Projeto III de MS960

Máquinas de Vetores de Suporte e Análise de Componentes Principais

Gianlucca Valente Ribeiro Tesseroli
g173047@dac.unicamp.br

Otávio Moreira Paiano
o185284@dac.unicamp.br

Instituto de Matemática, Estatística e Computação Científica
Universidade Estadual de Campinas

19 de Dezembro de 2020

Resumo

Nesse trabalho, usamos algorítmos de SVM ("Suport Vector Machines") e PCA ("Principal Component Analysis") para resolver problemas de classificação e redução de dimensionalidade (no caso do PCA).

No caso do SVM, usamos dois bancos de dados, onde usamos os Kernels linear e Gaussiano. Para o Kernel Gaussiano, também usamos um conjunto de validação para determinar os melhores hiperparâmetros, C e σ, os quais representam a constante de regularização e o coeficiente do Kernel Gaussiano, respectivamente.

Para o PCA, verificamos o problema de identificação de rostos. Visualizamos as 36 principais *eigenfaces* após reconstituição para o formato de imagem e reconstruímos 100 rostos aproximados a partir dos 100 primeiros *eigenfaces*. Além disso, observamos como a redução de dimensionalidade pode deixar mais eficiente os processos de detecção por reconstituição do dado original e aprendizado (no caso de usar a dimensão inferior para treinar um algorítmo supervisionado, por exemplo).

Sumário

I Introdução

Através de algorítmos de SVM ("Suport Vector Machines") e PCA ("Principal Component Analysis"), resolvemos problemas de classificação e redução de dimensionalidade (no caso do PCA).

Para o SVM, tratamos dois conjuntos de dados, um que foi possível resolver com uma fronteira de decisão linear e outro que usamos um Kernel Gaussiano para definir a fronteira. Para o Kernel Gaussiano, determinamos qual eram os melhores hiperparâmetros através do uso do conjunto de validação.

Através do PCA, aplicamos a redução de dimensionalidade (de 1024 para 100 atributos) e reconstruímos os dados originais de forma aproximada, utilizando apenas os 100 primeiros autovetores. O problema estudado foi o de identificação de rostos. Nesse contexto, é comum denominar os autovetores de *eigenfaces*.

II SVM

O conceito das Máquinas de Vetores de Suporte (SVM, "Suport Vector Machines") se baseia na busca por uma fronteira de decisão com base em um conjunto de dados de entrada e sabendo as respectivas classificações, que maximiza a margem, que é a distância entre os pontos mais próximos de cada uma das classes.

O algoritmo de SVM pode ser implementado com diferentes Kernels, entre os principais estão o *linear*, o *polinomial*, o *logístico ("sigmoid")* e o *gaussiano*. Nesse trabalho foram utilizados os Kernels linear e gaussiano.

Para resolver o problema proposto de SVM para classificação, usamos a biblioteca `sklearn` da linguagem `python`. A função utilizada foi `sklearn.svm.SVC`, uma implementação específica para problemas de classificação.

II.i Kernel Linear

O primeiro caso resolvido está apresentado na Figura (1). Nota-se que, para esse caso, existe um *outlier* que ficou do lado oposto ao indicado no seu rótulo.

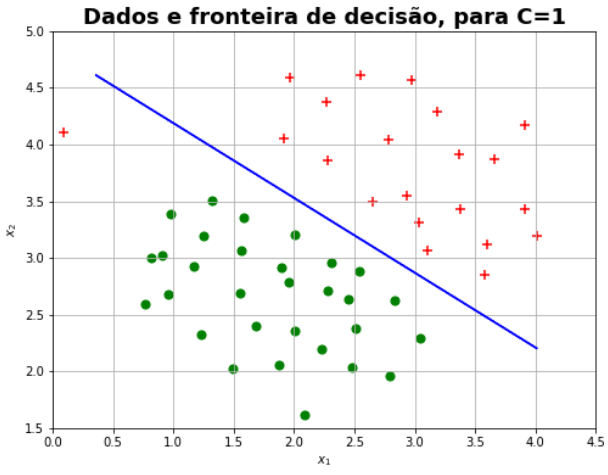

Figura 1: distribuição dos dados no espaço $(x_1, x_2) \in \mathbb{R}^2$ e a fronteira de decisão obtida para $C = 1$.

Ao aumentarmos o valor da constante de regularização para $C = 50$, observamos que o *outlier* não fica mais do lado oposto da fronteira de decisão. Entretanto, o custo para obter essa taxa de 100% de acerto foi deformar a fronteira de tal modo que, se o *outlier* for um dado rotulado incorretamente, novas previsões a partir desse treino poderão ser incorretas. Isso é fácil de perceber ao comparar as duas fronteiras. Por exemplo, se quisermos prever a classificação de um novo dado localizado em $\vec{x} = (x_1 = 2;\ x_2 = 3.5)$, obteríamos resultados diferentes de acordo com as duas fronteiras, como podemos notar nas Figuras (1 e 2).

