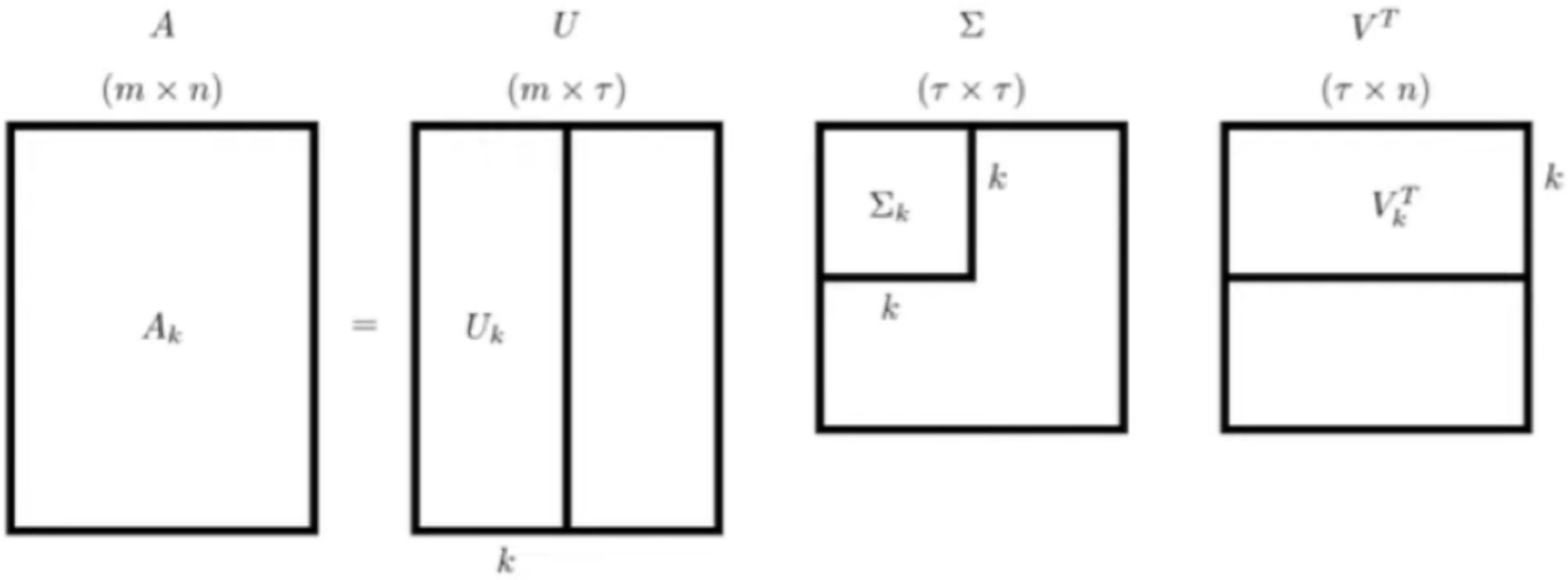
$X = VSU^T = Z^*U^T$

Теперь в зависимости о от того, какое k главных компонент мы хотим использовать, такое k сингулярных чисел нужно взять, пример на картинке следующего слайда

Связь с сингулярным разложением

$$A = U\Sigma V^T$$

$$A_k = U_k\Sigma_k V_k^T = (U_k\Sigma_k)V_k^T = U_k(\Sigma_k V_k^T)$$



По теореме Эккарта-Янга Ak – это лучшая аппроксимация матрицы A ранга k

```
import numpy as np
from sklearn.decomposition import PCA
X = np.array([[-1, -1, 1], [-2, -1, 3], [-3, 1, -2], [2, 1, 1], [1, 2, 1], [3, 3, 2]]) #матрица 6 * 3
```

```
pca = PCA(n_components=3) #приминяем метод главных компонент
Z = pca.fit_transform(X) #данные центрируются, потом применяется
                         #сингулярное разложние с k = n_components
                         # k = 3 сингулярными числами
#fit_transform создает вектора - главные компоненты
#затем проецирует данные на них
```

```
Z #если бы мы взяли, например, k=2, то Z состояла из первых двух стобцлв
```

```
array([[-1.66578638, -0.96625837,  0.8078438 ],
       [-2.00544699, -2.63099797, -0.6458662 ],
       [-3.28582985,  2.6786842 , -0.23611634],
       [ 1.79266035,  0.04886353,  0.9009767 ],
       [ 1.37266021,  0.60746729, -0.32847917],
       [ 3.79174267,  0.26224132, -0.49835879]])
```

```
pca.explained_variance_ratio_ #две главные компоненты сохранили
                              #около 96% иннформации
```

```
array([0.6760209, 0.2821519, 0.0418272])
```

```
sin = pca.singular_values_
sin #сингулярные числа
```

