Aula 19 - Redução de Dimensionalidade

João Florindo

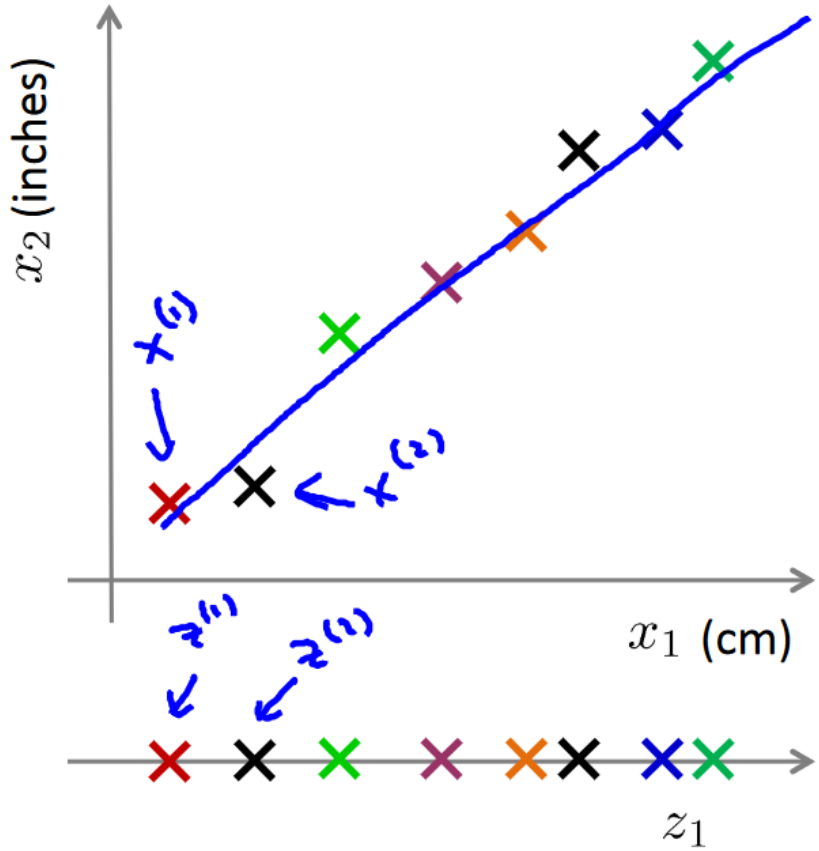
Instituto de Matemática, Estatística e Computação Científica
Universidade Estadual de Campinas - Brasil
florindo@unicamp.br

Outline

1. Motivação
2. Análise de Componentes Principais
3. Aplicação

Motivação I - Compressão

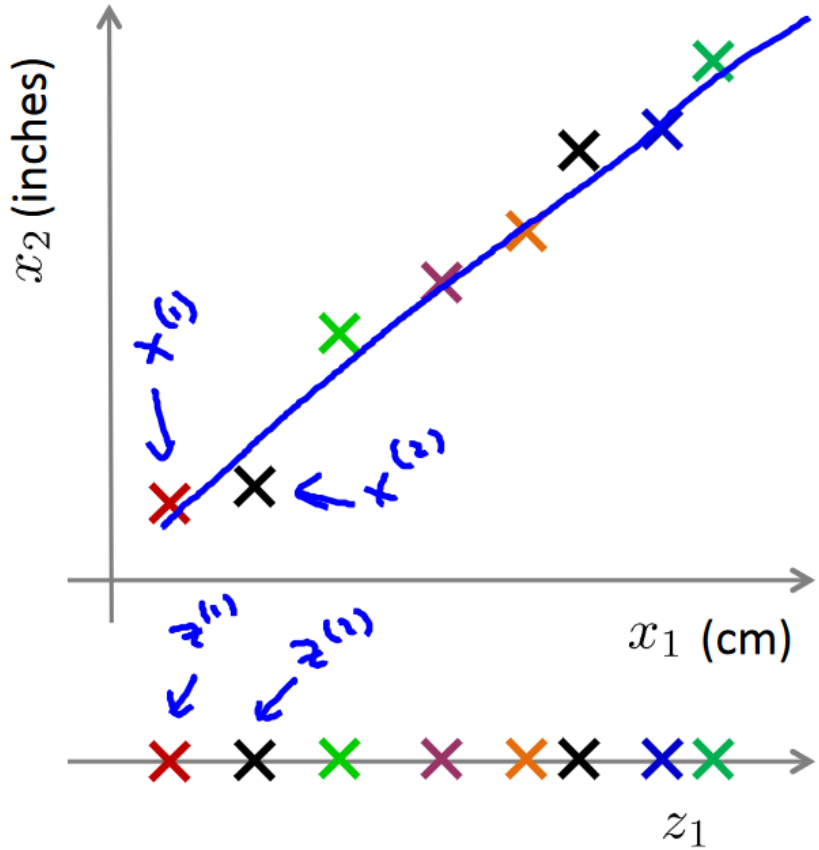
- Imagine que x_1 é um comprimento em cm e x_2 em polegadas:



- Mais realista: números de portas e de cômodos, horas de estudo e nota na prova, cor da roupa para prever o peso, etc.

Motivação I - Compressão

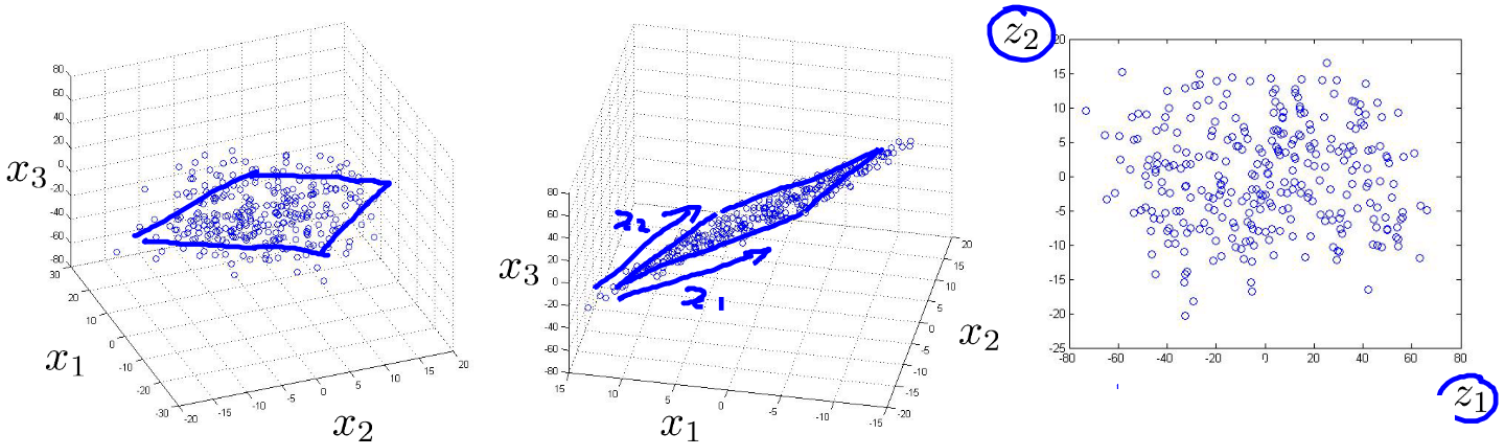
- Imagine que x_1 é um comprimento em cm e x_2 em polegadas:



- Mais realista: números de portas e de cômodos, horas de estudo e nota na prova, cor da roupa para prever o peso, etc.

