Relatório trabalho de grupo

Soft margins SVM com o SMO

UC: Otimização em Machine Learning
Docentes: Maria Fernanda Pires Costa e Gaspar José Brandão Queirós Azevedo Machado
Ano letivo: 2022/2023
Curso: Mestrado em Matemática e Computação

Alunos: Daniela Brasileiro (pg49172) · Luís Silva (pg49177) ·Diogo Rosário (pg49174)

Índice

Índice de figuras

Índice de tabelas

Introdução

Este trabalho de investigação foi proposto no âmbito da Unidade Curricular de Otimização em *Machine Learning* do curso Mestrado em Matemática e Computação. Este consiste na implementação de uma versão simplificada do algoritmo *Sequential Minimal Optimization Algorithm* (SMO) para treinar *Soft Margins Support Vector Machine* (C-SVM) na versão dual, sem e com *kernel*, para classificação binária.

Soft-Margin SVM

Máquinas de Vetores de Suporte SVM (*Support Vector Machine*) é um algoritmo de *Machine Learning* supervisionado de classificação. O objetivo do SVM é encontrar o hiperplano que melhor separa os dados de duas classes diferentes, de tal forma que a distância entre os elementos (vetores de atributos) mais próximas de cada classe (chamados vetores de suporte) seja maximizada.

No entanto, a premissa dos dados serem perfeitamente linearmente separáveis não é realista. Deste modo, existe o algoritmo da margem suave/flexível do SVM no qual permite a ocorrência de pontos dentro da margem.

Neste trabalho vai-se aplicar o algoritmo *Soft margins* SVM com SMO que permite fazer classificação binária em dados não linearmente separáveis, além disso é um algoritmo menos sensível a *outliers*. A chave deste algoritmo é a introdução de variáveis de folga (erro extra).

Recordar que nos casos em que os dados são linearmente separáveis, o hiperplano é calculado de modo a garantir que todos os pontos satisfazem a restrição (condição para o hiperplano ótimo):

$$y^i\left(w^T x^i + b\right) \geq 1, y^i \in \{-1,1\} \quad (1)$$

SVM pode lidar com pontos que não são linearmente separáveis, introduzindo variáveis de folga ξ_i na condição (1):

$$y^i(w^T x^i + b) \geq 1 - \xi_i,\ y^i \in \{-1,1\} \quad (2)$$

onde $\xi_i \geq 0$ é a variável de folga para ponto x^i, que indica quanto o ponto viola a condição de separabilidade. Os valores das variáveis de folga indicam três tipos de pontos: i) Se $\xi_i = 0$, então o ponto x^i está pelo menos $\frac{1}{||w||}$ distante do hiperplano; ii) Se $0 < \xi_i < 1$, então o ponto está dentro da margem e está corretamente classificado, isto é, está do lado correto do hiperplano. iii) Se $\xi_i \geq 1$, então o ponto está mal classificado e encontra-se no lado errado do hiperplano. Na Figura 1 visualiza-se uma representação gráfica do algoritmo *soft margins SVM* no qual podemos ver assinalado a vermelho um exemplo em que o $\xi_i = 0$, a amarelo um exemplo em que $0 < \xi_i < 1$ e a azul um caso em que $\xi_i \geq 1$.

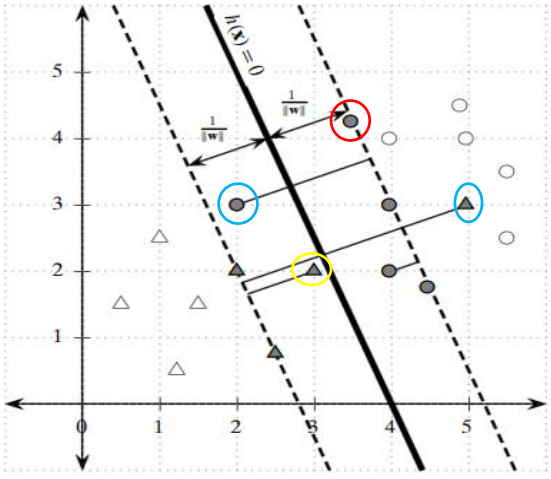

Figura 1- Soft Margin SVM

No *Soft-Margin SVM* o objetivo é encontrar o hiperplano com margem máxima que também minimiza a soma total dos erros ξi. E, desta forma a função objetivo é dada por:

$$\text{minimizar } \frac{1}{2}w^T w + C \sum_{i=1}^{N} \xi_i \quad (3)$$

$$sujeito\ a: \ y^i\left(w^T x^i + b\right) \geq 1 - \xi_i,\ \xi_i \geq 0, \text{ para } i = 1, \ldots, N \quad (4)$$

onde, o parâmetro $C > 0$ controla o *trade-off* entre a distribuição global dos pontos das duas classes e os pontos locais próximos à fronteira de cada classe.

