Universidade Estadual de Campinas - Unicamp

IMECC

MS571 - Aprendizado de Máquinas: Aspectos Teóricos e Práticos

Projeto Computacional II

Matheus Feres Turcheti 241727
Othavio Henrique de Jesus Ayres 246666
Ricardo Raimundo da Silva 120094

Novembro
2022

Conteúdo

1 Estrutura do Projeto

Os dois arquivos principais do projeto são `eigenfaces.ipynb` e `recomendation.ipynb` os quais contém a aplicação dos modelos implementados sobre os dados salvos na pasta `data`.

As funções utilizadas nestes dois *notebooks* foram implementadas em "módulos" separados (`pca_eigenfaces` e `movie`).

Em `pca_eigenfaces` nós implementamos as funções que são utilizadas para redução de dimensionalidade através de PCA e também para a visualização de imagenss (*eigenfaces*). Já em `movie`, nós implementamos o algoritmo de filtragem colaborativa a fim de fazer recomendações de filmes (embora o código em si seja mais genérico e possa ser facilmente reaproveitado para um outro sistema de recomendação).

Por fim, para testar o projeto solicitamos a consulta do arquivo `README.md` o qual descreve como as dependências são resolvidas e como os *notebooks* podem ser rodados.

2 *Principal Component Analysis* (PCA)

2.1 O que é o PCA?

O PCA é um método de análise estatística muito utilizado para fins de redução de dimensionalidade (projetando o conjunto original sobre um espaço de menor dimensão). O método em si permite a análise dos componentes principais (*principal component analysis*, em inglês) dos dados. Isto é, é verifica-se quais combinações lineares de atributos (componentes) representam melhor os dados (i.e. mantém variância).

A redução de dimensionalidade é feita ao obter explicitamente os componentes principais e utilizar deles para extrair as combinações lineares "principais" dos dados disponíveis, ou seja, projetando os dados originais sobre um espaço de dimensão menor (dimensão é o número de componentes principais).

Como podem ser obtidos os componentes principais de uma matriz de dados X?

2.2 Maximizando a variância

Consideremos novamente qual é o problema em questão: reduzir a dimensionalidade de X sem perder excessivamente a variância em X.

Uma forma de enxergar a redução de dimensionalidade é notar que para cada coordenada no espaço de menor dimensão, está associado um vetor de

"pesos" u. Temos tal vetor de pesos porque é necessário ponderar quanto de cada atributo será adicionado a cada componente (i.e. coordenada no novo espaço).

Assim, temos o problema de otimização

$$\max u^T \Sigma u$$
$$\text{s.a. } ||u||_2 = 1$$

Sendo Σ a matriz de covariância dos dados, a qual podemos calcular fazendo

$$\mu = \frac{1}{m}\sum_{i=1}^{m} x^{(i)} \text{ , sendo } x^{(i)} \text{ a i-ésima linha de } X \text{ como vetor coluna} \quad (1)$$

$$X = X - M \text{ sendo } M = \begin{bmatrix} \mu^T \\ \mu^T \\ \vdots \\ \mu^T \end{bmatrix} \in \mathbb{R}^{m \times n} \quad (2)$$

$$\Sigma = \frac{1}{m} X^T X \in \mathbb{R}^{n \times n} \quad (3)$$

Note que (2) do processo sobrescreve a matriz X para uma versão em que sua média por coluna é zero. Garantimos que isso acontece ao calcular $\mu \in \mathbb{R}^n$ o qual contém em cada coordenada a média de cada atributo (i.e. de cada coluna).

Reescrevendo o problema de maximização temos que

$$\max \frac{1}{m} u^T X^T X u$$
$$\text{s.a. } ||u||_2 = 1$$

O que equivale a

$$\arg \max_{||u||_2=1} \frac{1}{m} u^T X^T X u = \arg\max \frac{\frac{1}{m} u^T X^T X u}{u^T u}$$

Contudo

$$
\begin{aligned}
\max \frac{\frac{1}{m}u^T X^T X u}{u^T u} &= \max \frac{||\frac{1}{\sqrt{m}}Xu||_2}{||u||_2} \\
&= \frac{1}{\sqrt{m}}||X||_2 \\
&= \frac{1}{\sqrt{m}}\lambda_{\max}^{1/2}(X^T X) \\
&= \lambda_{\max}^{1/2}(\frac{1}{m}X^T X) \\
&= \lambda_{\max}^{1/2}(\Sigma) \\
&= \sigma_1(\Sigma)
\end{aligned} \tag{4}
$$

Note que essas passagens decorrem de resultados clássicos de álgebra linear numérica (vide capítulo 2 do livro do Golub [1]).

