Projeto 3 - MS960/MT862

Fernando Ribeiro de Senna — RA 197019
Rodolfo da Silva Santos — RA 228711

18 de dezembro de 2020

Na primeira parte do projeto foi implementado conceitos de Máquinas de Vetores de Suporte com Kernel Linear e Gaussiano, com o objetivo de separar elementos entre duas classes e otimizar hiperparâmetros. A implementação foi feita no arquivo *SVM_Projeto_3_Parte_1.ipynb* e está descrita na Seção 1.

No desenvolvimento da parte 1 do projeto foram utilizados funções, módulos e objetos das bibliotecas *scipy, numpy, sklearn* e *matplotlib*.

Já na segunda parte do projeto, foi implementada ferramenta de Análise de Componentes Principais (PCA), com o objetivo de realizar redução de dimensionalidade para trabalhar com reconhecimento facial. A implementação foi feita no arquivo *PCA_part2.ipynb* e está descrita na Seção 2.

Foram utilizados funções e objetos das bibliotecas *scipy, numpy* e *matplotlib*.

Toda a fundamentação teórica se baseia em conteúdo oferecido em vídeo-aulas e *slides* pelo Professor João Batista Florindo em ocasião de oferecimento da disciplina MS960 no segundo semestre de 2020 pelo Instituto de Matemática, Estatística e Computação Científica (IMECC) da Universidade Estadual de Campinas (UNICAMP).

1 Máquinas de Vetores de Suporte — Parte I

Os conceitos, métodos e ferramentas descritos nessa seção têm por objetivo a identificação de padrões e classificação de exemplos de treinamento ou teste em classes distintas. Para isso se procurou encontrar a melhor fronteira de decisão que separasse, de forma otimizada, as classes de objetos em estudo. Todas as considerações dessa seção estão descritas no arquivo *SVM_Projeto_3_Parte_1.ipynb*.

1.1 Máquinas de Vetores de Suporte – Linear

Na implementação da máquina de vetores de suporte linear foi utilizada a função SVC da biblioteca sklearn do python. Para o fator de regularização C (C = 1/lambda) utilizou-se os

valores 1, 50 e 100. As três fronteiras de decisão foram colocadas no gráfico abaixo. Neste gráfico se vê que para a SVM linear o melhor valor para o hiperparâmetro C foi 1 por manter a fronteira de decisão mais equidistante dos exemplos de treinamento de ambas as classes. Como C = 1/lambda é de se esperar que a regularização seja mais forte quando C for pequeno ao contrário de quando se usava lambda. Logo para se evitar overfitting deve-se diminuir o valor de C.

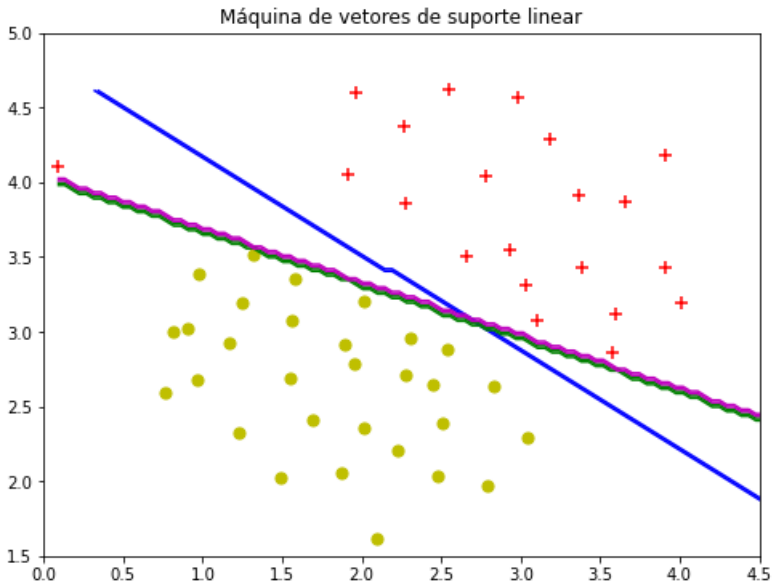

Figura 1: Azul C=1, Verde C= 50 e Magenta C= 100

Dessa forma, quanto maior o valor para a variável C mais a fronteira de decisão fica vulnerável a influencia de outliers, como visto, para as fronteiras de decisão com C valendo 50 e 100. Com o hiperparâmetros C muito grande a regularização fica praticamente anulada e o modelo fica sensível a overfitting. Ou seja, os outliers terão grande influência sobre o desenho da fronteira de decisão uma vez que não há regularização para reduzir o peso individual de cada exemplo de treinamento.

