

0.1 Introdução

O presente documento chamado Projeto III, tem como o objetivo apresentar os resultados obtidos durante a implementação e estudos realizados no tópico de métodos não supervisionados e Maquinas de Vetores de Suporte. A primeira parte do documento faz referencia ao método Máquinas de vetores de suporte. A segunda parte usa o método de aprendizagem não supervisionado chamado Análise de Componentes Principais (PCA) no reconhecimento facial.

Parte I

Quando usamos regressão linear polinomial temos para polinomios de grau n a função $y = \theta_0 + \theta_1 x + \dots + \theta_n x^n$. Defina

$$\begin{aligned}\phi : \mathbb{R} &\rightarrow \mathbb{R}^{n+1} \\ x &\mapsto \begin{bmatrix} 1 \\ x \\ \vdots \\ x^n \end{bmatrix}.\end{aligned}$$

Assim $y = \theta^T \phi(x)$ em que $\theta = [\theta_0, \theta_1, \dots, \theta_n]^T$.

Note que podemos escrever

$$\theta = \sum_{i=1}^m \beta_i \phi(x^{(i)}),$$

para $\beta_i \in \mathbb{R}$ e m o número de exemplos de treinamento.

No processo de aprendizado usando gradiente descendente com hiperparâmetro α temos para cada i

$$\beta_i := \beta_i + \alpha \left[y^{(i)} - \sum_{j=1}^m \beta_j \langle \phi(x^{(i)}), \phi(x^{(j)}) \rangle \right]$$

Definition 0.1.1. O kernel associado a ϕ é a função $\mathbb{R}^n \times \mathbb{R}^n \rightarrow \mathbb{R}$ dada por

$$K(x, z) = \langle \phi(x), \phi(z) \rangle.$$

A ideia do kernel pode ser aplicada para qualquer algoritmo de aprendizado de máquinas onde seja necessário o cálculo de produto interno entre vetores de atributos, por exemplo o kernel linear e o kernel gausiano dados por

$$k_{\text{linear}}(x, z) = 1 + \langle x, z \rangle + \langle x, z \rangle^2 + \langle x, z \rangle^3$$

$$k_{\text{gauss}}(x, z) = \exp \left\{ \frac{-\|x - z\|^2}{2\sigma^2} \right\},$$

em que σ é a variância. Notar que o kernel não é um método de aprendizado de maquinas, é um método para reescrever os métodos que inclui na função hipóteses o produto interno $\theta^T x$. Pode ser usado na regressão logística, linear ou no método Máquinas de Vetores de Suporte.

O método supervisionado *Maquinas de Vetores de Suporte* tem como objetivo encontrar uma fronteira que separe as classes, tal que a distância dos dados e a fronteira seja máxima. Para desenvolver a teoria necessária para encontrar essa fronteira, definimos a seguinte notação

- $x^{(i)}$: dados de treinamento i .
- $y^{(i)}$: classe do dado de treinamento i .
- m : número total de exemplos de treinamento.
- n : número de parâmetros.
- λ : hiperparâmetro de regularização.

Na regressão logística temos a função hipóteses

$$h_\theta(x) = \frac{1}{1 + e^{-\theta^T x}},$$

e função de custo

$$\arg \min_{\theta} \frac{1}{m} \left[\sum_{i=1}^m -y^{(i)} \log(h_\theta(x^{(i)})) - (1 - y^{(i)}) \log(1 - h_\theta(x^{(i)})) \right] + \frac{\lambda}{2m} \sum_{j=1}^n \theta_j^2. \quad (1)$$

Se definimos $C = \frac{1}{\lambda}$ o problema de otimização (1) é equivalente a

$$\arg \min_{\theta} C \left[\sum_{i=1}^m -y^{(i)} \log(h_\theta(x^{(i)})) - (1 - y^{(i)}) \log(1 - h_\theta(x^{(i)})) \right] + \frac{1}{2} \sum_{j=1}^n \theta_j^2. \quad (2)$$

Se redefinimos as classes binárias como $y^{(i)} \in \{-1, 1\}$, temos que para zerar o custo precisamos $\theta^T x \geq 1$ para $y^{(i)} = 1$ e $\theta^T x \leq -1$ para $y^{(i)} = -1$. Para valores de C grandes, então o primeiro termo de (2) vai para zero na solução final. Logo é possível reescrever-lo como um problema de otimização com restrições

$$\begin{aligned} \arg \min_{\theta} & \frac{1}{2} \|\theta\|_2^2 \\ \text{s.a} & \theta^T x^{(i)} \geq 1 \quad \text{se } y^{(i)} = 1 \\ & \theta^T x^{(i)} \leq -1 \quad \text{se } y^{(i)} = -1. \end{aligned} \quad (3)$$

O produto interno entre dois vetores pode ser escrito em termos de projeções, isso é $\theta^T x = p\|\theta\|_2$ em que p é a projeção do x no θ . Portanto o problema de otimização com restrições em termos de projeções é

$$\begin{aligned} & \arg \min_{\theta} \frac{1}{2}\|\theta\|_2^2 \\ \text{s.a} \quad & p^{(i)}\|\theta\|_2 \geq 1 \quad \text{se } y^{(i)} = 1 \\ & p^{(i)}\|\theta\|_2 \leq -1 \quad \text{se } y^{(i)} = -1. \end{aligned} \tag{4}$$

Como estamos minimizando $\|\theta\|$ então para a classe $y = 1$ precisamos ter $p^{(i)}$ positivo com magnitude grande, e para $y = -1$ um valor negativo; isso é, maximizamos os $p^{(i)}$. Daí, obtemos a margem larga entre a fronteira e os dados. Logo, o método de Maquinas de Vetores de Suporte se resume a resolver o problema de programação quadrática com restrições dado em (4).

