Лекция 6

Понижение размерности

В реальных задачах машинного обучения часто бывает слишком много параметров, на основе которых делается окончательный прогноз. С одной стороны, чем больше параметров присутствует в наборе данных, тем эффективнее обучается классификатор. Однако **большее количество параметров также означает более высокие вычислительные затраты, сложность визуализации и кроме того может привести к переобучению алгоритма**, когда он пытается создать модель, объясняющую влияние всех параметров.

Исправить это позволяют алгоритмы уменьшения размерности. Их главная задача - уменьшить количество рассматриваемых случайных признаков. Они выделяют только самые важные и влияющие на точность предсказания характеристики.

Основные способы понижения размерности

Понижение размерности можно осуществить двумя способами:

1. Выбор функций (Feature Selection). Сохранение только наиболее релевантных переменных из исходного набора данных.

Вот лишь некоторые примеры данных методов:

- *Фильтр с низкой дисперсией* (`VarianceThreshold`)

Например, если переменные какого-либо параметра не изменяются (т.е. параметр имеет нулевую дисперсию), то этот параметр никак не влияет на модель. Поэтому, можно рассчитать дисперсию каждого параметра в выборке, а затем отбросить параметры, имеющие низкую дисперсию по сравнению с другими параметрами в наборе данных. Например, по данным титаника видно, что наименьшую дисперсию дает параметр Parch:

```
Pclass       0.699015
Age        211.019125
SibSp        1.216043
Parch        0.649728
Fare      2469.436846
```

Age	Age in years
sibsp	# of siblings / spouses aboard the Titanic
parch	# of parents / children aboard the Titanic
ticket	Ticket number
fare	Passenger fare

- *Фильтр высокой корреляции*

Высокая корреляция между двумя переменными означает, что они имеют схожие тенденции и, вероятно, несут схожую информацию. Это может резко снизить производительность некоторых моделей (например, моделей линейной и логистической регрессии). Можно вычислить корреляцию между

независимыми числовыми переменными. Если коэффициент корреляции пересекает определенное пороговое значение (как правило 0,5–0,6), мы можем отбросить одну из переменных (*удаление переменной очень субъективно и всегда должно производиться с учетом предметной области*).

В качестве общего правила мы должны оставить те переменные, которые показывают приличную или высокую корреляцию с целевой переменной.

- *Выбор функции с помощью SelectFromModel*

SelectFromModel - это мета-преобразователь, который можно использовать вместе с любым оценщиком, который присваивает важность каждой функции (например, деревья решений или случайный лес). Функции считаются неважными и удаляются, если соответствующая важность значений функций ниже указанного *threshold* параметра.

Алгоритм действий:

1. **Обучение модели** – дерево решений (или ансамбль) обучается на данных.
2. **Оценка важности признаков** – модель вычисляет, насколько каждый признак влияет на предсказание (метод model.feature_importances_).
3. **Отбор наиболее значимых признаков** – выбираются только те признаки, чья важность превышает заданный порог.

Пример кода для построения гистограммы отсортированных по важности признаков

```
feature_importances = pd.Series(
    model.feature_importances_,
    index=X_train.columns
).sort_values(ascending=False)

plt.figure(figsize=(10, 6))
feature_importances.plot.bar()
plt.title("Важность признаков")
plt.ylabel("Важность")
plt.tight_layout()
plt.show()
```

- *Рекурсивное устранение признаков*

Цель исключения рекурсивных признаков (*RFE*) состоит в том, чтобы выбрать признаки путем рекурсивного рассмотрения все меньших и меньших наборов признаков. Модель обучается на начальном наборе признаков, и определяется важность каждого признака. Затем наименее важные признаки удаляются из текущего набора признаков. Эта процедура рекурсивно повторяется для

сокращенного набора, пока в конечном итоге не будет достигнуто желаемое количество признаков.

Пример *RFE*:

```
from sklearn.feature_selection import RFE
from sklearn.linear_model import LogisticRegression

model = LogisticRegression()
rfe = RFE(model, n_features_to_select=5)
X_new = rfe.fit_transform(X, y)
```

https://www.analyticsvidhya.com/blog/2018/08/dimensionality-reduction-techniques-python/

Алгоритмы выбора признаков реализованы в модуле ***sklearn.feature_selection***.

Преимущества Feature Selection

- ✓ Сохраняет интерпретируемость признаков.
- ✓ Уменьшает шум и избыточность данных
- ✓ Уменьшает переобучение и ускоряет обучение модели.
- ✓ Упрощает визуализацию данных.

2. Извлечение функций (Feature Extraction).

Нахождение меньшего набора новых признаков, каждый из которых представляет собой комбинацию входных признаков, содержащих в основном ту же информацию, что и входные признаки.

- ✓ ***PCA*** *(Principal Component Analysis)*
- ✓ ***LDA*** *(Linear Discriminant Analysis)*

Часто бывает так, что признаки довольно сильно зависят друг от друга и их одновременное наличие избыточно.

К примеру, расход топлива у нас меряется в литрах на 100 км, а в США в милях на галлон. На первый взгляд, величины разные, но на самом деле они строго зависят друг от друга. В миле 1600м, а в галлоне 3.8л. Один признак строго зависит от другого, зная один, знаем и другой.

Но гораздо чаще бывает так, что признаки зависят друг от друга не так строго и не так явно. Объем двигателя в целом положительно влияет на разгон до 100 км/ч, но это верно не всегда. А еще может оказаться, что с учетом не видимых на первый взгляд факторов (*улучшения качества топлива, использования более легких материалов и прочих современных достижений*), год выпуска автомобиля не сильно, но тоже влияет на его разгон.

