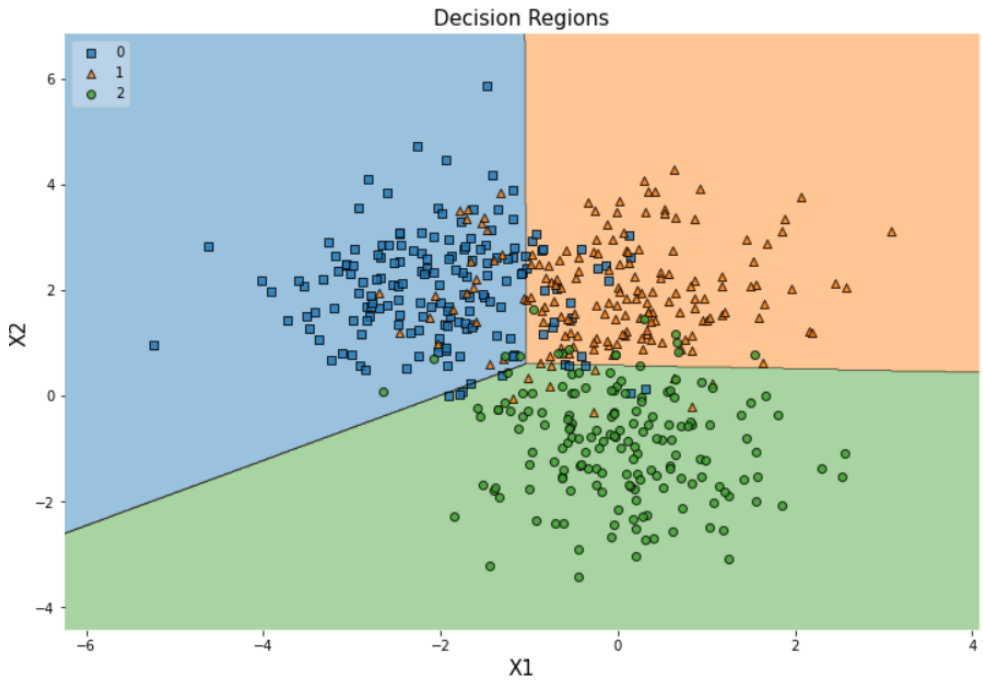
Support Vector Machine (SVM)

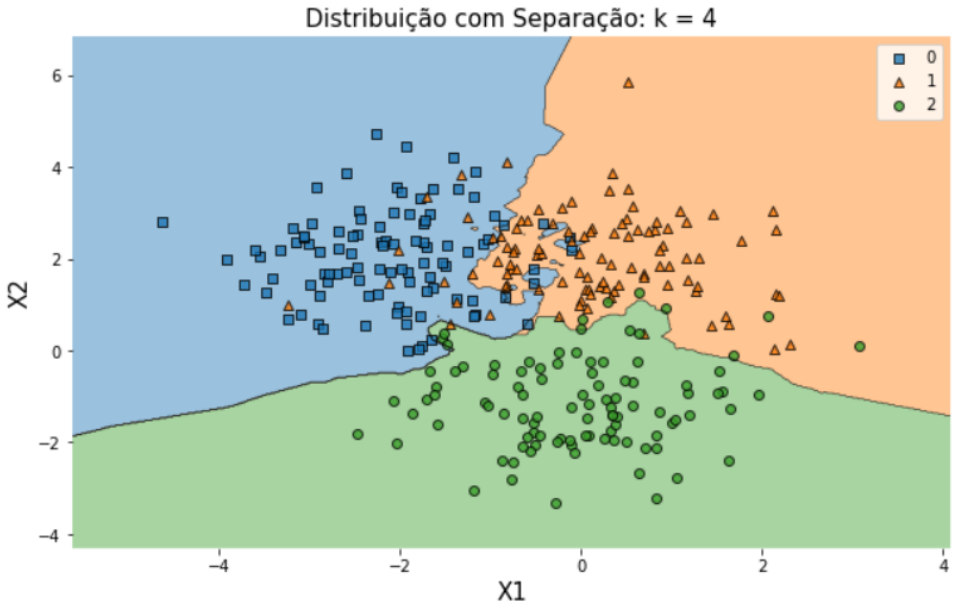
1. Introdução

Ao aplicar diferentes modelos de classificação, uma consequência da separação das classes em si é a criação de uma fronteira de decisão, onde de acordo com o modelo haverá diferentes métodos de separação. A Regressão Logística por exemplo tem uma fronteira de decisão na forma linear ou, de maneira mais genérica para qualquer dimensão, hiperplano.



