Universidade Estadual de Campinas

Institute of Mathematics, Statistics and Scientific Computing

(IMECC - Unicamp)

Projeto Computacional 3:

Aplicação dos métodos *Support Vector Machines* (SVM) e *Principal Component Analysis* (PCA) e seus resultados

Aluno : Gabriel Borin Macedo || RA : 197201
Aluno : Matheus Araujo Souza || RA : 184145
Aluno : Raul Augusto Teixeira || RA : 205177

Dezembro 2020

Resumo

Mesmo com o avanço das técnicas de *deep learning* em diversos problemas tais como no auxílio de diagnósticos médicos [6], em sistemas de recomendação [12] ou até mesmo em reconhecimento de objetos em imagens que utiliza modelos de *deep learning* e técnicas de *segmentação semântica* [4]. Outras técnicas tais como *Support Vector Machines* (SVM) e *Principal Component Analysis* (PCA) ainda são outras estratégias viáveis para se utilizar nesses problemas de classificação. A partir dessa ideia, esse projeto propõe um estudo na aplicação desses modelos em dois problemas. O primeiro seria a aplicação de uma SVM em um problema de classificação de dados e o outro seria a aplicação do PCA em um problema de reconhecimento facial. Para ambos os resultados, foi obtido um excelente desempenho para classificação desses dados.

1 Estrutura Matemática

1.1 *Support Vector Machines* (SVM)

As *Support Vector Machines* (SVM) ou conhecido em português como *Maquinas de vetores de suporte*, são modelos utilizados para margem larga. Geralmente, é mais utilizado em tarefas de *classificação*. A ideia proposta é encontrar fronteiras geométricas em que as mesmas classes de objetos estejam juntos e fiquem o mais longe dessa fronteiras.

1.1.1 Função de perda de duas classes utilizando SVM's

Assim, dada a equação da *regressão logística*, podemos deduzir a *função de perda* para as SVM's. Dada a equação da forma $z(x) = \theta^T x$, onde x é o vetor de entrada com $x \in R^n$ e θ é a matriz de pesos que melhor aproxima os dados, com $\theta \in R^n$. Assim, note que temos a seguinte expressão para a função de perda sem regularização e para apenas um único dado

$$J(\theta) = -y \log\left(\frac{1}{1+e^{-\theta^T x}}\right) - (1-y)\log\left(1 - \frac{1}{1+e^{-\theta^T x}}\right)$$

Com $y = 0$ se $z(x) \leq 0$ e $y = 1$ se $z(x) >> 0$. Dessa forma, iremos separar os termos da equação $J(\theta)$ em duas expressões

$$\text{custo}_0(z) = -\log\left(1 - \frac{1}{1+e^{-z}}\right)$$

$$\text{custo}_1(z) = -\log\left(\frac{1}{1+e^{-z}}\right)$$

Onde essas equações são denominadas de *hinge loss* que é a função de perda por partes. Além disso, iremos fazer duas readaptações. A primeira mudança é em relação ao valor de y

$$y = \begin{cases} -1 & \text{se } z(x) \leq 0 \\ 1 & \text{se } z(x) > 0 \end{cases}$$

Com $y = -1$ se ele não for o classe que buscamos e $y = 1$ se ele for da classe que buscamos. Assim, temos que a função $J(\theta)$ com o termo regularizador para m-dados e desconsiderando o termo $\frac{1}{m}$ (que neste caso não interfere nas contas), temos que

$$J(\theta) = \arg\min_{\theta} \sum_{i=1}^{m} y^{(i)} \text{custo}_1(\theta^T x^{(i)}) + (1 - y^{(i)}) \text{custo}_1(\theta^T x^{(i)}) + \frac{\lambda}{2} \sum_{i=1}^{n} \theta_i^2$$

Onde λ é um uma constante escolhida pelo usuário. Por fim, sabemos que minimizar uma equação da forma $A + \lambda B$ é a mesma coisa que minimizar a equação $CA + B$, com $C = \frac{1}{\lambda}$. Assim, teremos a equação final de $J(\theta)$ sendo

$$J(\theta) = \arg\min_{\theta} C \sum_{i=1}^{m} y^{(i)} \text{custo}_1(\theta^T x^{(i)}) + (1 - y^{(i)}) \text{custo}_1(\theta^T x^{(i)}) + \frac{1}{2} \sum_{i=1}^{n} \theta_i^2 \quad (1)$$

1.1.2 Intuição da Margem larga

Para zerar o custo da função, queremos que $z(x) = \theta^T x \geq 1$ para $y = 1$ e $z(x) = \theta^T x \geq -1$ para $y = -1$. Além disso, se uma SVM possui o valor de $C \longrightarrow \infty$ grande. Então, temos que a primeira parte do somatório de $J(\theta)$ é considerado zero, ou seja $\sum_{i=1}^{m} y^{(i)} \text{custo}_1(\theta^T x) + (1 - y^{(i)}) \text{custo}_1(\theta^T x) \longrightarrow 0$. Com isso, teremos a seguinte equação

$$\arg\min_{\theta} \frac{1}{2} \sum_{i=1}^{n} \theta_i^2 = \frac{1}{2} ||\theta||^2$$

S.A

$$\begin{cases} \theta^T x \geq 1; \text{ Se } y^{(i)} = 1 \\ \theta^T x \leq -1; \text{ Se } y^{(i)} = -1 \end{cases}$$

Uma outra forma de reescrever a equação acima é escrever em relação ao *produto interno*. Assim, teremos o produto

$$p = \frac{\theta^T x}{||\theta||}$$

Assim, podemos reescrever a equação na forma

$$\arg\min_{\theta} \frac{1}{2} \sum_{i=1}^{n} \theta_i^2 = \frac{1}{2}||\theta||^2$$

$$\text{S.A}$$

$$\begin{cases} p^{(i)}||\theta|| \geq 1; \text{ Se } y^{(i)} = 1 \\ p^{(i)}||\theta|| \leq -1; \text{ Se } y^{(i)} = -1 \end{cases}$$

Onde $p^{(i)}$ é a projeção de $x^{(i)}$ sobre θ. Além disso, sabemos que θ é *perpendicular* a fronteira de decisão $z(x) = \theta^T x = 0$.

