Data_Analysis_6

Erhon L. Aragão - 00288734
Henrique A. Boneto - 00288744

Teoria

Suport Vector Machine (SVM)

Support Vector Machine é utilizado para encontrar um hiperplano, tal que este maximize a margem entre as classes. SVMs são mais comumente usados para problemas de classificação. Eles também podem ser usados para regressão, detecção de outliers e agrupamento. O SVM funciona muito bem para pequenos conjuntos de dados.
Os vetores de suporte são os pontos de dados mais próximos do hiperplano. Esses pontos definirão melhor a linha de separação calculando as margens. Esses pontos são mais relevantes para a construção do classificador.
De modo a efetuar a utilização do algorítmo, se parte de informações obtidas anteriormente (exemplo: dados que sabemos se são e-mails ou spam) e nomeamos com labels diferentes, como $L = 1$ e $L = -1$, e a partir disso é possível encontrar os valores dos pesos

$$f_{(x)} = SGN((\vec{x} - \vec{p})\overrightarrow{w})$$

Temos conhecimento que $\vec{x_i}$ é o Vetor de Suporte, $\vec{w}$ é o peso, vvp é um ponto que temos, e y_i equivale a suas *labels*.

$$\begin{cases} (\vec{x_i} - \vec{p})\vec{w}) \geq 1 \; se \; y_i = 1 \\ (\vec{x_i} - \vec{p})\vec{w}) \leq 1 \; se \; y_i = -1 \end{cases}$$

Margem

Uma margem é uma lacuna entre as duas linhas nos pontos de classe mais próximos. Isso é calculado como a distância perpendicular da linha aos vetores de suporte ou pontos mais próximos. Se a margem for maior entre as classes, então é considerada uma boa margem, uma margem menor é uma margem ruim. De modo a se obter a melhor margem possível, podemos tanto maximizar a margem quanto minimizar, onde $i = 1, 2, ..., N$.

$$M = \frac{2}{||\vec{w}||}, \; se \; y_i(\vec{x_i} - \vec{p})\vec{w} \geq 1 \qquad \text{(Maximiza a Margem)}$$

$$M = \frac{||\mathbf{w}||^2}{2}, \; se \; y_i(\vec{x_i} - \vec{p})\vec{w} \geq 1, \qquad \text{(Minimiza a Margem)}$$

Multiplicadores de Lagrange

De modo a minimizar a função, utilizaremos a função de Lagrange, que pode ser escrita como:

$$L = \frac{1}{2}||\vec{w}|| - \sum_{i=1}^{N} \alpha_i [y_i(\vec{x_i} - \vec{p})\vec{w} - 1]$$

Onde α_i são os multiplicadores de Lagrange. De acordo com as seguintes condições, sabemos que:

$$\frac{\partial L}{\partial p} = 0 \Rightarrow \sum_{i=1}^{N} \alpha_i y_i = 0$$

$$\frac{\partial L}{\partial \vec{w}} = 0 \Rightarrow \vec{w} = \sum_{i=1}^{N} \alpha_i y_i \vec{x_i}$$

Então reescrevendo a função inicial, temos a função Dual de Lagrange como:

$$L = \sum_{i=1}^{N} \alpha_i - \frac{1}{2} \sum_{ij}^{N} \alpha_i \alpha_j y_i y_j \vec{x_i} \vec{x_j}$$

Onde apenas o α_i é desconhecido. Podemos descobrir seu valor através da Descida de Coordenada.

Descida de Coordenada

Começamos com valores aleatórios de α_i. A cada passo dado, subistituímos o valor de α_i de acordo com a equação:

$$\alpha_i \mapsto \alpha_i - \eta \frac{\partial L}{\partial \alpha_i}$$

Onde η é sua taxa de aprendizagem. A equação acima está sob a restrição de:

$$\sum_{i=1}^{N} \alpha_i y_i = 0$$

Sub Gradiente Descendente

O subgradiente descendente é um algoritmo de otimização iterativa de primeira ordem para resolver problemas de minimização convexa. Para encontrar um mínimo local de uma função usando a descida de gradiente, tomamos etapas proporcionais ao negativo do gradiente (ou gradiente aproximado) da função no ponto atual.
Métodos tradicionais de gradiente descendente (ou SGD) podem ser adaptados, onde em vez de dar um passo na direção do gradiente da função, um passo