Por outro lado, se reformulamos o problema (3) reescrevendo $\theta^T x$ como $w^T x + b$ em que b é o termo de bias e usando o fato que $y \in \{-1, 1\}$ temos

$$\begin{aligned} & \arg \min_{w,b} \frac{1}{2}\|w\|_2^2 \\ \text{s.a} \quad & y^{(i)}((w^{(i)})^T x^{(i)} + b) \geq 1 \quad i = 1, \dots, m. \end{aligned} \tag{5}$$

Ao reescrever a restrição anterior como $-y^{(i)}((w^{(i)})^T x^{(i)} + b) + 1 \leq 0$ podemos usar Multiplicadores de Lagrange para resolve-o. Daí a função lagrangiana nosso caso é

$$\mathcal{L}(w, b, \alpha) = \frac{1}{2}\|w\| - \sum_{i=1}^m \alpha_i (-y^{(i)}((w^{(i)})^T x^{(i)} + b) + 1)$$

em que os α_i são os multiplicadores de Lagrange associados às restrições de igualdade. Portanto, nosso problema de otimização quadrática torna-se

$$\min_{w,b} \mathcal{L}(w, b, \alpha) = \theta_{\mathcal{D}}(\alpha). \tag{6}$$

O anterior é conhecido como o problema primal. Lembre-se que para resolver (6) procuramos os pontos onde as derivadas parciais são zeradas. Daí, computando-as temos

$$\begin{aligned} \frac{\partial}{\partial w} \mathcal{L}(w, b, \alpha) &= w - \sum_{i=1}^m \alpha_i y^{(i)} x^{(i)} = 0 \\ \frac{\partial}{\partial b} \mathcal{L}(w, b, \alpha) &= \sum_{i=1}^m \alpha_i y^{(i)} = 0. \end{aligned}$$

O problema pode ser resolvido a traves do dual o qual tendo na conta as restrições anteriores e a regularização, temos

$$\begin{aligned} \max_{\alpha} W(\alpha) &= \max_{\alpha} \sum_{i=1}^m \alpha_i - \frac{1}{2} \sum_{i,j=1}^m y^{(i)} y^{(j)} \alpha_i \alpha_j \langle x^{(i)}, x^{(j)} \rangle \\ \text{s.a} \\ 0 \leq \alpha_i &\leq C \quad i = 1, \dots, m \\ \sum_{i=1}^m \alpha_i y^{(i)} &= 0. \end{aligned} \tag{7}$$

A partir do dual obtemos os α_i ótimos, para assim obter w e b do problema primal. Basados no método de coordenada ascendente temos o algoritmo que resolve o problema

Algorithm 1: Sequential minimal optimization

```

Input:  $X, m, y$ 
for  $i = 1$  até  $m$  do
|  $\alpha_i = \arg \max_{\tilde{\alpha}_i} W(\alpha_1, \dots, \alpha_{i-1}, \tilde{\alpha}_i, \alpha_{i+1}, \dots, \alpha_m)$ 
end
```

O método anterior pode ser resolvido usando kernels, onde cada feature é obtido a partir de um kernel K usado como medida de similaridade entre um ponto de treinamento $x^{(i)}$ e um ponto de referencia chamado landmarks $l^{(i)}$.

Vários pacotes são usados para resolver o problema anterior. O kernel usado será de acordo com a melhor fronteira que possa separar os dados, cuja decisão é tomada de acordo com o que é evidenciado na imagem plotada dos dados. Por exemplo, a base de dados *dado1.mat* são dados que podem ser separados linearmente tal como se observa na figura 1.

Lembre-se que o objetivo é encontrar uma fronteira tal que tenha margem máxima, isso é, a distancia entre dados e a fronteira seja máxima. Sabemos que a fronteira vai mudar de acordo ao valor C . Observa-se como muda a linha que separa os dados nas figuras 2, 3 e 4. Para valores de C maiores, ou seja λ pequenos, o problema tende ao underfitting e para valores C pequenos, tende ao overfitting.

Nosso caso, por ter um outlier da classe vermelha, observa-se como para $C = 50$ ou $C = 100$ (ver figuras 3 e 4) temos que a linha separa perfeitamente as classes, mas fica muito próxima dos dados, portanto, ao momento de ter um novo exemplo, poderia ser classificado incorretamente. No caso $C = 1$ temos que a fronteira separa

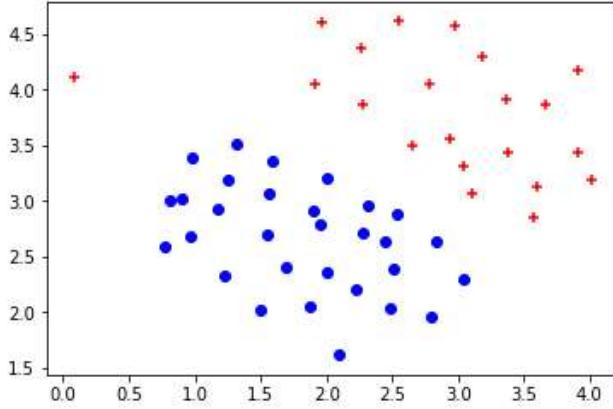


Figura 1: Dados plotados de acordo à classe.

os dados muito bem se fossemos ignorar o outlier, e a distância respeito os dados é boa. Nesse caso pode-se concluir que temos boa separação das classes, pois de forma geral temos margem máxima. É para esclarecer que às vezes os dados podem ter erros na medição, portanto os outliers podem ser dados errados. Conclui-se que a fronteira é sensível aos outliers, portanto é melhor usar valores para os hiperparâmetros como $C = 1$, para os quais se obtém uma boa separação ignorando outliers, ademais é um bom modelo que pode ser usado com dados novos.