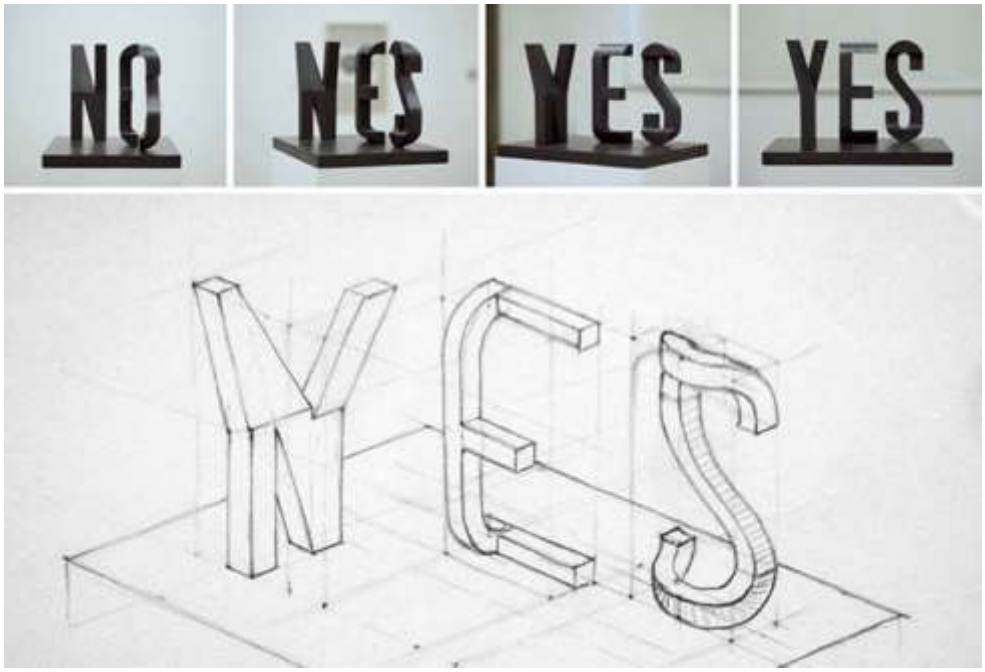
Зная зависимости и их силу, **можно выразить несколько признаков через один, так сказать, слить воедино**, и работать уже с более простой моделью. Конечно, избежать потерь информации, скорее всего не удастся, но минимизировать ее нам поможет как раз **метод PCA**.

Анализ главных компонент (**PCA**, *principal component analysis*)

Анализ главных компонент – это метод понижения размерности датасета, который преобразует больший набор переменных в меньший с минимальными потерями информативности.

Можно сказать, что метод главных компонент представляет собой метод, который осуществляет вращение данных с тем, чтобы преобразованные признаки не коррелировали между собой. Часто это вращение сопровождается выбором подмножества новых признаков в зависимости от их важности с точки зрения интерпретации данных.

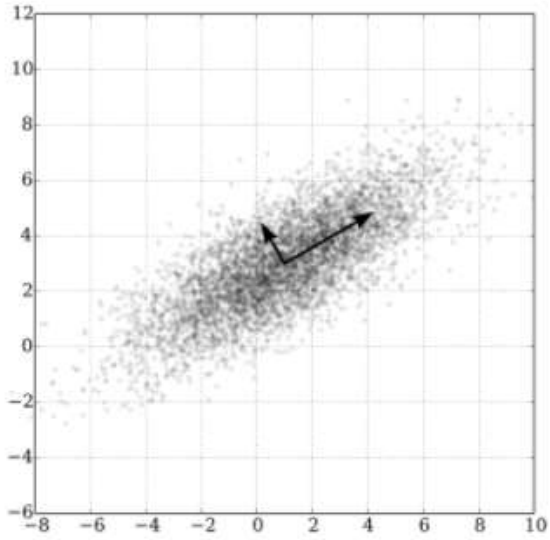
Например, можно провести аналогию с иллюзией, изображенной на рисунке:



Как работает метод главных компонент (PCA) на простом примере / Хабр (habr.com)

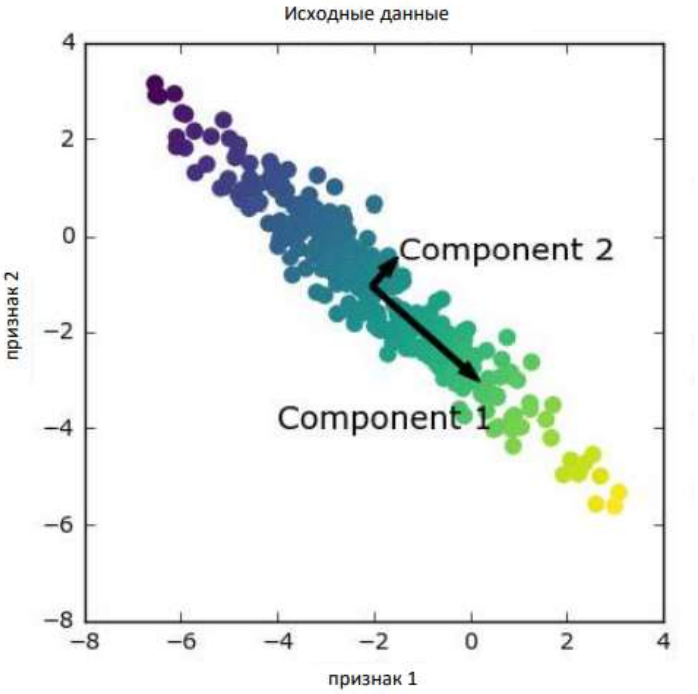
Итак, метод главных компонент — один из самых интуитивно простых и часто используемых методов для снижения размерности данных и проекции их на ортогональное подпространство признаков.