Fonte: Autoria Própria

De forma análoga, um algoritmo como *K-Nearest Neighbors* (K Vizinhos Próximos) à fronteira de decisão, que é construída com base na distância entre as observações do conjunto de dados, a estrutura da fronteira forma uma geometria conhecida como **Diagrama de Voronoi**.



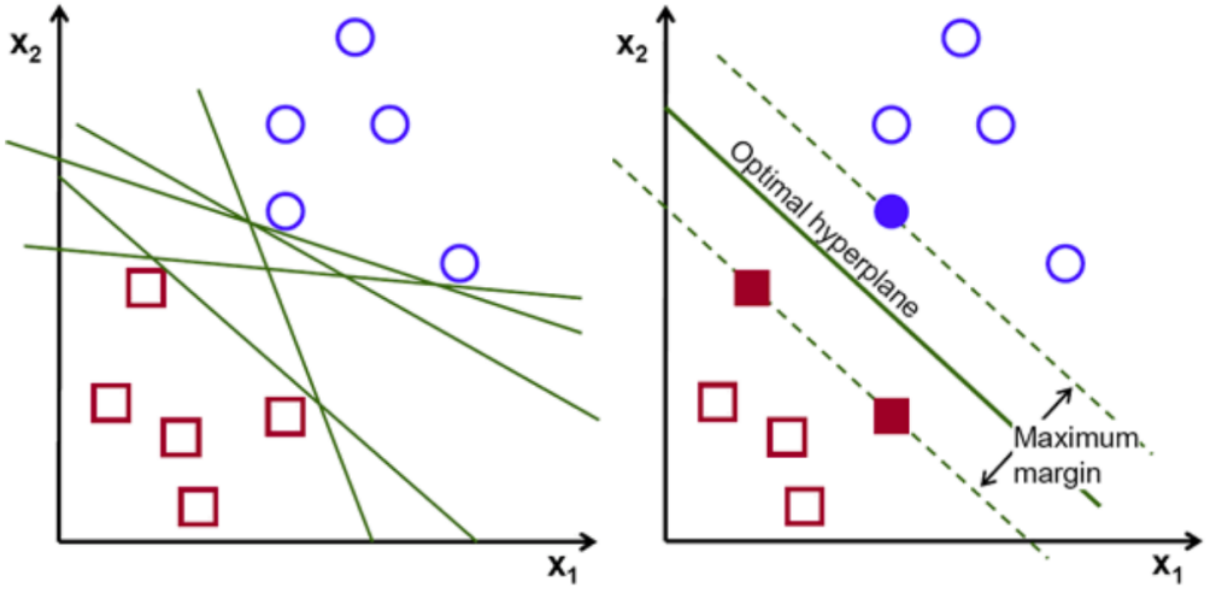
Fonte: Autoria Própria

Apesar disso, não só estes, mas muitos outros modelos de classificação têm uma fronteira de decisão, mas o modelo em foi pensado em cima deste conceito, acaba sendo uma consequência do próprio ato de gerar a separação das classes. Para isso, existe um algoritmo **Support Vector Machine**, onde a sua modelagem foi pensada em achar a melhor fronteira de decisão para o modelo.

2. *Support Vector Machine*

A Máquina de Vetores de Suporte (mais comumente conhecido por **Support Vector Machine** ou pela sigla **SVM**) é um algoritmo de *Machine Learning* que busca fazer a separação das classes de um determinado conjunto de dados a partir de uma fronteira de decisão linear (ou em hiperplanos para alta dimensionalidade). Vale ressaltar que o por mais que este material seja focado em aplicações do *SVM* como modelos de classificação, o *SVM* também pode ser utilizado para modelos de regressão.

Existem diversas possibilidades de retas (ou hiperplanos) que possam fazer a separação de duas ou mais classes, mas o algoritmo por trás do SVM busca determinar a reta com a separação de **margem máxima**, ou seja, a distância máxima entre os pontos extremos de um conjunto de dados. Maximizar a distância entre as classes garante uma confiança maior para a predição de dados futuros, ou seja, um modelo que generaliza melhor.