1.2 Máquinas de Vetores de Suporte – Gaussiano

A máquina de vetores de suporte gaussiano foi implementada com o uso da função SVC da biblioteca sklearn do python com a opção de kernel “rbf”. O modelo foi treinado com os dados presentes nas colunas X e y do arquivo dado2.mat, e os testes foram feitos com os dados presentes nas colunas Xval e yval do mesmo arquivo. Como resultado foi obtido o gráfico da imagem abaixo.

A fronteira de decisão foi capaz de separar as duas classes. Entretanto, de acordo com os valores de C e γ, uma parte isolada da fronteira de decisão aparece na parte inferior direita

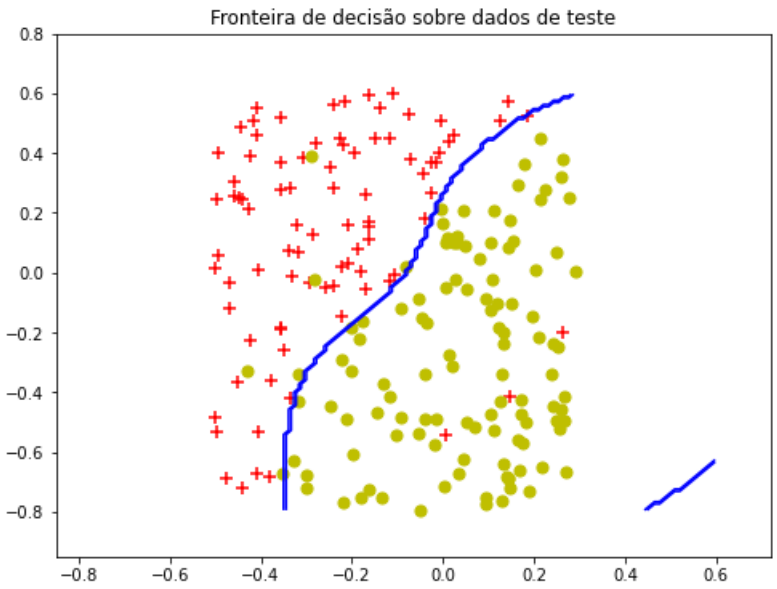

Figura 2: C= 1 e gamma = “scale”

do gráfico. Esta parte da fronteira de decisão desaparece com a calibragem dos hiperparâmetros C e γ. Isso pode ser visto na imagem a seguir.

Nesta imagem foram utilizados C = 19 e γ = 1,7. Entretanto, apesar da fronteira de decisão ficar restrita a divisão dos dois grupos de exemplos de treinamento, esses valores de C e γ não são os que melhor otimizam a fronteira de decisão. Esse fato será melhor analisado na próxima seção.

1.3 C e gamma ideais

Para otimizar os hiperparâmetros C e γ foi utilizado o módulo GridSearchCV da biblioteca sklearn. Esse módulo permite a combinação dos hiperparâmetros C e γ e a posterior avaliação dos resultados obtidos. Para isso o GridSearchCV necessita receber um dicionário com os valores de C e γ e a função SVC. O módulo retornará uma tabela com todas as combinações entre esses hiperparâmetros.

O GridSearchCV fornece funções como best_params_ que retorna os valores otimizados e best_score_ que apresenta a pontuação dos parâmetros escolhidos permitindo a comparação do desempenhos entre as combinações de parâmetros. A função best_estimator_ retorna diretamente a melhor combinação de parâmetros para a instância do módulo GridSearchCV e permiti o uso direto da função predict nela.

O hiperparâmetro gama define até onde chega a influência de um único exemplo de treinamento em relação à fronteira de decisão. Se ele tiver um valor baixo significa que cada ponto tem um alcance distante, e inversamente, um valor alto de gama significa que cada ponto tem um alcance próximo.