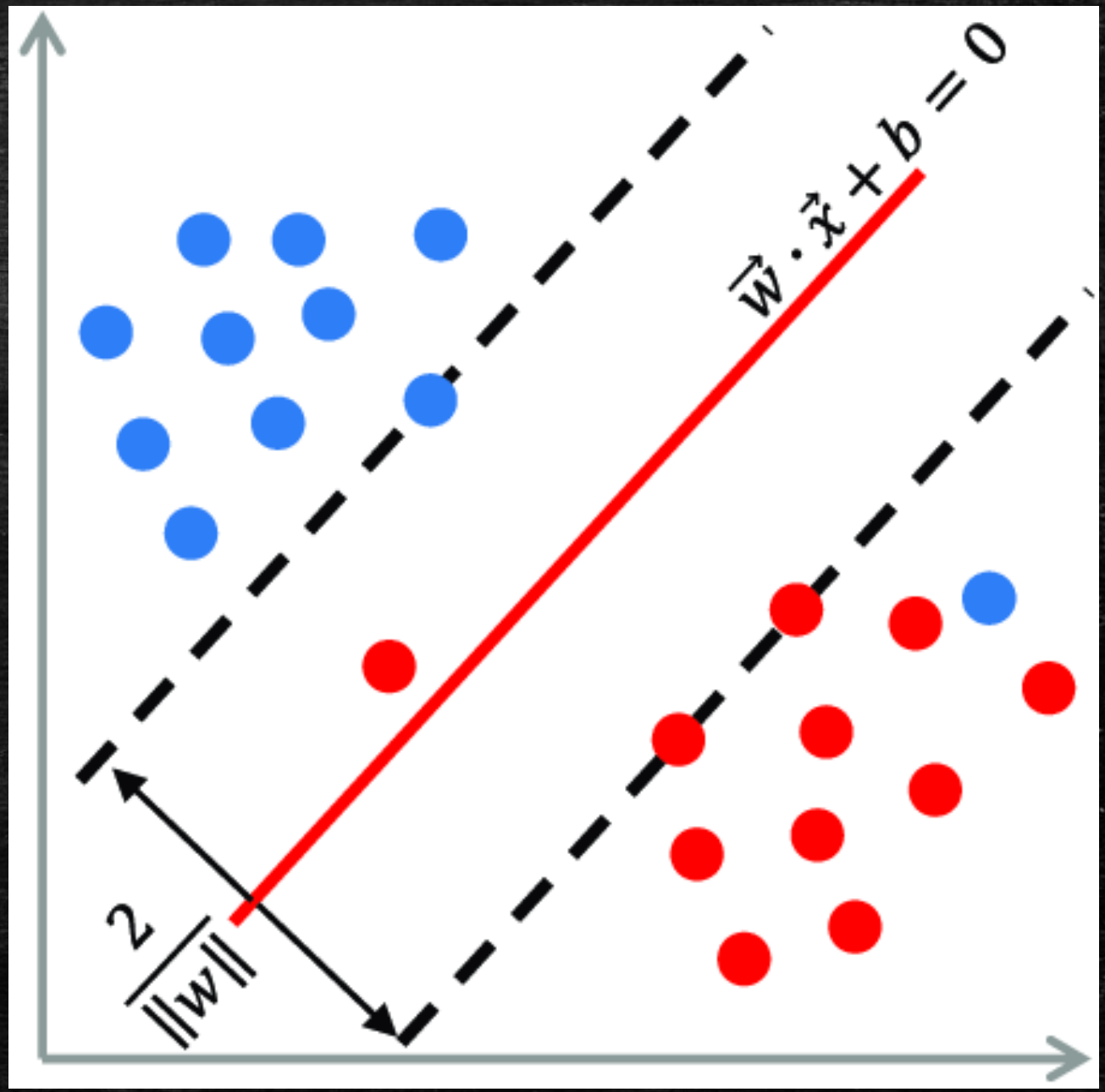
```
array([6.08838889, 3.93336107, 1.51443875])
```