Motivação I - Compressão

- Mesmo processo de 3D para 2D ou qualquer dimensão:



Motivação II - Visualização

- Atributos $x^{(i)} \in \mathbb{R}^{50}$ de país:

Country	x_1 GDP (trillions of US$)	x_2 Per capita GDP (thousands of intl. $)	x_3 Human Develop-ment Index	x_4 Life expectancy	x_5 Poverty Index (Gini as percentage)	x_6 Mean household income (thousands of US$)	...
Canada	1.577	39.17	0.908	80.7	32.6	67.293	...
China	5.878	7.54	0.687	73	46.9	10.22	...
India	1.632	3.41	0.547	64.7	36.8	0.735	...
Russia	1.48	19.84	0.755	65.5	39.9	0.72	...
Singapore	0.223	56.69	0.866	80	42.5	67.1	...
USA	14.527	46.86	0.91	78.3	40.8	84.3	...
...	...	...	...	...	...	...	

[resources from en.wikipedia.org]

- Para **visualização**, reduzimos para $z^{(i)} \in \mathbb{R}^2$:

País	z_1	z_2
Canadá	1.6	1.2
China	1.7	0.3
Índia	1.6	0.2
Rússia	1.4	0.5
Singapura	0.5	1.7
EUA	2	1.5
...	...	...

Motivação II - Visualização

- Atributos $x^{(i)} \in \mathbb{R}^{50}$ de país:

Country	x_1 GDP (trillions of US$)	x_2 Per capita GDP (thousands of intl. $)	x_3 Human Develop-ment Index	x_4 Life expectancy	x_5 Poverty Index (Gini as percentage)	x_6 Mean household income (thousands of US$)	...
Canada	1.577	39.17	0.908	80.7	32.6	67.293	...
China	5.878	7.54	0.687	73	46.9	10.22	...
India	1.632	3.41	0.547	64.7	36.8	0.735	...
Russia	1.48	19.84	0.755	65.5	39.9	0.72	...
Singapore	0.223	56.69	0.866	80	42.5	67.1	...
USA	14.527	46.86	0.91	78.3	40.8	84.3	...
...	...	...	...	...	...	...	

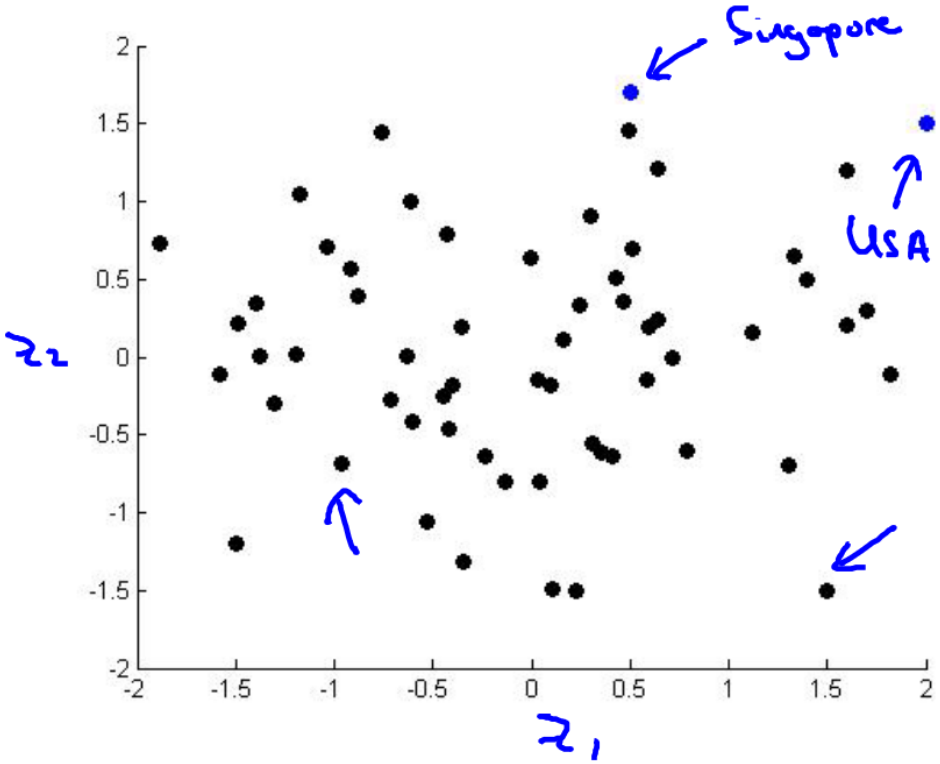
[resources from en.wikipedia.org]

- Para **visualização**, reduzimos para $z^{(i)} \in \mathbb{R}^2$:

País	z_1	z_2
Canadá	1.6	1.2
China	1.7	0.3
Índia	1.6	0.2
Rússia	1.4	0.5
Singapura	0.5	1.7
EUA	2	1.5
...	...	...

Motivação II - Visualização

- Aqui z_1 se relaciona com o PIB e z_2 com o PIB *per capita*:



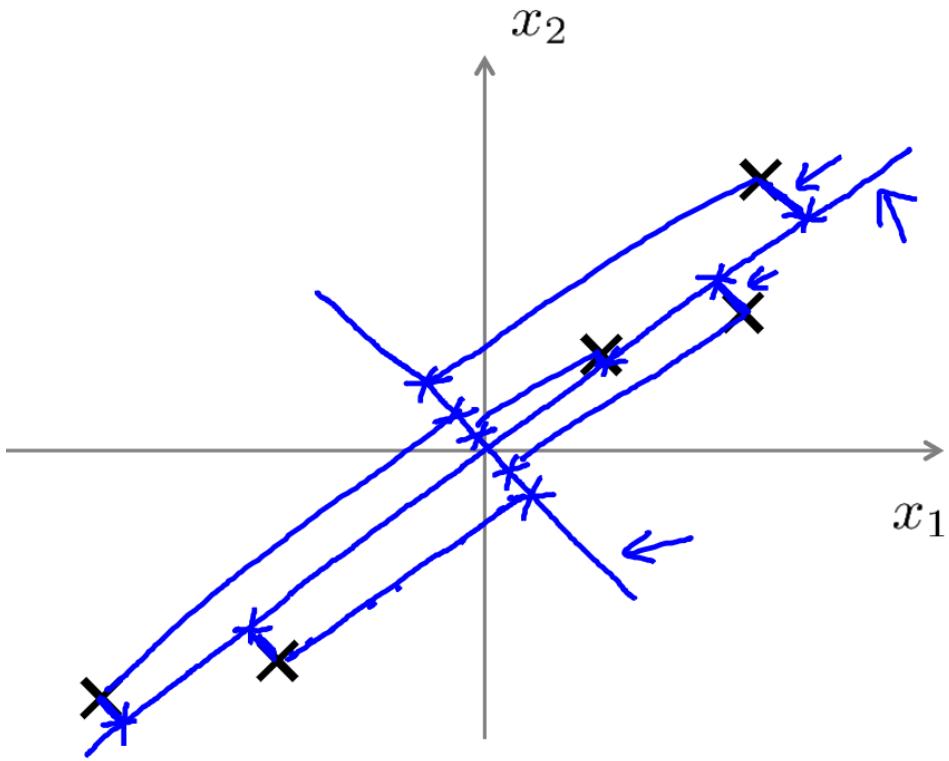
Outline

1 Motivação

2 Análise de Componentes Principais

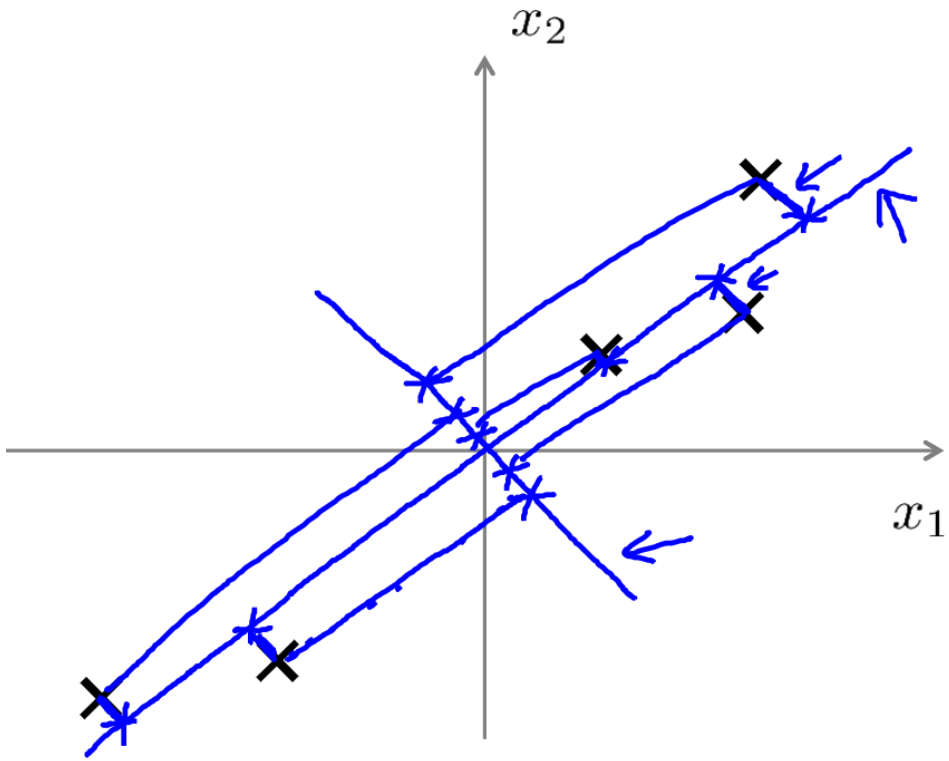
3 Aplicação

- EX.: Projetar dado $x \in \mathbb{R}^2$ para o vetor $u^{(1)} \in \mathbb{R}^n$: minimizar a soma das distâncias ao quadrado de cada ponto em x a u (**erro de projeção**).



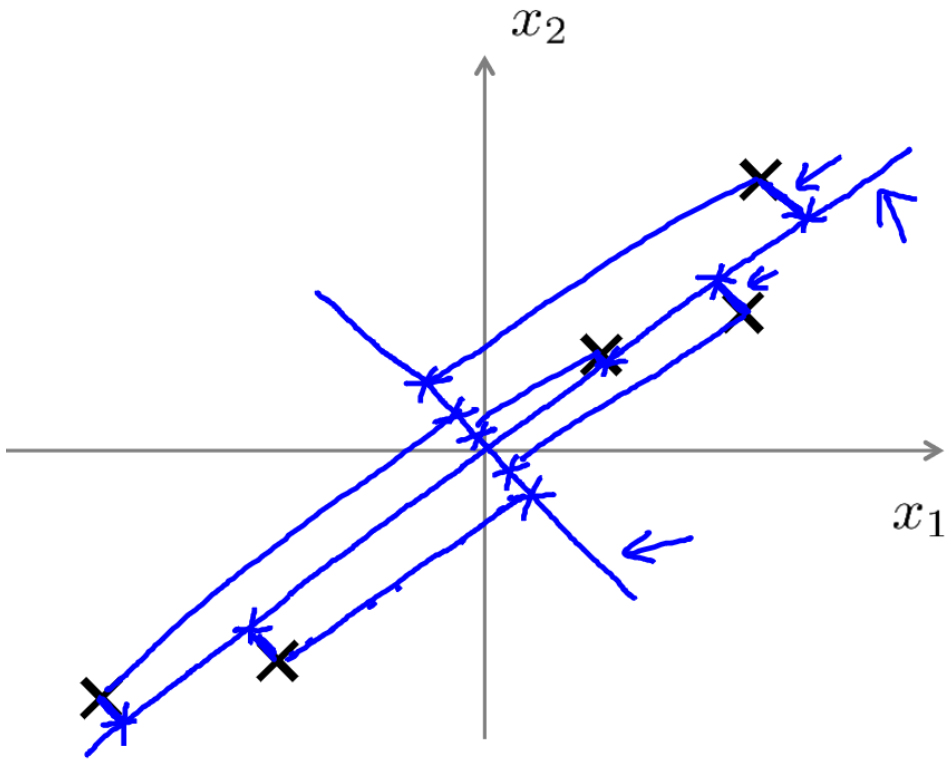
- Maximiza variância (preserva o máximo da variância original) - distância da projeção à origem.
- Em geral, reduzimos de n para k dimensões.
- Apenas a direção do vetor u importa.

- EX.: Projetar dado $x \in \mathbb{R}^2$ para o vetor $u^{(1)} \in \mathbb{R}^n$: minimizar a soma das distâncias ao quadrado de cada ponto em x a u (**erro de projeção**).