Podemos calcular a seguinte função Lagrangiana associada ao problema (3)-(4) introduzindo multiplicadores de Lagrange α_i e μ_i que satisfazem as condições KKT:

$$L(w, w_0, \xi, \alpha, \mu) = \frac{1}{2}w^T w + C\sum_{i=1}^{N} \xi_i - \sum_{i=1}^{N} \alpha_i [y^i(w^T x^i + b) - 1 + \xi_i] - \sum_{i=1}^{N} \mu_i \xi_i \quad (5)$$

O problema dual é dado por:

$$max_\alpha \sum_{i=1}^{m} \alpha_i - \frac{1}{2}\sum_{i=1}^{m}\sum_{j=1}^{m} \alpha_i \alpha_j y^i y^j {x^i}^T x^j \quad (6)$$

$$sujeito\ a: \ 0 \leq \alpha_i \leq C, i = 1, \ldots, m \quad (7)$$

$$\sum_{i=1}^{m} \alpha_i y^i = 0 \quad (8)$$

O valor de α_i é igual a zero para os pontos que não são vetores de suporte e $\alpha_i > 0$ apenas para os vetores de suporte que correspondem a todos os pontos x^i nos quais se verifica a condição $y^i(w^T x^i + b) = 1 - \xi_i$, notar que agora os vetores de suporte incluem todos os pontos que estão na margem ($\xi_i = 0$), bem como os pontos com variável de folga positiva ($\xi_i > 0$).

O vetor de pesos é dado por: $w = \sum_{\alpha_i > 0} \alpha_i y^i x^i$

A função do hiperplano é dado por:

$$f(x) = w^T x + b \quad (9)$$

onde, w é um vetor perpendicular ao hiperplano, b é um escalar, chamado viés. Assim sendo, obtemos:

$$f(x) = \sum_{i=1}^{m} \alpha_i y^i {x^i}^T x + b \quad (10)$$

No qual, podemos substituir o produto interno ${x^i}^T x$ pela função kernel $K(x^i, x)$.

Pelas condições KKT do problema primal tem-se que, $(C - \alpha_i)\xi_i = 0$ (11).

Assim para os vetores de suporte (com $\alpha_i > 0$) temos dois casos a considerar i) $\xi_i > 0$, o que implica que $C - \alpha_i = 0$, isto é, $\alpha_i = C$, ou ii) $C - \alpha_i > 0$, isto é, $\alpha_i < C$. Neste caso, por (11) temos $\xi_i = 0$. Ou seja, estes são precisamente os vetores que estão na margem.

Usando os vetores de suporte que estão na margem, isto é, $0 < \alpha_i < C$ e $\xi_i = 0$ podemos calcular o viés por: $\alpha_i(y^i(w^T x^i + b_i) - 1) = 0; y^i(w^T x^i + b_i) = 1;\ b_i = \frac{1}{y^i} - w^T x^i = y^i - w^T x^i$. Para obter o viés final podemos calcular a média de todos os b_i.

Depois de calculado o hiperplano ótimo, para ~~calcular~~ classificar os novos pontos z usamos a função sinal: $\hat{y} = sign(f(z)) = sign(w^T z + b)$.

Este é um problema de Programação Quadrática (PQ), e deste modo pode ser resolvido por um dos métodos de otimização especificamente desenvolvidos no contexto

do SVM como é o caso do *Sequential Minimal Optimization (SMO) algorithm* (este algoritmo será explicado posteriormente).
As condições KKT podem ser usadas para verificar a convergência para um ponto ótimo. Para este problema as condições KKT são:

- $\alpha_i = 0 \Rightarrow y^i(w^T x^i + b) \geq 1$ (12)
- $\alpha_i = C \Rightarrow y^i(w^T x^i + b) \leq 1$ (13)
- $0 < \alpha_i < C \Rightarrow y^i(w^T x^i + b) = 1$ (14)

Ou seja, quaisquer $\alpha_i's$ que satisfazem estas condições para todo o i, será uma solução ótima para o problema de otimização referido anteriormente.

O algoritmo SMO itera até que todas essas condições sejam satisfeitas (dentro de uma certa tolerância), garantindo assim a convergência.

Sequential Minimal Optimization (SMO)

O algoritmo SMO é um método iterativo que divide o problema de otimização em subproblemas menores que são resolvidos analiticamente. O algoritmo começa com uma solução inicial e, em cada iteração, seleciona dois vetores de suporte e otimiza os pesos do SVM para que esses vetores de suporte fiquem na fronteira de decisão.

SMO divide o grande problema de PQ em pequenos subproblemas (numa série subproblemas de PQ menores possíveis, incluindo apenas dois α's de cada vez), que podem ser resolvidos analiticamente. Esta abordagem é até uma ordem de grandeza computacionalmente mais rápida.

Grande parte deste algoritmo é dedicado a heurísticas para escolher quais α_i e α_j utilizar de forma a maximizar a função objetivo tanto quanto possível.