Por outro lado, temos outro tipo de exemplo quando temos dados que não podem ser separados linearmente, como é o caso da base *dado2.mat* (ver figura 5). Nesse caso é recomendado usar o kernel gaussiano.

Para usar o kernel gaussiano, ademais de ter o hiperparâmetro C , é necessário definir a variância σ . Logo, com o objetivo de procurar C e σ ideal, usamos o método cross-validation, que consiste em testar o modelo com dados novos. Os valores ideais são aqueles para os quais o erro de classificação foi menor. Para usar o método SVM com kernel gaussiano é necessário normalizar os atributos, isso é

$$x_i := \frac{x_i - \text{média}(x_i)}{\text{desvio padrão } (x_i)}.$$

Na librarria do Python temos o hiperparâmetro

$$\gamma = \frac{1}{2\sigma}.$$

Usando valores $C = 10$ e $\sigma = 0.01$, observa-se na figura 6 como a fronteira separa perfeitamente os dados, mas temos uma curva que tende a ter overfitting. A precisão

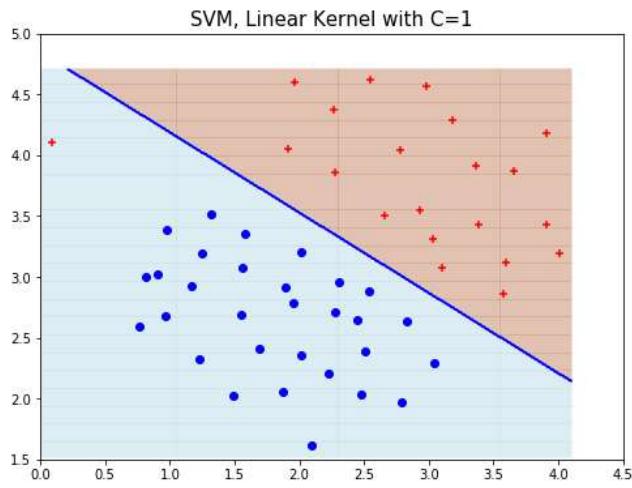


Figura 2: Fronteira linear para $C = 1$.

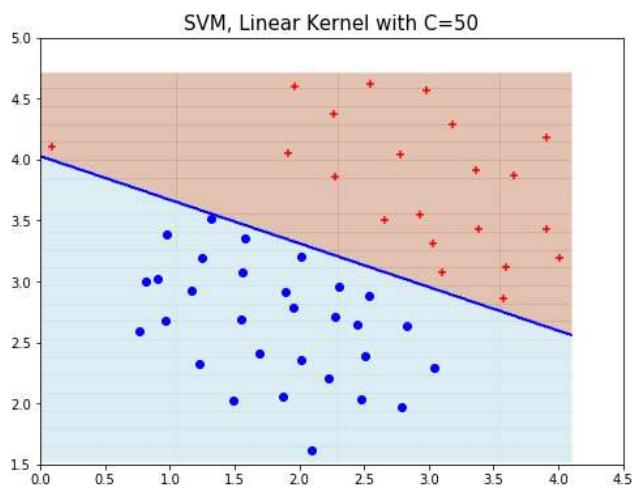


Figura 3: Fronteira linear para $C = 50$.

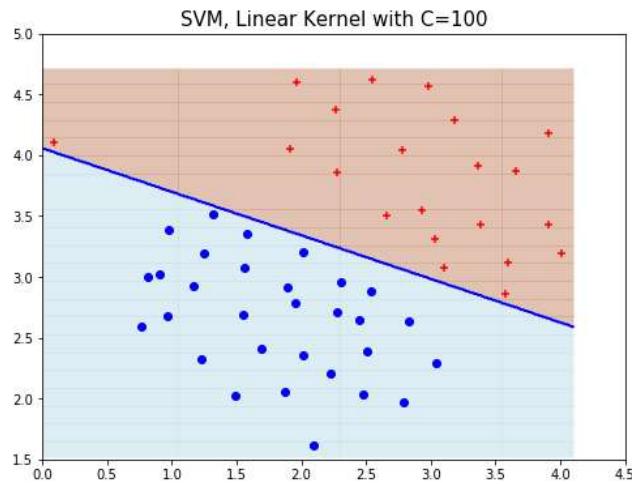


Figura 4: Fronteira linear para $C = 100$.

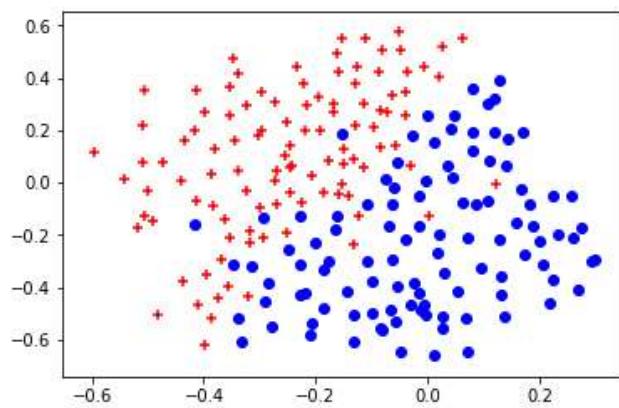


Figura 5: Dados plotados de acordo à classe, sem normalização.