Датасет Wine

- Набор данных вин содержит результаты химического анализа вин, выращенных в определенной области Италии. В 178 образцах представлены три вида вина, для каждого образца записаны результаты 13 химических анализов. Качественные переменные были преобразованы в категориальные переменные.
- Наша задача построить модель, чтобы классифицировать эти вина. Посмотрим, поможет ли PCA улучшить алгоритм

$$\frac{1}{2}\|W\|^2 + C\sum_{i=1}^{\ell}\xi_i \to \min_{W,\xi}$$

Метод опорных векторов ищет гиперплоскость wx+b=0 с такими параметрами w и b , чтобы максимизировать 2/||w|| и минимизировать сумму ошибок



```
from sklearn.svm import LinearSVC
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score
X_train, X_test, y_train, y_test = train_test_split(X, y, random_state=9)
```

```
%%time
pca = PCA(n_components=6) #приминяем метод главных компонент
pca.fit_transform(X)
X_train_pca = pca.transform(X_train) #получаем матрицу с новыми признаками
X_test_pca = pca.transform(X_test)

clf = LinearSVC(random_state=17).fit(X_train_pca, y_train) #обучаем модель
y_pred = clf.predict(X_test_pca) #предсказания
accuracy_pca = accuracy_score(y_pred, y_test) #точность
print(accuracy_pca)
```

```
0.9555555555555556
Wall time: 14 ms
```

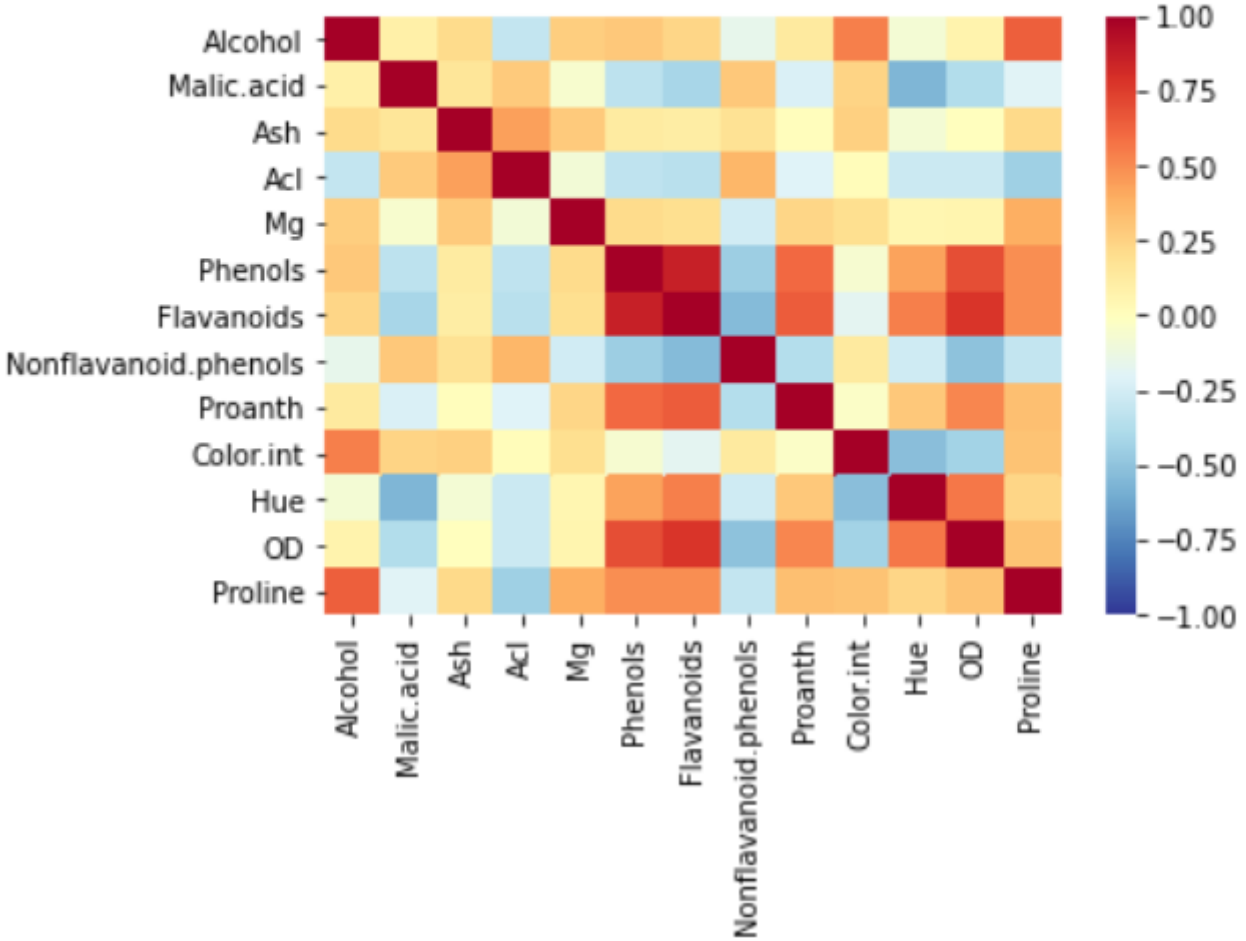
Можно заметить, если не применить PCA, то качество нашей модели будет не сильно лучше, но на большем количестве данных обучение будет происходить гораздо дольше

```
%%time
clf2 = LinearSVC(random_state=17).fit(X_train, y_train)
y_pred = clf2.predict(X_test)
accuracy_pca = accuracy_score(y_pred, y_test)
print(accuracy_pca)
```

```
0.9777777777777777
Wall time: 29.9 ms
```

