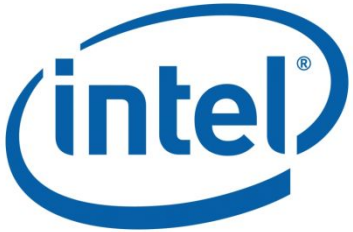
Лабораторная работа № 2
Анализ главных компонент



Основные приложения

- Dimensionality reduction
  Снижение размерности данных при сохранении всей или большей части информации
- Feature extraction
  Выявление и интерпретация скрытых признаков

Анализ заемщиков банка

- Задача : Проанализировать заемщиков банка на основе различных данных

Данные могут быть:

- Личные данные
- Семейное положение
- Образование
- Финансовое состояние
- Имущество
- Кредитная история
- …

Пример: Give Me Some Credit*

Variable Name	Description	Type
RevolvingUtilizationOfUnsecuredLines	Total balance on credit cards and personal lines of credit except real estate and no installment debt like car loans divided by the sum of credit limits	percentage
Age	Age of borrower in years	integer
NumberOfTime30-59DaysPastDueNotWorse	Number of times borrower has been 30-59 days past due but no worse in the last 2 years.	integer
DebtRatio	Monthly debt payments, alimony,living costs divided by monthy gross income	percentage
MonthlyIncome	Monthly income	real
NumberOfOpenCreditLinesAndLoans	Number of Open loans (installment like car loan or mortgage) and Lines of credit (e.g. credit cards)	integer
NumberOfTimes90DaysLate	Number of times borrower has been 90 days or more past due.	integer
NumberRealEstateLoansOrLines	Number of mortgage and real estate loans including home equity lines of credit	integer
NumberOfTime60-89DaysPastDueNotWorse	Number of times borrower has been 60-89 days past due but no worse in the last 2 years.	integer
NumberOfDependents	Number of dependents in family excluding themselves (spouse, children etc.)	integer

* https://www.kaggle.com/c/GiveMeSomeCredit

Пример: Give Me Some Credit*

RevolvingUtilizationOfUnsecuredLines	age	NumberOfTime30-59DaysPastDueNotWorse	DebtRatio	MonthlyIncome	NumberOfOpenCreditLinesAndLoans	NumberOfTimes90DaysLate	NumberRealEstateLoansOrLines	NumberOfTime60-89DaysPastDueNotWorse	NumberOfDependents
0.766126609	45	2	0.802982129	9120	13	0	6	0	2
0.957151019	40	0	0.121876201	2600	4	0	0	0	1
0.65818014	38	1	0.085113375	3042	2	1	0	0	0
0.233809776	30	0	0.036049682	3300	5	0	0	0	0
0.9072394	49	1	0.024925695	63588	7	0	1	0	0
0.213178682	74	0	0.375606969	3500	3	0	1	0	1
0.305682465	57	0	5710	NA	8	0	3	0	0
0.754463648	39	0	0.209940017	3500	8	0	0	0	0
0.116950644	27	0	46	NA	2	0	0	0	NA
0.189169052	57	0	0.606290901	23684	9	0	4	0	2
0.644225962	30	0	0.30947621	2500	5	0	0	0	0
0.01879812	51	0	0.53152876	6501	7	0	2	0	2
0.010351857	46	0	0.298354075	12454	13	0	2	0	2
0.964672555	40	3	0.382964747	13700	9	3	1	1	2
0.019656581	76	0	477	0	6	0	1	0	0
0.548458062	64	0	0.209891754	11362	7	0	1	0	2
0.061086118	78	0	2058	NA	10	0	2	0	0
0.166284079	53	0	0.18827406	8800	7	0	0	0	0
0.221812771	43	0	0.527887839	3280	7	0	1	0	2
0.602794411	25	0	0.065868263	333	2	0	0	0	0

* https://www.kaggle.com/c/GiveMeSomeCredit

Задача снижения размерности

- Представить набор данных меньшим числом признаков таким образом, чтобы потеря информации, содержащейся в оригинальных данных, была минимальной.

Principal Component Analysis

- Данные заданы матрицей $X = (x_i^j)$ размерности $n \times m$, где $i = \overline{1,n}$ и $j = \overline{1,m}$, n – число наблюдений (объектов), m – число признаков.

Principal Component Analysis

Обозначим за *C (m×m)* матрицу ковариаций признаков матрицы *X:*

$$c_{ij} = \frac{\sum_{p=1}^{n} x_k^i x_k^j}{n} - \mu_i \mu_j, \quad \forall i, j \in \{1 \ldots m\},$$

$$\mu_i - среднее\ значение\ признака\ i,\ i \in \{1 \ldots m\}$$

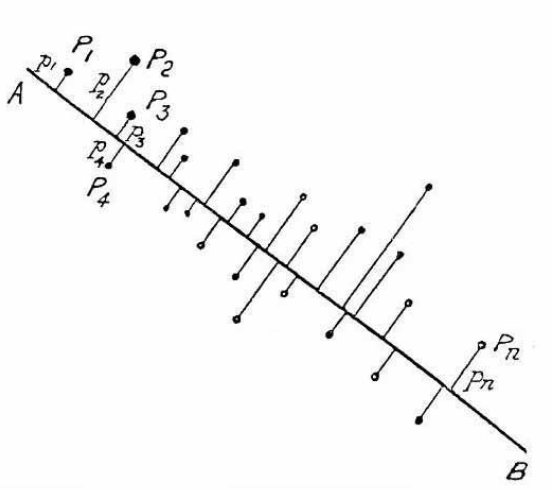
В матричном виде:

$$C = \frac{X^T X}{n} - \mu^T \mu, \qquad \mu = (\mu_1 \ldots \mu_m)$$

Principal Component Analysis

- Вариация *i*-го признака: $Var(x^i) = c_{ii}$
- Общая вариация данных $Var(X) = \sum_{i=1}^{m} c_{ii}$