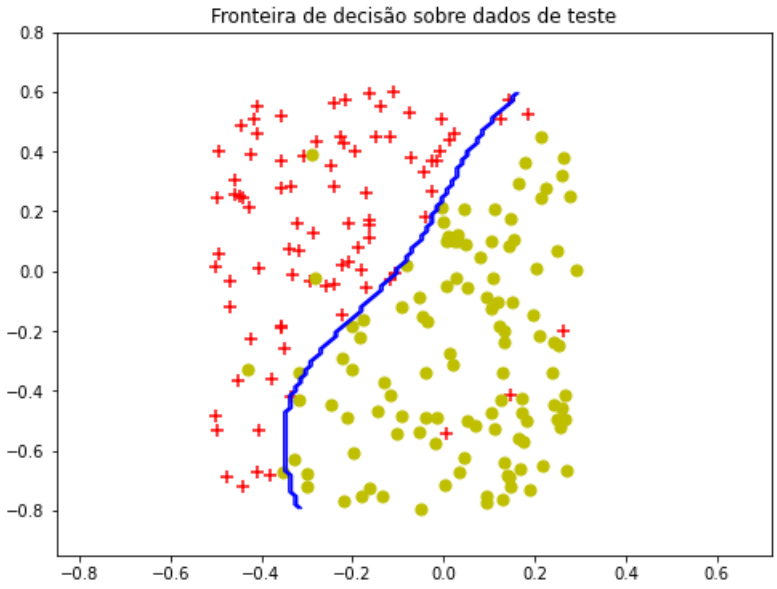

Figura 3: C= 19 e gamma = 1,7

Assim, se gama tem um valor muito alto a fronteira de decisão vai depender apenas dos pontos que estão muito próximos a ela e por causa disso terá um aspecto mais ondulado. Se gama tiver um valor muito pequeno a fronteira de decisão será muito afetada por pontos que estão distantes dela e a curva obterá um aspecto mais linear.

No treinamento dos modelos foram utilizados os valores [0.01, 0.03, 0.1, 0.3, 1, 3, 10, 30] tanto para C como para γ. Desses parâmetros foram devolvidos pela função best_params_ como melhores valores C = 30 e $\gamma = 3$. E a melhor pontuação foi 0,9194. O gráfico abaixo demonstra a fronteira de decisão desenhada com C e γ otimizados.

Mesmo com os hiperparâmetros otimizados a fronteira de decisão continuou dividida em duas partes isso demonstra que apesar desse fato essa fronteira divide melhor as duas classes.

2 Análise de Componentes Principais (PCA) — Parte II

As operações descritas nessa Seção estão no arquivo *PCA_part2.ipynb* e são relativas à segunda parte do projeto, que realiza Análise de Componentes Principais a fim de realizar redução de dimensionalidade para problemas de reconhecimento facial.

A Análise de Componentes Principais (PCA) tem por objetivo reduzir o excesso de atributos dos exemplos de treinamento a fim de diminuir os custos computacionais do treinamento a ser realizado. Para isso, é utilizado o conceito de projeção. A ideia básica é projetar o espaço de n atributos num espaço de menor dimensionalidade, com dimensão k, em que k é chamado número de componentes principais. O valor do número de componentes principais depende do problema a ser resolvido.

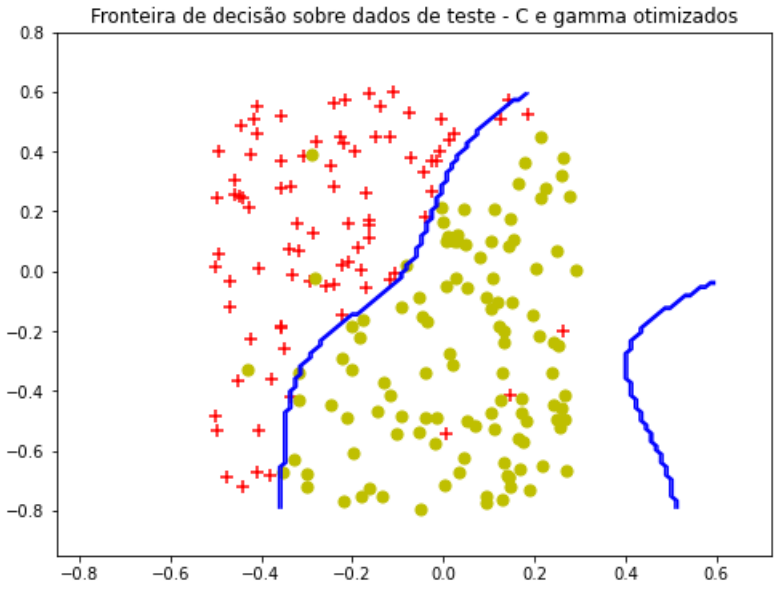

Figura 4: C= 30 e gamma = 3

No caso do reconhecimento facial, essa projeção consiste em obter k "faces padrão", chamadas *eigenfaces*, e definir cada uma das imagens originais de faces como uma combinação linear dessas *eigenfaces.*