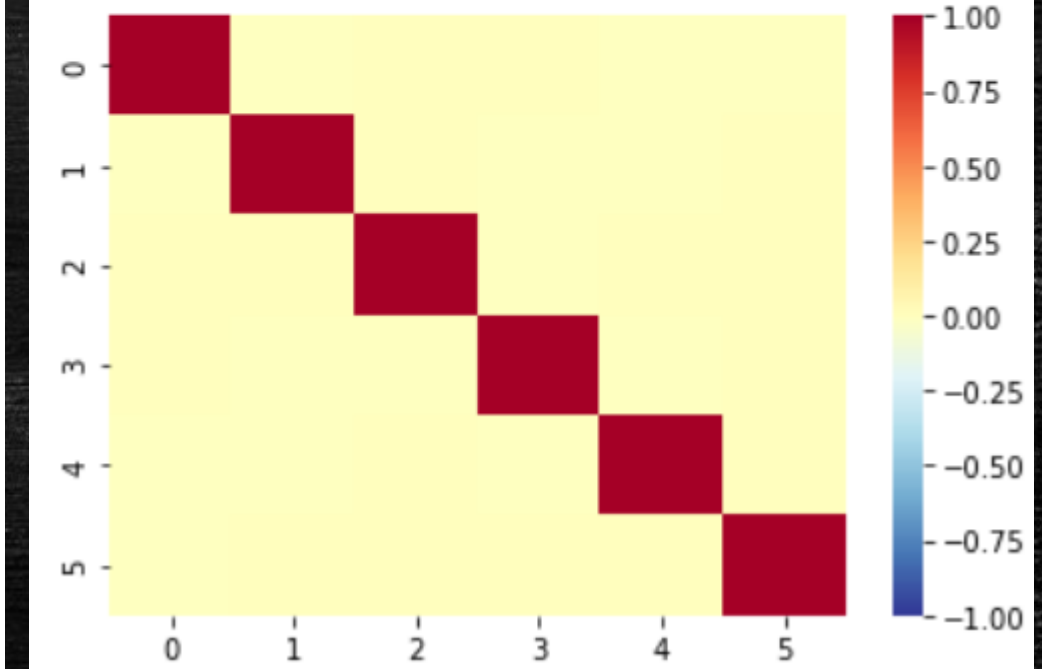
```
import pandas as pd
import seaborn as sns
wine = pd.read_csv('wine.csv')
X = wine.drop(['Wine'], axis=1)
sns.heatmap(X.corr(), cmap='RdYlBu_r', vmin=-1, vmax=1, center= 0)
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x156b3fb0>
```

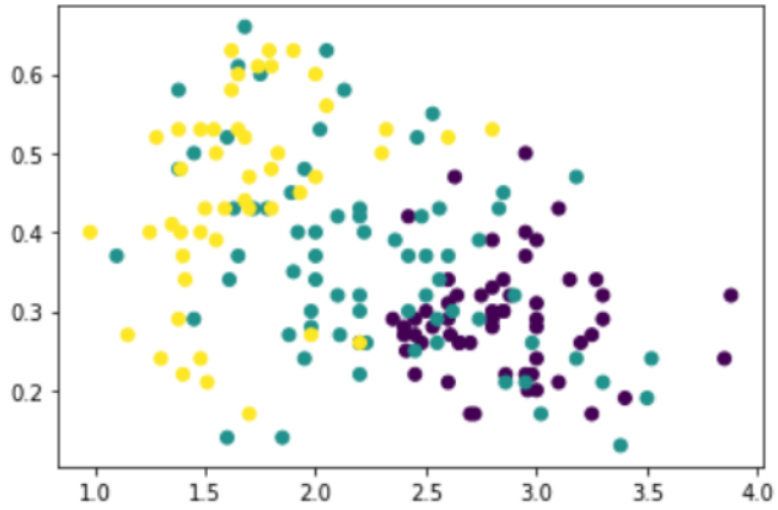



```
pca = PCA(n_components=6)
#приминяем метод главных компонент
pca.fit_transform(X)
X_pca = pd.DataFrame(pca.transform(X))
sns.heatmap(X_pca.corr(), cmap='RdYlBu_r',
            vmin=-1, vmax=1, center= 0)
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x168
```



```
plt.figure()
plt.scatter(wine['Phenols'], wine['Nonflavanoid.phenols'], c=y, )
plt.show()
```