- Задача: найти ортогональные векторы такие, что $v^T C v \to \max,$ т.е. проекция данных на которые позволит сохранить наибольшую вариацию



Principal Component Analysis

- Матрица *C* симметричная и положительно определена. Имеет место равенство:

$$C = V \Lambda V^T$$

$$\Lambda = \begin{bmatrix} \lambda_1 & 0 & \dots & 0 \\ 0 & \lambda_2 & \dots & 0 \\ \dots & \dots & \dots & \dots \\ 0 & 0 & \dots & \lambda_m \end{bmatrix}, \quad \lambda - \text{собственные значения матрицы } C, \quad \sum_{i=1}^{m} \lambda_i = \sum_{i=1}^{m} c_{ii}$$

$$\lambda_1 > \lambda_2 \geq \dots \geq \lambda_m \geq 0$$

$$V\,(m \times m) - \text{матрица собственных векторов матрицы } C$$

Principal Component Analysis

- Главные компоненты:

$$U = X \cdot \left\lfloor v^1, v^2, \ldots, v^k \right\rfloor^T, \; k < m$$

- Доля объясненной вариации:

$$\frac{\sum_{i=1}^{k} \lambda_i}{Var(X)}$$

Решение в R

- Подготовка R-сессии
- Подготовка данных

```
# Изменение опций по умолчанию
options(digits = 10, "scipen" = 10)

# Установка рабочей директории
setwd("path/to/wd")

# Загрузка пакетов
require(magrittr)
c("data.table", "dplyr", "ggplot2") %>%
   sapply(require, character.only = TRUE)
```

```
# Загрузка данных
X <- data.table::fread("cs-data.csv", drop = c(1), showProgress = FALSE)
# Удаление объектов с пропущенными значениями атрибутов
X <- na.omit(X)
# Размерность данных (число наблюдений и признаков)
dim(X)

[1]  201669   10

# Ранг матрицы X
Matrix::rankMatrix(X)[1]

[1] 10

# Стандартизация данных
Y <- scale(X, center = T, scale = T)
```

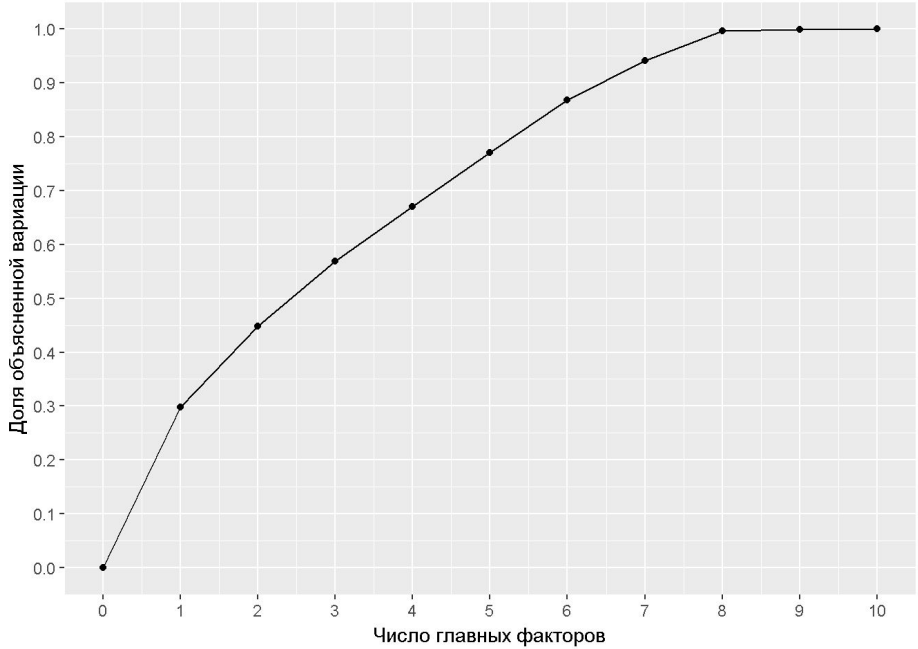
Решение в R

- Для выполнения PCA воспользуемся функцией *princomp* из пакета *stats* (встроен в базовый дистрибутив R)

```
# Вычисляем матрицу ковариаций
Y_cov <- cov.wt(Y)

# Выполняем PCA
Y_pca <- princomp(x = Y, covmat = Y_cov, scores = TRUE)
str(Y_pca)
```