Note que como $||\theta||$ é minimizador, precisamos encontrar valores de $p^{(i)}$ sempre com grande magnitude. Por isso que esta ideia é chamada de *margem larga*. Portanto, se aumentarmos o valor da fronteira implica que aumentamos a distância da fronteira dos dados. Assim, encontramos um cenário ideal para minimizar o erro do modelo e que fique mais otimizável para resolver. Dessa forma, iremos criar uma SVM da forma $f(x) = W^T x + b$, onde $b \in R$ é o *bias*, $W \in R^n$ é o vetor dos pesos e $x \in R^n$ é o vetor de entrada. Assim, temos o problema para minimizar

$$\min \frac{||W||^2}{2}$$
$$\text{S.A } y^{(i)}(W^T x + b) \geq 1; \; i = 1, 2, \ldots, m$$

Note que temos um problema de *programação quadrática*. Assim, considerando esse como o *problema primal* e iremos resolver o *problema dual* visto em [1], no qual é da forma

$$\max_{\alpha} F(\alpha) = \sum_{i=1}^{m} \alpha_i - \frac{1}{2} \sum_{i=1}^{m} \sum_{j=1}^{m} y^{(i)} y^{(j)} \alpha_i \alpha_j < x^{(i)}, x^{(j)} >$$

$$\text{S.A}$$

$$\begin{cases} \alpha_i \geq 0; \text{ i = 1, 2, ..., m} \\ \sum_{i=1}^{m} \alpha_i y^{(i)} = 0 \end{cases}$$

Com isso, a partir dos valores de α_i's do *problema dual*, podemos obter os vetores W e b do *problema primal*. Dessa forma, temos que a função, $f(x)$ será da forma

$$f(x) = \sum_{i=1}^{m} \alpha_i y^{(i)} < x^{(i)}, x > + b \quad (3)$$

Onde $< x^{(i)}, x >$ é o operador de *kernel* que é escolhido pelo usuário. Usualmente, é mais comum utilizar $< x^{(i)}, x >$ sendo o *produto escalar convencional*. Ou seja

$$< y, x >= x^T y = \sum_{i=1}^{m} x_i y_i \ ; \ \forall \ y, x \in R^m$$

1.1.3 *Sequential Minimal Optimization* (SMO)

SMO's são um tipo especial de SVM que são baseadas no método da *coordenada descendente*. Onde esse método consiste na em aplicar o algoritmo

Algorithm 1: Pseudo - Código para o método da *coordenada descendente*

```
while (Não converge) do
    for i = 0:m do
        α_i = arg max_{ᾱ_i} F(α_1, α_2, ..., α_{i-1}, ᾱ_i, α_{i+1}, ..., α_m)
    end
end
```

Onde as variáveis diferente de $\overline{\alpha}_i$ são fixas. Além disso, iremos considerar o *problema dual* do problema das SVM. Porém, levando em conta a regularização do modelo, teremos o seguinte problema

$$\max_\alpha F(\alpha) = \sum_{i=1}^{m} \alpha_i - \frac{1}{2} \sum_{i=1}^{m} \sum_{j=1}^{m} y^{(i)} y^{(j)} \alpha_i \alpha_j < x^{(i)}, x^{(j)} >$$

S.A

$$\begin{cases} C \geq \alpha_i \geq 0; \ \text{i = 1, 2, ..., m} \\ \sum_{i=1}^{m} \alpha_i y^{(i)} \qquad = 0 \end{cases}$$

A grande vantagem da SMO é aplicar o método das *coordenada descendente* em dois pares α_i e α_j. A partir disso, vamos analisar essa situação com um exemplo : suponha que começamos com α_1 e α_2 e fixando o restante dos valores $\alpha_3, \ldots, \alpha_m$. A partir da restrição $\sum_{i=1}^{m} \alpha_i y^{(i)} = 0$, é obtido

$$\alpha_1 y^{(1)} + \alpha_2 y^{(2)} = - \sum_{i=3}^{m} \alpha_i y^{(i)}$$

Como os valores $\alpha_3, \ldots, \alpha_m$ são fixados. Então, temos que o lado direito é uma constante $\zeta = -\sum_{i=3}^{m} \alpha_i y^{(i)}$. Com isso, temos como resultado final

$$\alpha_1 y^{(1)} + \alpha_2 y^{(2)} = \zeta \quad (2)$$

A partir da restrição $C \geq \alpha_i \geq 0$, temos que α_1 e α_2 estão limitados pela caixa $[0, C]$ x $[0, C]$. A partir de (2), podemos reescrever α_1 em função de α_2 na forma

$$y^{(1)}(\alpha_1 y^{(1)} + \alpha_2 y^{(2)}) = y^{(1)}\zeta \Longrightarrow \alpha_1 (y^{(1)})^2 + \alpha_2 y^{(1)} y^{(2)}) = y^{(1)}\zeta \Longrightarrow$$
$$\alpha_1 (y^{(1)})^2 = (\zeta - \alpha_2 y^{(2)}) y^{(1)}$$

Note que estamos fazendo classificação para um modelo binário, onde $y^{(i)} = 1$ ou $y^{(i)} = -1$. Portanto, temos que $(y^{(i)})^2 = 1$ para qualquer das tipo de classe. Dessa forma, podemos concluir que α_1 é da forma

$$\alpha_1 = (\zeta - \alpha_2 y^{(2)}) y^{(1)}$$

Por fim, iremos reescrever $F(\alpha)$ sendo

$$F(\alpha) = F(\alpha_1, \alpha_2, \ldots, \alpha_m) = F((\zeta - \alpha_2 y^{(2)}) y^{(1)}, \alpha_2, \ldots, \alpha_m)$$

Assim, se considerarmos que $\alpha_3, \ldots, \alpha_m$ são constantes. Então, teremos que $F(\alpha)$ é uma função quadrática em α_2, onde conseguimos maximizar apenas igualando a derivada igual a zero.