2.1 Importação e Visualização dos Exemplos de Treinamento

Inicialmente, os dados de treinamento foram extraídos do arquivo *dado3.mat*, utilizando a função *loadmat* da biblioteca *scipy.io.* A matriz de exemplos de treinamento é chamada de X, em que cada uma das 5000 linhas representa um exemplo de treinamento e cada uma das 1024 colunas representa um atributo. Nesse, caso, o vetor de atributos representa a linearização de imagens de dimensão 32×32.

Em seguida, foram selecionadas aleatoriamente 100 imagens utilizando a função *random.choice* da biblioteca *numpy*. A fim de visulizar uma fração dos exemplos de treinamento, essas 100 imagens foram ilustradas com auxílio de funções da biblioteca *matplotlib*. O resultado está na Figura 5.

2.2 Determinação e Visualização das *eigenfaces*

O processo de "treinamento" do PCA consiste em utilizar como base do novo espaço de atributos os autovetores da matriz de covariância empírica (Σ) dos atributos que correspondam aos k maiores autovalores de Σ.

Esse processo é realizado obtendo-se, inicialmente essa matriz de covariância através da Equação 1. Em seguida, utiliza-se a função *svd* da biblioteca *numpy.linalg* para obter uma matriz (U), em que cada coluna é um dos autovetores de Σ já ordenados com base na ordem

Figura 5: Representação de 100 imagens do conjunto de exemplos de treinamento

decrescente dos autovalores correspondentes, e um vetor (S) em que cada entrada é um valor singular de Σ obedecendo a mesma ordem da matriz de autovetores (U).

$$\Sigma = \frac{1}{m} X^t X \tag{1}$$

A Figura 6 apresenta as 36 principais *eigenfaces*, isto é, a representação gráfica dos 36 autovetores de Σ cujos correspondentes autovalores são os maiores.

Figura 6: Representação visual das 36 principais *eigenfaces*.

É interessante observar que as *eigenfaces* se parecem com imagens de faces com pouca nitidez e detalhes, o que é coerente com o que se espera, pois, com o PCA, deseja-se obter as *eigenfaces* a fim de ser possível representar uma imagem de face de uma pessoa como a combinação linear de faces consideradas comuns pelo algoritmo de treinamento.

2.3 Projeção das Faces

Uma forma útil de avaliar se de fato as *eigenfaces* obtidas são boas é visualizar as faces dos exemplos de treinamento como combinação linear das *eigenfaces*.

Para obter esse resultado, utilizamos as 100 principais *eigenfaces* e projetamos os

exemplos de treinamento sobre elas. Isso pode ser feito definindo a matriz U_{red} como as 100 primeiras colunas da matriz U obtida anteriormente e definindo a projeção das imagens do conjunto de treinamento como $z = XU_{red}$. A fim de ser possível visualizar essas projeções como imagens de faces, é necessário, então, restituí-las ao espaço de atributos original, o que pode ser feito fazendo $proj = zU_{red}^{t}$.

As Figuras 7 e 8 apresentam a comparação entre as 100 faces originais e os resultados obtidos com as projeções utilizando as 100 principais *eigenfaces.* Nessas figuras, as linhas ímpares apresentam as imagens originais e, abaixo, nas linhas pares, estão os resultados das projeções, a fim de facilitar a comparação. As faces utilizadas para realizar essa representação são as mesmas que as presentes na Figura 5, que foram selecionadas aleatoriamente.

Figura 7: Comparação entre imagens originais e obtidas a partir da projeção com 100 *eigenfaces.*

É interessante notar como as imagens obtidas pela projeção são similares às originais, indicando bom desempenho do processo de redução de dimensionalidade realizado. É evidente que as imagens originais apresentam mais detalhes e são mais nítidas. Contudo, as imagens originadas por projeção são bastante claras e permitem (a um humano) o reconhecimento facial das pessoas com facilidade.