В общем виде это можно представить, как предположение о том, что все наши наблюдения скорее всего выглядят как некий эллипсоид в подпространстве нашего исходного пространства и наш новый базис в этом пространстве совпадает с осями этого эллипсоида. Это предположение позволяет нам одновременно избавиться от сильно скоррелированных признаков, так как вектора базиса пространства, на которое мы проецируем, будут ортогональными.



В общем случае размерность этого эллипсоида будет равна размерности исходного пространства, но наше предположение о том, что данные лежат в подпространстве меньшей размерности, позволяет нам отбросить "лишнее" подпространство в новой проекции, а именно то подпространство, вдоль осей которого эллипсоид будет наименее растянут.

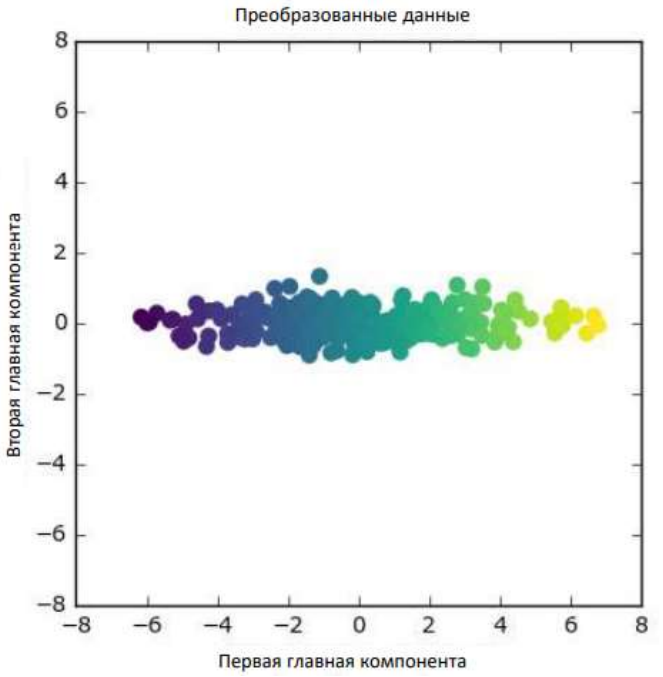
Рассмотрим на примере:



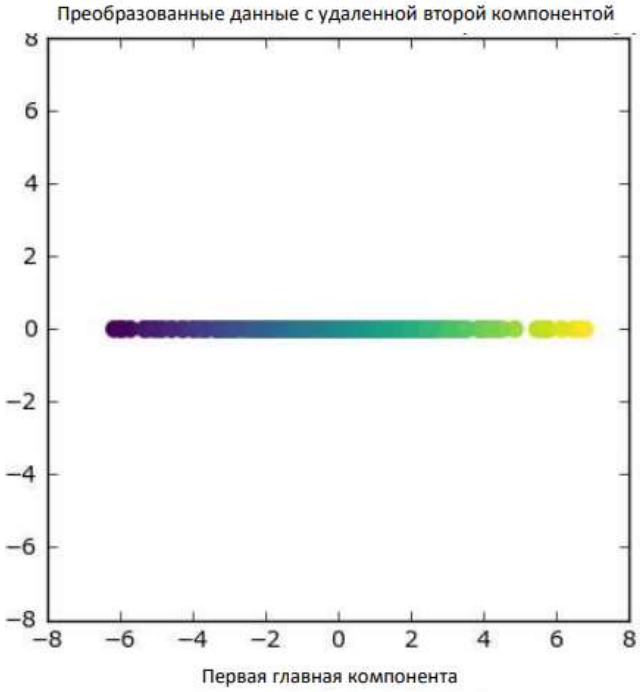
На рис. показаны исходные точки данных. Алгоритм начинает работу с того, что **сначала находит направление максимальной дисперсии**, помеченное как «компонента 1». Речь идет о направлении (или векторе) данных, который содержит большую часть информации, или другими словами, направление, вдоль которого признаки коррелируют друг с другом сильнее всего. **Затем алгоритм находит направление, которое содержит наибольшее количество информации, и при этом ортогонально** (расположено под прямым углом) первому направлению. В двумерном пространстве существует только одна возможная ориентация, расположенная под прямым углом, но в пространствах большей размерности может быть (бесконечно) много ортогональных направлений. Направления, найденные с помощью этого алгоритма, называются главными компонентами (principal components), поскольку они являются основными направлениями дисперсии данных. В целом

максимально возможное количество главных компонент равно количеству исходных признаков.

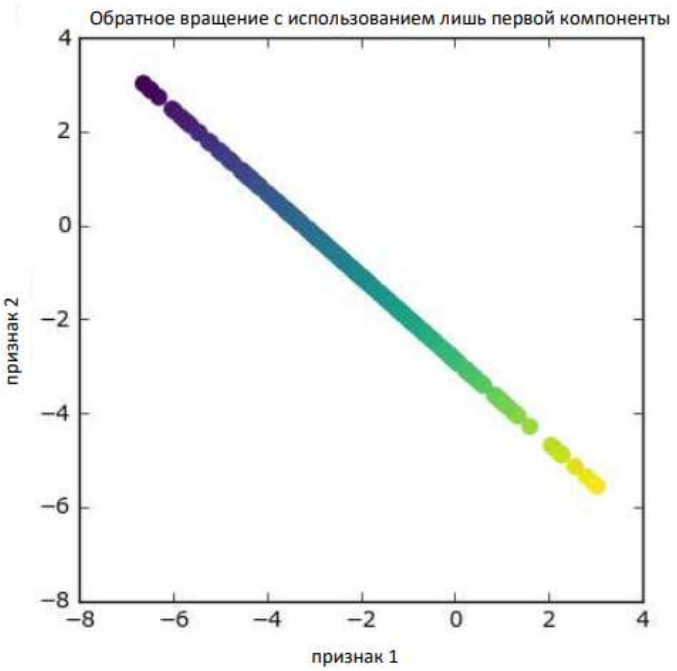
На данном рис. те же самые данные, но теперь повернутые таким образом, что первая главная компонента совпадает с осью x, а вторая главная компонента совпадает с осью y. *Перед вращением из каждого значения данных вычитается среднее, таким образом, преобразованные данные центрированы около нуля.* В новом представлении данных, найденном с помощью PCA, две оси становятся некоррелированными. Это означает, что в новом представлении все элементы корреляционной матрицы данных, кроме диагональных, будут равны нулю.