A última passagem de (4) indica que o valor da solução do problema de maximização é o maior valor singular da matriz Σ, ou seja, u deve ser o vetor singular (Σ é simétrica, então os vetores singulares à direita e à esquerda são iguais) associado ao maior valor singular.

Esta é, porém, uma solução parcial, uma vez que utilizando apenas o vetor u estaremos reduzindo a dimensão de $m \cdot n$ para m apenas (todos os n atributos seriam condensados em uma combinação linear). Assim, devemos encontrar demais vetores-peso u_j para descrever as demais coordenadas do espaço de dimensão reduzida. Em outras palavras precisamos dos demais vetores singulares de Σ.

2.3 *Singular Value Decomposition* (SVD)

Uma forma de encontrar todos os vetores singulares de Σ é realizando a decomposição em valores singulares (*singular value decomposition*, em inglês) da matriz.

$$\Sigma = USV^T$$

Em que U é ortogonal (colunas são ortonormais), $S = \text{diag}(\sigma_1, \ldots, \sigma_r)$ em que σ_j são os valores singulares ordenados e r é o posto de Σ (o qual será igual a n se X tiver posto completo, i.e. todos os atributos forem L.I), e temos $U = V$ pois Σ é simétrica.

Observe que o vetor u encontrado acima é a primeira coluna de U e, similarmente, as demais colunas são os demais vetores singulares de Σ.

2.4 Compressão e descompressão

Quando obtivemos o vetor-peso u concluímos que utilizá-lo seria reduzir todo o conjunto de dados a um espaço de dimensão m. Mas como exatamente esse procedimento decorreria? Por definição, o vetor u deve ser o peso que os atributos terão no novo espaço, ou seja, estamos implicitamente fazendo $z = Xu$, obtendo o novo vetor z representativo de *todos* os exemplos para tal compressão.

Claramente, comprimindo com apenas um vetor singular acarreta em perdas significativas de informação (leia, variância). Perda essa proporcional à diferença entre σ_1 e $\text{tr}(S)$, isto é, a variância preservada depende da quantidade de valores singulares utilizados - mais do que isso, depende da magnitude dos valores singulares utilizados, uma vez que eles vem ordenados e podem ter valores muito distintos.

Assim, para projetar (comprimir) o conjunto de dados X com a menor perda possível, devemos determinar um k tal que $Z = XU_k \in \mathbb{R}^{m \times k}$ exiba uma preservação mínima de variância.

Nesse caso, U_k equivale à matriz com as k primeiras colunas de U e fazemos a escolha deste k tomando o menor valor tal que

$$\frac{\sum_{j=1}^{k} \sigma_j}{\sum_{j=1}^{n} \sigma_j} \geq t$$

para algum um *threshold* de variância mínima a ser mantida t.

A descompressão dos dados implica na transformação contrária, isto é, a transposta: $X_r = ZU_k^T$ em que X_r representa os dados recuperados da projeção Z.

Desta forma, temos agora uma ferramenta que permite a representação aproximada de dados da "melhor" forma possível, isto é, preservando o máximo de variância possível.

2.5 *Eigenfaces*

Neste projeto aplicamos o método do PCA descrito acima sobre uma base de imagens contendo apenas a face de algumas pessoas.

No caso particular de imagens, nós temos que cada exemplo é um vetor de pixels (um quadrado perfeito, nesse caso) e trabalhamos com ele na ordem em que vem. Apenas transformamos em uma matriz para fins de visualização.

Antes de começarmos a trabalhar com os dados disponíveis, aplicamos o *z-score* sobre nossa matriz de dados, procedimento esse que tem por finalidade apenas reproduzir o que é feito em (2), a fim de permitir que todos os demais passos descritos acima sejam válidos para serem implementados.

A primeira análise que realizamos foi a comparação de 100 imagens aleatórias originais com a suas versões com 99% da variância retida, o que implicou em usar $k = 335$ (o que representa menos de 1/3 da dimensão original dos dados) e obtivemos resultados surpreendentemente bons para o olho nu.

Consideramos interessante que a maioria das imagens é muito bem recuperada e observamos que aquelas que não apresentam tanta nitidez são, por frequência, imagens cujas diferentes de tons é baixa (ou seja, a variância nos pixels da própria imagem é baixa, logo seria necessário individualmente o uso de um k maior para preservar os mesmos 99% da variância do conjunto todo).

Em seguida nós analisamos os 36 principais *eigenfaces*, isto é, transformamos os primeiros 36 vetores singulares em imagens. Nessas imagens podemos visualizar como os atributos foram combinados a fim de preservar o máximo de variância possível. Isto é, as imagens são formas que relembram rostos borrados e representam as combinações lineares feitas com os atributos, de forma que cada uma dessas imagens funciona como um filtro para as imagens no nosso conjunto de dados.