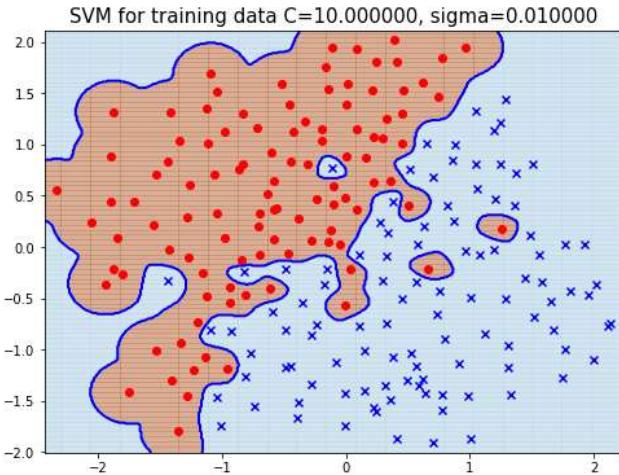


Figura 6: SVM com kernel gaussiano para $C = 10$ e $\sigma = 0.01$.

de classificação de treino é 100%, mas a precisão de validação é 88%. Se tomamos diferentes valores como $C = 5$ e $\sigma = 1$ temos uma curva mais simples (ver figura 7), com precisão de treino 94.31% e uma precisão de validação maior 91%. Daí concluímos que mesmo a precisão de treino seja perfeita, o modelo pode ser propenso a variância alta.

Como foi concluído, os valores do C e σ são importantes no momento de obter um bom modelo. Logo, foram selecionados valores $C = (0.01, 0.03, 0.1, 0.3, 1, 3, 10, 30)$ e $\sigma = (0.01, 0.03, 0.1, 0.3, 1, 3, 10, 30)$ para procurar o menor erro de validação. Após usar o algoritmo que testa os hiperparâmetros ideais, foi obtido $C = 10$ e $\sigma = 0.3$ com erro de validação 7.5%. A fronteira testada nos dados de treinamento está na figura 8 e para os dados de validação na figura 9. De forma geral, temos uma boa separação das classes para os dados de validação, pois o conjunto de validação contém muitos dados da classe azul que podem ser considerados outlier, pois estão muito mais próximos dos vermelhos que dos azuis.

Conclui-se então que ao ter dados que não são separados linearmente, temos que obter uma fronteira com margem larga pode se tornar mais complexa, como foi obtido nas bases de dados *dado1.mat* e *dado2.mat*. Ao não ter uma fronteira linear, os dados podem ficar na fronteira mesma deixando de fora a margem longa. Logo, o objetivo deste tipo de dados passa a encontrar uma fronteira de forma a separar melhor os dados.

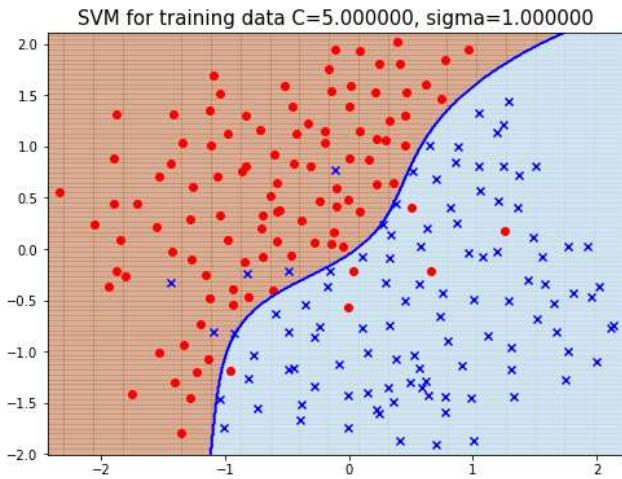


Figura 7: SVM com kernel gaussiano para $C = 5$ e $\sigma = 1$.

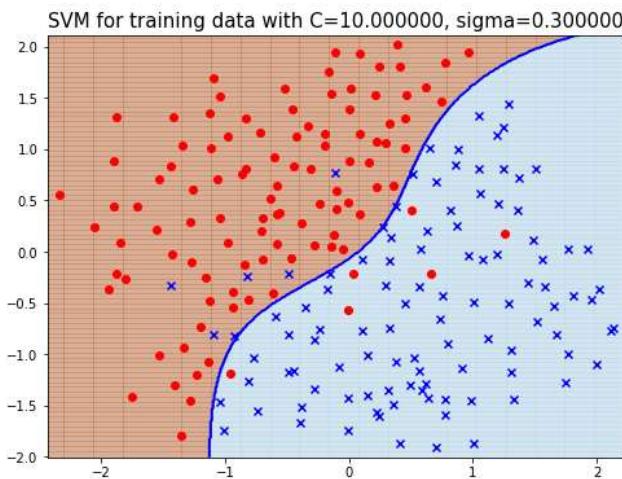


Figura 8: SVM com kernel gaussiano com hiperparâmetros ideais $C = 10$ e $\sigma = 0.3$ para os dados de treinamento.