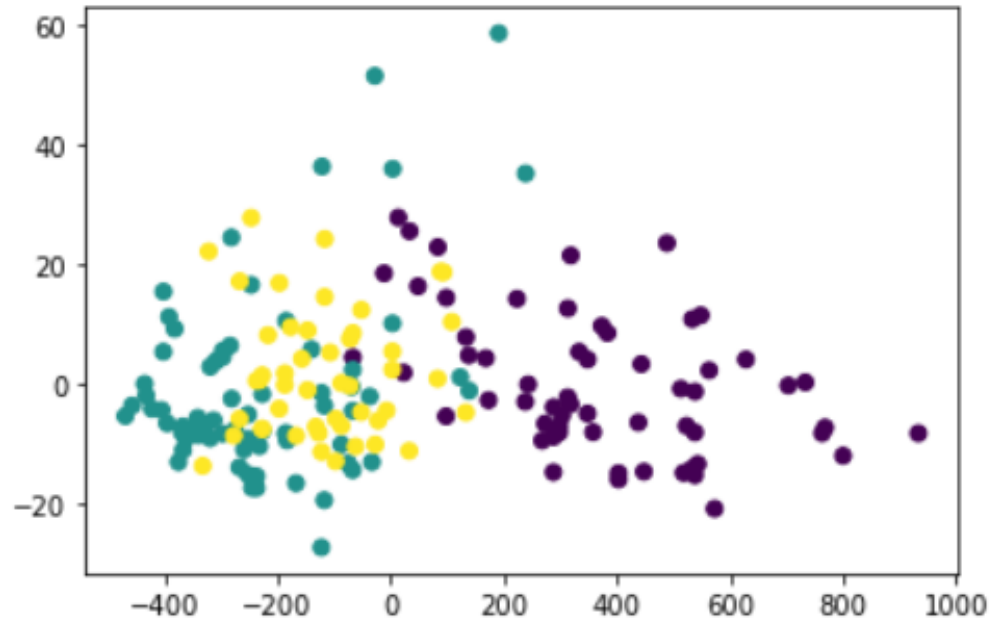



```
from sklearn import datasets

wine = datasets.load_wine() #загружаем данные
X, y = wine.data, wine.target #делим на признаки и целевую переменную

pca = PCA(n_components=2) #приминяем метод главных компонент
Z = pca.fit_transform(X)

plt.figure() #рисуем
plt.scatter(Z[:, 0], Z[:, 1], c=y, )
plt.show()
```



Наши объекты, которые описываются 13 признакам, мы представили в двумерном пространстве

Новые признаки – это линейные комбинации старых

```
for component in pca.components_:
    print("Meaning of the component:")
    print(" + ".join("%.3f x %s" % (value, name)
                     for value, name in zip(component,wine.feature_names)))
```

```
Meaning of the component:
0.002 x alcohol + -0.001 x malic_acid + 0.000 x ash + -0.005 x alcalinity_of_ash + 0.018 x magnesium + 0.001 x total_phenols +
0.002 x flavanoids + -0.000 x nonflavanoid_phenols + 0.001 x proanthocyanins + 0.002 x color_intensity + 0.000 x hue + 0.001 x
od280/od315_of_diluted_wines + 1.000 x proline
Meaning of the component:
0.001 x alcohol + 0.002 x malic_acid + 0.005 x ash + 0.026 x alcalinity_of_ash + 0.999 x magnesium + 0.001 x total_phenols + -
0.000 x flavanoids + -0.001 x nonflavanoid_phenols + 0.005 x proanthocyanins + 0.015 x color_intensity + -0.001 x hue + -0.003
x od280/od315_of_diluted_wines + -0.018 x proline
```