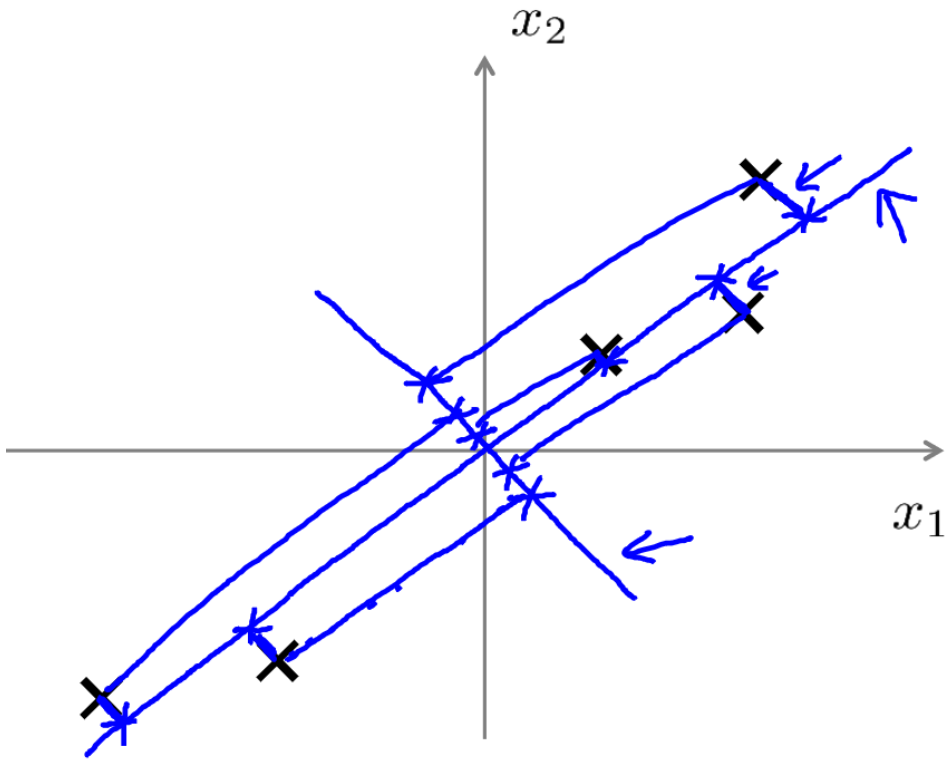
- Maximiza variância (preserva o máximo da variância original) - distância da projeção à origem.
- Em geral, reduzimos de n para k dimensões.
- Apenas a direção do vetor u importa.

- EX.: Projetar dado $x \in \mathbb{R}^2$ para o vetor $u^{(1)} \in \mathbb{R}^n$: minimizar a soma das distâncias ao quadrado de cada ponto em x a u (**erro de projeção**).



- Maximiza variância (preserva o máximo da variância original) - distância da projeção à origem.
- Em geral, reduzimos de n para k dimensões.
- Apenas a direção do vetor u importa.

- EX.: Projetar dado $x \in \mathbb{R}^2$ para o vetor $u^{(1)} \in \mathbb{R}^n$: minimizar a soma das distâncias ao quadrado de cada ponto em x a u (**erro de projeção**).

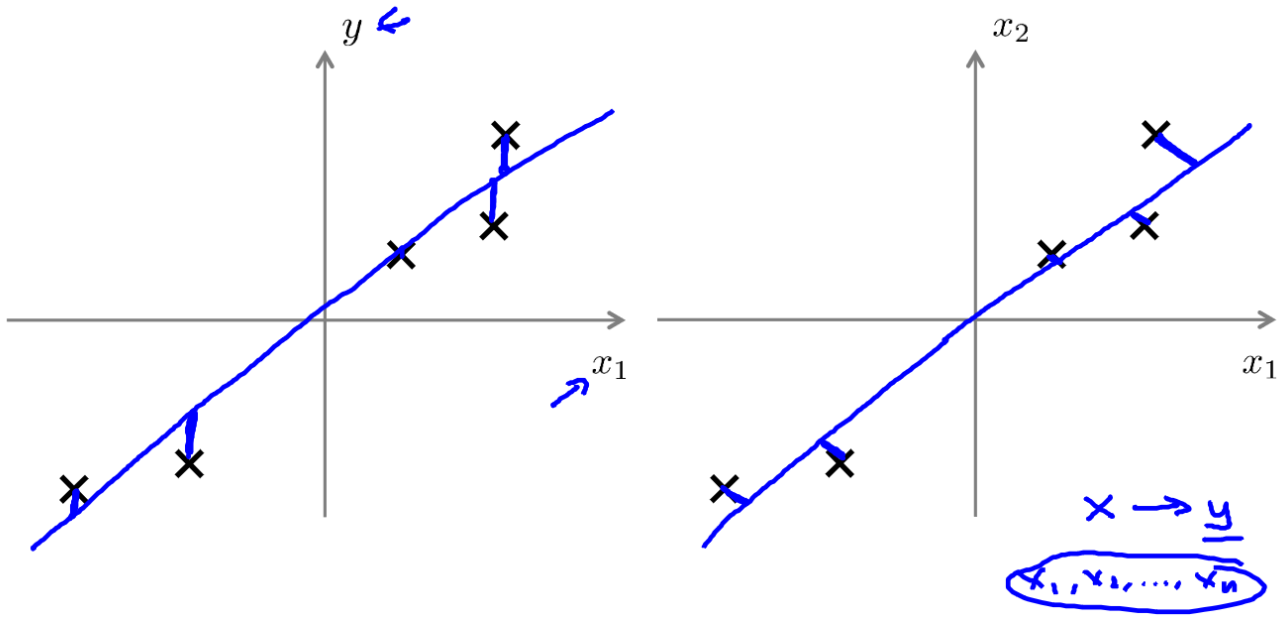


- Maximiza variância (preserva o máximo da variância original) - distância da projeção à origem.
- Em geral, reduzimos de n para k dimensões.
- Apenas a direção do vetor u importa.

NOTA

PCA **NÃO** é regressão linear:

1. PCA minimiza a distância PERPENDICULAR ao vetor/reta, enquanto a regressão linear minimiza a distância paralela ao eixo y.
2. Na regressão linear temos uma variável de resposta diferenciada y, enquanto no PCA todas as variáveis estão no mesmo nível.



Algoritmo

- Treinamento $\{x^{(1)}, x^{(2)}, \cdots, x^{(m)}\}$ (não supervisionado).
- Normalização por média/escala:

$$\mu_j = \frac{1}{m} \sum_{i=1}^{m} x_j^{(i)}$$

e substituímos cada $x_j^{(i)}$ por $x_j^{(i)} - \mu_j$.

- Se os atributos estão em escalas diferentes, normalizar escala:

$$x_j^{(i)} \leftarrow \frac{x_j^{(i)} - \mu_j}{s_j}$$

em que s_j pode ser $max(x_j) - min(x_j)$, mas mais comumente é

$$s_j = std(x_j).$$

Algoritmo

- Treinamento $\{x^{(1)}, x^{(2)}, \cdots, x^{(m)}\}$ (não supervisionado).
- Normalização por média/escala:

$$\mu_j = \frac{1}{m} \sum_{i=1}^{m} x_j^{(i)}$$

e substituímos cada $x_j^{(i)}$ por $x_j^{(i)} - \mu_j$.

- Se os atributos estão em escalas diferentes, normalizar escala:

$$x_j^{(i)} \leftarrow \frac{x_j^{(i)} - \mu_j}{s_j}$$

em que s_j pode ser $max(x_j) - min(x_j)$, mas mais comumente é

$$s_j = std(x_j).$$

Algoritmo

- Treinamento $\{x^{(1)}, x^{(2)}, \cdots, x^{(m)}\}$ (não supervisionado).
- Normalização por média/escala:

$$\mu_j = \frac{1}{m} \sum_{i=1}^{m} x_j^{(i)}$$

e substituímos cada $x_j^{(i)}$ por $x_j^{(i)} - \mu_j$.