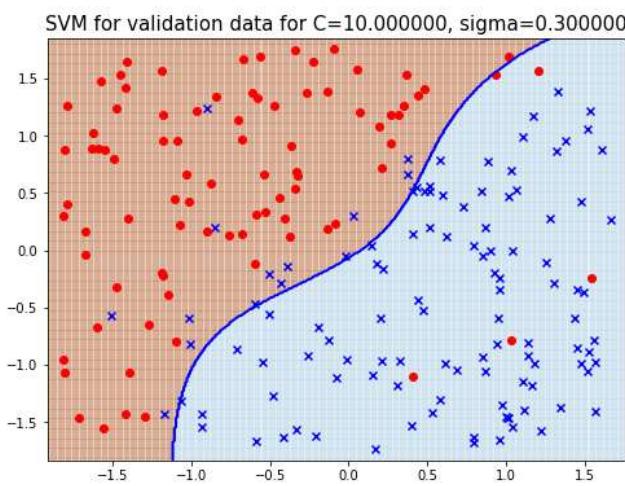


Figura 9: SVM com kernel gaussiano com hiperparâmetros ideais $C = 10$ e $\sigma = 0.3$ para os dados de validação.

Parte II

O aprendizado não supervisionado é um tipo de algoritmo de aprendizado de máquina usado para fazer inferências de conjuntos de dados de entrada sem classes rotuladas.

Um dos métodos é o PCA, Analises de Componentes Principais (principal components analysis) que é uma técnica para reduzir a dimensionalidade de conjuntos de dados, aumentando a interpretabilidade, mas minimizando a perda de informações. O método cria novas variáveis basado nos autovetores, tal que sejam não correlacionadas e maximizam a variância. Nesse caso a variância refere-se à distância entre a origem e a projeção do dado a um hiperplano.

A ideia do método é minimizar a distância perpendicular dos dados a um hiperplano. Para isso, seguimos os passos

- Dados de treinamento $\{x^{(1)}, x^{(2)}, \dots, x^{(m)}\}$.
- Normalizar os dados pela média, isso é para cada $i \in \{1, \dots, m\}$

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

em que $std(x_j)$ é o desvio padrão e

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

- Como a distância da projeção $x^{(i)}$ sobre u é dado pelo produto interno $(x^{(i)})^T u$. Portanto, maximizamos

$$\max_u u^T \left(\frac{1}{m} \sum_{i=1}^m x^{(i)} (x^{(i)})^T \right) u, \quad (8)$$

sujeito a $\|u\|_2 = 1$.

- Dado que redefinimos os dados com media zero, defina a matriz de covariância

$$\Sigma = \frac{1}{m} \sum_{i=1}^n (x^{(i)})((x^{(i)})^T). \quad (9)$$

- O problema (8) tem como solução o autovetor principal da matriz (9). Para reduzir os erros computacionais é recomendado obter os autovetores da decomposição SVD (no Python $U, S, V = svd(X)$) da matriz X , que retorna nas colunas da matriz V^T os autovetores do Σ . Pode ser demostrado que o processo descrito pode ser substituído pela decomposição SVD da matriz Σ , e usar a matriz U obtida no Python. Como as dimensões das matrizes Σ e X são $n \times n$ e $m \times n$ respectivamente, então computacionalmente, podemos procurar o minimo entre n e m e resolver a decomposição SVD da matriz correspondente (de aqui em diante vamos supor que a decomposição usada foi feita na matriz Σ , isso é, vamos usar os autovetores obtidos das colunas da matriz U).
- Seja k o numero de componentes principais dado por as primeiras k colunas da matriz U . Isso é, os primeiros k autovetores da matriz $\Sigma\Sigma^T$ ($U_{red} = U[1 : k]$).

A projeção de cada exemplo x sobre o vetor u é dada por

$$z = U_{red}^T x.$$

Note que $z \in \mathbb{R}^k$, em que $k \leq n$. O anterior mapeia um vetor de dados de dimensão n para um novo espaço de k variáveis que são não correlacionadas. Esse novo vetor z contem informação do exemplo x , mas ocupando uma quantidade menor de memória do computador. Sendo uma ganancia computacional, pois n poderia ser muito grande e gerar problemas de memoria física ou armazenamento do computador. Pelas características do z , é possível aproximar cada dado partindo de z . Para isso computamos

$$x_{approx} = U_{red} z. \quad (10)$$

Uma das aplicações do PCA é no reconhecimento facial (eigenfaces). A base de dados *dado3.mat* contém 5000 imagens de tamanho 32×32 e foram vetorizadas. Na figura 10 temos uma amostra aleatória de 100 imagens da base de dados.

Realizando o processo descrito anteriormente, obtemos uma matriz $\Sigma \in \mathbb{R}^{1024 \times 1024}$. Se visualizamos cada autovetor, observa-se na figura 11 que cada autovetor extrai os contornos de cada face, ressaltando as características faciais prototípicas como olhos, nariz, sobrancelhas e boca. Esses autovetores nos dará um tipo de raio-x dos



Figura 10: 100 faces aleatórias da base de dados *dado3.mat*.



Figura 11: Representação gráfica dos primeiros 36 autovetores da matriz Σ .

rostos. Ao se tratar de autovetores, então eles formam uma base. Logo, cada rosto é uma combinação linear dos eigenfaces. Por essa razão os autovetores são imagens com características faciais. Portanto, se tivéssemos um rosto novo e quiséramos reconhece-o, então seria suficiente com projetar o vetor que representa o rosto no eigenspace para obter uma impressão digital com os rastros relevantes do rosto. A aproximação da face é obtida através da equação (10).