Na versão simplificada do SMO aqui apresentada itera-se simplesmente sobre todo $\alpha_i, i = 1, \ldots, m.$ Se α_i não cumprir as condições KKT dentro de alguns limites numéricos de tolerância seleciona-se α_j dos restantes m-1 α's e tenta-se otimizar em conjunto α_i e α_j. Se nenhum dos α's for alterado após algumas iterações sobre todos os $\alpha_i's$, então o algoritmo termina.

Tendo escolhido os multiplicadores de Lagrange α_i e α_j para otimizar, primeiro calculam-se as restrições para esses valores e depois resolve-se o problema de maximização.

Primeiro, pretende encontrar os limites L e H tais que $L \leq \alpha_j \leq H$ deve ser garantido para que α_j satisfaça a restrição $0 \leq \alpha_j \leq C$. Isto é dado por:

- $Se\ y^i \neq y^j,\ L = max(0, \alpha_j - \alpha_i),\ H = min(C, C + \alpha_j - \alpha_i)$ (15)
- $Se\ y^i = y^j,\ L = max(0, \alpha_i + \alpha_j - C),\ H = min(C, \alpha_i + \alpha_j)$ (16)

De seguida, pretende-se encontrar α_j de maneira a maximizar a função objetivo. Se esse valor se encontrar fora dos limites L e H, simplesmente diminui-se o valor de α_j para ficar dentro desse intervalo.
O α_j ótimo é dado por:

- $\alpha_j := \alpha_j - \frac{y^i(E_i - E_j)}{\eta}$ (17)

onde

- $E_k = f(x^k) - y^k$ (18)
- $\eta = 2x^{i^T}x^j - x^{i^T}x^i - x^{j^T}x^j$ (19)

Pode-se pensar em E_k como o erro entre a saída SVM no k-ésimo exemplo e o verdadeiro rótulo/legenda/classificação de y^k. Isto pode ser calculado usando a equação (2). Ao calcular o parâmetro η pode-se utilizar uma função *kernel* K em vez do produto interno. Em seguida, diminui-se α_j de modo a se encontrar no intervalo [L, H].

- $$\alpha_j := \begin{cases} H \ \ se\ \alpha_j > H \\ \alpha_j \ \ se\ L \leq \alpha_j \leq H \\ L \ \ se\ \alpha_j < L \end{cases} \quad (20)$$

Finalmente, tendo calculado α_j, o valor de α_i que é dado por:

- $$\alpha_i := \alpha_i + y^i y^j (\alpha_j^{ant} - \alpha_j) \quad (21)$$

Depois de otimizar α_i e α_j, seleciona-se o valor limiar b de modo que as condições KKT sejam satisfeitas para os pontos x^i e x^j. Se, após a otimização, α_i não está nos limites (ou seja, $0 < \alpha_i < C$), então o seguinte valor limiar b_1 é válido, pois força o SVM a produzir y^i quando o input é x^i

$$b_1 = b - E_i - y^i(\alpha_i - \alpha_i^{ant})(x^{i^T} x^i) - y^j(\alpha_j - \alpha_j^{ant})(x^{i^T} x^j) \quad (22)$$

Analogamente, o seguinte valor limiar b_2 é valido se $0 < \alpha_j < C$

$$b_2 = b - E_j - y^i(\alpha_i - \alpha_i^{ant})(x^{i^T} x^j) - y^j(\alpha_j - \alpha_j^{ant})(x^{j^T} x^j) \quad (23)$$

Se $0 < \alpha_i < C$ e $0 < \alpha_j < C$, ambos estes valores limiares são válidos e serão iguais.

Se ambos os novos α's estão nos limites (ou seja, $\alpha_i = 0$ ou $\alpha_i = C$ e $\alpha_j = 0$ ou $\alpha_j = C$), então todos os valores limiares entre b_1 e b_2 satisfazem as condições KKT, assim considera-se b:= (b_1 +b_2)/2. Deste modo, a equação completa para b:

- $$b := \begin{cases} b_1 \ \ se\ 0 < \alpha_i < C \\ b_2 \ \ se\ 0 < \alpha_j < C \\ (b_1 + b_2)/2 \quad caso\ contrário \end{cases} \quad (24)$$

Pseudo-código

A seguir, apresenta-se o algoritmo do SMO simplificado que foi implementado neste trabalho usando a linguagem de programação em *matlab* para treinar o modelo.

Algoritmo: SMO Simplificado

Input: C, *tol*: tolerância numérica, *max_passes*: número máximo de vezes para iterar sobre α's sem alterar $(x^1, y^1), \ldots, (x^m, y^m)$: dados de treino

Output: $\alpha \in \mathbb{R}^m$, $b \in \mathbb{R}$

◦ Inicializar: $\alpha_i = 0$, ∀i, b = 0, *passes*= 0.