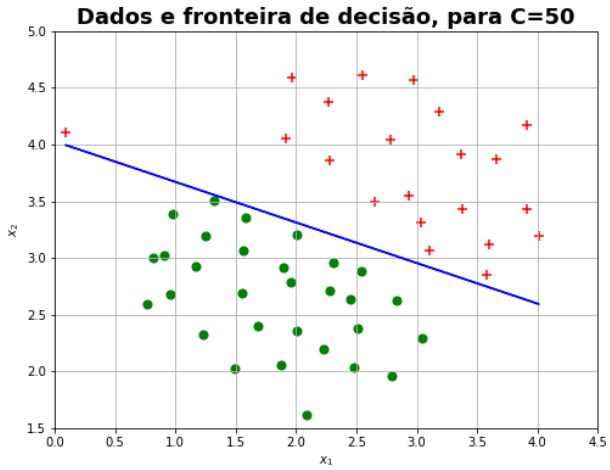

Figura 2: distribuição dos dados no espaço $(x_1, x_2) \in \mathbb{R}^2$ e a fronteira de decisão obtida para $C = 50$.

Ao aumentar o valor da constante de regularização para $C = 100$, nota-se pouca/nenhuma diferença na fronteira de decisão, como podemos observar nas Figuras (2 e 3). Isso acontece porque o aumento de C diminui a penalização imposta pela regularização na função de custo, visto que $C = 1/\lambda$, onde λ é o coeficiente linear dos termos de regularização. Assim, à medida que C aumenta, o peso da regularização diminui, de modo que de $C = 50$ para $C = 100$ pouco mudou na fronteira de decisão, pois o peso da regularização já estava caminhando assintoticamente para ficar irrelevante.

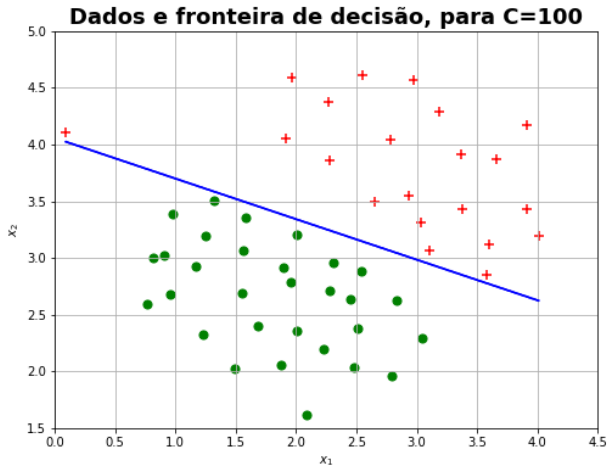

Figura 3: distribuição dos dados no espaço $(x_1, x_2) \in \mathbb{R}^2$ e a fronteira de decisão obtida para $C = 100$.

Por fim, é útil lembrar que a solução para o problema de SVM Linear supõe que para C "muito grande"conseguimos zerar toda a função de custo[1], f, exceto o termo vinculado à regularização, de forma que o problema se torna minimizar os termos de regularização em uma região delimitada por restrições lineares [1].

Entretanto, pode não ser possível minimizar completamente o somatório associado à constante de regularização[2], C -o primeiro somatório de f. Nesse caso, o problema de SVM linear não pode ser reduzido a um problema de programação não linear para $\frac{1}{2}\sum_{i=1}^{n}\theta_j^2$ com restrições lineares dadas por[1]

$$\begin{cases} \theta^T x^{(i)} \geq 1, & \text{se} \quad y^{(i)} = +1 \\ \theta^T x^{(i)} \leq 1, & \text{se} \quad y^{(i)} = -1 \end{cases} \tag{1}$$

porque a região seria infactível, *i.e.*, o espaço disponível para θ seria vazio.

Como podemos ver nas Figuras (1, 2, 3), a região dadas pelas restrições lineares para esse problema é factível, pois existem soluções que classificam corretamente todos os exemplos para treino, visto que encontramos fronteira de decisão que separa os dados corretamente para "$C \to \infty$.".

Se esse não fosse o caso, deveríamos considerar a função de custo f com o primeiro somatório não nulo, de forma que não seria possível transformar o problema de minimizar f para $\theta \in \mathbb{R}^n$ no problema de minimizar $\frac{1}{2}\sum_{i=1}^{n}\theta_j^2$ sujeita às restrições das Inequações(1).

II.ii Kernel Gaussiano

O problema resolvido a seguir requer fronteiras de decisão não lineares para ser resolvido. Para isso, usaremos o Kernel Gaussiano. Dessa forma, além do parâmetro de regularização C, agora temos o paraêmtro σ associado a esse Kernel.

O hiperparâmetro σ do Kernel Gaussiano está localizado na seguinte equação:

$$K(\mathbf{x}, \mathbf{z}) = \exp\left(-\frac{\|\mathbf{x} - \mathbf{z}\|}{2\sigma^2}\right) \tag{2}$$

A primeira abordagem foi resolver o problema para $C \in \{1, 50, 200\}$, mantendo constante o parâmetro $\sigma = \frac{1}{sn} \approx 0.1411$, onde $n = 2$ é o número de atributos e s é a variância do conjunto de dados (no caso, provavelmente é a soma dos elementos da matriz de covariância dada pelos exemplos de treino).