- Se os atributos estão em escalas diferentes, normalizar escala:

$$x_j^{(i)} \leftarrow \frac{x_j^{(i)} - \mu_j}{s_j}$$

em que s_j pode ser $max(x_j) - min(x_j)$, mas mais comumente é

$$s_j = std(x_j).$$

Algoritmo

- Lembre-se que a distância da projeção de $x^{(i)}$ sobre u à origem é $x^T u$.
- Tomamos u como vetor unitário e devemos então maximizar

$$\frac{1}{m}\sum_{i=1}^{m}({x^{(i)}}^T u)^2 = \frac{1}{m}\sum_{i=1}^{m} u^T x^{(i)} {x^{(i)}}^T u$$
$$= u^T \left(\frac{1}{m}\sum_{i=1}^{m} x^{(i)} {x^{(i)}}^T\right) u,$$

  sujeito à restrição $\|u\|_2 = 1$.
- Mostra-se que a solução é o auto-vetor principal de $\frac{1}{m}\sum_{i=1}^{m} x^{(i)} {x^{(i)}}^T$.
- Mesmo raciocínio se estende para dimensões maiores.

Algoritmo

- Lembre-se que a distância da projeção de $x^{(i)}$ sobre u à origem é $x^T u$.
- Tomamos u como vetor unitário e devemos então maximizar

$$\frac{1}{m}\sum_{i=1}^{m}({x^{(i)}}^T u)^2 = \frac{1}{m}\sum_{i=1}^{m} u^T x^{(i)} {x^{(i)}}^T u$$
$$= u^T \left(\frac{1}{m}\sum_{i=1}^{m} x^{(i)} {x^{(i)}}^T\right) u,$$

  sujeito à restrição $\|u\|_2 = 1$.

- Mostra-se que a solução é o auto-vetor principal de $\frac{1}{m}\sum_{i=1}^{m} x^{(i)} {x^{(i)}}^T$.
- Mesmo raciocínio se estende para dimensões maiores.

Algoritmo

- Lembre-se que a distância da projeção de $x^{(i)}$ sobre u à origem é $x^T u$.
- Tomamos u como vetor unitário e devemos então maximizar

$$\frac{1}{m}\sum_{i=1}^{m}({x^{(i)}}^T u)^2 = \frac{1}{m}\sum_{i=1}^{m} u^T x^{(i)} {x^{(i)}}^T u$$
$$= u^T \left(\frac{1}{m}\sum_{i=1}^{m} x^{(i)} {x^{(i)}}^T\right) u,$$

  sujeito à restrição $\|u\|_2 = 1$.

- Mostra-se que a solução é o auto-vetor principal de $\frac{1}{m}\sum_{i=1}^{m} x^{(i)} {x^{(i)}}^T$.
- Mesmo raciocínio se estende para dimensões maiores.

Algoritmo

- Lembre-se que a distância da projeção de $x^{(i)}$ sobre u à origem é $x^T u$.
- Tomamos u como vetor unitário e devemos então maximizar

$$\frac{1}{m}\sum_{i=1}^{m}({x^{(i)}}^T u)^2 = \frac{1}{m}\sum_{i=1}^{m} u^T x^{(i)} {x^{(i)}}^T u$$
$$= u^T \left(\frac{1}{m}\sum_{i=1}^{m} x^{(i)} {x^{(i)}}^T\right) u,$$

  sujeito à restrição $\|u\|_2 = 1$.

- Mostra-se que a solução é o auto-vetor principal de $\frac{1}{m}\sum_{i=1}^{m} x^{(i)} {x^{(i)}}^T$.
- Mesmo raciocínio se estende para dimensões maiores.

Algoritmo

- **Matriz de covariância** empírica (dado que $x^{(i)}$ tem média zero):

$$\Sigma = \frac{1}{m}\sum_{i=1}^{n}(x^{(i)})(x^{(i)})^T.$$

- Calcular os "auto-vetores" da matriz Σ. No Python:

```
U,S,V = svd(Sigma)
```

NOTA

Poderia também usar a função *eig*. Mas o SVD (*Singular Value Decomposition*) é mais estável em casos gerais. No caso específico do PCA, porém, são equivalentes pois Σ é *semi-positiva definida*.

Algoritmo

- **Matriz de covariância** empírica (dado que $x^{(i)}$ tem média zero):

$$\Sigma = \frac{1}{m}\sum_{i=1}^{n}(x^{(i)})(x^{(i)})^T.$$

- Calcular os "auto-vetores" da matriz Σ. No Python:

```
U,S,V = svd(Sigma)
```

NOTA

Poderia também usar a função *eig*. Mas o SVD (*Singular Value Decomposition*) é mais estável em casos gerais. No caso específico do PCA, porém, são equivalentes pois Σ é *semi-positiva definida*.

Algoritmo

- **Matriz de covariância** empírica (dado que $x^{(i)}$ tem média zero):

$$\Sigma = \frac{1}{m}\sum_{i=1}^{n}(x^{(i)})(x^{(i)})^T.$$

- Calcular os "auto-vetores" da matriz Σ. No Python:

```
U,S,V = svd(Sigma)
```

NOTA

Poderia também usar a função *eig*. Mas o SVD (*Singular Value Decomposition*) é mais estável em casos gerais. No caso específico do PCA, porém, são equivalentes pois Σ é *semi-positiva definida*.

Algoritmo

- Os auto-vetores são armazenados na matriz U (pela ordem dos auto-valores):

$$U = \begin{bmatrix} | & | & & | \\ u^{(1)} & u^{(2)} & \dots & u^{(n)} \\ | & | & & | \end{bmatrix} \in \mathbb{R}^{n \times n}.$$

- Definimos a matriz $U_{reduzida}$ com as k primeiras colunas de U e o vetor $z^{(i)}$:

$$z^{(i)} = U_{reduzida}^T x^{(i)} = \begin{bmatrix} | & | & & | \\ u^{(1)} & u^{(2)} & \dots & u^{(k)} \\ | & | & & | \end{bmatrix}^T x^{(i)}.$$

Note que $z \in \mathbb{R}^k$, como desejado.

Algoritmo

- Os auto-vetores são armazenados na matriz U (pela ordem dos auto-valores):

$$U = \begin{bmatrix} | & | & & | \\ u^{(1)} & u^{(2)} & \dots & u^{(n)} \\ | & | & & | \end{bmatrix} \in \mathbb{R}^{n \times n}.$$

- Definimos a matriz $U_{reduzida}$ com as k primeiras colunas de U e o vetor $z^{(i)}$:

$$z^{(i)} = U_{reduzida}^T x^{(i)} = \begin{bmatrix} | & | & & | \\ u^{(1)} & u^{(2)} & \dots & u^{(k)} \\ | & | & & | \end{bmatrix}^T x^{(i)}.$$

Note que $z \in \mathbb{R}^k$, como desejado.