Podemos usar a aproximação das imagens de acordo a (10), usando número de componentes $k = 100$. Obtemos uma matriz $U_{red} \in \mathbb{R}^{1024 \times 100}$. Comparando os exemplos originais e os aproximados, observa-se uma amostra deles na figura 12 em que o lado direito é a foto original e o lado esquerdo a imagem aproximada. Observa-se que a aproximação elimina elementos tais como os óculos, forma exata dos olhos e linhas de expressão, mas conserva a forma da cara.

Esse tipo de métodos aplicado as imagens de faces pode ser aplicado na pratica ao



Figura 12: Comparação das faces originais e aproximadas para $k = 100$.

reconhecimento facial. Kirby and Sirovich demonstrou que o PCA é um método ótimo tal que minimiza o erro quadrático meio entre a aproximação e as imagens reais [2]. As novas imagens de teste (face a ser reconhecida) são combinadas com imagens no banco de dados, projetando-as na base de auto-vetores e encontrar a imagem compactada mais próxima no subespaço, com o objetivo de reconhecer a face. Para esse caso, pode-se usar o método SVM, em que se procura a classe (pessoa) mais próxima do novo dado. Para isso, será necessário ter várias imagens de cada pessoa para criar cada cluster (a seção 0.2 contem um pequeno exemplo mostrando como usar o método PCA para reconhecimento facial).

Sabendo a importância deste método, uma escolha relevante na implementação do método PCA é a seleção do valor de k ou número de componentes principais. Dado que os autovalores correspondem à variação relativa, procuramos os primeiros k autovetores que capturam pelo menos 99% da variância total, isso é, os primeiros k autovetores tais que a soma dos autovalores seja maior ou igual que 0.99. Para isso implementa-se o algoritmo 2.

Algorithm 2: Definição do número de componentes principais com 99% da variância retida.

Input: Σ

$$U, S, V = svd(\Sigma)$$

Tome o menor valor de k para

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

Lembre-se que a fatorização SVD dada no algoritmo 2 pode ser feita na matriz X , selecionando aquela que computacionalmente requer um espaço de armazenamento menor, pois os dois problemas são equivalentes.

Nosso caso, foi obtido que o melhor valor para as componentes principais é 335. Repetindo o processo anterior, mas usando $U_{red} \in \mathbb{R}^{1024 \times 335}$. Note-se na figura 13 que nesse caso a aproximação dos exemplos é mais detalhada comparada para o caso $k = 100$ (ver figura 12), tendo aspectos tais como óculos na foto 3 e linhas de expressão. Note ademais que a aproximação é quase exata. Portanto, usar esse número de componentes é ideal, pois foi possível conseguir uma redução da dimensionalidade de aproximadamente 68%.

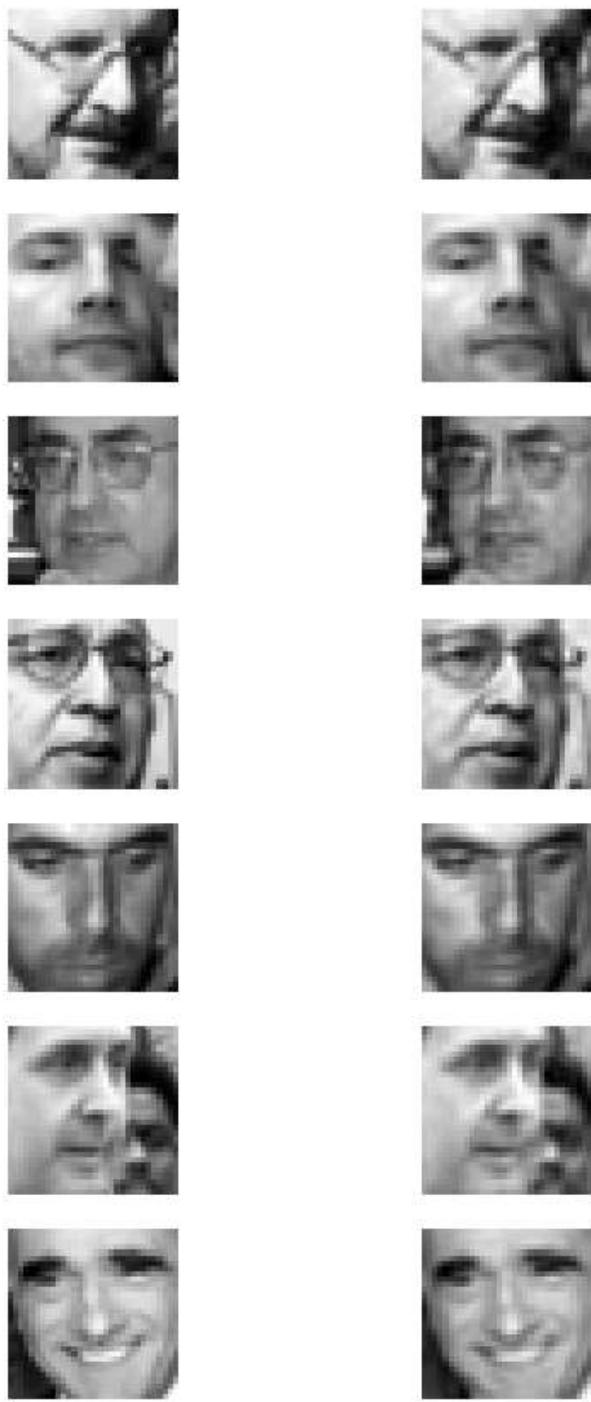


Figura 13: Comparação das faces originais e aproximadas para $k = 335$.