Os resultados para as fronteiras e para os dados de treino estão nas Figuras (4, 5, 6). A performance dessas fronteiras nos conjuntos de treino e teste/validação foram as seguintes:

- $C = 1$: treino: 94.3%, validação: 95.0%;
- $C = 50$: treino: 93.8%, validação: 96.0%;
- $C = 200$: treino: 93.8%, validação: 95.0%;

[1] $f(\theta) = C\sum_{j=1}^{m}\left\{y^{(i)} f_1(\theta^T x^{(i)}) + (1 - y(i) f_2(\theta^T x^{(i)})\right\} + \frac{1}{2}\sum_{i=1}^{n}\theta_j^2$, onde m é o número de exemplos para treino e n é o número de atributos dos dados.

[2] Veja a nota de rodapé 1.

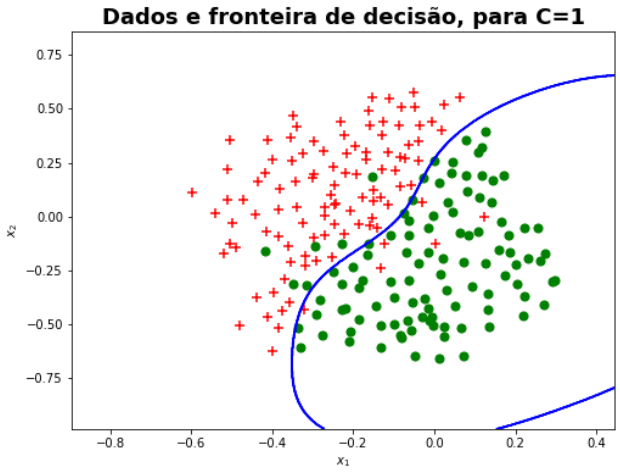

Figura 4: distribuição dos dados no espaço $(x_1, x_2) \in \mathbb{R}^2$ e a fronteira de decisão obtida para $C = 1$.

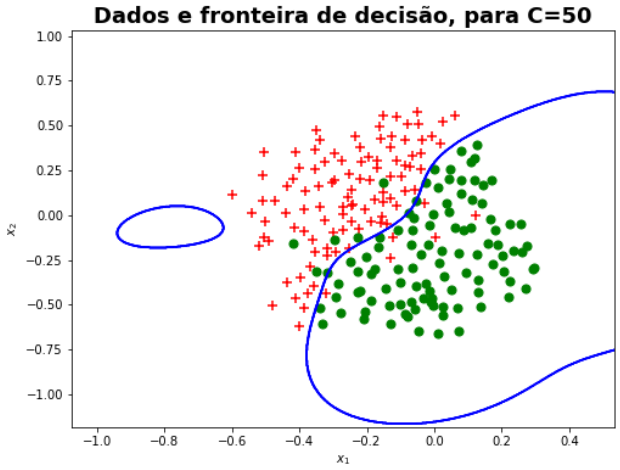

Figura 5: distribuição dos dados no espaço $(x_1, x_2) \in \mathbb{R}^2$ e a fronteira de decisão obtida para $C = 50$.

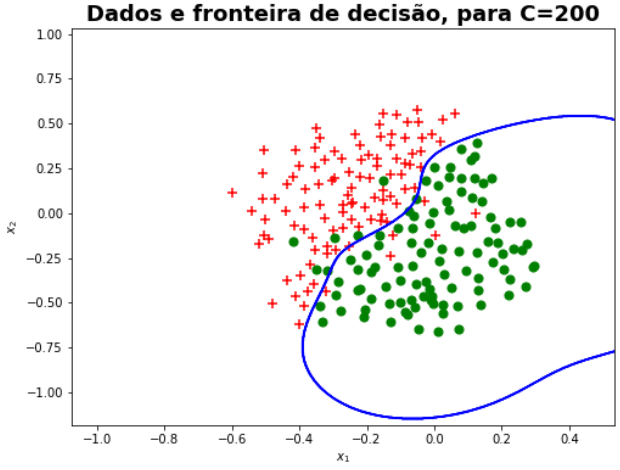

Figura 6: distribuição dos dados no espaço $(x_1, x_2) \in \mathbb{R}^2$ e a fronteira de decisão obtida para $C = 200$.

Dois aspectos chamam atenção nesses resultados. O primeiro é que o conjunto de treino performou pior do que o de teste/validação em todos os cenários. O segundo é que, à medida em que aumentamos C para diminuir o peso da regularização e permitir um maior *overfitting*, além da fronteira de decisão ter mais detalhes, ela se tornou desconexa em $C = 50$ e voltou a ser conexa em $C = 100$, mesmo sem a presença de dados na região.

O fato do conjunto de treino ter performado pior deve-se ao maior número de *outliers* e pontos muito próximos à fronteira, em comparação com o conjunto de teste/validação, que era mais "amigável", como podemos ver na Figura (7).

Desses três cenários, o melhor é $C = 1$.

O cenário para $C = 200$ é estatisticamente o pior, pois performou igual ou pior do que os outros dois para os dois conjuntos, de treino e de teste/validação.

O cenário $C = 50$, apesar de ter apresentado uma performance 1% melhor no conjunto de teste/validação e 0.5% pior no conjunto de treino, apresentou a região desconexa, consequência clara de *overfitting*. Como essa região não possui dados de treino e está do lado oposto da fronteira esperada, é muito provável que dados novos caiam nessa região e sejam classificados de forma errada. Dessa forma, a fronteira para $C = 50$ não é confiável.