Algoritmo

- Os vetores $x^{(i)}$ podem ser representados em uma matriz X:

$$X = \begin{bmatrix} (x^{(1)})^T \\ \vdots \\ (x^{(m)})^T \end{bmatrix}$$

permitindo o cálculo vetorizado de Σ:

```
Sigma = (1/m)*X'*X;
```

- Em geral:

```
Sigma = (1/m)*X'*X;
U,S,V = svd(Sigma);
Ureduzida = U(:,1:k);
z = Ureduzida'*x;
```

IMPORTANTE: NÃO temos mais a convenção $x_0 = 1$ (atributos tratados por igual).

Algoritmo

- Os vetores $x^{(i)}$ podem ser representados em uma matriz X:

$$X = \begin{bmatrix} (x^{(1)})^T \\ \vdots \\ (x^{(m)})^T \end{bmatrix}$$

permitindo o cálculo vetorizado de Σ:

```
Sigma = (1/m)*X'*X;
```

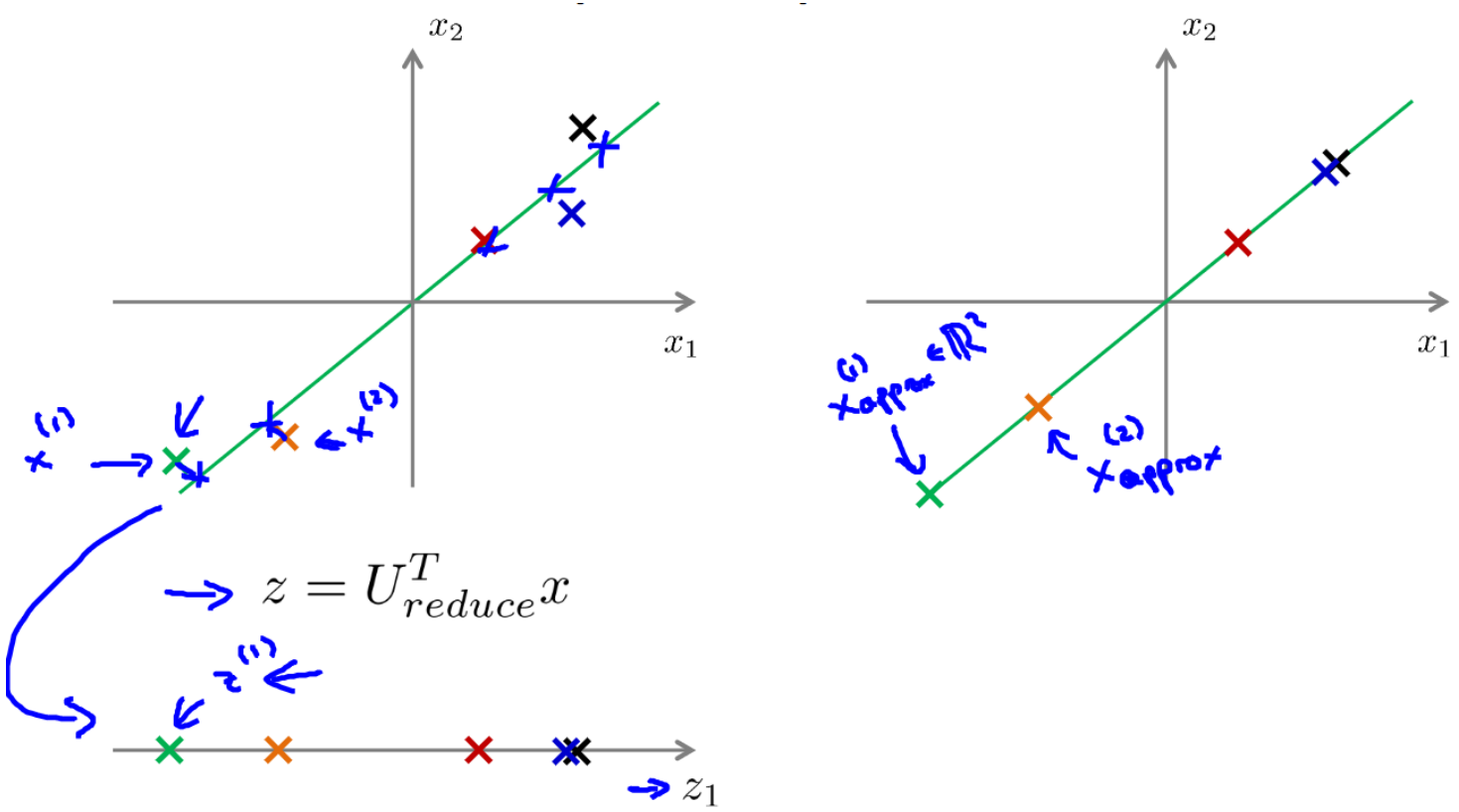
- Em geral:

```
Sigma = (1/m)*X'*X;
U,S,V = svd(Sigma);
Ureduzida = U(:,1:k);
z = Ureduzida'*x;
```

IMPORTANTE: NÃO temos mais a convenção $x_0 = 1$ (atributos tratados por igual).

Reconstrução

- Projeção de x sobre o vetor u:

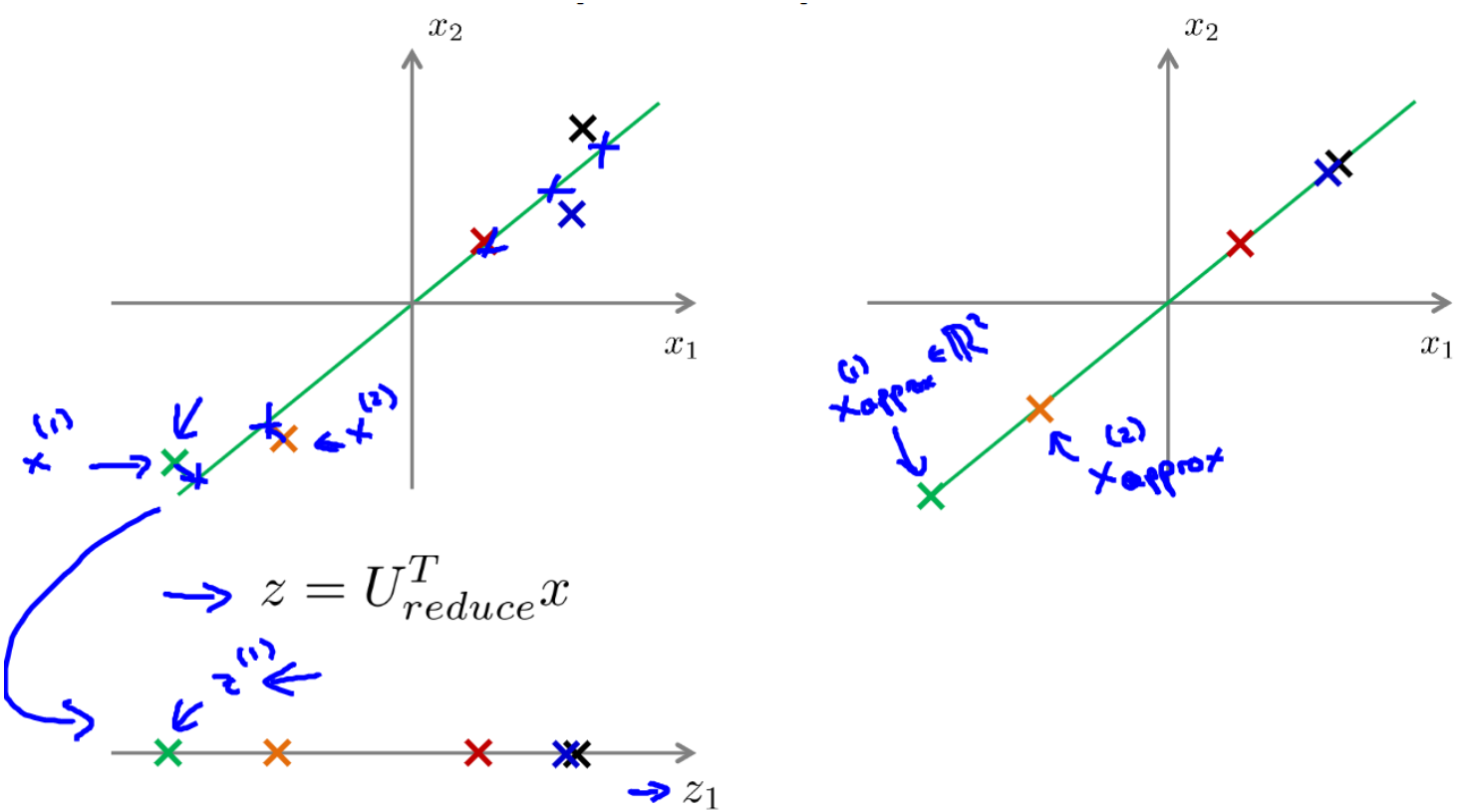


- $x^{(i)}_{aproximado} \approx x^{(i)}$ é dado por

$$x_{aproximado} = U_{reduzida} z^{(i)}.$$

Reconstrução

- Projeção de x sobre o vetor u:



- $x^{(i)}_{aproximado} \approx x^{(i)}$ é dado por

$$x_{aproximado} = U_{reduzida} z^{(i)}.$$

Número de Componentes Principais

- k é chamado de **número de componentes principais**.
- Definimos:
  - Erro de projeção quadrático médio:

$$\frac{1}{m}\sum_{i=1}^{m}\|x^{(i)} - x^{(i)}_{aproximado}\|^2$$

  - Variação total no dado:

$$\frac{1}{m}\sum_{i=1}^{m}\|x^{(i)}\|^2$$

- Podemos escolher k como o menor valor tal que

$$\frac{\frac{1}{m}\sum_{i=1}^{m}\|x^{(i)} - x^{(i)}_{aproximado}\|^2}{\frac{1}{m}\sum_{i=1}^{m}\|x^{(i)}\|^2} \leq 0.01 \quad (1\%).$$

Diz-se que "99% da variância é retida".

Número de Componentes Principais

- k é chamado de **número de componentes principais**.
- Definimos:
  - Erro de projeção quadrático médio:

$$\frac{1}{m}\sum_{i=1}^{m}\|x^{(i)} - x^{(i)}_{aproximado}\|^2$$

  - Variação total no dado:

$$\frac{1}{m}\sum_{i=1}^{m}\|x^{(i)}\|^2$$

- Podemos escolher k como o menor valor tal que

$$\frac{\frac{1}{m}\sum_{i=1}^{m}\|x^{(i)} - x^{(i)}_{aproximado}\|^2}{\frac{1}{m}\sum_{i=1}^{m}\|x^{(i)}\|^2} \leq 0.01 \quad (1\%).$$

Diz-se que "99% da variância é retida".

Número de Componentes Principais

- k é chamado de **número de componentes principais**.
- Definimos:
  - Erro de projeção quadrático médio:

$$\frac{1}{m}\sum_{i=1}^{m}\|x^{(i)} - x^{(i)}_{aproximado}\|^2$$

  - Variação total no dado:

$$\frac{1}{m}\sum_{i=1}^{m}\|x^{(i)}\|^2$$

- Podemos escolher k como o menor valor tal que

$$\frac{\frac{1}{m}\sum_{i=1}^{m}\|x^{(i)} - x^{(i)}_{aproximado}\|^2}{\frac{1}{m}\sum_{i=1}^{m}\|x^{(i)}\|^2} \leq 0.01 \quad (1\%).$$

Diz-se que "99% da variância é retida".

Número de Componentes Principais

- k é chamado de **número de componentes principais**.
- Definimos:
  - Erro de projeção quadrático médio:

$$\frac{1}{m}\sum_{i=1}^{m}\|x^{(i)} - x^{(i)}_{aproximado}\|^2$$

  - Variação total no dado:

$$\frac{1}{m}\sum_{i=1}^{m}\|x^{(i)}\|^2$$

- Podemos escolher k como o menor valor tal que

$$\frac{\frac{1}{m}\sum_{i=1}^{m}\|x^{(i)} - x^{(i)}_{aproximado}\|^2}{\frac{1}{m}\sum_{i=1}^{m}\|x^{(i)}\|^2} \leq 0.01 \quad (1\%).$$

Diz-se que "99% da variância é retida".

Número de Componentes Principais

- k é chamado de **número de componentes principais**.
- Definimos:
  - Erro de projeção quadrático médio:

$$\frac{1}{m}\sum_{i=1}^{m}\|x^{(i)} - x^{(i)}_{aproximado}\|^2$$

  - Variação total no dado:

$$\frac{1}{m}\sum_{i=1}^{m}\|x^{(i)}\|^2$$

- Podemos escolher k como o menor valor tal que

$$\frac{\frac{1}{m}\sum_{i=1}^{m}\|x^{(i)} - x^{(i)}_{aproximado}\|^2}{\frac{1}{m}\sum_{i=1}^{m}\|x^{(i)}\|^2} \leq 0.01 \quad (1\%).$$

  Diz-se que "99% da variância é retida".