0.2 Apêndice

Com o objetivo de testar como usar o algoritmo definido na Parte II para o reconhecimento facial, foi criada uma base de dados pequena com imagens de duas pessoas da disciplina Aprendizado de Maquinas. Para isso foram vetorizadas as imagens, convertendo-as em tamanho 50×50 e obtendo um vetor para cada imagem de dimensão 2500.

Observa-se na imagem [14](#) que os autovetores obtidos da decomposição SVD sobre a matriz Σ , são misturas das imagens das duas pessoas que contêm características faciais. Dado que eles são uma base, cada imagem é uma combinação linear dos eigenfaces.

Como os primeiros eigenfaces são imagens mais claras, foram usados para projetar cada foto no plano. Daí, obtemos que as imagens de cada pessoa projetadas estão dadas na figura [15](#).

Dado que os dados tem etiqueta, então é possível usar o método SVM da Parte I com o objetivo de separar as classes (pessoas) encontrando a fronteira que separa os dados. Daí o novo dado que não tem etiqueta é projetado como foi descrito anteriormente e de acordo com a seção que pertence, a previsão será feita.

Já tendo treinado os dados, temos a classificação na figura [16](#), obtendo uma fronteira linear. Daí, as imagens por classificar foram projetados no plano (ver figura [17](#)) e classificadas posteriormente na figura [18](#). Os resultados obtidos da predição comparando as fotos dos estudantes são dados no gráfico [19](#). Nesse caso, as previsões foram certeiras na totalidade.

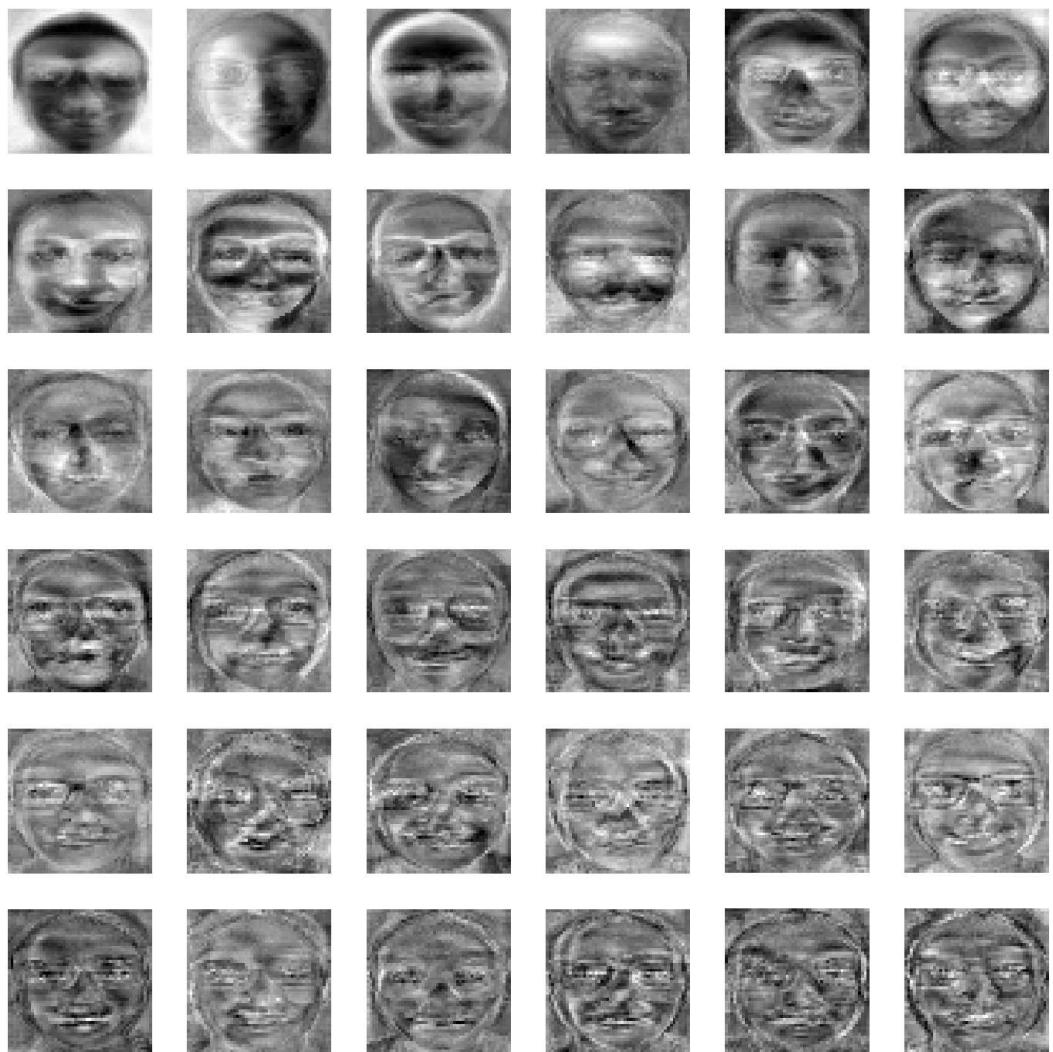


Figura 14: Representação gráfica dos primeiros autovetores da matriz Σ .