◦ **Enquanto** (*passes < max_passes && it<maxit)*

- ◦ it = it +1
- ◦ changed_alphas=0
- ◦ **para** i = 1, . . . N,
  - ◦ Calcular $E_i = f(x^i) - y^i$
  - ◦ **se** $((y^i E_i < -tol \ \&\& \ \alpha_i < C) \ || \ (y^i E_i > tol \ \&\& \ \alpha_i > 0))$
    - ◦ **para** j =[1:i-1,i+1:m]
      - ◦ Calcular $E_j = f(x^j) - y^j$, utilizar (10)
      - ◦ Guardar: $\alpha_i^{ant} = \alpha_i, \ \alpha_j^{ant} = \alpha_j$.
      - ◦ Calcular L e H usando (15) e (16).

◦ **se** (L == H)
continuar para o próximo valor de i.
◦ Calcular η usando (19).
◦ **se** (η >= 0)
continuar para o próximo valor de i.
◦ Calcula o novo valor para α_j usando (17) e (20).
◦ **se** ($|\alpha_j - \alpha_j^{ant}| < tol$)
continuar para o próximo valor de i.
◦ Calcula novo o valor de α_i usando (21).
◦ Calcular b_1 *e* b_2 usando (22) e (23) respectivamente.
◦ Calcular b usando (24).
◦*num_changed_alphas*:= *num_changed_alphas* + 1.
◦ **fim para**
◦ **fim se**
◦ **fim para**
◦ **se** (*num_changed_alphas* == 0)
passes := passes + 1
◦ **caso contrário**
passes := 0
◦ **fim enquanto**

Implementação Prática do algoritmo SVM com SMO

Para a implementação do algoritmo SVM com SMO utilizou-se o *dataset* Iris.

O *dataset* Iris contém as características da flor Iris, sendo que o objetivo é classificar (definir a classe / rótulo) o tipo dessa flor, entre três tipos possíveis: Versicolor, Setosa e Virginica.

Numa primeira abordagem a classificação será feita entre a espécie Iris-Setosa *versus* outras espécies. Numa segunda abordagem será realizada a classificação entre Iris-versicolor *versus* outras espécies.

Iris-Setosa versus Outras Espécies

Primeiramente, realizou-se o tratamento dos dados com o intuito de classificar como 1 os dados correspondentes à espécie Setosa e como -1 os dados referentes às outras espécies. Além disso, criou-se um *dataset* contendo esta informação bem como a informação dos atributos sepal width e sepal length através do código presente na Figura 2.

```
iris = sns.load_dataset('iris')
#Fazer label enconding aos dados
X = iris.drop(['species'], axis=1).values
y = iris['species'].values
def label_encoding(y):
    y_encoded = np.zeros((y.shape[0]))
    for i in range(y.shape[0]):
        if y[i] == "setosa":
            y_encoded[i] = 1
        else:
            y_encoded[i] = -1
    return y_encoded
y = label_encoding(y)
X= X[:,0:2]
dataset = np.concatenate((X, y.reshape(-1,1)), axis=1)
df = pd.DataFrame(dataset)
df[2] = df[2].round(0).astype(int)
df.to_csv('dataset1.csv', index=False, header=False)
```

Figura 2- Criação do dataset

De seguida, importou-se o *dataset* criado (dataset1) para o *matlab* e representou-se o gráfico contendo as flores da espécie Setosa versus outras espécies para os atributos sepal_width e sepal_length. Pelo gráfico presente na Figura 3 pode-se deduzir que para esta classificação (Setosa versus outras espécies) os dados são linearmente separáveis.

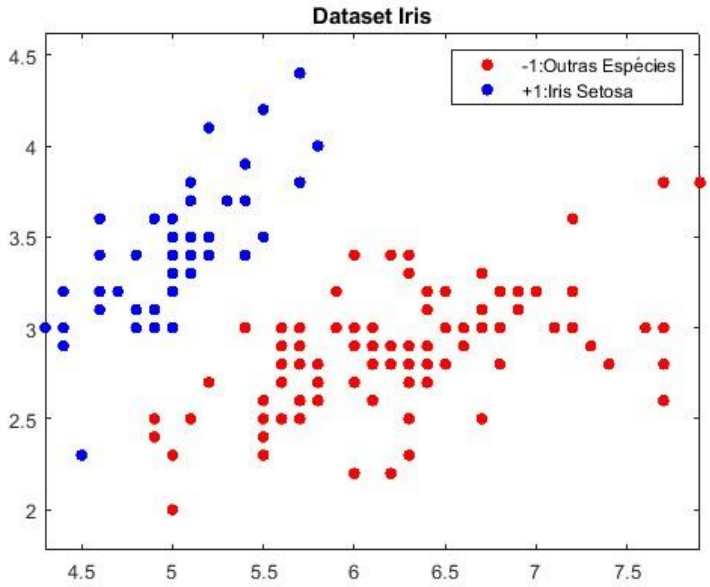

Figura 3- Dados de treino

Depois disto, dividiu-se os dados em 80% para dados de treino e 20% para validação. É importante referir que esta separação é feita de forma aleatória o que poderá levar a resultados ligeiramente diferentes, visto que o modelo é treinado com dados diferentes.