Доля объясненной вариации

```
# Доля объясненной вариации
e_var <- c(0,
          sapply(1:(length(Y_pca$sdev^2)),
                 function(x, y) {sum(y[1:x])/sum(y)},
                 y = Y_pca$sdev^2))

ggplot(mapping = aes(x = 0:(length(e_var)-1), y = e_var)) +
    geom_point() + geom_line() +
    xlab("Число главных факторов") +
    ylab("Доля объясненной вариации") +
    scale_x_continuous(breaks = 0:(length(e_var)-1)) +
    scale_y_continuous(breaks = seq(0, 1, 0.1))
```



Интерпретация главных факторов

```
# Число главных компонент
k <- 6

# Матрица нагрузок
L <- Y_pca$loadings[ ,1:k] %*%
    t(diag(Y_pca$sdev[1:k])) %>%
    as.data.frame

rownames(L) <- colnames(X)
colnames(L) <- paste("u", 1:k, sep = "")

# Округлим значения для удобного просмотра
round(L, 3)
```

	u1	u2	u3	u4	u5	u6
RevolvingUtilizationOfUnsecuredLines	0.001	-0.014	-0.037	0.275	-0.953	0.118
age	0.089	0.345	0.718	0.027	-0.017	-0.043
NumberOfTime30-59DaysPastDueNotWorse	-0.989	0.078	0.011	-0.002	-0.001	0.005
DebtRatio	0.003	0.024	-0.009	-0.838	-0.298	-0.457
MonthlyIncome	0.017	0.218	-0.096	0.472	0.029	-0.847
NumberOfOpenCreditLinesAndLoans	0.117	0.819	0.034	-0.059	0.006	0.137
NumberOfTimes90DaysLate	-0.993	0.053	0.019	0.000	-0.001	0.000
NumberRealEstateLoansOrLines	0.080	0.793	-0.202	-0.045	-0.019	0.119
NumberOfTime60-89DaysPastDueNotWorse	-0.994	0.064	0.021	-0.001	-0.001	0.001
NumberOfDependents	0.000	0.122	-0.804	-0.027	0.033	0.039

Интерпретация главных факторов

- Исходя из структуры матрицы корреляций, можно предложить следующую интерпретацию:
  - U1: История просроченных выплат по кредитам
  - U2: Имеющиеся кредиты
  - U3: Показатель независимости
  - U4: Задолженности
  - U5: Показатель расточительности
  - U6: Доход

Решение в pyDAAL

```
import numpy as np
from sklearn.preprocessing import scale

# Чтение данных
data = np.genfromtxt("cs-data.csv",
                     delimiter = ',',
                     dtype=np.double,
                     skip_header = 1,
                     usecols=list(range(1, 11)))
# Удаление объектов с пропусками
data = data[~np.isnan(data).any(axis = 1)]
# Стандартизация
data = scale(data)
print("Размерность данных \n", data.shape, "\n")

# Матрица ковариаций признаков
cov_data = np.cov(data.transpose())
# Перевод в NumericTable
cov_nt = HomogenNumericTable(cov_data)
```

```
# Выполнение PCA
from daal.algorithms.pca import Batch_Float64CorrelationDense,
                                data, eigenvalues, eigenvectors
algorithm = Batch_Float64CorrelationDense()
algorithm.input.setDataset(data, cov_nt)
result = algorithm.compute()

# Перевод в NumPy объект
loadings = getArrayFromNT(result.get(eigenvectors))
ev = getArrayFromNT(result.get(eigenvalues)

# Вклад каждой компоненты в объяснение вариации
var = np.round(ev/np.sum(ev), decimals=5)

print("Вклад каждой компоненты в объяснение вариации \n ", var, " \n ")
```

```
## Размерность данных
## 201669 10

## Вклад каждой компоненты в объяснение вариации
## [[ 0.38183 0.15477 0.1445 0.11345 0.10845 0.06558 0.03138 0.00003 0.00001 -0. ]]
```

Singular value decomposition

- Данные заданы матрицей $X = (x_i^j)$ размерности $n \times m$, где $i = \overline{1,n}$ и $j = \overline{1,m}$, n – число наблюдений (объектов), m – число признаков.

- Требуется среди всех матриц такого же размера $n \times m$ и ранга $\le k$ найти матрицу Y, для которой норма матрицы $\|X - Y\|$ будет минимальной.

Singular value decomposition

- Решение зависит от матричной нормы
- Наиболее подходящие: Евклидова норма и норма Фробениуса

Евклидова норма:

$$\|A\|_2 = \sqrt{\lambda_{\max}(A^T \cdot A)},$$

где $\lambda_{\max}$ - максимальное собственное значение матрицы A

Норма Фробениуса:

$$\|A\|_F = \sqrt{\sum_i \sum_i a_{ij}^2}$$

Singular value decomposition

Существуют такие матрицы *U* и *V*, что выполняется равенство

$$X = U \cdot S \cdot V^T,$$

где *U* – матрица собственных векторов матрицы $X \cdot X^T$,
V – матрица собственных векторов матрицы $X^T \cdot X$,
а матрица *S* размерности *n*×*m* имеет на главной диагонали элементы $\sigma_1, \sigma_2, \ldots, \sigma_m$ и все остальные σ_i нули, где σ_i^2 - сингулярные числа матрицы *X*, а - собственные числа матрицы