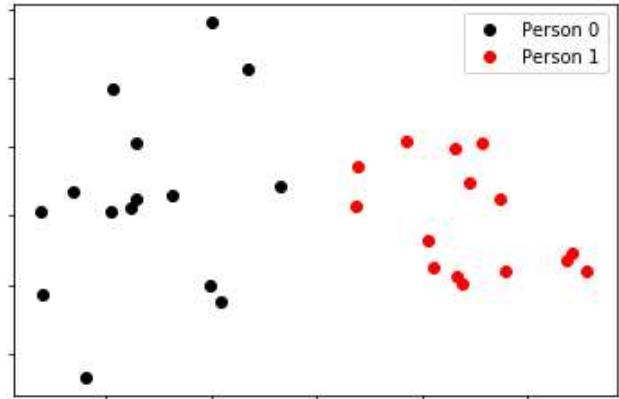


Figura 15: Projeção das imagens de cada pessoa no plano respeito aos dois primeiros eigenfaces.

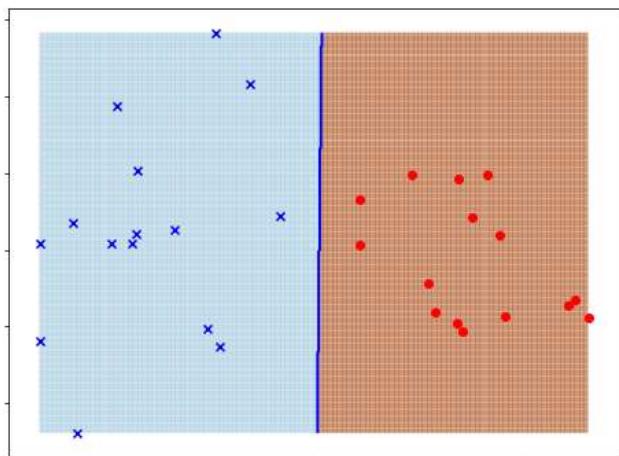


Figura 16: SVM das imagens projetadas.

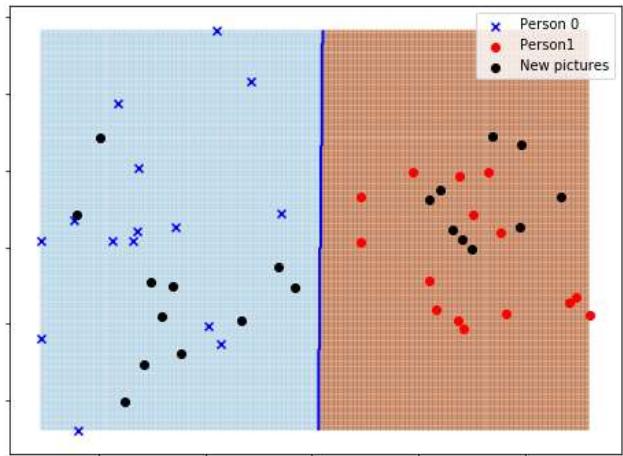


Figura 17: Dados novos projetados no plano.

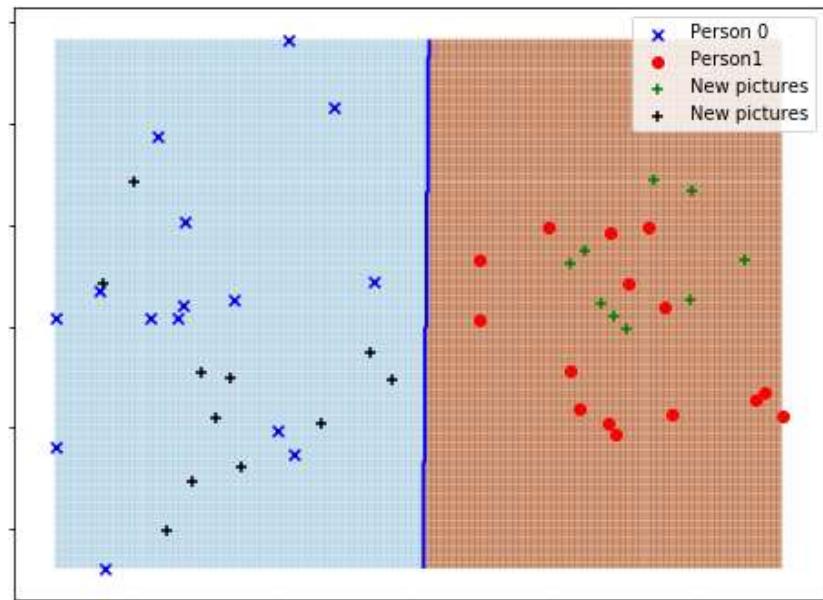


Figura 18: Predição de acordo a SVM.



Figura 19: Predição a partir da separação linear SVM.

Bibliografia

- [1] Enrique Carmona. Tutorial sobre máquinas de vectores soporte (svm), 11 2016.
- [2] Bruce Draper, Kyungim Baek, and J. Beveridge. Recognizing faces with pca and ica. *Computer Vision and Image Understanding*, 91:115–137, 07 2003.
- [3] Ian T. Jolliffe and Jorge Cadima. Principal component analysis: a review and recent developments. 374, 2016.
- [4] Jonathon Shlens. A tutorial on principal component analysis, 2014.