Uma vez que os dados são linearmente separáveis decidiu-se aplicar o *Kernel* linear e definir os parâmetros da seguinte forma: σ=1, tolerância =1e-4, máximo de iterações= 100, máximo de passagens = 1, C=10 e C= 1000. Fez-se o treino do algoritmo SMO para dois valores diferentes de C com o intuito de comparar os resultados obtidos.

Resultados obtido para C=10:

SVM - Vetores de suporte:

Tabela 1- Vetores de suporte com C=10

n:	alpha_n	Xsv	Y_n
5	10.0000	x=[5.4000, 3.0000]	-1
10	0.4968	x=[5.5000, 3.5000]	1
13	2.2022	x=[5.7000, 3.8000]	1
15	2.5328	x=[4.9000, 2.5000]	-1
24	4.8905	x=[5.0000, 3.0000]	1
26	5.0886	x=[5.6000, 2.9000]	-1
32	10.0000	x=[6.0000, 3.4000]	-1
44	0.3059	x=[5.9000, 3.2000]	-1
51	6.2671	x=[4.5000, 2.3000]	1

54	7.6230	x=[5.8000, 4.0000]	1
60	10.0000	x=[4.9000, 3.1000]	1
63	6.1098	x=[5.1000, 3.4000]	1
75	1.7645	x=[5.4000, 3.7000]	1
92	10.0000	x=[5.0000, 2.3000]	-1
97	8.5733	x=[5.0000, 3.3000]	1
112	10.0000	x=[5.2000, 2.7000]	-1

Erro de validação: 0.0000e+00

Representação gráfica:

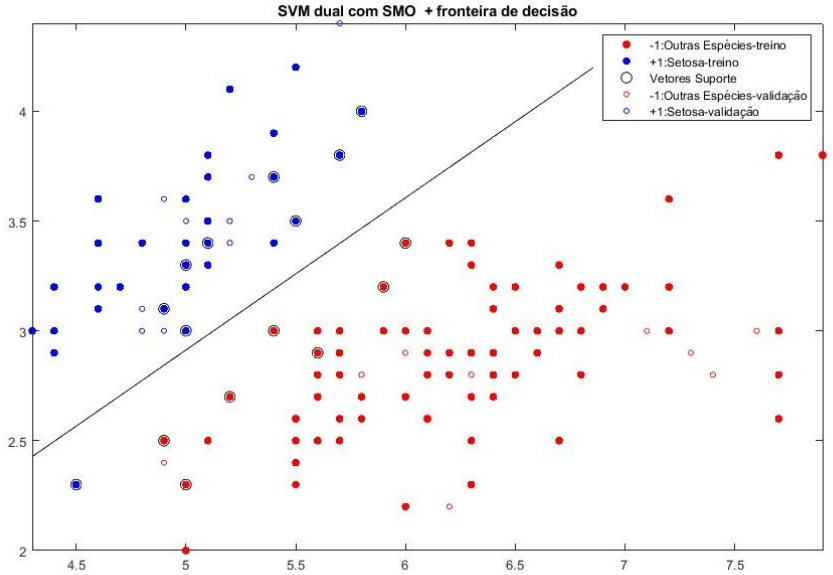

Figura 4- Representação dos resultados

Resultados obtidos para C = 1000:

SVM - Vetores de suporte:

Tabela 2- Vetores de Suporte com C=1000

n:	alpha_n	Xsv	Y_n
30	0.5459	x=[5.4000, 3.4000]	1
34	6.3143	x=[5.6000, 2.9000]	-1
41	4.5103	x=[5.0000, 3.0000]	1
49	1.2582	x=[4.9000, 3.0000]	1
54	10.0000	x=[4.9000, 2.5000]	-1
73	2.9586	x=[6.3000, 3.4000]	-1
74	0.3403	x=[5.0000, 3.3000]	1
78	2.6183	x=[5.0000, 3.2000]	1
81	10.0000	x=[5.4000, 3.4000]	1
86	7.6412	x=[5.0000, 3.4000]	1
90	4.8095	x=[5.7000, 3.0000]	-1

94	2.8317	x=[6.0000, 3.4000]	-1
118	0.3883	x=[5.0000, 3.5000]	1
119	6.4751	x=[4.9000, 3.1000]	1
120	6.8634	x=[5.9000, 3.0000]	-1

Erro de validação: 0.0000e+00

Representação gráfica dos resultados:

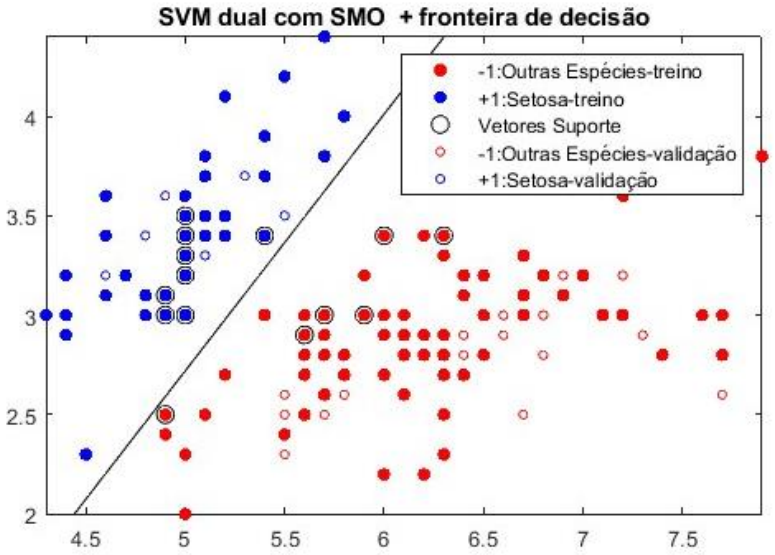

Figura 5- Representação dos resultados

Conclusões:

Apesar de em ambos os casos (C=10 e C=1000) o erro de validação ser igual a zero, por norma um valor de C pequeno desvaloriza o termo do erro, permitindo erros maiores e que mais pontos se tornem vetores de suporte com base nos quais o hiperplano é criado. Desta a fronteira de decisão reflete melhor a distribuição global dos pontos das duas classes.

Iris-versicolor versus outras espécies

Primeiramente, criou-se um *dataset* contendo a classificação de 1 para as plantas da espécie versicolor e -1 para as restantes espécies. Seguidamente, aplicou-se a PCA (Análise de Componentes Principais) ao *dataset* com o objetivo de reduzir este a duas componentes principais. Na Figura 6 observa-se a criação do referido *dataset* em python no qual se recorreu à função PCA já existente em *python* para reduzir a dimensão.

```
iris = datasets.load_iris()
X = iris.data
pca = PCA(n_components=2)
X_r = pca.fit(X).transform(X)
iris = sns.load_dataset('iris')
y = iris['species'].values
def label_encoding(y):
    y_encoded = np.zeros((y.shape[0]))
    for i in range(y.shape[0]):
        if y[i] == "versicolor":
            y_encoded[i] = 1
        else:
            y_encoded[i] = -1
    return y_encoded
y = label_encoding(y)
dataset = np.concatenate((X_r, y.reshape(-1,1)), axis=1)
df = pd.DataFrame(dataset)
df[2] = df[2].round(0).astype(int)
df.to_csv('dataset2.csv', index=False, header=False)
```

Figura 6- Aplicação de PCA e criação de datase

Na figura 7 visualiza-se os dados de treino do *dataset* referido. No qual se observa que os dados não são linearmente separáveis.

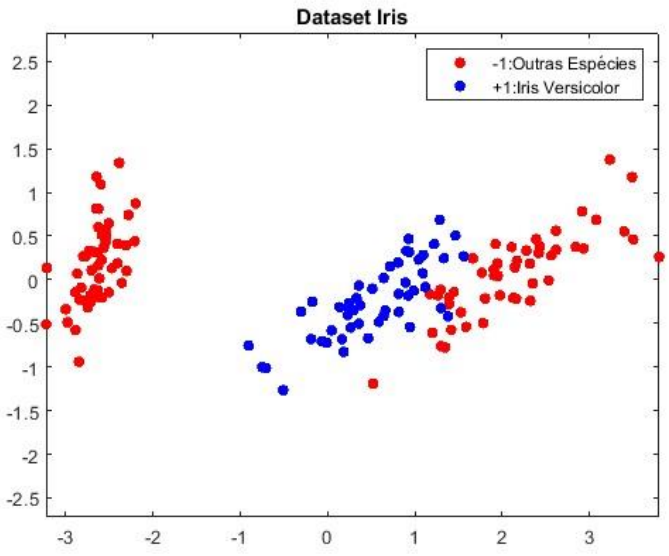

Figura 7- Dados de treino

Implementou-se o algoritmo para $\sigma^2 = 0.05$, com kernel gaussiano, C= 10, maxit=100 e tolerância $10e^{-4}$.