Singular value decomposition

Запишем матрицы *U* и *V* в векторном виде:

$$U = \left\lfloor u^1, u^2, \ldots, u^n \right\rfloor, \; V = \left\lfloor v^1, v^2, \ldots, v^m \right\rfloor$$

Тогда SVD разложение можно представить как

$$X = \sigma_1 u^1 (v^1)^T + \sigma_2 u^2 (v^2)^T + \ldots + \sigma_m u^m (v^m)^T$$

Singular value decomposition

Теорема Шмидта-Мирского:

Решением матричной задачи наилучшей аппроксимации в норме Евклида и в норме Фробениуса является матрица

$$X^* = \sigma_1 u^1 (v^1)^T + \sigma_2 u^2 (v^2)^T + \ldots + \sigma_k u^k (v^k)^T$$

Ошибки аппроксимации:

$$\|X - X^*\|_2 = \sigma_{k+1},$$

$$\|X - X^*\|_F = \sqrt{\sigma_{k+1}^2 + \sigma_{k+2}^2 + \ldots + \sigma_m^2}\,.$$

Выбор числа *k* главных факторов

- Общая вариация данных:

$$Var(X) = \sigma_1^2 + \sigma_2^2 + \ldots + \sigma_m^2$$

- Доля объясненной вариации:

$$\frac{\sigma_1^2 + \sigma_2^2 + \ldots + \sigma_k^2}{Var(X)}, k < m$$

- Хорошим значением считается доля объясненной вариации ≥ 80%

Решение в R

- Для выполнения SVD разложения воспользуемся функцией *svd()* из пакета *stats* (встроен в базовый дистрибутив R)

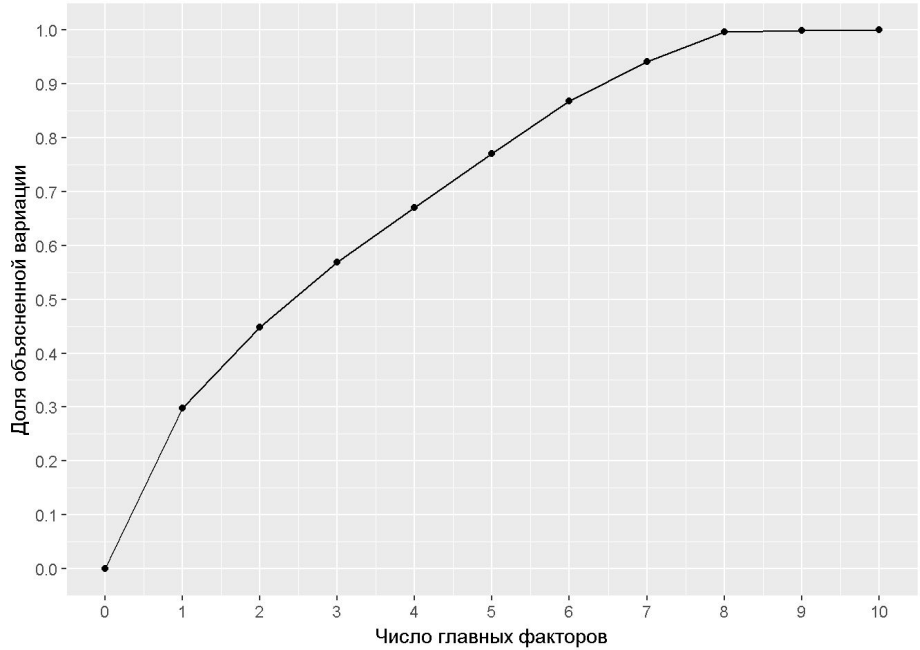
```
Y_svd <- svd(Y)
str(Y_svd)
```

- Функция возвращает объект класса *list*, состоящий из трех элементов:
  - d – сингулярные числа $\sigma_1, \sigma_2, ..., \sigma_m$ матрицы Y
  - u – матрица правых собственных векторов (матрицы $X^T \cdot X$)
  - v – матрица левых собственных векторов (матрицы $X \cdot X^T$)

```
## List of 3
## $ d: num [1:10] 775 549 495 451 449 ...
## $ u: num [1:201669, 1:10] 0.0000588 -0.0000238 -0.0004799 -0.0000522 -0.0000534 ...
## $ v: num [1:10, 1:10] 0.000416 0.051466 -0.572698 0.001871 0.009995 ...
```