Можно использовать PCA для уменьшения размерности, сохранив лишь несколько главных компонент. В данном примере мы можем оставить лишь первую главную компоненту. Это уменьшит размерность данных: из двумерного массива данных получаем одномерный массив данных.



И, наконец, можно отменить вращение и добавить обратно среднее значение к значениям данных. В итоге получим данные, показанные на рис. Эти точки располагаются в пространстве исходных признаков, но мы оставили лишь информацию, содержащуюся в первой главной компоненте. Это преобразование иногда используется, чтобы удалить эффект шума из данных или показать, какая часть информации сохраняется при использовании главных компонент.

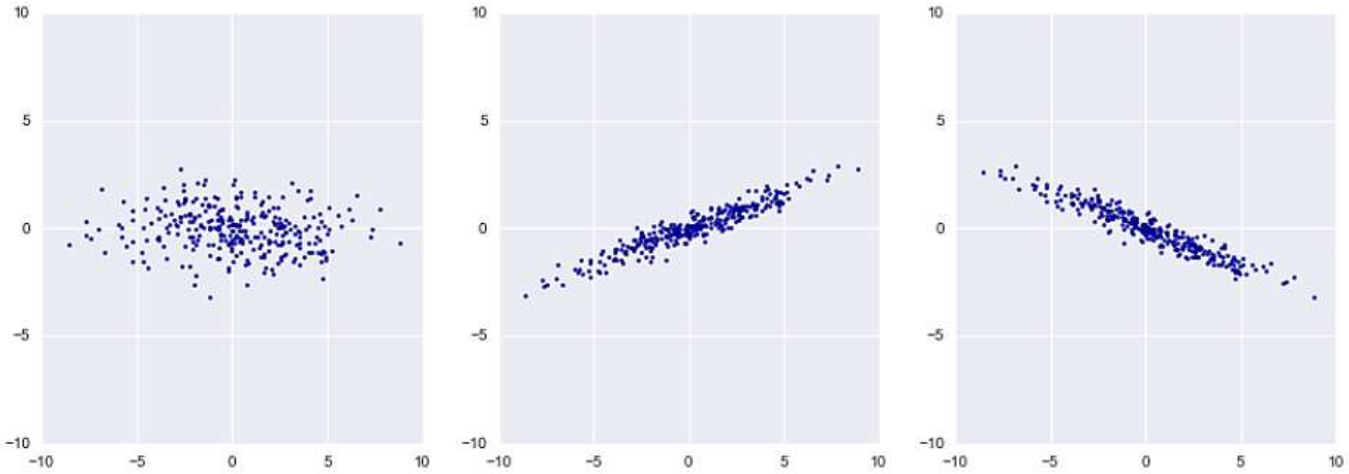


PCA сохраняет корреляции между параметрами. Основные компоненты, создаваемые методом PCA, представляют собой линейные комбинации исходных параметров.

Немного математики

Вспомним, что для описания случайной величины используются мат. ожидание и дисперсия. Можно сказать, что мат. ожидание – это «центр тяжести» величины, а дисперсия – это ее «размеры». Грубо говоря, мат. ожидание задает положение случайной величины, а дисперсия – ее размер (точнее, разброс).

В случае с многомерной случайной величиной (случайным вектором) положение центра все так же будет являться мат. ожиданиями ее проекций на оси. А вот для описания ее формы уже недостаточно только ее дисперсий по осям. *На этих графиках у всех трех случайных величин одинаковые мат.ожидания и дисперсии, а их проекции на оси в целом окажутся одинаковы!*