Portanto, nota-se que, ao permitir o aumento de C, temos uma fronteira mais detalhada, porém mais instável, o que diminui o nível de confiança dessa configuração de parâmetros.

Otimização dos hiperparâmetros através do conjunto de validação

Para determinar a melhor combinação possível de hiperparâmetros (σ, C), foi testada a performance de cada par possível em um espaço de $[10^{-3}; 10^{5}] \times [10^{-3}; 10^{5}]$ discretizado por 1000 pontos em cada eixo em escala logarítmica. O resultado foi que existe uma região ótima, e não apenas um único valor, como podemos observar na Figura (8).

A melhor performance obtida sobre o conjunto de teste/validação foi de 97.0%, para $C = 1.72 \times 10^{-1}$ e $\sigma = 5.99 \times 10^{-2}$. Entretanto, foram detectados, no total, 451 pontos com essa mesma performance, de um total de 10^6. O alto número de combinações com a mesma performance deve-se, em parte, ao tamanho relativamente pequeno dos conjuntos, de cerca de 200 elementos (para cada um dos dois), que discretizou muito as taxas de acerto possíveis.

Por fim, após determinar qual a melhor configuração de hiperparâmetros, apresentamos o resultado com a fronteira de decisão e com os dados dos dois conjuntos na Figura (7). Observamos que, apesar do valor de C ser muito menor do que nos outros casos, a fronteira de decisão é muito mais detalhada do que nos casos anteriores, mesmo quando havia *overfitting*.

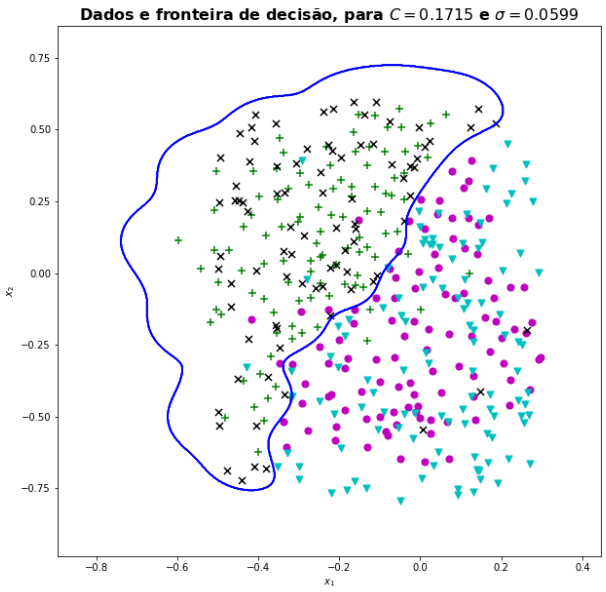

Figura 7: fronteira de decisão (em azul) obtida a partir do conjunto de treino com $C = 1.72 \times 10^{-1}$ e $\sigma = 5.99 \times 10^{-2}$. Os valores dos hiperparâmetros C e σ foram obtidos por otimização realizada através do conjunto de validação (mais detalhes estão disponíveis na Figura (8)).
Legenda para os pontos do gráfico: os símbolos × e os triângulos pertencem ao conjunto de teste/validação, e os outros dois seguem a mesma convenção das imagens anteriores (conjunto de treino).