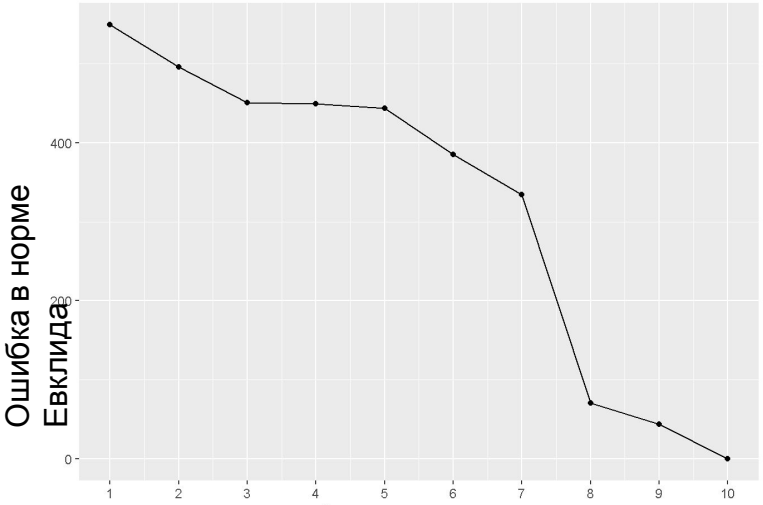
Доля объясненной вариации

```
e_var <- c(0,
           sapply(1:(length(Y_svd$d^2)),
           function(x, y) {sum(y[1:x])/sum(y)},
           y = Y_svd$d^2))

ggplot(mapping = aes(x = 0:(length(e_var)-1), y = e_var)) +
  geom_point() + geom_line() +
  xlab("Число главных факторов") +
  ylab("Доля объясненной вариации") +
  scale_x_continuous(breaks = 0:(length(e_var)-1)) +
  scale_y_continuous(breaks = seq(0, 1, 0.1))
```

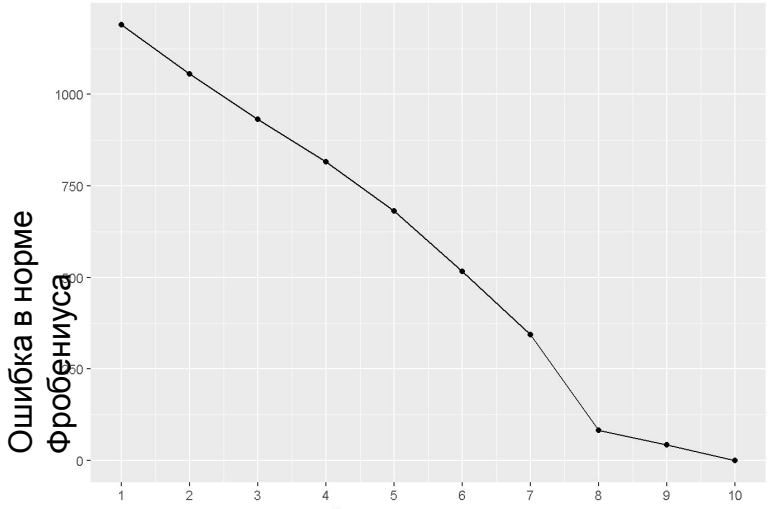


Ошибки аппроксимации

```
## Ошибка аппроксимации в норме Евклида
err2 <- c(Y_svd$d[2:length(Y_svd$d)], 0)

ggplot(mapping = aes(x = 1:length(err2), y = err2)) +
  geom_point() +
  geom_line() +
  xlab("Число главных компонент") +
  ylab("Ошибка в норме Евклида") +
  scale_x_continuous(breaks = 1:length(err2))
```




```
## Ошибка аппроксимации в норме Фробениуса
errF <- c(sapply(1:(length(Y_svd$d^2)-1),
                 function(x, y) {sqrt(sum(y[(x+1):length(y)]))},
                 y = Y_svd$d^2),
          0)
ggplot(mapping = aes(x = 1:length(errF), y = errF)) +
  geom_point() +
  geom_line() +
  xlab("Число главных компонент") +
  ylab("Ошибка в норме Фробениуса") +
  scale_x_continuous(breaks = 1:length(errF))
```



Интерпретация главных факторов

Результаты, полученные методом svd, совпадают с результатами, полученными после применения pca.

```
# Матрица нагрузок
L <- Y_svd$v[ ,1:k] %*% t(diag(Y_svd$d[1:k]))

# Переход к матрице корреляций
L <- L/apply(Y, 2, norm, '2') %>%
    as.data.frame

rownames(L) <- colnames(X)
colnames(L) <- paste("u", 1:k, sep = "")

# Округлим значения для удобного просмотра
round(L, 3)
```

