PCA и что это такое

Всем снова привет) У меня в планах изначально было за неделю сделать хотя бы две статьи, и раз уж в будни ничего нормального не получилось, то давайте сделаем это сейчас) Ах да, всё равно раз уж мою статью и так никто ничает, но если вдруг ты это читаешь, то знай, что ты огромное талантище, и всё только в твоих руках) поэтому для начала самым крутым порекомендую книгу (сам скоро тоже начну читать, присоединяйся!), Которая называется "ЗАПОМНИТЬ ВСЁ", которая поможет тебе и мне в улучшении и тренировки нашей памяти, чтобы мы всё схатывали на лету и были мега умниками)

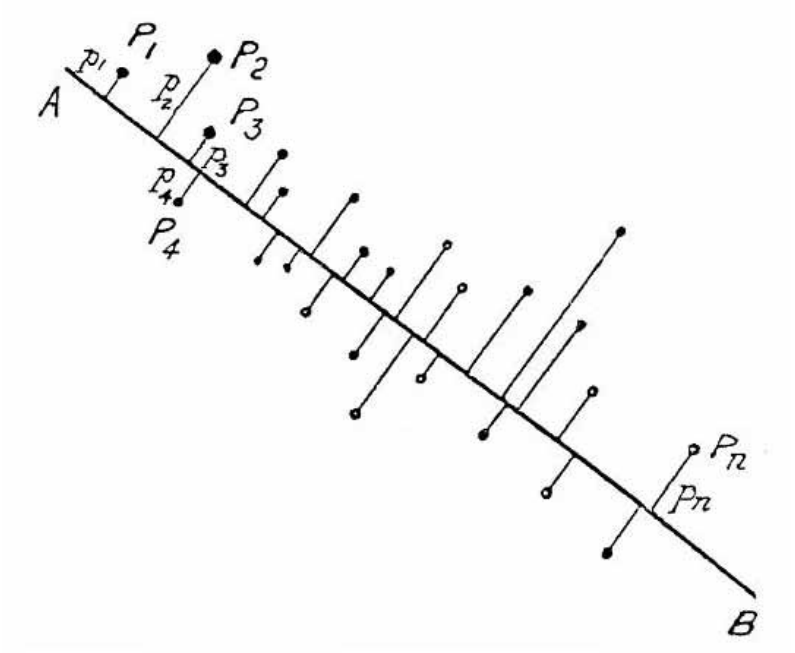
Ах да, у нас тут аналитика, точно) Для начала давайте подумаем, что вообще такое данные.
В больших фирмах за несколько недель может набраться данных о пользователях на целые ТБ-ы, поэтому если вы попробуете проанализировать эти данные методами машинного обучения, то ваш комп просто встанет, и будет считать одну формулу для миллионов значений несколько дней подряд. **Но почему?**

В настоящее время алгоритмы работают с выборками, которые зависят от десятков, или даже сотен различных признаков. Поэтому вам в голову может придти мысль какую-то метрику не использовать вовсе, *убрать и забыть*. Но можно ли вообще так поступать?

К примеру, не думаю, что для анализа эффективности смазки для любовных утех вы будете учитывать количество пальцев партнера. А также не забывайте такую вещь как корреляция данных - *взаимосвязь двух или более случайных величин*. И именно о них мы дальше и будем говорить.

Как нам сократить эти данные, чтобы увеличить скорость обработки данных? Так и возникает задача снижения размерности - хотим заменить несколько таких "связанные" фичей на одну. Без потери качества.

Карлом Пирсоном в 1901 был придуман Principal Component Analysis (PCA), или метод главных компонент. Пирсон решал задачу аппроксимации (приближения) экспериментальных данных линейными преобразованиями, т.е. прямыми. Пирсон пытался перейти от двумерной задачи к одномерной. Ниже картинка из оригинальной работы, на которой демонстрируется главная компонента, которая представляет собой прямую линию



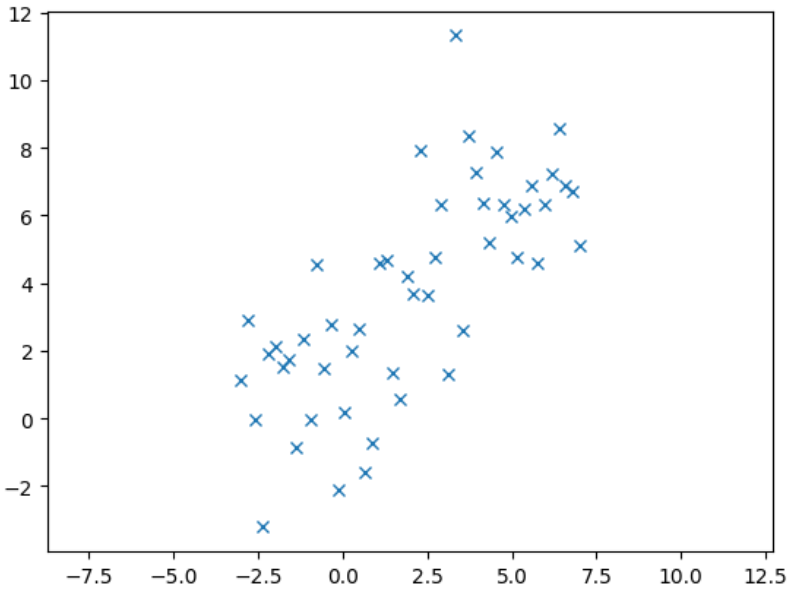
Таким образом у нас получится уменьшить размерность данных! Давайте сразу проверим на практике

```
import numpy as np
import matplotlib.pyplot as plt
import pickle
%matplotlib inline

with open('8.8_eigen.pkl', 'rb') as f:
```

```
    X = pickle.load(f)
print(X[:5,:5],"\n")
plt.plot(X[:,0], X[:,1], 'x')
plt.axis('equal') # Для того чтобы масштаб был 1:1
plt.show()
```

```
[[-3.          1.1474225 ]
 [-2.79591837  2.91847142]
 [-2.59183673 -0.05209246]
 [-2.3877551  -3.2127219 ]
 [-2.18367347  1.91180977]]
```