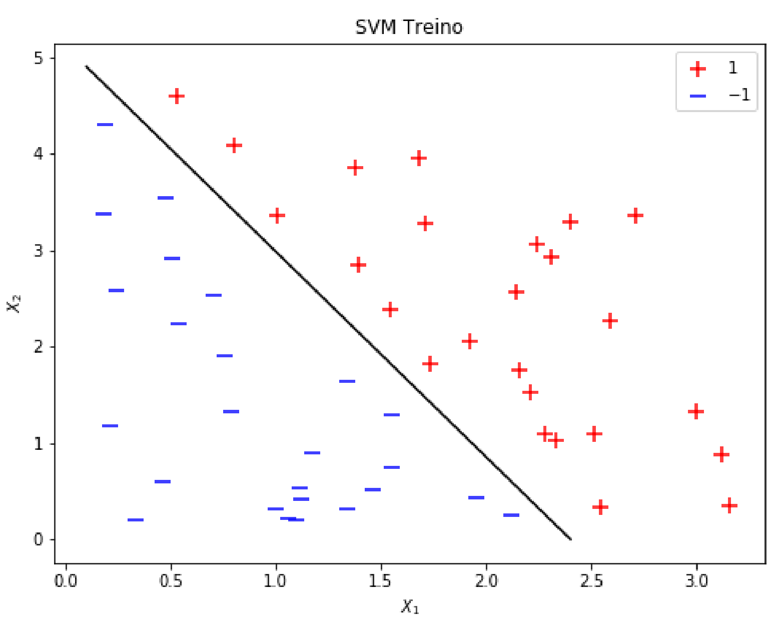
é dado na direção de um vetor selecionado do sub-gradiente da função. Esta abordagem tem a vantagem de que, para certas implementações, o número de iterações não é escalonado com N (número de pontos de dados). O método utilizado se baseia na equação:

$$f_{(\vec{w},p)} = \frac{1}{N}\sum_{i=1}^{N}[max(0, 1 - y_i(\vec{w}\vec{x_i} - p))] + \lambda||\vec{w}||^2$$

Prática

Treino

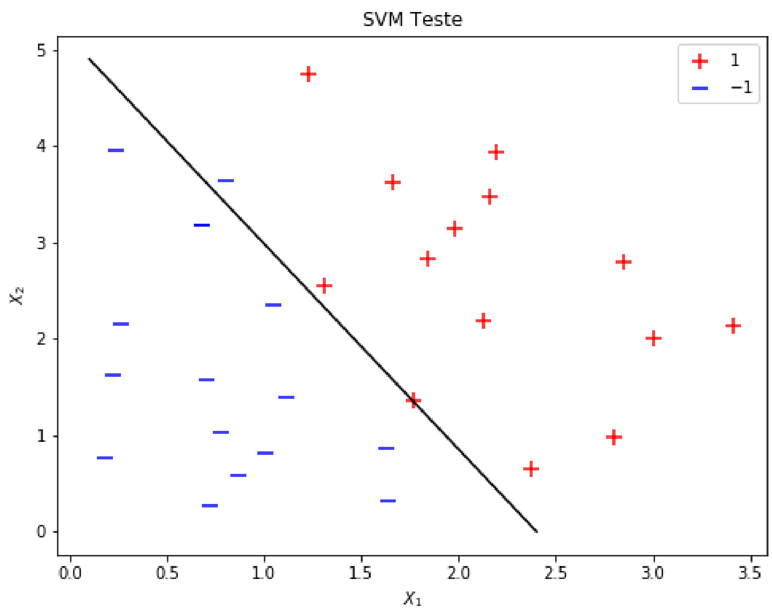
A prática desta atividade consiste na análise de dados com SVM para prever, através de um treino, prever a classificação de dados posteriores. Os dados utilizados para o treino foram pontos colocados aleatoriamente em um gráfico 2D (expressos no programa como "x_train"). Ao todo foram utilizados 50 pontos para o teste, 25 correspondendo a label "-1" e 25 correspondendo a "1". Podemos ver estes através do gráfico:



De modo a efetuar o treino, foi implementado no programa a classe "LinearSVM", cujos resultados são demonstrados na seguinte seção.