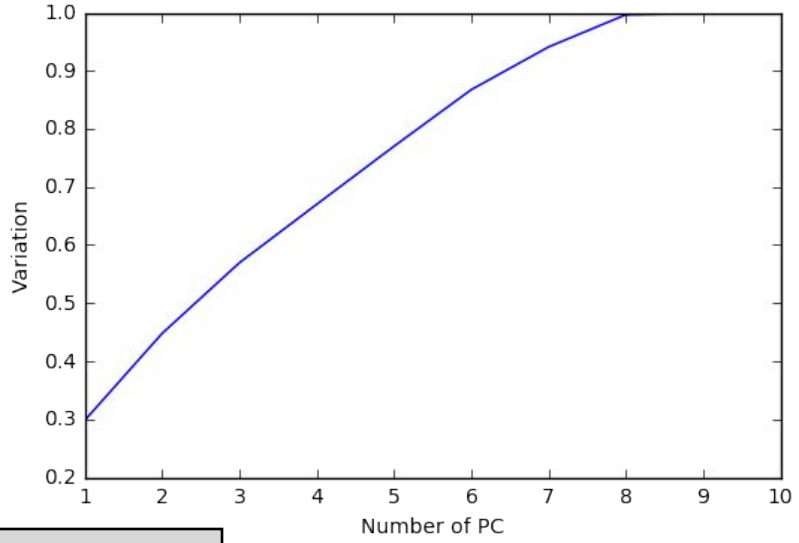
	u1	u2	u3	u4	u5	u6
RevolvingUtilizationOfUnsecuredLines	0.001	-0.014	-0.037	0.275	-0.953	0.118
age	0.089	0.345	0.718	0.027	-0.017	-0.043
NumberOfTime30-59DaysPastDueNotWorse	-0.989	0.078	0.011	-0.002	-0.001	0.005
DebtRatio	0.003	0.024	-0.009	-0.838	-0.298	-0.457
MonthlyIncome	0.017	0.218	-0.096	0.472	0.029	-0.847
NumberOfOpenCreditLinesAndLoans	0.117	0.819	0.034	-0.059	0.006	0.137
NumberOfTimes90DaysLate	-0.993	0.053	0.019	0.000	-0.001	0.000
NumberRealEstateLoansOrLines	0.080	0.793	-0.202	-0.045	-0.019	0.119
NumberOfTime60-89DaysPastDueNotWorse	-0.994	0.064	0.021	-0.001	-0.001	0.001
NumberOfDependents	0.000	0.122	-0.804	-0.027	0.033	0.039

Решение в scikit-learn

```
%matplotlib inline
import numpy as np
import scipy as sp
from sklearn.decomposition import PCA
import matplotlib.pyplot as plt
from sklearn.preprocessing import scale
import time

np.set_printoptions(precision=10,
                    threshold = 10000,
                    suppress = True)

# Загружаем данные
data = np.genfromtxt("cs-data.csv", delimiter = ',',
                     skip_header = 1, usecols=list(range(1, 11)))
# Удаляем наблюдения с пропущенными значениями
data = data[~np.isnan(data).any(axis = 1)]
# Приводим к стандартному виду
data = scale(data)

print("Размерность данных \n", data.shape, "\n")
```

Решение в scikit-learn

```
# Выполняем метод главных компонент
pca = PCA(svd_solver='full')
pca.fit(data)

# Вклад каждого фактора в объяснение вариации
print("Вклад каждого фактора в объяснение вариации \n",
      pca.explained_variance_ratio_, "\n")
# Рост доли объясненной вариации с увеличением числа главных факторов
var = np.round(np.cumsum(pca.explained_variance_ratio_), decimals=4)

plt.plot(np.arange(1,11), var)
plt.ylabel('Variation')
plt.xlabel('Number of PC')
```




```
## Размерность данных
## (201669, 10)

## Вклад каждого фактора в объяснение вариации
## [ 0.2979766397 0.1496007962 0.1217110055 0.1007219879 0.0999739517  0.0975640598 0.0735527529
      0.055468798  0.0024871325 0.0009428757]

## Рост доли объясненной вариации с увеличением числа главных факторов
## [ 0.298  0.4476  0.5693  0.67  0.77  0.8675  0.9411  0.9966  0.9991  1 ]
```

Решение в pyDAAL

```
import numpy as np
from sklearn.preprocessing import scale

# Чтение данных
data = np.genfromtxt("cs-data.csv", delimiter = ',',
                                dtype=np.double,
                                skip_header = 1,
                                usecols=list(range(1, 11)))
data = data[~np.isnan(data).any(axis = 1)]
data = scale(data)

data_nt = HomogenNumericTable(data)
print("Размерность данных \n",
        data_nt.getNumberOfRows(),
        data_nt.getNumberOfColumns(), "\n")
```

```
# Выполнение PCA
from daal.algorithms.pca import Batch_Float64SvdDense,
                                        data, eigenvalues, eigenvectors
algorithm = Batch_Float64SvdDense()
algorithm.input.setDataset(data, data_nt)
result = algorithm.compute()

# Перевод в NumPy объект
loadings = getArrayFromNT(result.get(eigenvectors))
ev = getArrayFromNT(result.get(eigenvalues)

print("Вклад каждого фактора в объяснение вариации \n", ev/np.sum(ev), "\n")
      var = np.round(np.cumsum(ev/np.sum(ev)), decimals=4)
print("Рост доли объясненной вариации с увеличением числа главных факторов \n",
       var, "\n")
```

```
## Размерность данных
## 201669 10

## Вклад каждого фактора в объяснение вариации
## [[ 0.29797664   0.1496008   0.12171101   0.10072199   0.09997395   0.09756406   0.07355275   0.0554688
     0.00248713   0.00094288]]

## Рост доли объясненной вариации с увеличением числа главных факторов
## [ 0.298  0.4476  0.5693  0.67  0.77  0.8675  0.9411  0.9966  0.9991  1 ]
```

Latent Semantic Analysis

- Document-term matrix X: строки – документы, столбцы – слова (после предобработки, нормализации, удаления стоп-слов)
- Элементы матрицы x_{dt} :
  - Term frequency (tf): сколько раз слово *t* встречается в документе *d* или производная от этого величина
  - TF-IDF: $tfidf(t,d,D) = tf(t,d) \cdot idf(t,D),$

$$idf(t,D) = \log \frac{N}{1 + \left|\{d \in D : t \in d\}\right|}, \text{ где } D - \text{множество документов}, N = \left|D\right|$$