Resultados:

SVM - Vetores de suporte:

Tabela 3- Vetores de Suporte (kernel Gaussiano)

n:	alpha_n	Xsv	Y_n
1	1.9825	x=[1.7643, 0.0789]	-1
2	3.2007	x=[-2.6489, 0.8134]	-1
5	1.2540	x=[2.9326, 0.3555]	-1
6	1.6178	x=[-2.8890, -0.1449]	-1

7	3.0021	x=[-2.9974, -0.3419]	-1
8	1.1604	x=[-2.6841, 0.3194]	-1
9	3.2736	x=[-2.3558, -0.0373]	-1
10	3.4464	x=[2.1076, 0.3723]	-1
11	1.8735	x=[2.3212, -0.2438]	-1
12	0.9265	x=[-2.7453, -0.3183]	-1
13	1.8863	x=[-2.1998, 0.8728]	-1
14	0.5334	x=[3.4999, 0.4607]	-1
16	0.1452	x=[-2.2809, 0.7413]	-1
17	0.5102	x=[3.7956, 0.2573]	-1
20	0.9065	x=[0.9002, 0.3285]	1
22	1.3588	x=[-2.5880, 0.5136]	-1
23	4.1943	x=[-2.6728, -0.1138]	-1
24	3.9296	x=[2.2754, 0.3350]	-1
25	6.7444	x=[0.9447, -0.5431]	1
26	9.8747	x=[-0.9065, -0.7561]	1
28	8.1811	x=[-0.0087, -0.7231]	1
30	10.0000	x=[1.0951, 0.2835]	1
31	10.0000	x=[0.2990, -0.3489]	1
32	10.0000	x=[-0.1739, -0.2549]	1
33	10.0000	x=[1.2982, -0.3278]	1
36	10.0000	x=[1.5578, 0.2675]	1
38	5.6856	x=[-0.0681, -0.7052]	1
40	7.4953	x=[-0.7045, -1.0122]	1
41	8.1963	x=[1.3800, -0.4210]	1
46	4.7644	x=[-0.1896, -0.6803]	1
48	5.9507	x=[1.2848, 0.6852]	1
49	3.2941	x=[0.6426, 0.0177]	1
53	1.3836	x=[0.8151, -0.3720]	1
55	3.0259	x=[0.9279, 0.4672]	1
57	3.7014	x=[0.3579, -0.0689]	1
58	2.3427	x=[0.8133, -0.1634]	1
68	0.6888	x=[0.3762, -0.2932]	1
69	4.6040	x=[0.9217, -0.1827]	1
75	4.2331	x=[0.6603, -0.3530]	1
77	3.8678	x=[0.7149, 0.1491]	1
82	3.1089	x=[-2.4688, 0.1310]	-1
83	5.4566	x=[1.2911, -0.1167]	-1
84	9.2614	x=[1.6618, 0.2422]	-1
85	2.0639	x=[1.9497, 0.0419]	-1
87	2.3124	x=[-2.5381, 0.5038]	-1
93	4.3087	x=[2.6167, 0.3439]	-1

94	2.1535	x=[1.9009, 0.1166]	-1
95	4.4506	x=[1.3008, -0.7611]	-1
97	3.7790	x=[-2.6320, -0.1970]	-1
98	5.4328	x=[2.1594, -0.2173]	-1
99	6.5772	x=[0.5212, -1.1928]	-1
101	4.2322	x=[-3.2238,-0.5114]	-1
102	4.6494	x=[1.8034, -0.2156]	-1
103	2.9021	x=[-2.6395, 0.3120]	-1
104	4.8527	x=[-2.5069, 0.6451]	-1
107	5.5886	x=[1.4152, -0.5749]	-1
109	0.3288	x=[1.9051, 0.0493]	-1
110	7.0007	x=[1.5272, -0.3753]	-1
111	7.0371	x=[3.0765, 0.6881]	-1
112	1.7014	x=[-2.7141, -0.1770]	-1
113	8.6878	x=[1.1693, -0.1650]	-1
114	1.3122	x=[2.3500, -0.0403]	-1
116	6.2085	x=[3.3970, 0.5508]	-1
117	6.2085	x=[0.9325, 0.3183]	1
118	4.6808	x=[1.5859, -0.5396]	-1
120	1.2332	x=[1.2207, 0.4076]	1

Erro de validação: 0.0000e+00

Representação gráfica dos resultados:

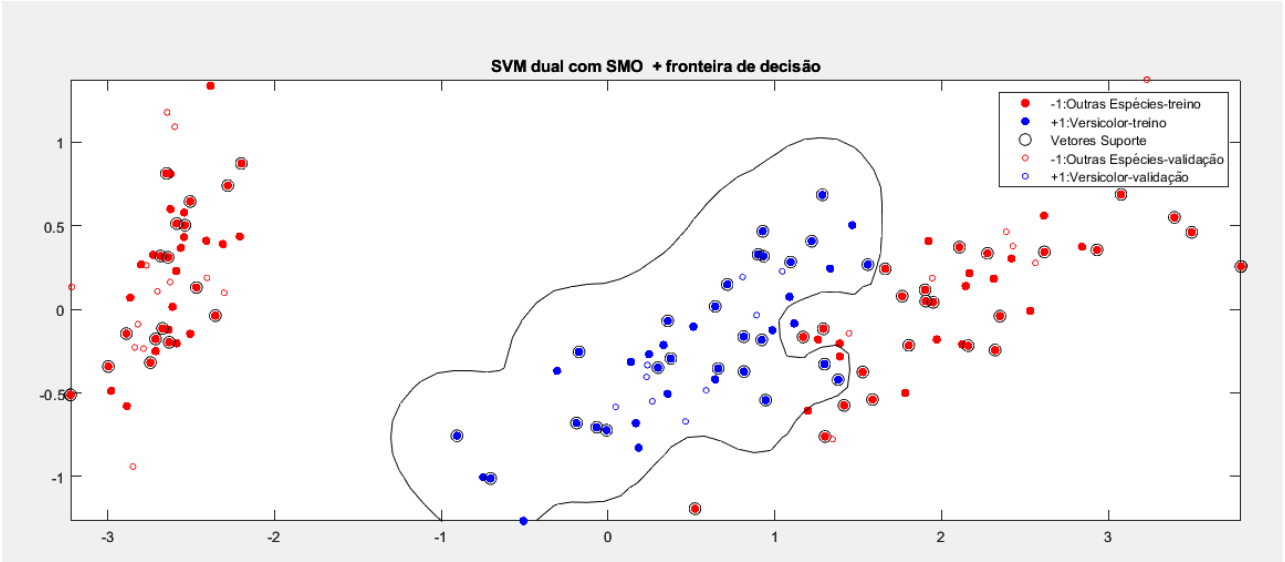

Figura 8- Representação dos resultados (kernel Gaussiano)

Aplicou-se novamente algoritmo com os mesmo parâmetros à exceção do kernel que se alterou para polinomial de grau 2 obteve-se o seguinte gráfico:

Erro de validação: 4.0000e+00

Representação gráfica dos resultados:

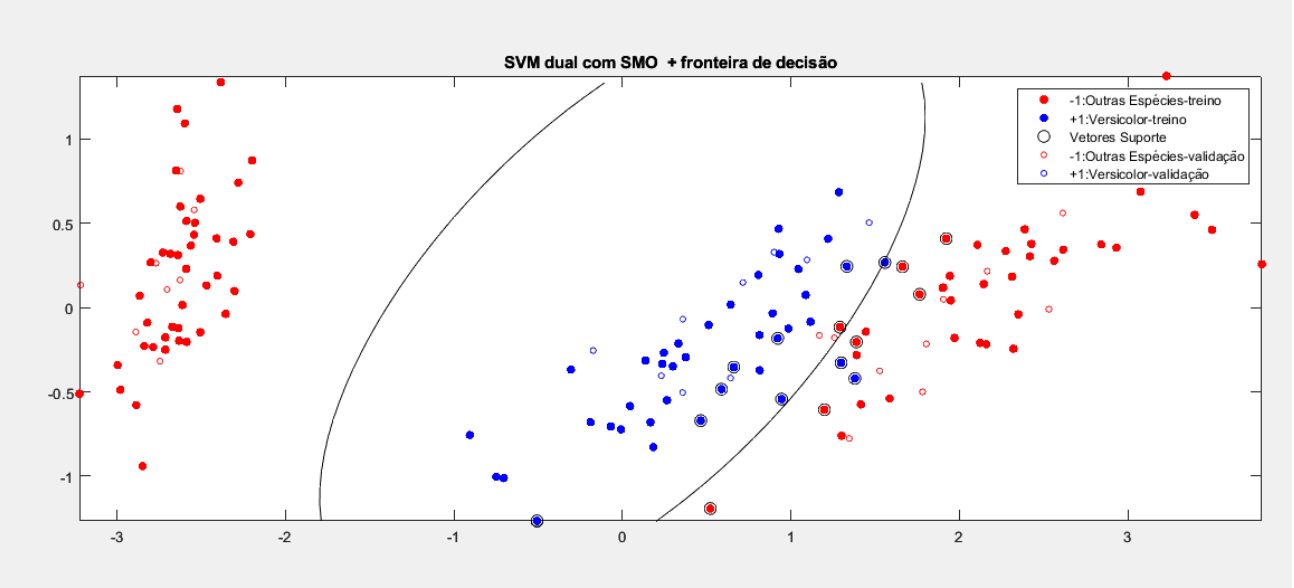

Figura 9- Representação dos resultados (com kernel polinomial)

Conclusão:

Podemos concluir que para estes dados o algoritmo permite encontrar um classificador melhor quando é aplicado o kernel gaussiano, isto é possível se concluir pela comparação do erro para os dados de teste, no caso gaussiano o erro é zero e no caso polinomial o erro é de 4. Além disso, também se pode verificar através da visualização do gráfico uma vez que com o kernel gaussiano se verifica uma melhor separabilidade das duas classes.

Webgrafia

- Data Mining and Machine Learning: Fundamental Concepts and Algorithms (2020), Mohammed J. Zaki,Wagner Meira Jr.
- CS 229, Autumn 2009, The Simplified SMO Algorithm. Disponível em: http://cs229.stanford.edu/materials/smo.pdf
- Material disponibilizado pelos docentes.