Em particular é interessante ressaltar que as *eigenfaces* parecem indicar as regiões da face que são mais relevantes para a reconstrução de um resto humano. No caso, notamos que olhos, boca e nariz são as regiões mais marcadas (mais definidas, com contrastes de cor mais fortes, e menos borradas) nas *eigenfaces*.

Nós também realizamos uma rodada de compressão e descompressão com apenas $k = 100$ componentes principais. Não surpreendentemente, essa recuperação foi menos clara do que a anterior, uma vez que estamos usando agora cerca de 10% da dimensão original. Contudo, devido ao ordenamento dos valores singulares, fica claro que a melhora na qualidade da recuperação é progressivamente menor conforme aumentamos a quantidade de vetores singulares na matriz `U_r`, uma vez que os primeiros componentes são responsáveis por explicar mais variância do que os subsequentes (por isso são os "principais").

Possíveis usos práticos de eigenfaces podem incluir, principalmente, sistemas de reconhecimento facial e geração de imagens (realistas ou com animações adicionadas por cima).

Para o caso de reconhecimento facial, uma possível abordagem é manter uma base de dados com imagens comprimidas (a fim de maximizar o tamanho possível da base) e, para cada imagem nova, projetar essa imagem no espaço das *eigenfaces* e então medir uma distância (como a norma-2) entre a imagem projetada e imagens na base de dados. Isso é uma métrica de proximidade entre os valores de contribuição de cada *eigenface* do novo exemplo

comparado com a base de dados. Se o novo exemplo tem uma distância $\delta < \epsilon$ para um *threshold* de decisão ϵ, quando comparada a uma imagem da base, então podemos predizer que a nova imagem é da mesma pessoa que está na base (caso contrário, é uma pessoa desconhecida).

3 Sistemas de Recomendação

3.1 Algoritmos de Recomendação

Algoritmos de recomendação analisam o que os usuários desejam em um espaço muito vasto e denso de dados, por isso os algoritmos levam em consideração a análise crítica com base no perfil dos usuários, padrões comportamentais, itens visualizados e mais uma série de outros fatores que fazem o algoritmo entender quais resultados da busca são, em teoria, mais relevantes para quem está realizando a mesma. Os resultados muitas vezes não são completamente precisos, mas trazem dentre os primeiros itens da lista o que provavelmente o usuário deseja. A cada ano tais algoritmos são aperfeiçoados de modo que a acurácia se mantém cada vez mais elevada. Alguns exemplos famosos de aplicação são a *Netflix*, *Amazon*, *Facebook*, todos estes *sites* utilizam de algoritmos de recomendação para prever possíveis interesses de um usuário com base em dados previamente adquiridos a respeito daquele usuário e dos usuários com os quais ele interage.

3.2 Filtragem por Conteúdo

O método mais simples de recomendação é a filtragem por conteúdo, alguns softwares tem como função gerar descrições de um determinado item, ou seja, atribuir conteúdos a um item qualquer. A filtragem de conteúdo consiste em realizar análises comparativas envolvendo o diversos conteúdo dos itens, como por exemplo, palavras similares, temas similares, gêneros similares e afins; de maneira geral, busca identificar a similaridade entre dois ou mais itens através de um coeficiente numérico. A recomendação baseada na filtragem por conteúdo utiliza informações anteriores do usuário em relação a um item para recomendar itens similares. A recomendação priorizará os itens que mais se aproximam de itens avaliados positivamente anteriormente pelo usuário.

3.3 Filtragem Colaborativa

Filtragem colaborativa é uma forma de realizar predições a respeito dos interesses de um certo indivíduo a partir da coleta de dados de interesses de um grupo de indivíduos. Partindo do pressuposto que se um indivíduo X possui o mesmo grau de interesse sobre um determinado assunto K tal como um indivíduo Y, então é provável que o indivíduo X possua o mesmo grau de interesse sobre outro assunto N assim como o indivíduo Y, do que outro indivíduo qualquer Z.

A filtragem colaborativa tem como base:

- Dado $x^{(1)}, \ldots, x^{(n_m)}$, estimar $\theta^{(1)}, \ldots, \theta^{(n_u)}$
- Dado $\theta^{(1)}, \ldots, \theta^{(n_u)}$, estimar $x^{(1)}, \ldots, x^{(n_m)}$

Mas como o θ depende de x e vice versa, podemos inicialmente aleatoriamente tomar um valor para θ e iterativamente calcular os novos valores para x e θ.