Por fim, precisamos considerar também que $L \leq \alpha_2 \leq H$. Assim, iremos calcular o valor de α_2 considerando os limites inferiores e superiores da forma sendo da seguinte forma

$$\alpha_2^{\text{final}} = \begin{cases} H & \text{se } \alpha_2 > H \\ L & \text{se } \alpha_2 < L \\ \alpha_2 & \text{se } L \leq \alpha_2 \leq H \end{cases}$$

Assim, esse valor α_2^{final} é utilizado para calcular o valor de α_1. Portanto, teremos que

$$\alpha_1 = \alpha_1^{\text{final}} = (\zeta - \alpha_2^{\text{final}} y^{(2)}) y^{(1)}$$

1.1.4 SVM com *kernels*

As SVM's permitem que se utilize features . Assim, para cada *feature* da forma $f_i(x)$, é obtido alguma medida de similaridade $< x, l >$ entre os pontos de treinamento x e um ponto de referência $l^{(i)}$ chamado de *landmark*. Um exemplo desses *landmarks* seria usar os próprios dados de treinamento. Assim, dado um par de elementos do treinamento $(x^{(i)}, y^{(i)})$ com $i = 1, 2, \ldots, m$, um *landmark* possível seria tomar $l^{(i)} = x^{(i)}$. Assim, teremos que a função de predição será da forma

$$f(x) = \sum_{i=0}^{m} \theta_i f_i(x)$$

Além disso, a SVM prevê $y = 1$ se $f(x) \geq 0$ e $y = -1$ caso contrário. Assim, temos que a função de perda durante o treinamento será da forma

$$J(\theta) = \arg\min_{\theta} C \sum_{i=1}^{m} y^{(i)} \text{custo}_1(\theta^T f(x^{(i)})) + (1 - y^{(i)}) \text{custo}_1(\theta^T f(x)) + \frac{1}{2} \sum_{i=1}^{m} \theta_i^2$$

1.1.5 SVM para multi-classes

Os modelos de SVM são praticamente aplicados em problemas de classificação binária, no qual divide classifica o dado em 1 se for da classe que procuramos e -1 caso contrário. Para o problema de múltiplas classes, a ideia é aplicar o mesmo princípio para todas as classes. Dessa forma, o problema é decomposto em vários problemas de classificação binária.

Nessa parte, é comum utilizar técnicas que são mais convencionais nesses problemas. Nesse artigo, a abordagem para classificação de multi-classes será a de *one-vs-one*[3]

2 Modelo de *PCA*

A ideia principal dessa metodologia é reduzir a redundância de dados. Em termos matemáticos, queremos projetar um vetor (que neste caso é um dado) $x \in R^m$ para o vetor $\mu \in R^n$ e minimizar a soma das distâncias ao quadrado de cada ponto em x a μ.

Em outra palavras, o *PCA* minimiza a distância perpendicular a reta. Além disso, a variância é maximizada e a quantidade de variáveis é diminuída. Portanto, o *PCA* é uma excelente alternativa para acelerar o processamento dos dados aplicando em modelos de redes neurais, pois diminui a dimensão do problema, e é um meio válido para visualizar a relação de algumas variáveis. Por fim, o mapeamento do método é feito somente para os dados de treinamento e depois aplicado no conjunto de teste.

2.1 Intuição do algoritmo

Dado um conjunto de treinamento da forma $\{x^{(1)}, x^{(2)}, \ldots, x^{(m)}\}$, é necessário normalizar os dados de tal forma que a média seja 0 e a variância seja 1. Com isso, iremos adotar que a relação

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

Se os dados estiverem na mesma escala, basta tomar que $x_j^{(i)} := x_j^{(i)} - \mu_j$. Caso contrário, iremos tomar que $x_j^{(i)} := \dfrac{x_j^{(i)} - \mu_j}{S_j}$, onde S_j é o desvio padrão em relação a x_j.

Além disso, sabemos que a distância da projeção de $x^{(i)}$ sobre μ à origem é o resultado do produto $x^T\mu$. Assim, tomando μ sendo o vetor unitário, então podemos maximizar o termo

$$\frac{1}{m}\sum_{i=1}^{m}((x^{(i)})^T\mu)^2 = \sum_{i=1}^{m}(\mu^T x^{(i)}(x^{(i)})^T\mu)^2 = \mu\sum_{i=1}^{m}(x^{(i)}(x^{(i)})^T)^2\mu^T.$$

Com a restrição $||\mu||_2 = 1$. Com isso, a solução é da forma $\frac{1}{m}\sum_{i=1}^{m} x^{(i)}(x^{(i)})^T$. Dessa forma, a matriz de covariância empírica, dado que $x^{(i)}$ tem média 0 é da forma

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

Assim, iremos calcular os *auto-valores* da matriz Σ utilizando a *decomposição SVD*[10]. Assim, teremos as matrizes

$$U, S, V = \text{svd}(\Sigma)$$

.

Onde a matriz U é a matriz que contém os *auto-valores* em cada coluna e que estão organizados em ordem da maior norma para menor e S é a matriz diagonal. Além disso, será definido uma matriz U_{red} que a matriz U reduzida contendo os as k primeiras colunas de U e o vetor $z^{(i)}$ sendo

$$z^{(i)} = U_{\text{red}}^T x^{(i)} = \begin{vmatrix} \mu^{(1)} \\ \mu^{(2)} \\ \vdots \\ \mu^{(k)} \end{vmatrix} x^{(i)}.$$

Onde $\mu^{(1)} \in R^{1 \times n}$ e $x^{(i)} \in R^n$. Com isso, teremos que $U_{\text{red}}^T \in R^{k \times n}$ e consequentemente $z^{(i)} \in R^k$. Note que podemos vetorizar a matriz do dados sendo da seguinte forma

$$X = \begin{vmatrix} x^{(1)} \\ x^{(2)} \\ \vdots \\ x^{(m)} \end{vmatrix}.$$

Uma observação desse modelo é que não existe mais a convenção de que $x_0 = 1$ quando ocorre atributos iguais. Por fim, teremos que $x_{\text{aprox}}^{(i)} = U_{\text{red}} z^{(i)}$

2.2 Número de componente principais

Para sabermos qual o valor ótimo de k para se utilizar na matriz U_{red}. Assim, dado que sabemos os elementos da diagonal da matriz S. Então, iremos encontrar o primeiro valor de k que satisfaça a relação

$$\frac{\sum_{i=1}^{k} S_{ii}}{\sum_{i=1}^{n} S_{ii}} \geq (100 - t).$$

Onde $t \in [0, 100] \subset R$ e é escolhi pelo usuário. Dessa forma, temos que $(100-t)\%$ da variância é retida. Ou seja, quanto maior o valor de t, maior será o erro. Em compensação, o modelo terá menos componentes.

2.3 Algoritmo do PCA

Assim, temos o algoritmo com os elementos vetorizados.