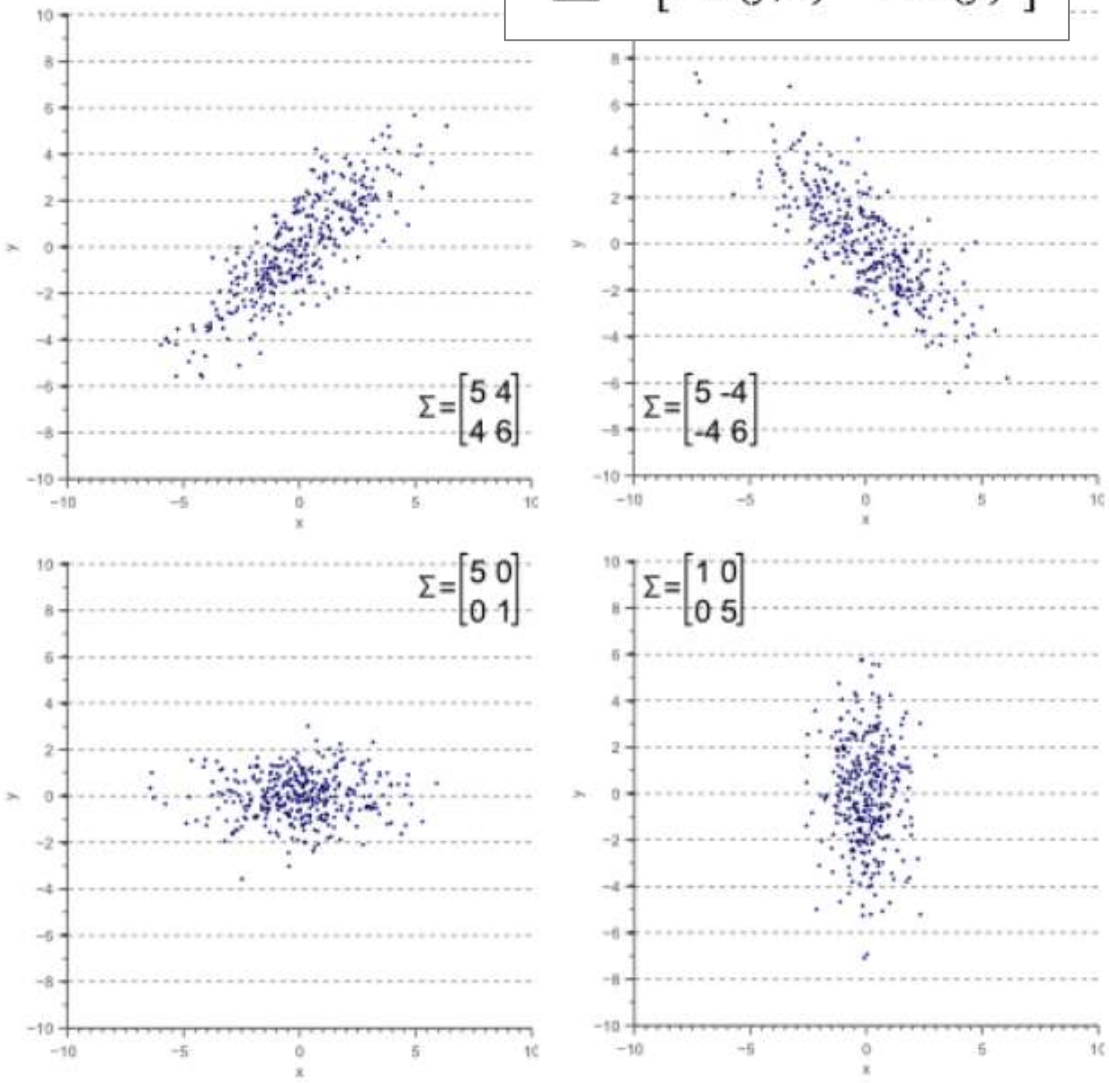
Для описания формы случайного вектора необходима **ковариационная матрица**.

Ковариация характеризует степень зависимости для зависимых величин. Чем больше ковариация отличается от нуля, тем больше зависимость.

Матрица ковариаций для нескольких случайных величин X, Y,...Z всегда симметрична, причем на главной диагонали этой матрицы лежат дисперсии соответствующих признаков, а вне диагонали — ковариации соответствующих пар признаков

$$\begin{pmatrix} \sigma_x^2 & \sigma_{xy} & \cdots & \sigma_{xz} \\ \sigma_{yx} & \sigma_y^2 & & \sigma_{yz} \\ \vdots & & \ddots & \vdots \\ \sigma_{zx} & \sigma_{zy} & \cdots & \sigma_z^2 \end{pmatrix}$$

$$\sum = \begin{bmatrix} Var(x) & Cov(x,y) \\ Cov(y,x) & Var(y) \end{bmatrix}$$



Ковариантная матрица описывает форму нашей случайной величины, из которой мы можем получить ее размеры по **x** и **y**, а также примерную форму на плоскости.

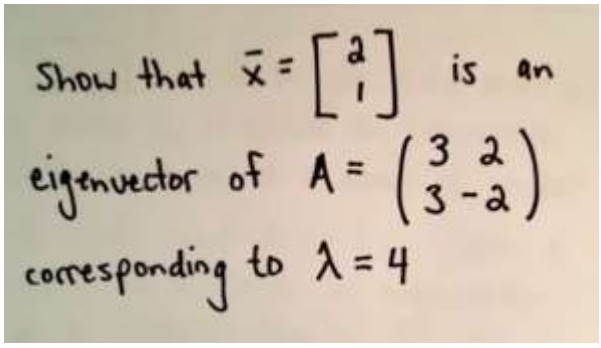
Теперь надо найти такой вектор (в нашем случае только один), при котором максимизировался бы размер (дисперсия) проекции нашей выборки на него.

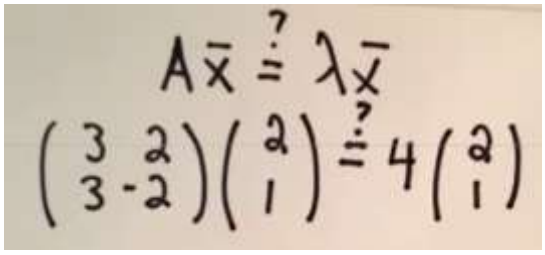
Собственный вектор $\overline{X}$ матрицы ковариации будет показывать направление максимальной дисперсии, а собственное число λ – величину дисперсии.

Рассмотрим квадратную матрицу порядка n с постоянными действительными элементами a_{ij}

$$A = \begin{pmatrix} a_{11} & \dots & a_{1n} \\ \dots & \dots & \dots \\ a_{n1} & \dots & a_{nn} \end{pmatrix}.$$

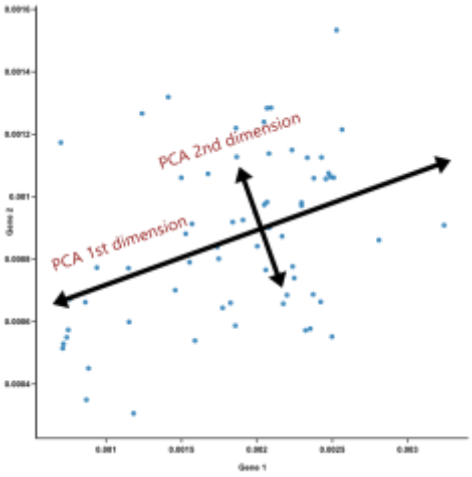
Определение. Число λ называется собственным значением, а ненулевой вектор $\vec{h}$ называется соответствующим собственным вектором матрицы A если выполняется равенство: $A \cdot \bar{h} = \lambda \cdot \bar{h}$. (1)