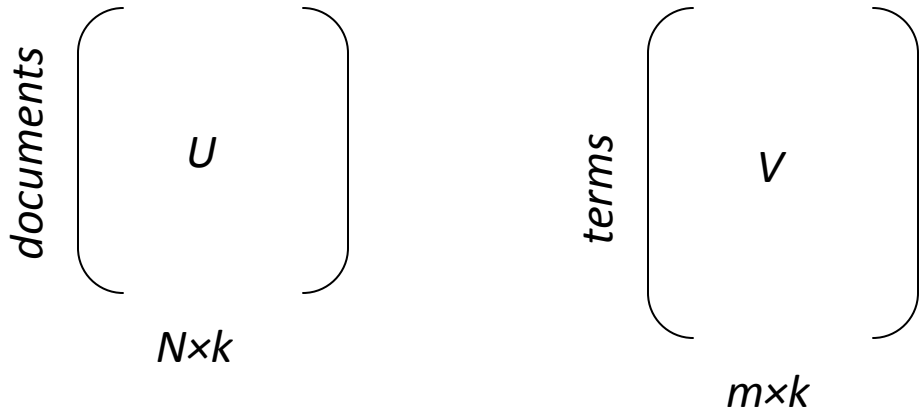
Latent Semantic Analysis

- Задача: представить документы в пространстве *k* признаков, где *k* много меньше размера словаря, с максимальным сохранением информации.
- Решение: применить SVD к document-term матрице и взять первые *k* столбцов матриц *U* и *V*.

NB: Также данным подходом частично решается проблема синонимов и полисемии.

Latent Semantic Analysis

- Применяя SVD к document-term матрице, мы одновременной находим представление в *k*-мерном пространстве как для документов, так и для слов



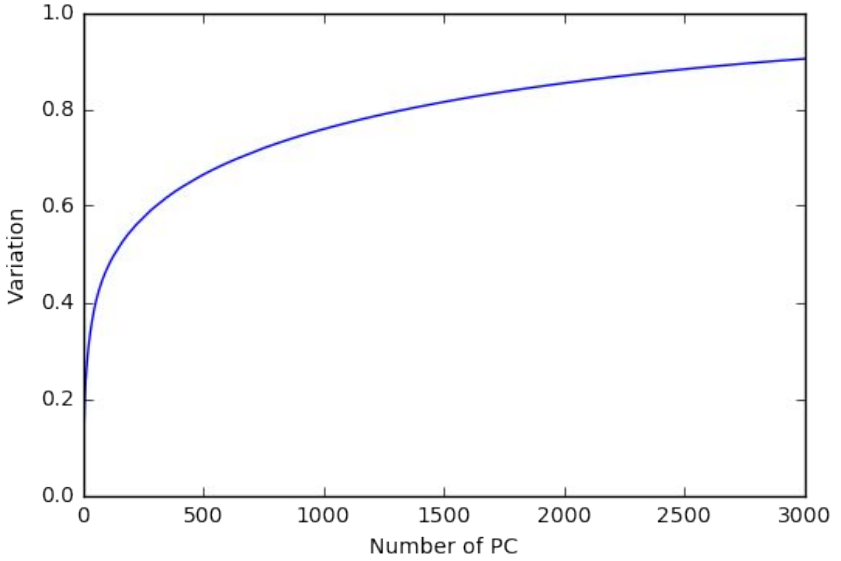
Пример: 20 Newsgroups

- Набор новостных статей «20 Newsgroups»
- 18000 новостных статей из 20 различных рубрик.
- URL: http://qwone.com/~jason/20Newsgroups/

Решение в scikit-learn

```
# Загрузка данных
from sklearn.datasets import fetch_20newsgroups_vectorized
newsgroups = fetch_20newsgroups_vectorized(subset='train',
                                           remove = ('headers',
                                                     'footers',
                                                     'quotes'))
# Размерность данных
print("Размерность данных \n", newsgroups.data, "\n")
# Применяем SVD к данным
from sklearn.decomposition import TruncatedSVD
svd = TruncatedSVD(n_components = 3000, algorithm = "randomized")
t = time.process_time()
svd.fit(newsgroups.data)
t = time.process_time() – t
# Доля объясненной вариации
print(" Доля объясненной вариации \n", svd.explained_variance_ratio_.sum(), "\n")
# График роста доли объясненной вариации
var_nwsd = np.round(np.cumsum(svd.explained_variance_ratio_), decimals=4)
plt.plot(np.arange(1,3001), var_nwsd)
plt.ylabel('Variation'); plt.xlabel('Number of PC')
# Время выполнения
print("Время выполнения (секунд) \n", t, "\n")
```