Número de Componentes Principais

- Usa-se também ≤ 0.05 (5%) e ≤ 0.10 (10%).
- Procedimento geral:

  Testar PCA com $k = 1$

  Calcular $U_{reduzida}, z^{(1)}, z^{(2)}, \cdots, z^{(m)}, x^{(1)}_{aproximado}, \cdots, x^{(m)}_{aproximado}$

  Checar se $\frac{\frac{1}{m}\sum_{i=1}^{m} \|x^{(i)} - x^{(i)}_{aproximado}\|^2}{\frac{1}{m}\sum_{i=1}^{m} \|x^{(i)}\|^2} \leq 0.01$

  Repetir esse processo para $k = 2, 3, 4, \cdots$

Número de Componentes Principais

- Usa-se também ≤ 0.05 (5%) e ≤ 0.10 (10%).
- Procedimento geral:

Testar PCA com $k = 1$

Calcular $U_{reduzida}, z^{(1)}, z^{(2)}, \cdots, z^{(m)}, x^{(1)}_{aproximado}, \cdots, x^{(m)}_{aproximado}$

Checar se $\frac{\frac{1}{m}\sum_{i=1}^{m} \|x^{(i)} - x^{(i)}_{aproximado}\|^2}{\frac{1}{m}\sum_{i=1}^{m} \|x^{(i)}\|^2} \leq 0.01$

Repetir esse processo para $k = 2, 3, 4, \cdots$

Número de Componentes Principais

- Modo muito mais eficiente: rodando

  U,S,V = svd(Sigma)

  a matriz S é diagonal e a checagem da variância equivale a

$$1 - \frac{\sum_{i=1}^{k} S_{ii}}{\sum_{i=1}^{n} S_{ii}} \leq 0.01$$

  ou ainda

$$\frac{\sum_{i=1}^{k} S_{ii}}{\sum_{i=1}^{n} S_{ii}} \geq 0.99$$

  e o algoritmo então é

  U,S,V = svd(Sigma)
  Tome o menor valor de k para o qual
  $\frac{\sum_{i=1}^{k} S_{ii}}{\sum_{i=1}^{n} S_{ii}} \geq 0.99$
  (99% da variância retida)

Outline

1 Motivação

2 Análise de Componentes Principais

3 Aplicação

Aplicação

- Acelerar algoritmos supervisionados. EX.: Dados os exemplos

$$(x^{(1)}, y^{(1)}), (x^{(2)}, y^{(2)}), \cdots, (x^{(m)}, y^{(m)})$$

podemos fazer

$$x^{(1)}, x^{(2)}, \cdots, x^{(m)} \in \mathbb{R}^{(10000)}$$

$$\downarrow PCA$$

$$z^{(1)}, z^{(2)}, \cdots, z^{(m)} \in \mathbb{R}^{1000}$$

e o novo conjunto de dados será

$$(z^{(1)}, y^{(1)}), (z^{(2)}, y^{(2)}), \cdots, (z^{(m)}, y^{(m)}).$$

Aplicação

- IMPORTANTE: O mapeamento PCA $x^{(i)} \rightarrow z^{(i)}$ deve ser definido APENAS sobre o conjunto de treinamento
- $U_{reduzida}$ e os parâmetros de normalização (média, desvio) devem ser calculados sobre o conjunto de treino apenas e aplicados sobre o teste.

Receber o conjunto de treino $\{(x^{(1)}, y^{(1)}), (x^{(2)}, y^{(2)}), \cdots, (x^{(m)}, y^{(m)})\}$
Rodar PCA para reduzir a dimensão de $x^{(i)}$ gerando $z^{(i)}$
Treinar a regressão logística (ou qualquer outro algoritmo) sobre $\{(z^{(1)}, y^{(1)}), \cdots, (z^{(m)}, y^{(m)})\}$
Mapear $x^{(i)}_{teste}$ para $z^{(i)}_{teste}$
Rodar $h_\theta(z)$ em $\{(z^{(1)}_{teste}, y^{(1)}_{teste}), \cdots, (z^{(m)}_{teste}, y^{(m)}_{teste})\}$

Aplicação

- IMPORTANTE: O mapeamento PCA $x^{(i)} \rightarrow z^{(i)}$ deve ser definido APENAS sobre o conjunto de treinamento
- $U_{reduzida}$ e os parâmetros de normalização (média, desvio) devem ser calculados sobre o conjunto de treino apenas e aplicados sobre o teste.

Receber o conjunto de treino $\{(x^{(1)}, y^{(1)}), (x^{(2)}, y^{(2)}), \cdots, (x^{(m)}, y^{(m)})\}$
Rodar PCA para reduzir a dimensão de $x^{(i)}$ gerando $z^{(i)}$
Treinar a regressão logística (ou qualquer outro algoritmo) sobre
$\{(z^{(1)}, y^{(1)}), \cdots, (z^{(m)}, y^{(m)})\}$
Mapear $x^{(i)}_{teste}$ para $z^{(i)}_{teste}$
Rodar $h_\theta(z)$ em $\{(z^{(1)}_{teste}, y^{(1)}_{teste}), \cdots, (z^{(m)}_{teste}, y^{(m)}_{teste})\}$

Aplicação

- Uso geral:
  - Compressão:
    - Reduzir o consumo de memória/armazenamento
    - Acelerar o algoritmo de aprendizado

    Escolhe-se k pela porcentagem da variância retida.
  - Visualização: $k = 2$ ou $k = 3$.
  - Redução de ruídos; *eigenfaces*.

Maus Usos

- Prevenir *overfit*.
- Pode até funcionar, mas não é uma boa prática: PCA não leva em conta o rótulo y e um atributo importante pode desaparecer.
- Melhor é a **regularização**.
- Sempre, antes de qualquer coisa, rodar todo o algoritmo de aprendizado sobre o dado original/bruto $x^{(i)}$.
- Só se estiver com problema de desempenho de CPU/memória considerar usar o dado $z^{(i)}$ do PCA.

Maus Usos

- Prevenir *overfit*.
- Pode até funcionar, mas não é uma boa prática: PCA não leva em conta o rótulo y e um atributo importante pode desaparecer.
- Melhor é a **regularização**.
- Sempre, antes de qualquer coisa, rodar todo o algoritmo de aprendizado sobre o dado original/bruto $x^{(i)}$.
- Só se estiver com problema de desempenho de CPU/memória considerar usar o dado $z^{(i)}$ do PCA.

Maus Usos

- Prevenir *overfit*.
- Pode até funcionar, mas não é uma boa prática: PCA não leva em conta o rótulo y e um atributo importante pode desaparecer.
- Melhor é a **regularização**.
- Sempre, antes de qualquer coisa, rodar todo o algoritmo de aprendizado sobre o dado original/bruto $x^{(i)}$.
- Só se estiver com problema de desempenho de CPU/memória considerar usar o dado $z^{(i)}$ do PCA.

Maus Usos

- Prevenir *overfit*.
- Pode até funcionar, mas não é uma boa prática: PCA não leva em conta o rótulo y e um atributo importante pode desaparecer.
- Melhor é a **regularização**.
- Sempre, antes de qualquer coisa, rodar todo o algoritmo de aprendizado sobre o dado original/bruto $x^{(i)}$.
- Só se estiver com problema de desempenho de CPU/memória considerar usar o dado $z^{(i)}$ do PCA.

Maus Usos

- Prevenir *overfit*.
- Pode até funcionar, mas não é uma boa prática: PCA não leva em conta o rótulo y e um atributo importante pode desaparecer.
- Melhor é a **regularização**.
- Sempre, antes de qualquer coisa, rodar todo o algoritmo de aprendizado sobre o dado original/bruto $x^{(i)}$.
- Só se estiver com problema de desempenho de CPU/memória considerar usar o dado $z^{(i)}$ do PCA.