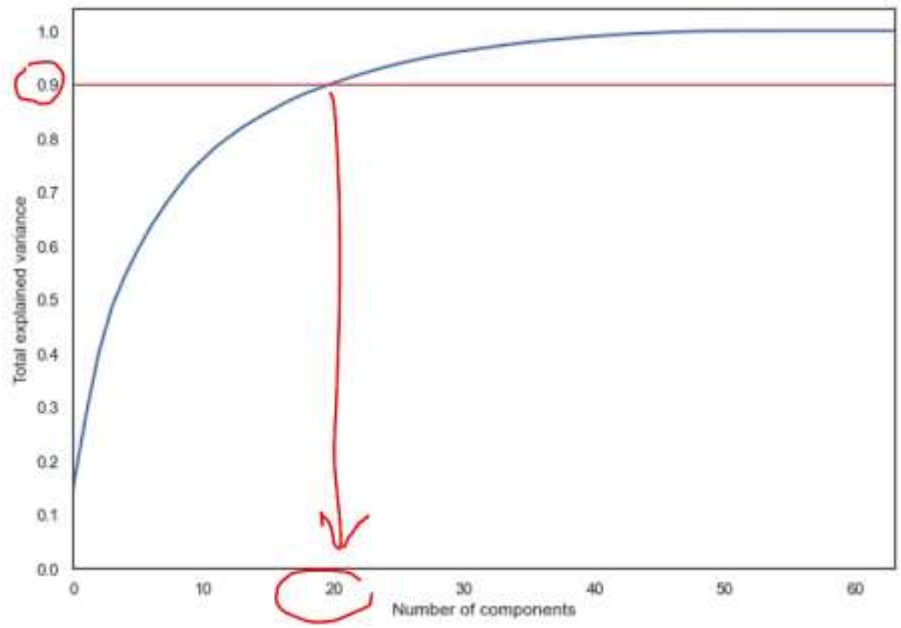
Итоги: метод **PCA** используется для разложения многомерного набора данных на набор последовательных ортогональных компонентов, упорядоченных по величине дисперсии. Причем первая компонента в наибольшей степени описывает набор данных (потому что ее направление совпадает с направлением максимальной дисперсии данных и несет наибольшее количество информации об изменении данных), и так по убывающей от компоненты к компоненте.

Если методу **PCA** не указывать количество компонент, то он найдет все компоненты, описывающие данные, и при реконструкции мы получим первоначальный набор данных. НО цель применения метода **PCA** именно в сокращении размерности модели (уменьшении количества параметров), поэтому при указании количества компонент будут найдены главные компоненты с достаточной степенью точности описывающие набор данных.



Как правило, выбирают количество главных компонент, чтобы оставить 90% дисперсии исходных данных. В примере с распознаванием рукописных цифр для обеспечения 90% дисперсии достаточно взять примерно 20 главных компонентов, то есть снизить размерность с 64 признаков до 20.

```
# Построим график процента отклонения от количества компонент
pca = decomposition.PCA().fit(X)
plt.figure(figsize=(10,7))
# pca.explained_variance_ratio_ Процент отклонения, объясняемый
# каждым из выбранных компонентов
plt.plot(np.cumsum(pca.explained_variance_ratio_), color='b', lw=2)
plt.xlabel('Number of components')
plt.ylabel('Total explained variance')
plt.xlim(0, 63)
plt.yticks(np.arange(0, 1.1, 0.1))
plt.axhline(0.9, c='r')
plt.show();
```



Примеры использования PCA

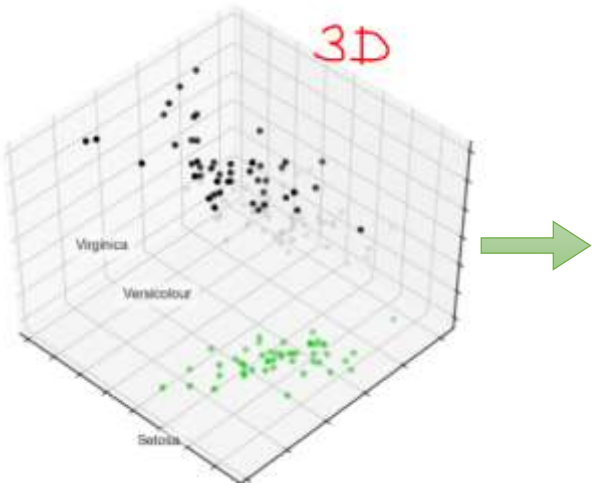
Открытый курс машинного обучения. Тема 7. Обучение без учителя: PCA и кластеризация / Хабр (habr.com)

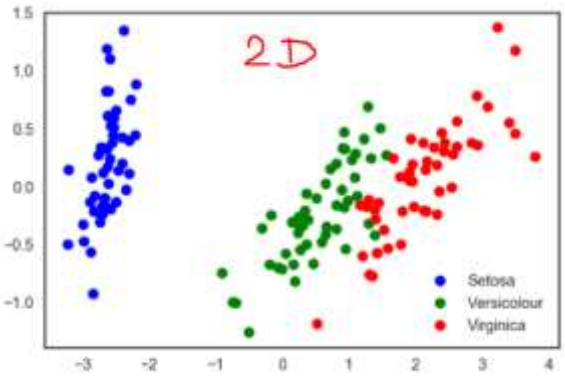
Рассмотрим на примере набора данных по цветкам ириса как изменится точность вычисления при использовании простой модели неглубокого дерева решений:

```
# Для примера возьмём неглубокое дерево решений
clf = DecisionTreeClassifier(max_depth=2, random_state=42)
clf.fit(X_train, y_train)
preds = clf.predict_proba(X_test)
print('Accuracy: {:.5f}'.format(accuracy_score(y_test,
                                               preds.argmax(axis=1))))
```

```
Out: Accuracy: 0.88889
```