Algorithm 2: Pseudo - Código para aplicação do método do *PCA*

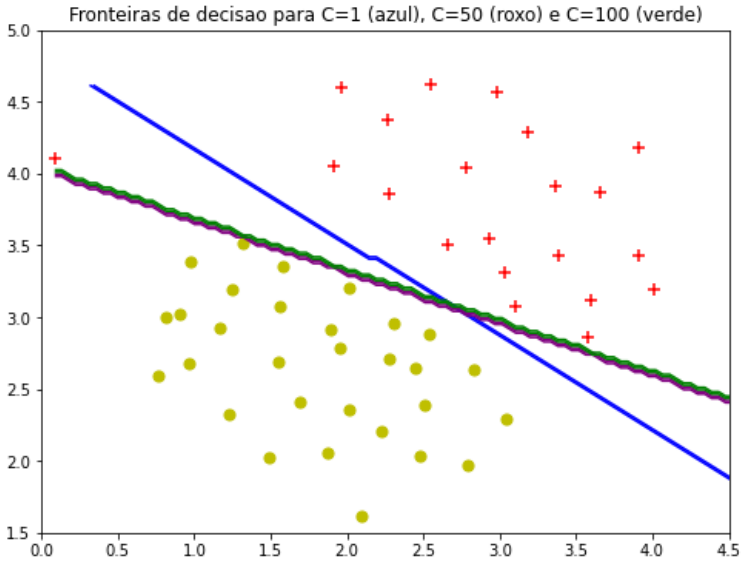
$\Sigma = \frac{1}{m} X^T X$
$U, S = \text{svd}(\Sigma)$
$k = 1$
t = escolha do usuário
for $(\frac{\sum_{i=1}^{k} S_{ii}}{\sum_{i=1}^{n} S_{ii}} \geq (100 - t))$ **do**
  $k := k + 1$
end
$U_{\text{red}} = U[:, 1 : k]$
$z = U_{\text{red}}^T X$

3 Resultados práticos

3.1 *Support Vector Machine*

3.1.1 Parte 1-a

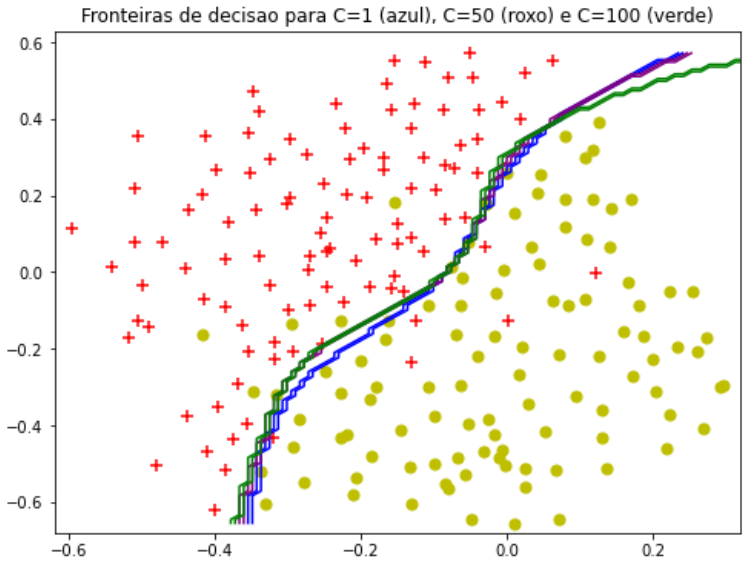
A implementação dessa foi baseada na aula da implementação do método *SVM* em *Python* visto em [1]. Porém, com uma pequena modificação para treinar o modelo com os valores sugeridos de $C = 1, C = 50 e C = 100$. Ao final, as três diferentes fronteiras de decisão resultantes são plotadas em um mesmo gráfico, juntas com as variáveis de treinamento, representadas com o símbolo $+$ quando $y(i) = 1$ e com um o quando $y(i) = 0$. Neste gráfico, a linha azul representa o resultado para $C = 1$, a linha roxa, $C = 50$, e a linha verde, $C = 100$. O gráfico resultante está representado abaixo.



Analisando o gráfico, a linha azul parece ser a aproximação mais racional para a fronteira de decisão, apesar de errar a classificação de um *outlier*. Para as linhas roxa e verde, vemos que o modelo busca se aproximar mais do *outlier* conforme é aumentado o valor de C, porém a fronteira de decisão em relação ao resto dos dados fica cada vez mais não-homogênea em relação ao distanciamento de todos os outros pontos de treinamento. Outro ponto importante a se notar é que as linhas roxa e verde estão muito próximas, então quase não há diferenças entre $C = 50$ e $C = 100$.

3.1.2 Parte 1-b

Para esta parte, utilizamos a mesma implementação da parte 1-a, mudando apenas o *kernel* dos *classifiers* para *kernel* gaussiano. O conjunto de treinamento considerado foi o que está armazenado nas variáveis X e y. O gráfico resultante está representado abaixo.



Observando o gráfico, é difícil dizer qual curva está melhor do que as outras. Algo que podemos notar é que, quando $C = 100$, a fronteira de decisão desvia do resultado das outras, pois busca se aproximar melhor dos *outliers* que estão presentes em ambas as regiões positiva e negativa.

3.1.3 Parte 1-c

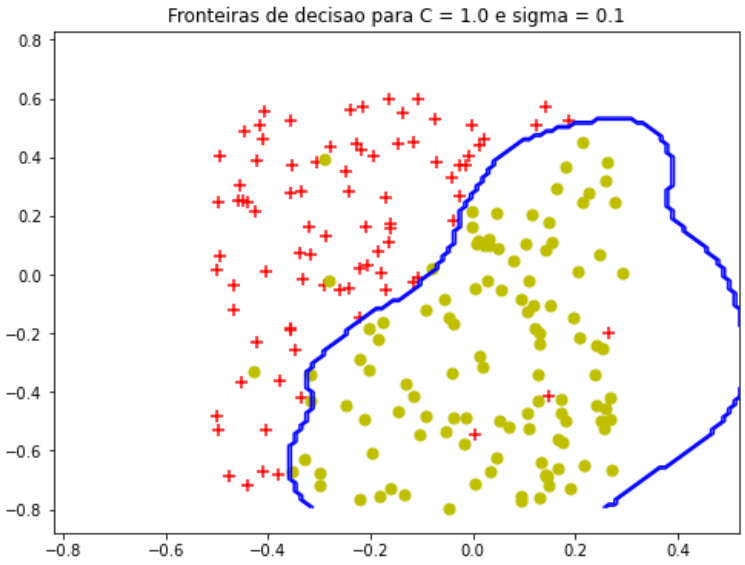
Nesta parte, tomamos como base a implementação anterior. A partir dela, construímos um programa com um *loop* que considera todas as combinações dos valores sugeridos para C e Σ e os usa para treinar o modelo a partir das variáveis de treinamento presentes em X e y, e a partir disso calcula a acurácia em relação às variáveis de validação X_{val} e y_{val}. Para esses cálculos, chamamos a função com o seguinte comando :