Teste

Após ser efetuado o treino algumas vezes para o incremento da precisão do algorítmo, foi obtido o resultado na análise de outro conjunto de dados denominado "*x_test*" que equivalem a 30 valores, e então, é analisado o gráfico:



A análise da acurácia do código foi efetuada na última linha do programa, "*accuracy_score*", e observamos que este possui uma acurácia extremamente apurada de $96,667\%$

Programa

```python
# Importando Bibliotecas
import numpy as np
import matplotlib.pyplot as plt
from sklearn.utils import shuffle
from tqdm import tqdm
import random
import pandas as pd
from sklearn.metrics import accuracy_score, recall_score, precision_score



# Definição do SVM SGD
```

```python
class LinearSVM:

    def __init__(self, x, y):
        self.x = x
        self.y = y
        self.n = x.shape[1]

    def _stochastic_subgradient_descent(self):
        epochs = 10000
        learning_rate = 1e-8
        w = np.zeros(self.n)

        for epoch in tqdm(range(1, epochs+1)):

            X = shuffle(self.x) #arrumar o shuffle
            Y = shuffle(self.y)

            for i, xi in enumerate(X):
                yi = Y[i]
                if(yi*(np.dot(w.T, xi))<=1):
                    w = (1 - learning_rate)*w + learning_rate*yi*xi
                else:
                    w = (1 - learning_rate)*w

        return w

    def fit(self):
        w = self._stochastic_subgradient_descent()
        norm = np.linalg.norm(w)
        b = self.y - np.dot(self.x, w)
        #normalize
        b = b.sum() / b.size
        self.w, self.b = w / norm, b / norm


    def predict(self, x):
        #w, b = self._fit()
        y = np.sign(np.dot(self.w, x.T) + self.b * np.ones(x.shape[0]))
        return y



# Dados para Treino
x_train =  np.array ([ \
        [1.00, 0.31], [1.10, 0.20], [1.12, 0.42], [1.55, 0.75], [1.11, 0.53], #-1
        [1.34, 0.32], [0.19, 4.30], [0.51, 2.91], [0.24, 2.58], [1.17, 0.89], #-1
```

```
        [1.95, 0.44], [2.12, 0.25], [1.06, 0.21], [0.33, 0.20], [0.46, 0.59], #-1
        [0.79, 1.33], [0.76, 1.91], [1.34, 1.64], [0.21, 1.18], [0.70, 2.53], #-1
        [0.54, 2.23], [1.55, 1.29], [1.46, 0.51], [0.18, 3.37], [0.48, 3.54], #-1
        [2.51, 1.10], [2.33, 1.03], [2.21, 1.52], [1.92, 2.05], [2.24, 3.06], # 1
        [2.71, 3.35], [0.53, 4.60], [0.80, 4.08], [1.01, 3.35], [1.68, 3.95], # 1
        [1.38, 3.86], [2.14, 2.56], [2.16, 1.76], [1.39, 2.84], [1.71, 3.27], # 1
        [1.73, 1.82], [2.28, 1.09], [3.00, 1.33], [2.54, 0.34], [3.16, 0.35], # 1
        [3.12, 0.88], [2.59, 2.27], [2.40, 3.29], [2.31, 2.92], [1.54, 2.38]  # 1
    ])

y_train = np.array([ \
        [-1], [-1], [-1], [-1], [-1],
        [-1], [-1], [-1], [-1], [-1],
        [-1], [-1], [-1], [-1], [-1],
        [-1], [-1], [-1], [-1], [-1],
        [-1], [-1], [-1], [-1], [-1],
        [1], [1], [1], [1], [1],
        [1], [1], [1], [1], [1],
        [1], [1], [1], [1], [1],
        [1], [1], [1], [1], [1],
        [1], [1], [1], [1], [1]