Теперь уменьшим размерность с 3D до 2D





```
# Прогоним встроенный в sklearn PCA
pca = decomposition.PCA(n_components=2)
X_centered = X - X.mean(axis=0)
pca.fit(X_centered)
X_pca = pca.transform(X_centered)
```

```
# Повторим то же самое разбиение на валидацию и тренировочную выборку.
X_train, X_test, y_train, y_test = train_test_split(X_pca, y, test_size=.3,
                                                    stratify=y,
                                                    random_state=42)

clf = DecisionTreeClassifier(max_depth=2, random_state=42)
clf.fit(X_train, y_train)
preds = clf.predict_proba(X_test)
print('Accuracy: {:.5f}'.format(accuracy_score(y_test,
                                               preds.argmax(axis=1))))
```

Смотрим на возросшую точность классификации:

```
Out: Accuracy: 0.91111
```

Таким образом в примере с простой моделью уменьшение размерности методом PCA дает увеличение точности.

Пример уменьшения размерности набора данных **Breast**, который содержит 30 признаков, до 2ух компонент:

было

	0	1	2	3	4	5
0	1.097064	-2.073335	1.269934	0.984375	1.568466	3.283515
1	1.829821	-0.353632	1.685955	1.908708	-0.826962	-0.487072
2	1.579888	0.456187	1.566503	1.558884	0.942210	1.052926
3	-0.768909	0.253732	-0.592687	-0.764464	3.283553	3.402909
4	1.750297	-1.151816	1.776573	1.826229	0.280372	0.539340
...	...	...	...	...	...	...
564	2.110995	0.721473	2.060786	2.343856	1.041842	0.219060
565	1.704854	2.085134	1.615931	1.723842	0.102458	-0.017833
566	0.702284	2.045574	0.672676	0.577953	-0.840484	-0.038680
567	1.838341	2.336457	1.982524	1.735218	1.525767	3.272144
568	-1.808401	1.221792	-1.814389	-1.347789	-3.112085	-1.150752

569 rows × 30 columns

стало

	0	1
0	9.192837	1.948583
1	2.387802	-3.768172
2	5.733896	-1.075174
3	7.122953	10.275589
4	3.935302	-1.948072
...	...	...
564	6.439315	-3.576817
565	3.793382	-3.584048
566	1.256179	-1.902297
567	10.374794	1.672010
568	-5.475243	-0.670637

569 rows × 2 columns

Главные компоненты можно получить из атрибута **components_** модели **pca**.

```
from sklearn.decomposition import PCA
# оставляем первые две главные компоненты
pca = PCA(n_components=2)
# подгоняем модель PCA на наборе данных breast cancer
pca.fit(X_scaled)
# преобразуем данные к первым двум главным компонентам
X_pca = pca.transform(X_scaled)

# посмотрим на содержимое атрибута components_:
print("компоненты PCA:\n{}".format(pca.components_))
```

```
компоненты PCA:
[[ 0.21890244  0.10372458  0.22753729  0.22099499  0.14258969  0.23928535
   0.25840048  0.26085376  0.13816696  0.06436335  0.20597878  0.01742803
   0.21132592  0.20286964  0.01453145  0.17039345  0.15358979  0.1834174
   0.04249842  0.10256832  0.22799663  0.10446933  0.23663968  0.22487053
   0.12795256  0.21009588  0.22876753  0.25088597  0.12290456  0.13178394]
 [-0.23385713 -0.05970609 -0.21518136 -0.23107671  0.18611302  0.15189161
   0.06016536 -0.0347675   0.19034877  0.36657547 -0.10555215  0.08997968
  -0.08945723 -0.15229263  0.20443045  0.2327159   0.19720728  0.13032156
   0.183848    0.28009203 -0.21986638 -0.0454673  -0.19987843 -0.21935186
   0.17230435  0.14359317  0.09796411 -0.00825724  0.14188335  0.27533947]]
```

Оценка вклада каждого главного компонента и объема потерянной инфрмации

После того, как рассчитаны главные компоненты, можно найти explained_variance_ratio. Он предоставит объем информации или дисперсии, которую обеспечивает каждый главный компонент после проецирования данных в подпространство более низкой размерности.

```
pca.explained_variance_ratio_
```

```
array([0.44272026, 0.18971182])
```

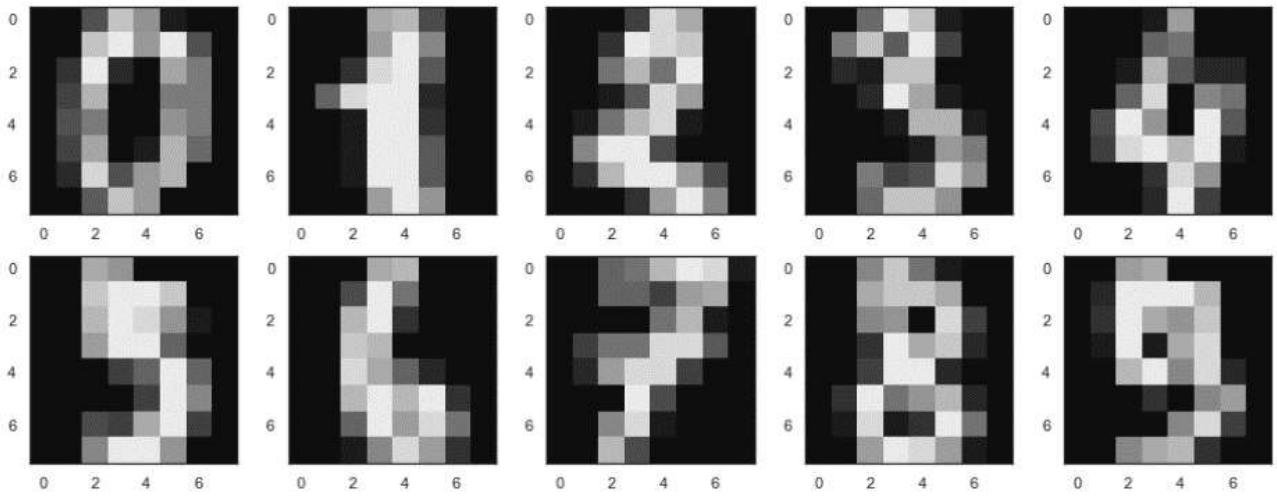
Можно заметить, что первая главная компонента содержит 44,2% информации, в то время как вторая главная компонента содержит только 19% информации. Кроме того, следует отметить, что при проецированием тридцатимерных данных на двумерные данные было потеряно 36,8% информации.