$$\text{classifier} = \text{SVC}(\gamma = 1/(2 * \Sigma * *2), C = C, \textit{kernel} = \text{"rbf"})$$

De modo que, de acordo com a documentação do pacote *sklearn*, SVC(*kernel* = "rbf") utiliza um *kernel* similar ao gaussiano, com o *kernel* sendo

$$K(x,z) =< x,z >= e^{(-\gamma * ||x-z||'2)}$$

Assim, consideramos todos os valores possíveis de sigma e chamamos a função assim como na linha de código acima. O programa imprime os valores de acurácia para cada combinação de C e Σ. Depois, armazena o conjunto de C e Σ que geraram a melhor acurácia de todas, *printando-os* junto de sua acurácia obtida, e em seguida roda novamente o treinamento e plota a fronteira de decisão obtida em cima dos dados de validação. Para a nossa implementação, obtivemos $C_{\text{ótimo}} = 1$ e $\Sigma_{\text{ótimo}} = 0.1$, para os quais obtivemos acurácia = 96%. O gráfico resultante está abaixo.



Observando o gráfico e os resultados obtidos, vemos que a fronteira de decisão ficou ótima em comparação aos dados de validação.

3.2 Modelo de *PCA*

3.2.1 Parte 1-a

Para a primeira parte do projeto plotamos as 100 imagens de forma aleatória, isso significa que a cada vez que o código é executado 100 imagens diferentes ocuparão o *plot*.

Figure 1: Exemplo de imagens obtidas aleatoriamente do conjunto de dados

3.2.2 Parte 1-b

Na visualização dos *eigenfaces* correspondentes aos 36 primeiros componentes principais, podemos ver que as imagens dos rostos já podem ser visualizadas, mesmo com poucas colunas da nossa matriz de autovetores. Assim, podemos ver que resultados expressivos podem ser obtidos

Figure 2: Alguns dos resultados obtidos para os 36 primeiros componentes principais

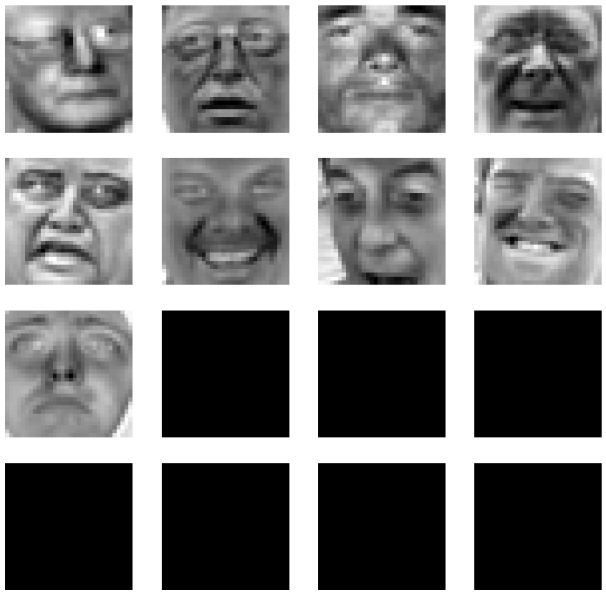
3.2.3 Parte 1-c

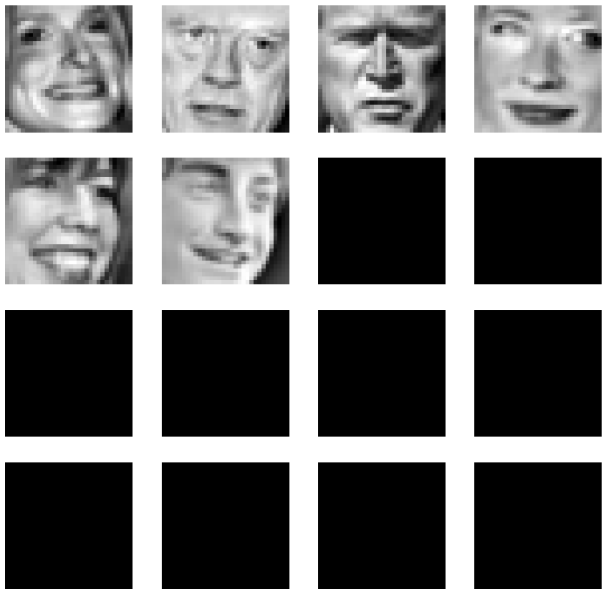
Temos que a matriz X projetada sobre os 100 componentes principais e a comparação e podemos visualizar as fotos reais, no caso pegamos as 36 primeiras imagens do nosso X para comparar

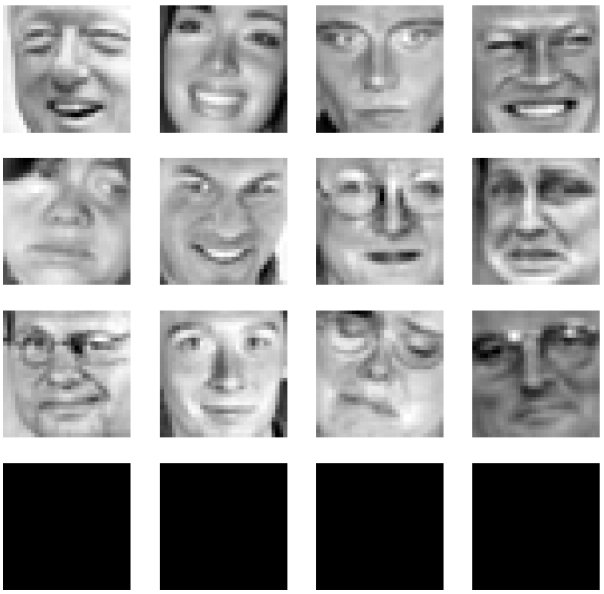
Figure 3: Comparação entre as imagens respectivamente obtidas pelo *PCA* e com as imagens originais do conjunto de dados