    ])



# Definição dos dados de treino: cada ponto "-" corresponde ao label -1 e
# cada ponto "+" corresponde ao label 1
x1_train_pos = []
x2_train_pos = []
x1_train_neg = []
x2_train_neg = []
X1trpos, X2trpos,X1trneg, X2trneg=[],[],[],[]

for i in range(len(x_train)):
    if y_train[i] == -1:
        X1trneg, X2trneg = x_train[i][0], x_train[i][1]
        x1_train_neg.append(X1trneg)
        x2_train_neg.append(X2trneg)

    else:
        X1trpos, X2trpos = x_train[i][0], x_train[i][1]
        x1_train_pos.append(X1trpos)
        x2_train_pos.append(X2trpos)

#Plot do gráfico treino
def plot_treino():
```

```
    fig, ax = plt.subplots(figsize=(8,6))
    plt.title("SVM Treino")
    plt.xlabel('$X_1$')
    plt.ylabel('$X_2$')
    plt.scatter(x1_train_pos,x2_train_pos, s=120, marker='+', linewidths=2, label = "$1$", c
    plt.scatter(x1_train_neg,x2_train_neg, s=120, marker='_', linewidths=2, label = "$-1$",
    plt.plot([2.4,0.1],[0.0,4.9],'k')
    plt.legend()
    plt.savefig('SVMTrain.png')
plot_treino()



# Dados para Teste
x_test =  np.array ([ \
        [1.01, 0.82], [0.87, 0.58], [1.63, 0.87], [1.64, 0.32], [0.78, 1.03], #-1
        [1.11, 1.39], [0.70, 1.57], [0.22, 1.62], [0.26, 2.15], [1.05, 2.35], #-1
        [0.68, 3.17], [0.68, 3.17], [0.24, 3.96], [0.18, 0.77], [0.72, 0.26], #-1
        [0.80, 3.64], [1.66, 3.62], [2.19, 3.94], [2.16, 3.48], [1.98, 3.15], # 1
        [2.85, 2.79], [2.13, 2.19], [1.84, 2.83], [1.31, 2.54], [1.77, 1.36], # 1
        [2.80, 0.98], [2.37, 0.64], [3.41, 2.13], [3.00, 2.00], [1.23, 4.74]  # 1
    ])
# "Embaralhamento" dos dados de teste
x_test_shuffle = np.random.shuffle(x_test)
x_test

y_test_s = np.array([1.,1.,-1.,1.,1.,-1.,1.,-1.,-1.,1.,1.,-1.,-1.,
1.,1., -1.,1.,-1.,-1.,1.,1.,-1.,-1.,-1.,-1.,1.,-1.,1.,-1.,1.,])



# Aplicação do SVM Linear
lsvm = LinearSVM(x=x_train, y=y_train)
lsvm.fit()
y_pred = lsvm.predict(x=x_test)



# Definição dos dados de teste: cada ponto "-" corresponde ao label -1 e
# cada ponto "+" corresponde ao label 1
x1_test_pos = []
x2_test_pos = []
x1_test_neg = []
x2_test_neg = []
X1tspos, X2tspos,X1tsneg, X2tsneg=[],[],[],[]

```

```
for i in range(len(x_test)):
    if y_ppred[i] == -1:
        X1tsneg, X2tsneg = x_test[i][0], x_test[i][1]
        x1_test_neg.append(X1tsneg)
        x2_test_neg.append(X2tsneg)

    else:
        X1tspos, X2tspos = x_test[i][0], x_test[i][1]
        x1_test_pos.append(X1tspos)
        x2_test_pos.append(X2tspos)

#Plot do gráfico teste
def plot_ppred():
    fig, ax = plt.subplots(figsize=(8,6))
    plt.title("SVM Teste")
    plt.xlabel('$X_1$')
    plt.ylabel('$X_2$')
    plt.scatter(x1_test_pos,x2_test_pos, s=120, marker='+', linewidths=2, label = "$1$", co
    plt.scatter(x1_test_neg,x2_test_neg, s=120, marker='_', linewidths=2, label = "$-1$", c
    plt.plot([2.4,0.1],[0.0,4.9],'k')
    plt.legend()
    plt.savefig('SVMTest.png')
plot_ppred()



# Análise da Acurácia do Programa
accuracy_score(y_test_s, y_pred)
```