A função objetivo do algoritmo da filtragem colaborativa terá como base uma aproximação por vizinhança das notas dadas pelos usuários:

- Dado $x^{(1)}, \ldots, x^{(n_m)}$, estimar $\theta^{(1)}, \ldots, \theta^{(n_u)}$

$$\underset{\theta^{(1)},\ldots,\theta^{(n_u)}}{\arg\min} \frac{1}{2} \sum_{j=1}^{n_u} \sum_{i:r(i,j)=1} ((\theta^j)^T x^{(i)} - y^{(i,j)})^2 + \frac{\lambda}{2} \sum_{j=1}^{n_u} \sum_{k=1}^{n} (\theta_k^{(j)})^2 \quad (5)$$

- Dado $\theta^{(1)}, \ldots, \theta^{(n_u)}$, estimar $x^{(1)}, \ldots, x^{(n_m)}$

$$\underset{x^{(1)},\ldots,x^{(n_m)}}{\arg\min} \frac{1}{2} \sum_{i=1}^{n_m} \sum_{j:r(i,j)=1} ((\theta^j)^T x^{(i)} - y^{(i,j)})^2 + \frac{\lambda}{2} \sum_{i=1}^{n_m} \sum_{k=1}^{n} (x_k^{(i)})^2 \quad (6)$$

Que deverão ser somados em uma única função $J(x^{(1)}, \ldots, x^{(n_m)}, \theta^{(1)}, \ldots, \theta^{(n_u)})$.

E finalmente podemos descrever o algoritmo da filtragem colaborativa sendo:

- Passo 1: Inicializar $x^{(1)}, \ldots, x^{(n_m)}, \theta^{(1)}, \ldots, \theta^{(n_u)}$ com valores aleatórios;
- Passo 2: Minimizar $J(x^{(1)}, \ldots, x^{(n_m)}, \theta^{(1)}, \ldots, \theta^{(n_u)})$

  Utilizando gradiente descendente como algoritmo de otimização, fazemos, para todo $j = 1, \ldots, n_u, i = 1, \ldots, n_m$:

$$x_k^{(i)} = x_k^{(i)} - \alpha \left(\sum_{j:r(i,j)=1} ((\theta^{(j)})^T x^{(i)} - y(i,j))\theta_k^{(j)} + \lambda x_k^{(i)} \right) \tag{7}$$

$$\theta_k^{(j)} = \theta_k^{(j)} - \alpha \left(\sum_{i:r(i,j)=1} ((\theta^{(j)})^T x^{(i)} - y(i,j))x_k^{(i)} + \lambda \theta_k^{(j)} \right) \tag{8}$$

- Passo 3: Para um indivíduo com parâmetro $\theta^{(j)}$ e um assunto com atributos $x^{(i)}$ (já treinados), predizer a nota $(\theta^{(j)})^T(x^{(i)})$.

3.4 Vetorização

Na filtragem colaborativa, as notas dada pelos usuários são salvas em uma matriz, onde a linha i representa o usuário i, e a coluna j, representa o filme j. Além disso, definindo uma matriz coluna X, que contem os exemplos e uma matriz coluna θ que contem os pesos para cada respectivo exemplo, temos que a matriz de predição é dada por $X\Theta^T$.

3.5 Resultados

Adicionamos notas artificiais para alguns filmes a fim de ilustrar qual seria o *output* do nosso modelo para um usuário cujas notas são conhecidas explicitamente. Para isso acrescentamos tais notas aos dados que já possuíamos e realizamos o treinamento usando o método de gradiente conjugado da função `scipy.optimize.minimize` para minimizar a função de custo do modelo, definida por `_loss` no módulo `movies`.

Ao final realizamos predições para todos os "usuários" da nossa base de dados, mas apenas analisamos a saída para o nosso usuário, cujas notas foras manualmente adicionadas. Notamos que a nota predita não é representativa de uma nota de 1 a 5 como inicialmente temos nos dados, até porque são valores acima de 5 que são mostrados. Essas "notas" são apenas indicativas

da pontuação relativa entre os filmes, isto é, servem apenas para ordenar os filmes de forma a poder obter uma relação de prioridade e, por consequência, ter a capacidade de recomendar certos filmes em detrimento de outros. E também notou-se que obtivemos diferentes recomendações de filmes em comparação ao treinamento utilizando gradiente descendente, como no exemplo demonstrado em sala de aula.

Referências

[1] Gene H. Golub; Charles F. Van Loan. *Matrix Computations*. The Johns Hopkins University Press, Baltimore, 2013.

[2] Leonardo Herdy Marinho, Rodrigo Campos, Rodrigo Pereira dos Santos, Mônica Ferreira da Silva, and Jonice Oliveira. Conceitos, implementaçao e dados privados de algoritmos de recomendaçao. *Sociedade Brasileira de Computação*, 2019.