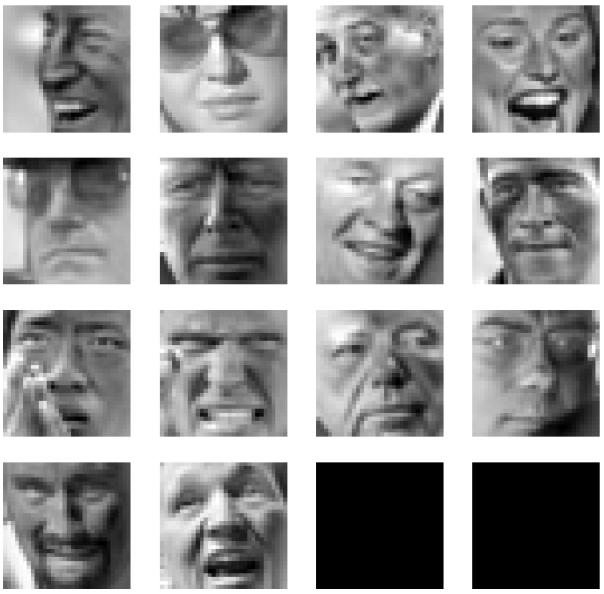
3.3 Utilização do *k-means* junto com o *PCA*

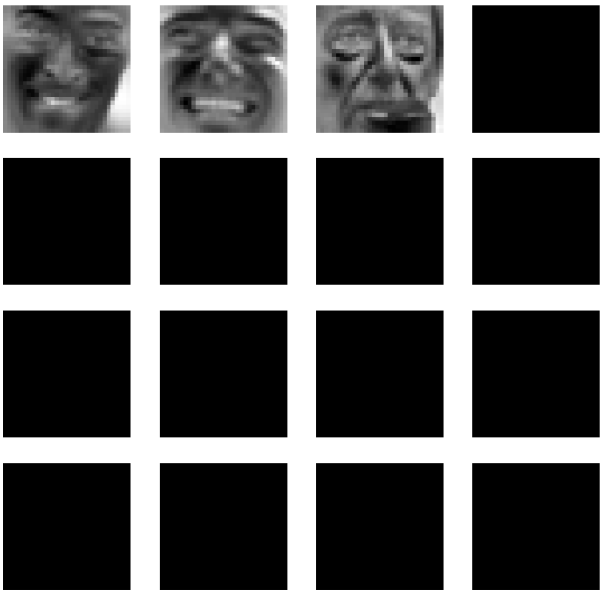
Nesta parte, utilizamos o método de *k-means*[9] para separar nossos grupos, comprimimos nossos dados para um total de k=2 componentes principais, criamos 10 centroides com o objetivo de separar em 10 grupos um total de 100 faces, também deixamos um total de 200 iterações:

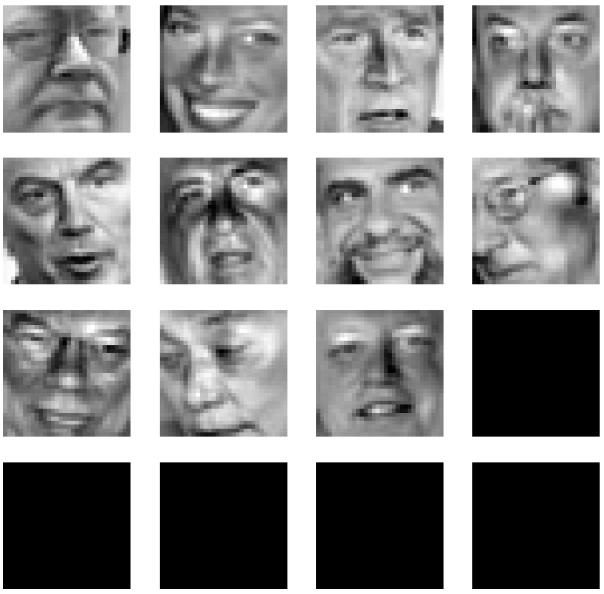












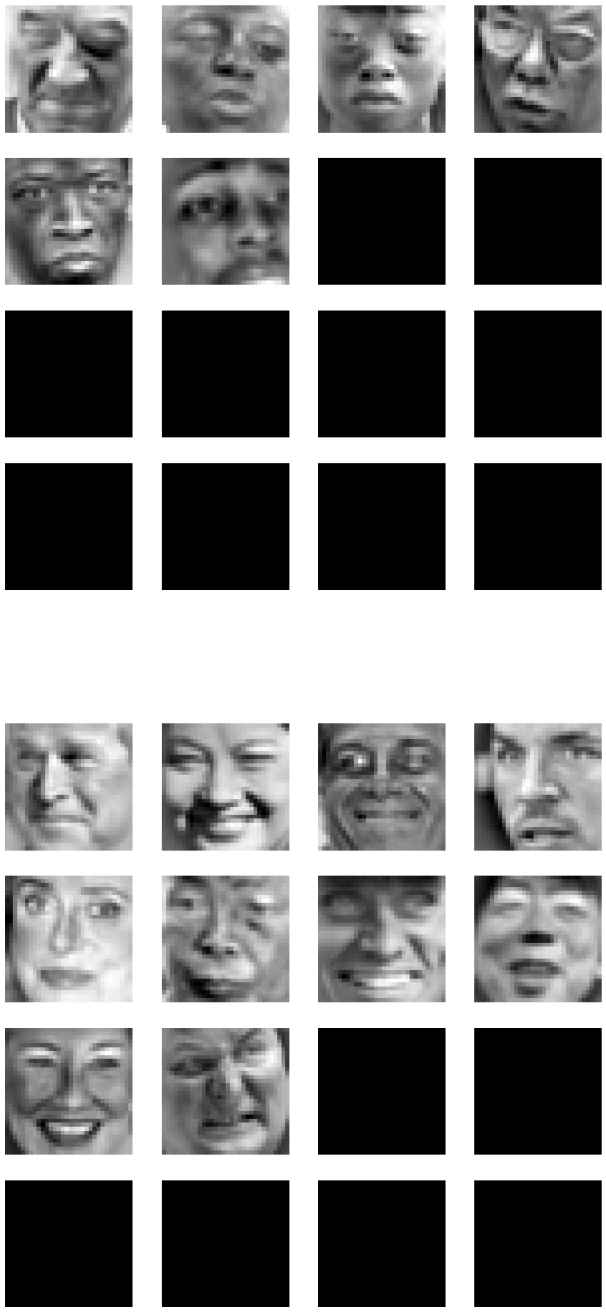
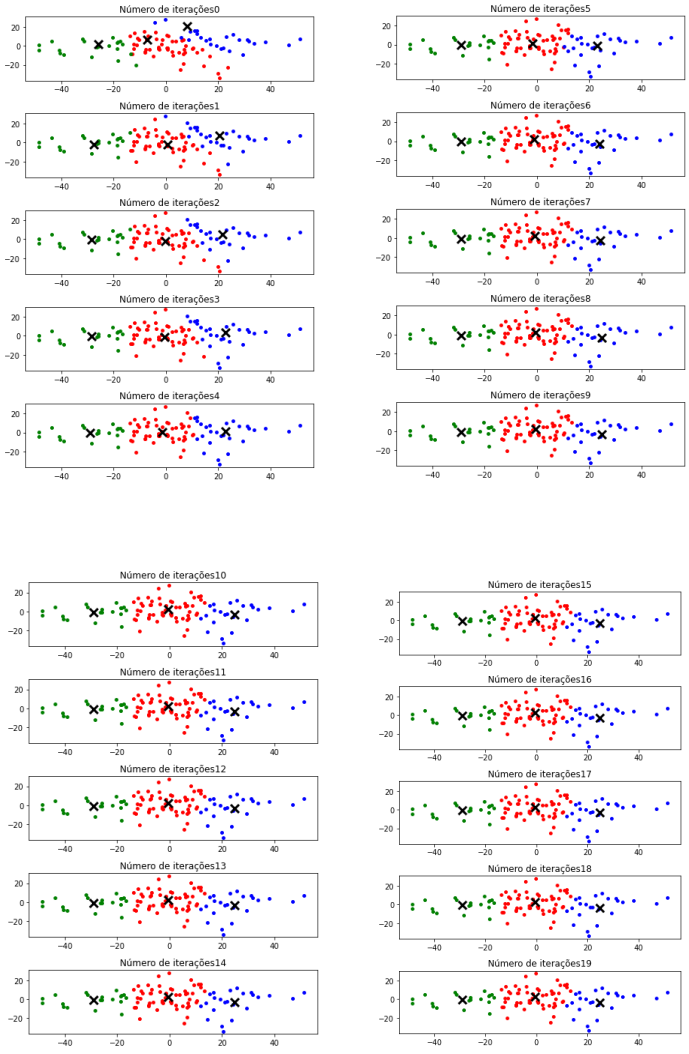


Figure 4: Resultados obtidos utilizando *k-means* para os 10 grupos

Com o objetivo de utilizar a regressão logística junto com o *k-means*, o nosso próximo experimento foi separar em 3 grupos, com um total de 100 faces e uma compressão de $k = 2$ componentes principais, desses 3 grupos selecionamos 9 imagens de um e 1 imagem de outro e formamos uma outra matriz com os dados. Nosso objetivo aqui era que depois que fizermos os testes com os dados de treinamento, os dados correspondentes ao grupo diferente sempre se destacasse dos demais. Fazendo assim com que ser acerto seja superior aos que eram do mesmo grupo, por terem características semelhantes. Para esse teste, utilizamos na regressão logarítmica $\alpha = 0,1$ e 1000 iterações



Além disso, foi obtido três grupos separados a seguir

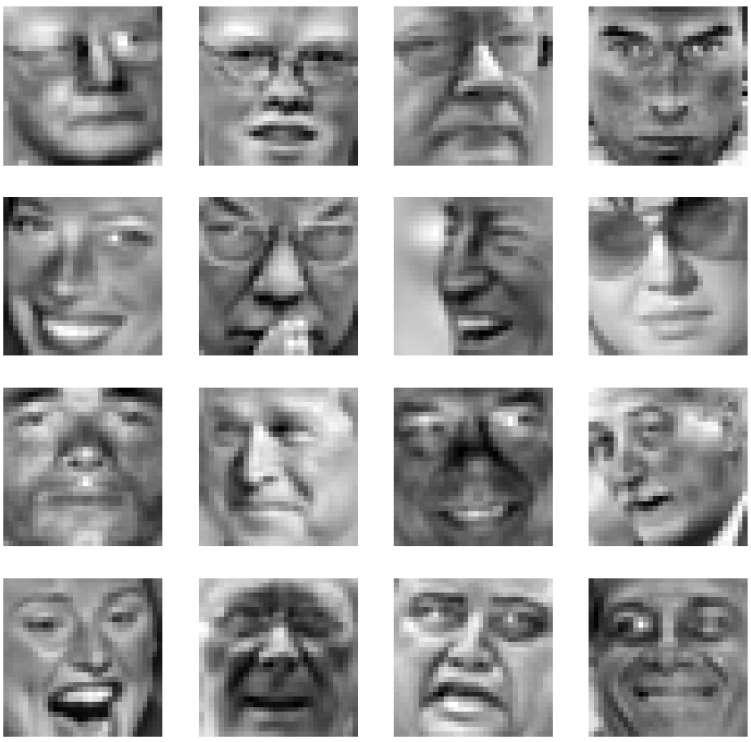


Figure 5: Imagens do grupo A)

Figure 6: Imagens do grupo B)

Figure 7: Imagens do grupo C)

Com a distribuição de probabilidade sendo $\alpha = 0,1$ e com 1000 iterações.