Figura 8: Comparação entre imagens originais e obtidas a partir da projeção com 100 *eigenfaces.*

Considerando que o objetivo dessa redução de dimensionalidade é justamente possibilitar o treinamento de algoritmos de reconhecimento facial com menor custo computacional, a possibilidade de reconhecer as pessoas a partir das imagens projetas em um espaço de dimensionalidade de menos de 10% das dimensões do espaço original é altamente significativa, devido à grande redução de custo computacional que isso implica.

É importante, todavia, ressaltar que seria necessário realizar testes sobre os algoritmos de reconhecimento facial para verificar qual é o número de componentes principais ideal que permita ao algoritmo reconhecer as faces com custo computacional adequado. A Seção 2.4 discute um pouco essa questão.

2.4 Fração da Variância Original Mantida

Uma forma de medir a perda de informação gerada pelo PCA é avaliar a fração da variância original mantida. Quanto maior for essa fração, menor a perda de informação. Essa fração pode ser obtida pela equação 2, em que k é o número de componentes principais utilizado e n é o número de atributos do dado original.

$$\frac{\sum_{i=1}^{k} S_i}{\sum_{i=1}^{n} S_i} \tag{2}$$

A Figura 9 apresenta um gráfico da evolução da fração da variância original mantida em função do número de componentes principais. Conforme esperado, quanto maior o número de componentes principais, maior essa fração e, portanto, menor a perda de informação. É interessante observar, contudo, como um número relativamente baixo de componentes principais já garante um valor grande dessa fração e consequente pequena perda de informação. Para k=100, por exemplo, como utilizado na Seção 2.3, há cerca de 94% da manutenção da variância original. Com 150 componentes principais, o que corresponde a pouco mais do que 10% da dimensão original dos dados, mais de 96% da variância original é mantida e, a partir de 350 *eigenfaces*, que representa cerca de um terço do número original de atributos, já há manutenção de mais do que 99% da variância original.

Portanto, há grande potencial de aplicação da técnica de PCA para facilitar o treinamento de algoritmos para realização de reconhecimento facial (e de fato essa técnica é muito utilizada na prática).

3 Referências

Vídeo-aulas e *slides* pelo Professor João Batista Florindo em ocasião de oferecimento da disciplina MS960 no segundo semestre de 2020 pelo Instituto de Matemática, Estatística e Computação Científica (IMECC) da Universidade Estadual de Campinas (UNICAMP).

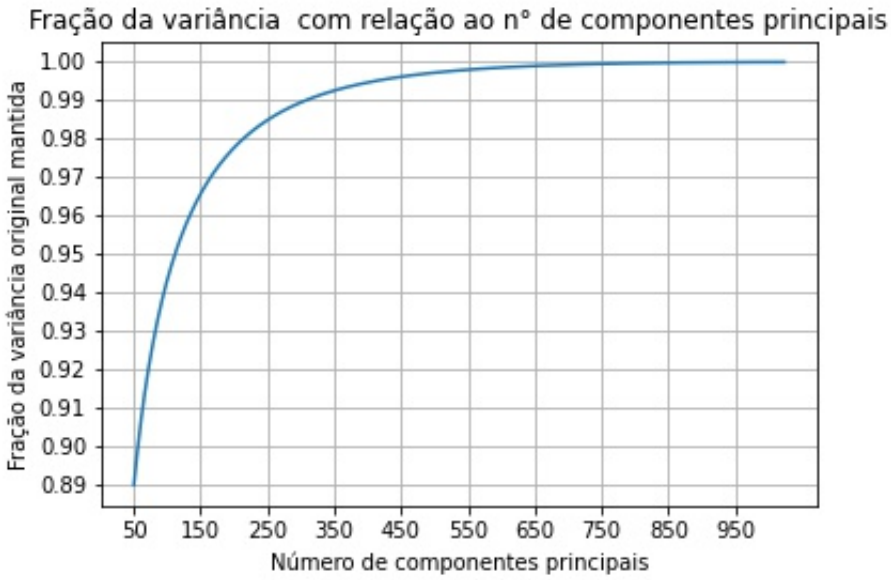

Figura 9: Evolução da fração da variância original mantida em relação ao número de componentes principais.