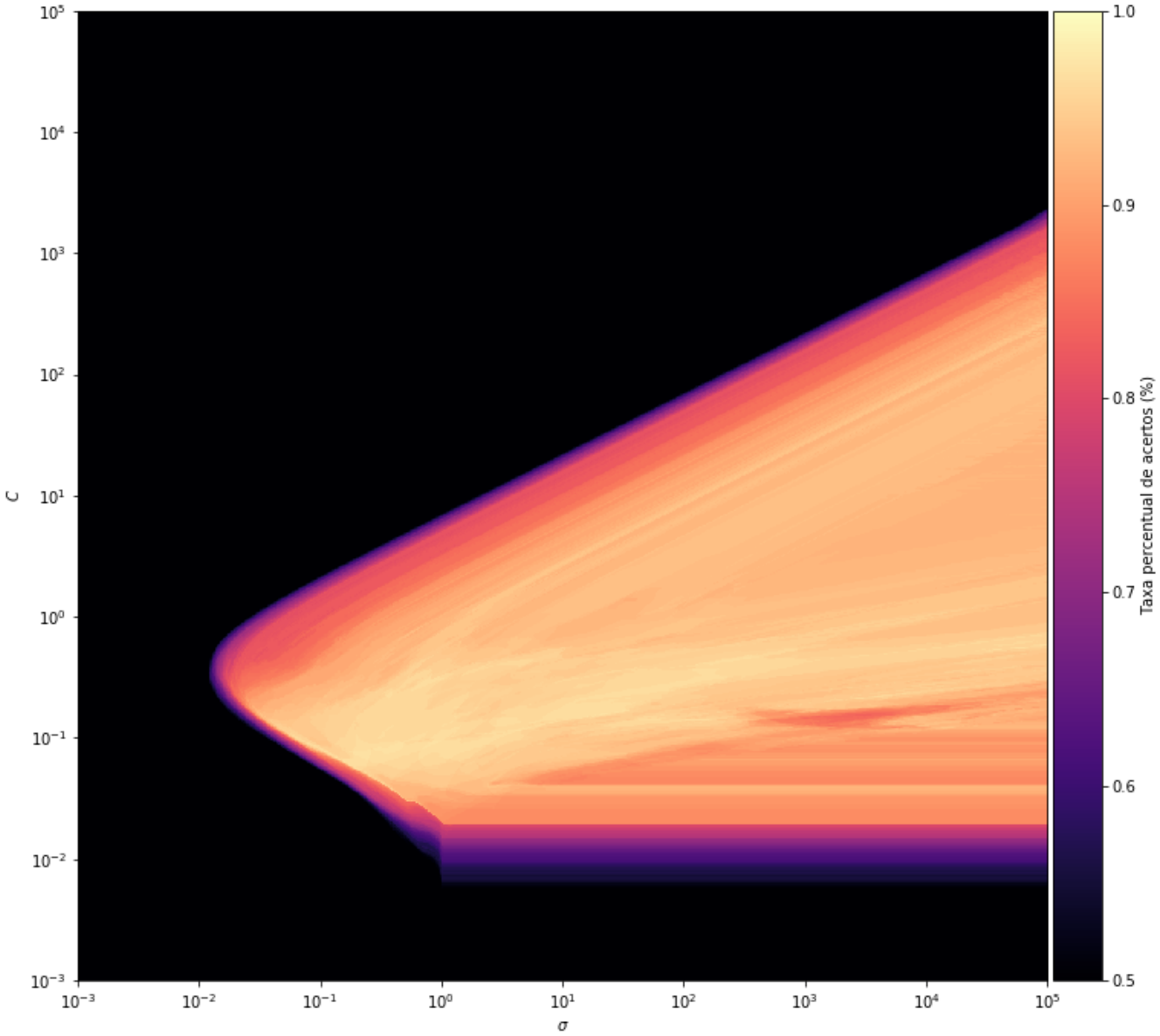

Figura 8: taxa percentual de acertos sobre o conjunto de validação para cada para $(\sigma, C) \in [10^{-3}; 10^5] \times [10^{-3}; 10^5]$. A combinação ótima encontrada foi $C = 1.72 \times 10^{-1}$ e $\sigma = 5.99 \times 10^{-2}$, com 97% de taxa de acerto sobre o conjunto de validação (entretanto, essa não foi a única combinação que apresentou essa taxa de acerto, mas foi a primeira -o algorítmo só atualizava se encontrasse um valor de acerto maior). Apesar disso, notamos que existe uma região com uma boa taxa de acertos. Como os conjuntos de teste e validação não eram muito numerosos (cerca de 200 exemplos em cada, o que inclusive contribuiu para que existissem várias combinações ótimas, visto que os acertos ficaram muito discretizados, podendo assumir apenas valores i/m_v, onde $i \in [0, m_v] \cap \mathbb{N}$ e $m_v \approx 200$ é o número de elementos no conjunto de validação), é possível que, para outra amostragem de dados os valores ótimos de (σ, C) se desloquem um pouco nessa região clara da imagem. Por fim, é interessante notar que as piores performances correspondem a cerca de 50%, visto que é um problema de classificação binária e os elementos estavam distribuídos de forma aproximadamente equiprovável nos conjuntos. Mesmo assim, foram detectadas várias combinações de hiperparâmetros com taxa de acerto de $\approx$ 40%, resultado pior do que uma escolha completamente aleatória, assumindo que a probabilidade de ocorrência de qualquer um dos dois tipos é igual.

III PCA

O PCA é um algorítmo de redução de dimensionalidade e classificação não supervisionada. Com isso, é possível reduzir o tempo de execução do script (visto que um algorítmo de aprendizado teria que lidar com menos atributos após a redução de dimensões). Este é um procedimento que utiliza transformação ortogonal para converter um conjunto de observações de variáveis possivelmente correlacionadas em um conjunto de valores de variáveis linearmente não correlacionadas, chamadas de componentes principais. Para isto, utilizamos o método *.svd()* da biblioteca numpy.linalg, de Python. Com sua utilização, obtemos a matriz de decomposição ortogonal U e um vetor que contém os autovalores S, precentes na matriz diagonal. Note que este método já ordena os pares de acordo com a magnitude do autovalor, o que é imprescindível já queremos os componentes principais.

Após a obtenção dos valores de U e S, vamos particionar a matriz U, considerando apenas as k componentes principais, previamente estabelecidads. Com isso, podemos calcular os atibutos reduzidos z e, assim, conseguir recalcular os valores de nossos atributos. Por fim, calculamos a variância retida, uma vez que definimos como valor mínimo de 99% de retidão.

III.i Impressão de imagens

Em primeiro lugar, apenas imprimimos 100 imagens, através do redimensionamento dos dados de entrada, conforme Figura (11).

É possível perceber que estas imagens estão nítidas e conseguimos enxergar os traços de cada rosto.

III.ii Eigenfaces - autovetores das componentes principais

Após analisar os dados em sua forma original, demos início à implementação do PCA, de fato. Com ele, consideramos 36 componentes principais, isto é, os 36 autovetores associados aos 36 maiores autovalores, para realizar o redimensionamento dos atributos.