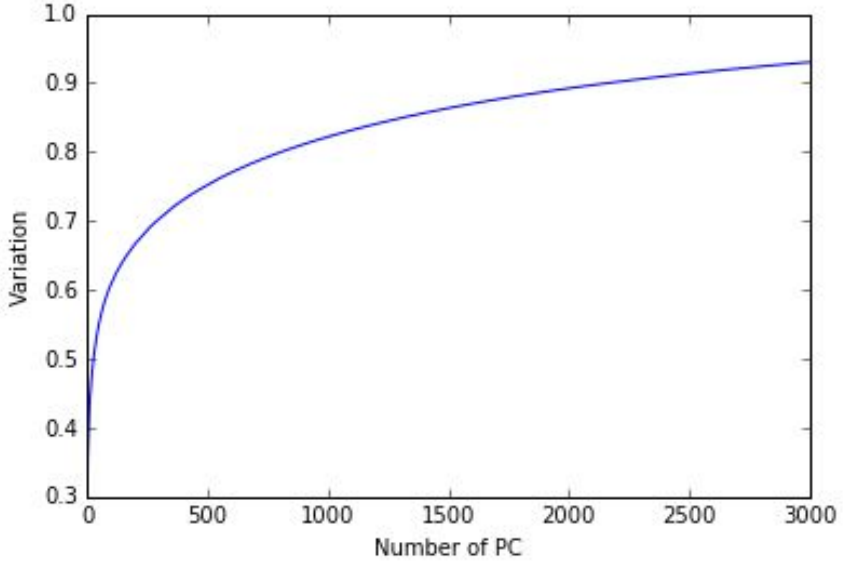
```
## Размерность данных
## <11314x101631 sparse matrix of type '<class 'numpy.float64'>'
##                 with 1103627 stored elements in Compressed Sparse Row format>
## Доля объясненной вариации
## 0.9055
## Время выполнения (секунд)
## 1611.6875
```



Решение в pyDAAL

```
# Транспонируем матрицу данных и переводим из sparse в dense
nwsd = newsgroups.data
nwsd = nwsd.transpose()
nwsd_dense = nwsd.toarray()
print("Размерность данных \n", nwsd_dense.shape, "\n")
# Перевод в NumericTable
nwsd_dense_nt = HomogenNumericTable(nwsd_dense)
# Выполнение SVD
from daal.algorithms.svd import Batch, data, singularValues, rightSingularMatrix, leftSingularMatrix
algorithm = Batch()
algorithm.input.set(data, nwsd_dense_nt)
t = time.process_time()
result = algorithm.compute()
t = time.process_time() – t
# Доля объясненной вариации
ev = np.square(getArrayFromNT(result.get(singularValues)))
var_all = ev.sum()
var = ev/var_all
var_explained = np.round(np.cumsum(var), decimals=4)
print(" Доля объясненной вариации \n", var_explained[2999].sum(), "\n")
plt.plot(np.arange(1,3000), var_explained[0:2999])
plt.ylabel('Variation'); plt.xlabel('Number of PC')
# Время выполнения
print("Время выполнения (секунд) \n", t, "\n")
```

```
## Размерность данных
##    (101631, 11314)
## Доля объясненной вариации
## 0.9303
## Время выполнения (секунд)
## 11688.90625
```



Пример: 20 Newsgroups

- k = 3000 главных факторов дает долю объясненной вариации ~ 90%
- k = 1500 главных факторов дает долю объясненной вариации ~ 80%
- При помощи SVD удалось эффективно снизить размерность пространства признаков в 30-60 раз.

Скорость вычислений

Скорость выполнения метода главных компонент в R, Scikit-learn и pyDAAL для набора данных Give Me Some Credit на 1000 запусков.

	Median time (seconds)
R: svd()	0.09
R: princomp()	0.16
Scikit-learn: sklearn.decomposition.PCA	0.61
pyDAAL: pca_svd_dense_batch	0.11
pyDAAL: pca_correlation_dense_batch	$< 10^{-8}$

Задания

1. Воспроизведите вычисления, представленные в лекционных материалах. Подтвердите выводы.
2. Рассмотрите набор данных Turkiye Student Evaluation:
   a) Опишите исследуемые данные
   b) Выберите данные по одному предмету (class) и выполните анализ главных компонент. Выделите главные факторы, дайте интерпретацию (или покажите, что этого сделать нельзя).
   c) Выберите два предмета, которые проводил один и тот же преподаватель. Снова выполните анализ главных компонент, выделите главные факторы, постарайтесь дать интерпретацию. Сравните результаты с предыдущим пунктом.
   d) Выполните PCA для всего набора данных. Также сравните результаты с пунктами выше.
   e) Повторите вычисления из пунктов b - d, но для нестандартизованных данных. Сравните с соответствующими результатами, полученными на стандартизованных данных.
3. Сравните время выполнения SVD в R, Scikit-learn и DAAL на полном наборе данных Turkiye Student Evaluation.

Приложение

```
from daal.data_management import HomogenNumericTable, BlockDescriptor_Float64, readOnly
import numpy as np

# Определим необходимые функции

# Данная функция переводит из NumericTable в numpy array
def getArrayFromNT(table, nrows=0):
    bd = BlockDescriptor_Float64()
    if nrows == 0:
        nrows = table.getNumberOfRows()
    table.getBlockOfRows(0, nrows, readOnly, bd)
    npa = bd.getArray()
    table.releaseBlockOfRows(bd)
    return npa

# Вывод NumericTable в консоль
def printNT(table, nrows = 0, message=''):
    npa = getArrayFromNT(table, nrows)
    print(message, '\n', npa)
```