	0	1	2	3	4	5	6	7	8	9
0	0.999997	5.1563e-12	0.520162	0.917571	0.400232	1.20796e-07	6.16347e-13	0.0110958	0.0164659	0.99376
1	4.78734e-23	0.999904	1.7262e-12	1.96885e-11	4.31119e-14	1.32459e-08	0.027472	1.75635e-73	0.00690448	2.76521e-10
2	1.11076e-12	0.000255715	1.06805e-07	1.64492e-06	5.41008e-09	0.000153393	2.01585e-05	4.82951e-23	3.2007e-05	5.11775e-11
3	7.19412e-10	2.71769e-06	2.03941e-06	3.35322e-05	1.28056e-07	0.00129574	2.5825e-06	1.69232e-10	9.73852e-06	4.8384e-09
4	8.33668e-15	0.0045566	1.28841e-08	1.97066e-07	5.21985e-10	8.90407e-05	0.0001209	1.65239e-29	3.84656e-05	1.59936e-12
5	5.45147e-18	0.000416218	2.17883e-09	5.54791e-08	2.99471e-11	0.949333	0.0316697	0.0238513	8.07401e-09	6.31149e-15
6	2.68479e-27	0.999647	1.08559e-13	2.11067e-12	7.90825e-16	0.00417868	0.956272	1.19196e-47	1.01391e-06	1.88157e-21
7	3.85543e-10	4.37354e-07	2.4542e-06	4.77777e-05	1.17386e-07	0.0795716	8.48707e-06	0.964624	5.73583e-07	2.78295e-09
8	4.35551e-11	0.811856	4.43311e-08	2.73365e-07	9.94761e-09	2.05952e-14	3.81965e-08	2.04735e-83	0.993249	1.25782e-09
9	5.68096e-11	0.000250287	3.63472e-07	4.7119e-06	2.84408e-08	2.71803e-06	1.76091e-06	1.91727e-30	0.000549082	9.33687e-10

Figure 8: Valores da distribuição de probabilidade obtidos utilizando *k-means* para os 10 grupos para $\alpha = 0,1$ na regressão logarítmica

Já as imagens classificadas erroneamente, temos que a primeira imagem seria a imagem do grupo distinto (imagem 9) e as demais do mesmo grupo, essas foram as que se confundiram com a imagem 9:

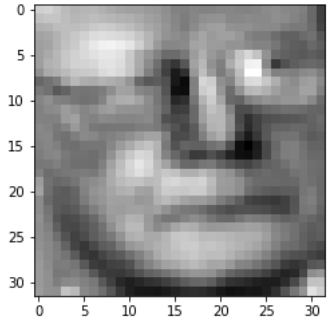


Figure 9: Imagem 10 obtida como classificação errada

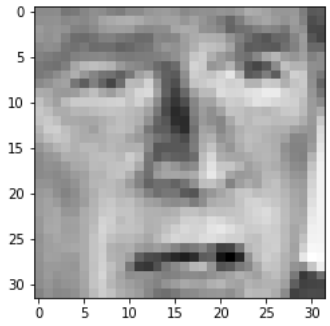


Figure 11: Imagem 12 obtida como classificação errada

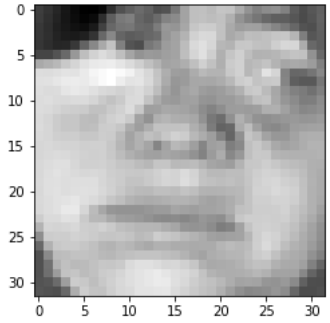


Figure 12: Imagem 13 obtida como classificação errada

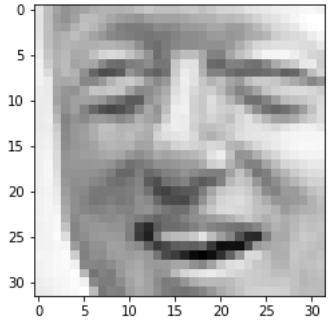


Figure 10: Imagem 11 obtida como classificação errada

4 Conclusão

Com os resultados obtidos, foi possível notar que o modelo de SVM alcançou uma precisão em torno de 96% na tarefa de classificação, o que é um resultado com uma taxa elevada de acerto.

Já para o modelo do PCA, foi realizado uma comparação visual dos resultados. Com isso, foi notado que o PCA conseguiu identificar grande parte dos rostos e realizou uma classificação correta em grande parte das imagens. Apenas em poucos casos (mostrado anteriormente nos resultados obtidos) obteve uma classificação errada, no qual não identificou nenhum rosto na imagem.

No geral, ambos os modelos apresentaram um rendimento excelente para essas tarefas de classificação abordadas neste relatório

References

[1] Joao B. Florindo. Ms960 : Tópicos especiais em processamento de imagens. https://www.youtube.com/playlist?list=PLGwGFVrptiyRmFoDWxruNGgTu2cSPnTLX, 2020.

[2] Aurelien Geron. Hands-on machine learning with scikit-learn and tensorflow: Concepts, tools, and techniques to build intelligent systems, 2017.

[3] Anthony Gidudu, Gregory Hulley, and Tshilidzi Marwala. Image classification using svms: One-against-one vs one-against-all. *CoRR*, abs/0711.2914, 11 2007.

[4] Kaiming He, Georgia Gkioxari, Piotr Dollár, and Ross Girshick. Mask r-cnn, 2018.

[5] Bagesh Kumar, Ayush Sinha, Sourin Chakrabarti, Aashutosh Khandelwal, Harsh Jain, and Prof. O. P. Vyas. Sequential minimal optimization for one-class slab support vector machine, 2020.

[6] Yiming Liu, Pengcheng Zhang, Qingche Song, Andi Li, Peng Zhang, and Zhiguo Gui. Automatic segmentation of cervical nuclei based on deep learning and a conditional random field. *IEEE Access*, PP:1–1, 2018.

[7] Hucker Marius. Multiclass classification with support vector machines (svm), dual problem and kernel functions. https://towardsdatascience.com/multiclass-classification-with-support-vector-machines-svm-kernel-trick-kernel-functior

[8] Andrzej Maćkiewicz and Waldemar Ratajczak. Principal components analysis (pca). *Computers Geosciences*, 19(3):303 – 342, 1993.

[9] O. J. Oyelade, O. O. Oladipupo, and I. C. Obagbuwa. Application of k means clustering algorithm for prediction of students academic performance, 2010.

[10] Rowayda A. Sadek. Svd based image processing applications: State of the art, contributions and research challenges, 2012.

[11] Stephen J. Wright. Coordinate descent algorithms, 2015.

[12] Shuai Zhang, Lina Yao, Aixin Sun, and Yi Tay. Deep learning based recommender system. *ACM Computing Surveys*, 52(1):1–38, Feb 2019.