Após seguir o algoritmo descrito anteriormente, plotamos os dados presentes nos autovetores das componentes principais, conforme Figura (10). Nesse conjunto de imagens, podemos perceber apenas os principais traços de cada rosto. É possível identificar nariz, boca e olhos, porém, por estarmos utilizando poucos atributos (apenas 3,5% do total), perdemos a nitidez e a clareza nas imagens.

III.iii Reconstrução de imagem

Por fim, testamos o algoritmo PCA na reconstrução da imagem, de fato. Isto é, após normalizarmos a entrada e seguirmos com o raciocínio do método, utilizamos 100 componentes principais para aproximar a visualização.

Podemos medir sua eficácia, pela taxa de retidão da variância, que é dada pela seguinte equação:

$$\frac{\sum_{i=1}^{k} S_i}{\sum_{i=1}^{n} S_i} \geq 99,00\%$$

Para esse valor de componentes principais, obtivémos 93,19% para a taxa mencionada acima. Dessa forma, entendemos que 100 componentes principais não são suficientes para reter, ao menos, 99,00% da variância. Além disso, podemos perceber a diferença entre as imagens pelas Figuras (11 e 12).

Para finalizar a análise, buscamos qual o menor número k (componentes principais) que retém os 99,00% de variância. Com isso, pudemos construir o gráfico abaixo:

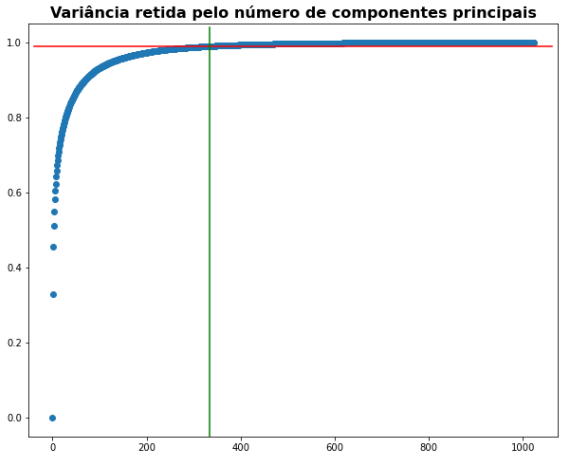

Figura 9: Valor de retenção da variância, a partir da variação do número de componentes principais (k).

A linha vermelha limita os 99,00% e a verde demonstra o menor valor de k que satisfez a condição imposta e, nesse caso, $k = 335$, cuja taxa de retenção foi de $99,0004\%$.

Além disso, é nítido o comportamento assintótico para os 100% à medida em que aumenta-se o número de componentes principais, o que faz sentido, já que quanto maior esse valor, mais próximo de todos os atributos estamos e, com isso, maior a similaridade entre as imagens.