Пример с рукописными цифрами:

```
from sklearn.datasets import load_digits
# загрузим набор данных по рукописным цифрам
data = load_digits()
X, y = data.data, data.target
```

```
Картинки здесь представляются матрицей 8 x 8 (интенсивности белого цвета для
каждого пикселя). Далее эта матрица "разворачивается" в вектор длины 64,
получается признаковое описание объекта.
```

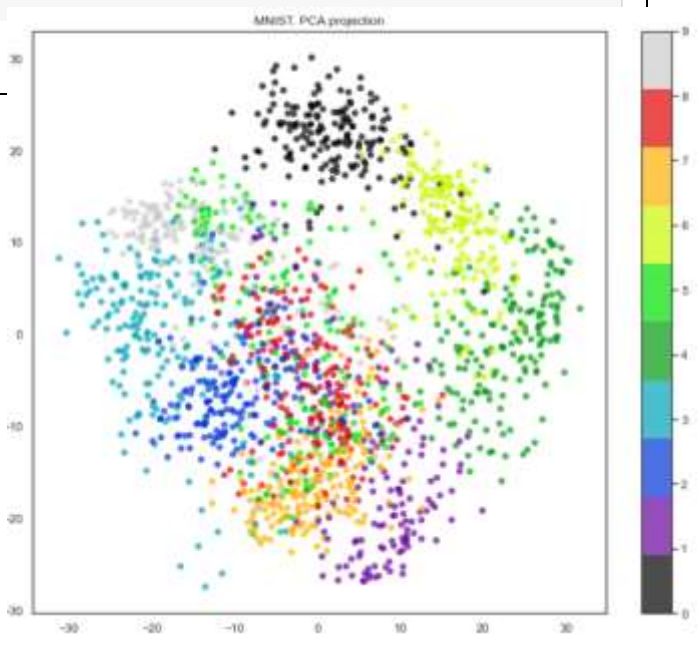
```
plt.figure(figsize=(16, 6))
for i in range(10):
    plt.subplot(2, 5, i + 1)
    plt.imshow(X[i,:].reshape([8,8]));
```



Картинки здесь представляются матрицей 8 x 8 (интенсивности белого цвета для каждого пикселя). Далее эта матрица "разворачивается" в вектор длины 64, получается признаковое описание объекта.

Получается, размерность признакового пространства здесь – 64. **Снизим размерность всего до 2 при помощи метода PCA.**

```
pca = decomposition.PCA(n_components=2)
X_reduced = pca.fit_transform(X)
```



Работа с изображениями

https://towardsdatascience.com/eigenfaces-recovering-humans-from-ghosts-17606c328184

Есть набор данных из 1000 изображений размером 64x64 пикселя. Таким образом размер нашей выборки составит $(1000, 4096)$.

Визуализация компонент

Мы не сможем понять весь содержательный смысл этих компонент, мы можем догадаться, какие характеристики изображений лиц были выделены некоторыми компонентами. **Похоже, что первая компонента главным образом**

кодирует контраст между лицом и фоном, а вторая компонента кодирует различия в освещенности между правой и левой половинами лица и т.д. Хотя это представление данных в отличие от исходных значений пикселей немного содержательнее, оно по-прежнему весьма далеко от того, как человек привык воспринимать лицо. **Важно помнить, что, как правило, алгоритмы в отличие от человека интерпретируют данные (в частности, визуальные данные, например, изображения людей) совершенно по-другому**.

Еще один способ понять, что делает модель PCA – реконструировать исходные данные, используя лишь некоторые компоненты.

Здесь визуализируем результаты реконструкции некоторых лиц, используя 10, 50, 100 и 500 компонент:

Рис. 3.11 Реконструкция трех изображений лица с помощью постепенного увеличения числа главных компонент

Таким образом можно сказать, что из первоначальных 4096 признаков изображения достаточно описать 500 признаками. Уменьшили размерность в 8! раз.

Недостатки Principal Component Analysis

PCA часто называют неконтролируемым алгоритмом. Всё из-за того, что он не гарантирует разделимость классов. Проще говоря, PCA он не понимает, является ли проблема, которую мы пытаемся решить, задачей регрессии или классификации.

Хотя PCA - очень полезный метод, его следует избегать для контролируемых алгоритмов, поскольку он полностью затрудняет получение данных. Поэтому, если мы хотим интерпретировать технику извлечения функций, стоит обратить внимание на LDA.

Кратко главные отличия PCA и LDA:

- LDA - контролируемый алгоритм, PCA - нет.
- LDA - описывает направление максимальной разделимости данных. PCA - описывает направление максимальной дисперсии данных.
- LDA требует информации о метке класса, в отличие от PCA.