Применяем алгоритм `sklearn.decomposition.PCA` к набору данных:

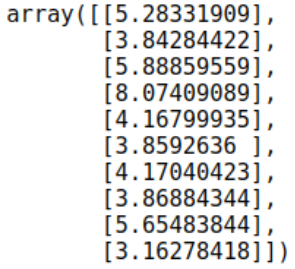
```
from sklearn.decomposition import PCA

pca = PCA(n_components=1).fit(X)
X_pca = pca.transform(X)
X_pca[:10]
```

```
array([[5.28331909],
       [3.84284422],
       [5.88859559],
       [8.07409089],
       [4.16799935],
       [3.8592636 ],
       [4.17040423],
       [3.86884344],
       [5.65483844],
       [3.16278418]])
```

Как видите, мы сжали с помощью PCA двумерные данные в одномерный массив.

```
array([[-0.66642088,  0.44491071],
       [-0.88927281, -0.53789631],
       [-0.84150307,  0.84666476],
       [ 1.38974939,  0.39091086],
       [-1.22821053, -1.37589924],
       [-0.11931996,  0.28957531],
       [ 1.23620879, -0.18126413],
       [-0.90744735, -0.80065208],
       [-1.16059805, -0.82904383],
       [-0.76099026,  1.24978287]])
```



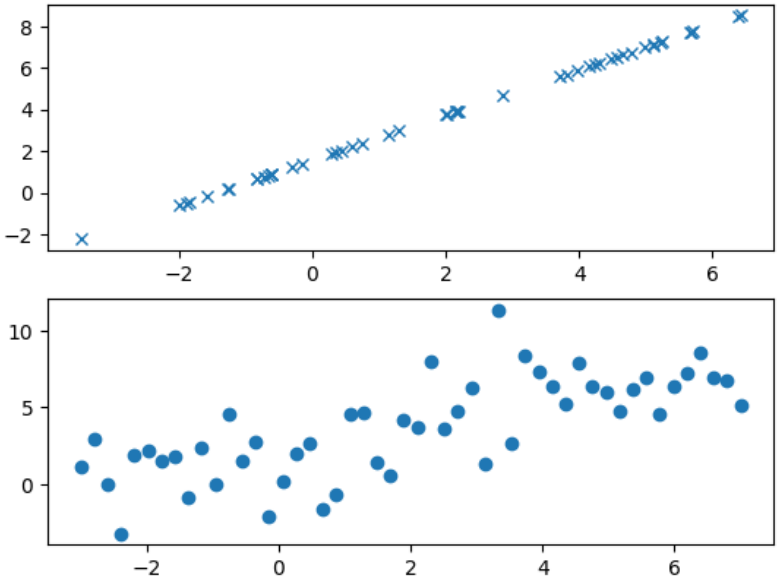
```
array([[5.28331909],
       [3.84284422],
       [5.88859559],
       [8.07409089],
       [4.16799935],
       [3.8592636 ],
       [4.17040423],
       [3.86884344],
       [5.65483844],
       [3.16278418]])
```

Давайте визуализируем этот массив - для этого надо произвести обратное преобразование из одномерного массива в двумерный. Отобразим "восстановленные" данные на графике (вторая картинка - исходный, несжатый массив):

```
X_new = pca.inverse_transform(X_pca)

plt.figure(1)
plt.subplot(211)
plt.plot(X_new[:,0], X_new[:,1], 'x')

plt.subplot(212)
plt.plot(X[:,0], X[:,1], 'o')
plt.show()
```



Алгоритм PCA выявил, что переменная от x_2 линейным образом зависит от переменной x_1 - то есть вместо переменной x_2 можно использовать линейное преобразование $x_2 = x_1 w_1 + w_0$ (эта переменная на первом графике). "Сжатые данные" представляют собой график линейной функции $x_2 = ax_1 + b$.

Данные, представляли собой облако точек, но алгоритм PCA ужал их до одномерного случая. Другие интересные визуализации можно глянуть по ссылке

Как применять этот алгоритм на практике? PCA можно применить к любым данным, но как определить, что вам *действительно* нужно сжимать размерность?

- обучаете свой алгоритм (например, линейную регрессию) на исходных данных и вычисляете качество решения по RMSE (например)
- сжимаете данные с помощью PCA
- снова обучаете линейную регрессию с теми же параметрами, проверяете качество решения RMSE. Если качество выросло - PCA сжал данные "правильно", пропали шумы. Если качество уменьшилось - PCA сказывается негативно, вместо убирания шумов вы теряете часть информации

Ну и перед тем как закончить мою статью, хочется еще разобрать несколько моментов в PCA: он плохо работает, когда переменные находятся в разный мастшабах, к примеру, когда кол-во минут пользователя по своему значению численному меньше, чем какая-нибудь цена за подписку, то цена может быть намного ниже по значению, чем минуты, и по итогу

погрешность будет большая. Желательно тогда использовать эти данные после перемасштабирования в какие-нибудь абсолютные значения, либо если мы используем похожие значения по величинам. Ладно, на сегодня мы точно уж закончили, до завтра!