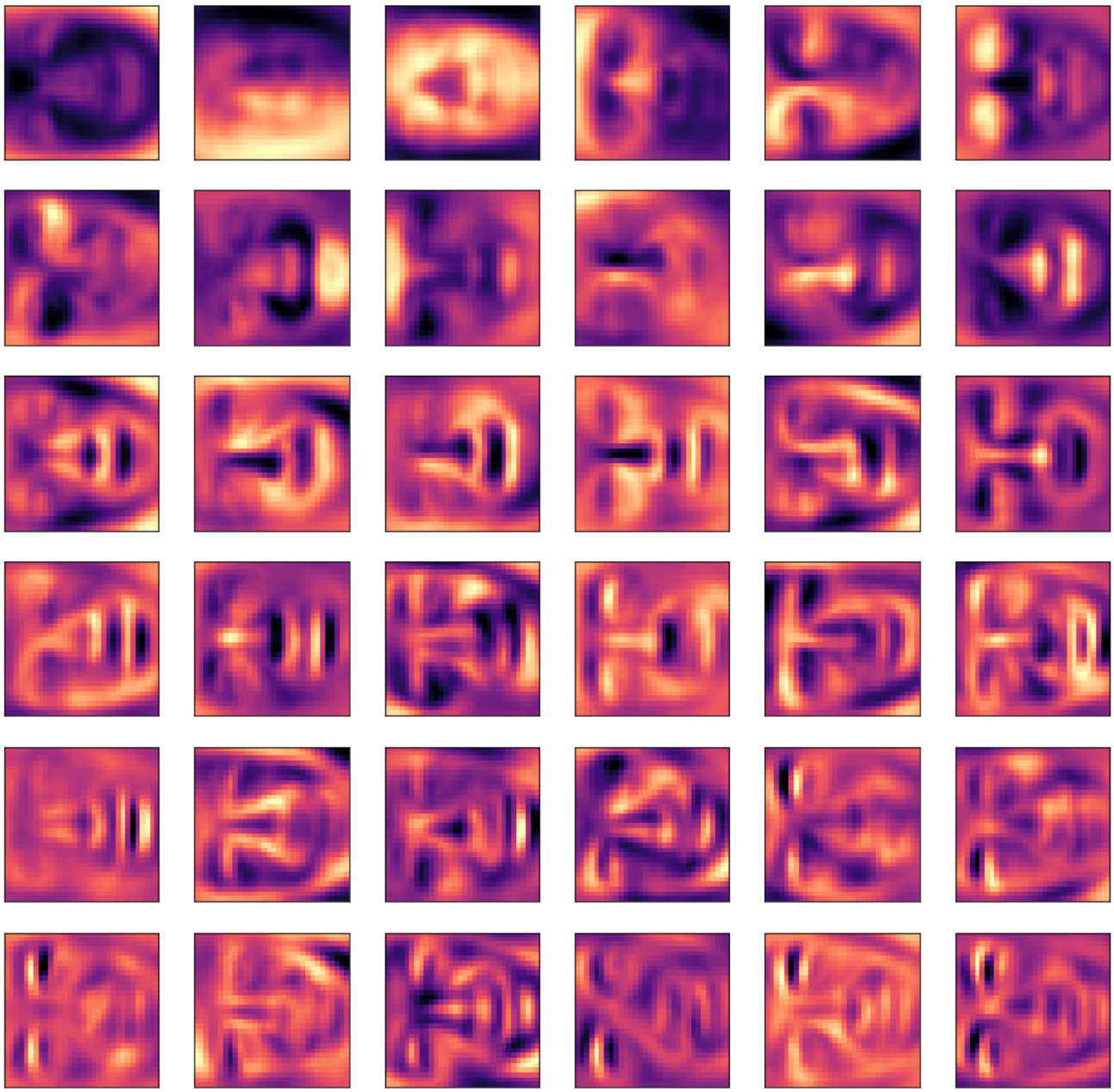

Figura 10: representação dos 36 primeiros *eigenfaces* (ordenados do maior para o menor autovalor) após reconstruir as imagens de 32×32 pixels.

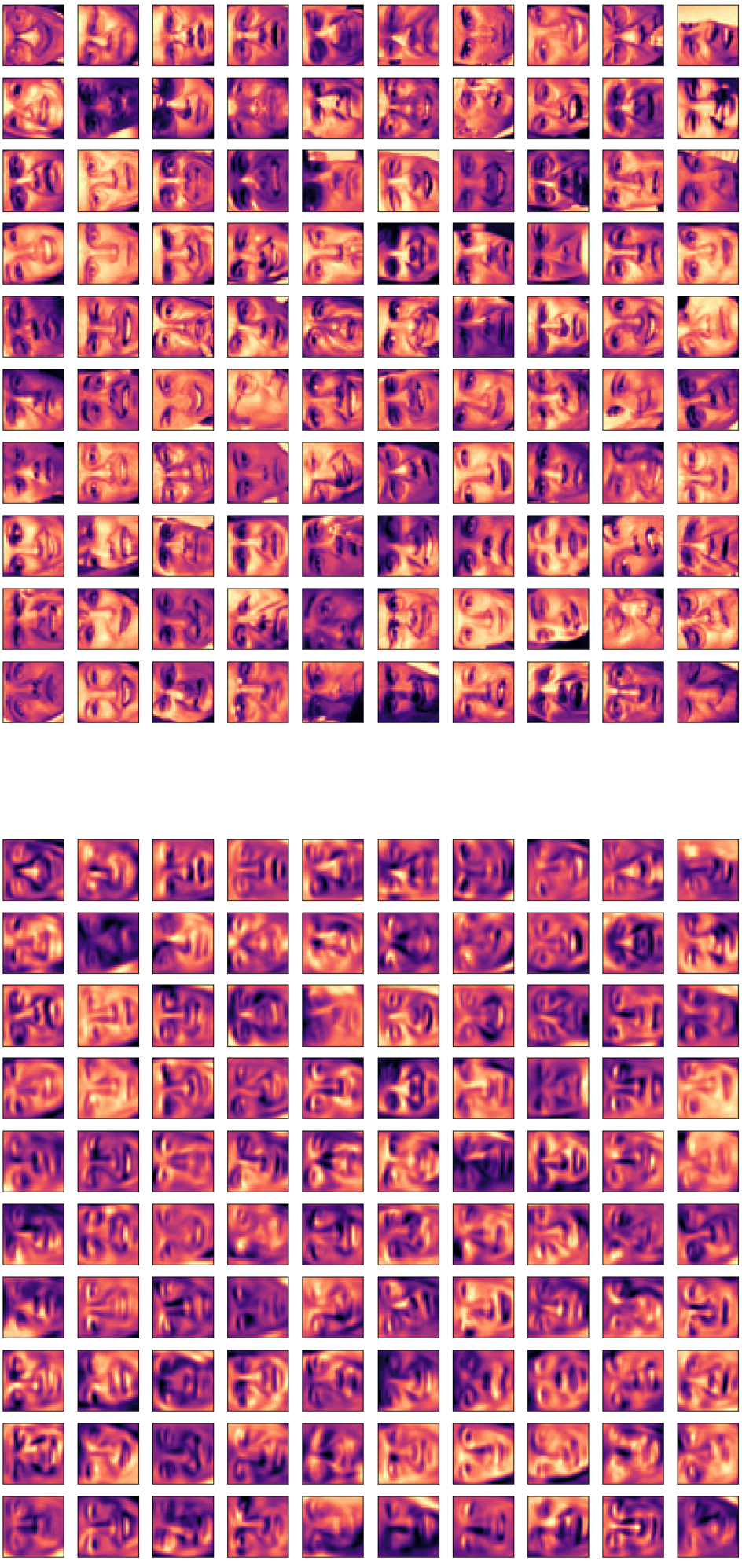

Figura 11: 100 rostos (acima) e as aproximações correspondentes (abaixo) a partir das primeiras 100 *eigenfaces*.

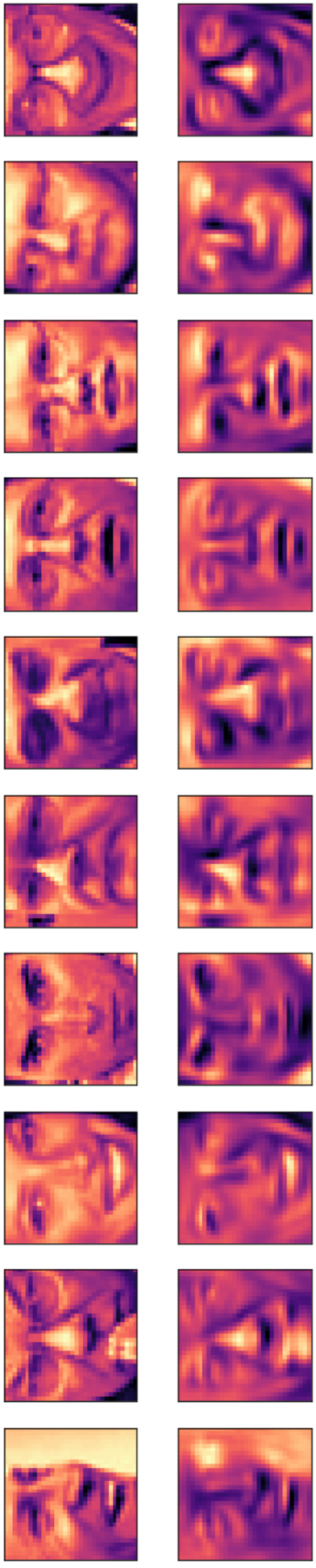

Figura 12: 10 exemplos de comparação lado a lado dos rostos originais com as reconstruções feitas a partir de 100 *eigenfaces*. .

IV Conclusão

Para o SVM de Kernel Linear, concluímos que a presença de *outliers* em um conjunto de dados que tem uma região factível para o problema de minimização dos termos de regularização, ou seja, quando é possível ter uma taxa de acerto de 100%, deforma a fronteira de decisão se a regularização tiver um peso pequeno ("$C \to \infty$").

Para o SVM de Kernel Gaussiano, a fronteira de decisão no limite entre os dois tipos estudados não muda muito com o aumento de C, talvez porque a presença de muitos *outliers* amenize o efeito de *overfitting* nessa região. Entretanto, em áreas mais afastadas dos dados estudados, foi evidente o efeito do *overfitting* para valores de C maiores. Por fim, ao procurar os melhores hiperparâmetros, encontramos uma região ótima para a performance de predição sobre o grupo de validação, e não apenas um valor único.

Para o PCA, observamos como as componentes principais são espectros de faces, e como podemos reconstituir as faces originais a partir dessas componentes. Quanto mais componentes principais usarmos, mais precisa será a reconstituição da face. Por exemplo, nas Figuras(11 e 12) temos as faces originais e as reconstruídas a partir dos 100 espectros mais relevantes. Para as faces originais, precisamos de $32^2 = 1024$ unidades de informação para cada imagem para armazenar, treinar, etc. Após usar os primeiros 100 espectros, precisamos de apenas 100 unidades de informação (o peso de cada espectro na composição da imagem original). Obviamente existe perda de informação, mas para muitos casos é possível reduzir consideravelmente a dimensão do problema (o que diminui muito o tempo de processamento para um algoritmo de aprendizado) sem perder quantidades consideráveis de performance.

Notamos também, pela Figura (9), que a variância retida converge relativamente rápido em função do número de componentes principais, k. É por esse motivo que esse método é eficiente em reduzir a dimensão do problema, pois mesmo para k relativamente baixo as informações principais dos dados ainda são preservadas.

Assim, com a redução de dimensionalidade, é possível melhorar a performance de treinamento de outros algorítmos de aprendizado (no caso, se quisermos treinar uma rede neural para detecção facial após determinar os 100 *eigenfaces* mais relevantes, bastaria "alimentar"a rede com o peso que cada *eigenface* tem na reconstituição de cada face).

Uma situação análoga à reconstituição de um dado a partir de componentes principais é no uso da transformada de Fourier em $\mathbb{R}^2$ para imagens. Nesse caso, as frequências baixas seriam mais relevantes para a reconstrução da forma geral da imagem original, enquanto as frequências altas iriam compor os detalhes da imagem.

Referências

1. João Batista Florindo, Aulas de MS960, segundo semestre de 2020, UNICAMP.

2. T. Hastie, R. Tibshirani and J. H. Friedman, The Elements of Statistical Learning; Data Mining, Inference, and Prediction. Springer, 2009.

3. S. Shalev-Shwartz and S. Ben-David, Understanding Machine Learning: From Theory to Algorithms. Cambridge University Press, 2014.

4. C. Bishop, Pattern Recognition and Machine Learning. Springer, 2006.