Fonte: Alvarez

O treinamento de um modelo SVM, utilizando a implementação do sklearn, pode ser feita de maneira bem simples, segundo o código em *Python* a seguir:

```
# Carrega a função do SVM
from sklearn.svm import SVC

# Instancia o modelo
model = SVC()

# Fit do modelo com os dados de treino
model.fit(X_train, y_train)

# Gera as predições com os dados de teste
y_pred = model.predict(X_test)
```

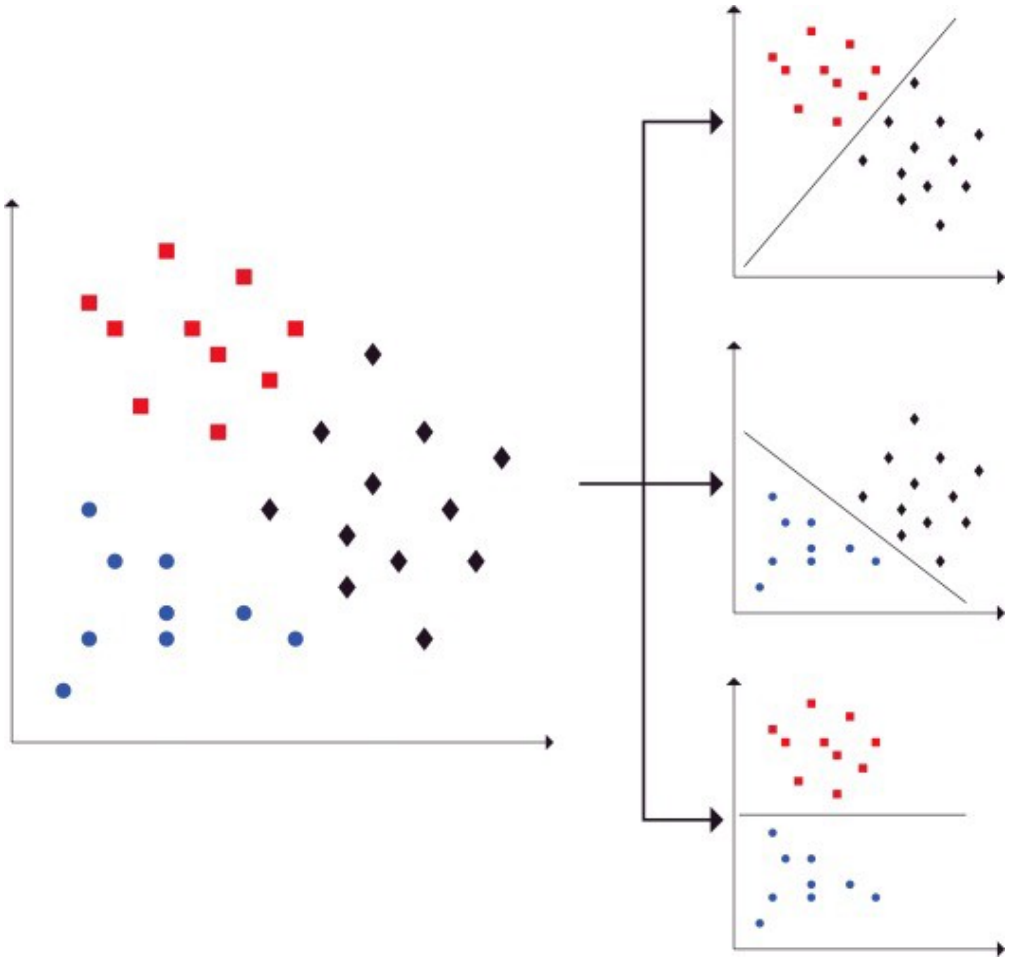
Exemplo Prático

```
# Aplicando o modelo de SVM na classificação das espécies da Iris

# Carregando as bibliotecas necessárias
import pandas as pd
import seaborn as sns
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import train_test_split
```

```
from sklearn.svm import SVC
from sklearn.metrics import classification_report

# carrega o conjunto de dados
iris = sns.load_dataset('iris')

# Separação dos dados em atributos (X) e variável resposta (y)
X = iris.drop('species', axis = 1)
y = iris['species']

# Separação da base em treino e teste
X_train, X_test, y_train, y_test = train_test_split(X,
# atributos
                                                    y,
# Variável resposta
                                                    test_size = 0.3,
# proporção base de treino
                                                    random_state = 42,
# semente aleatório
                                                    stratify = y)
# mantém a proporção das classes

# Instancia a Normalização
scaler = StandardScaler()

# Normaliza os dados
# OBS.: Em modelos que trabalham com distâncias, deve-se normalizar os
# dados para que todos os atributos estejam na mesma escala e a escala
# não influencie o desempenho do modelo
X_train_std = scaler.fit_transform(X_train)
X_test_std = scaler.transform(X_test)

# Instancia o modelo
model = SVC()

# Fit do modelo com os dados de treino
model.fit(X_train_std, y_train)

# Gera as predições com os dados de teste
y_pred = model.predict(X_test_std)

# Avaliando o desempenho do modelo
print(classification_report(y_test, y_pred))
```

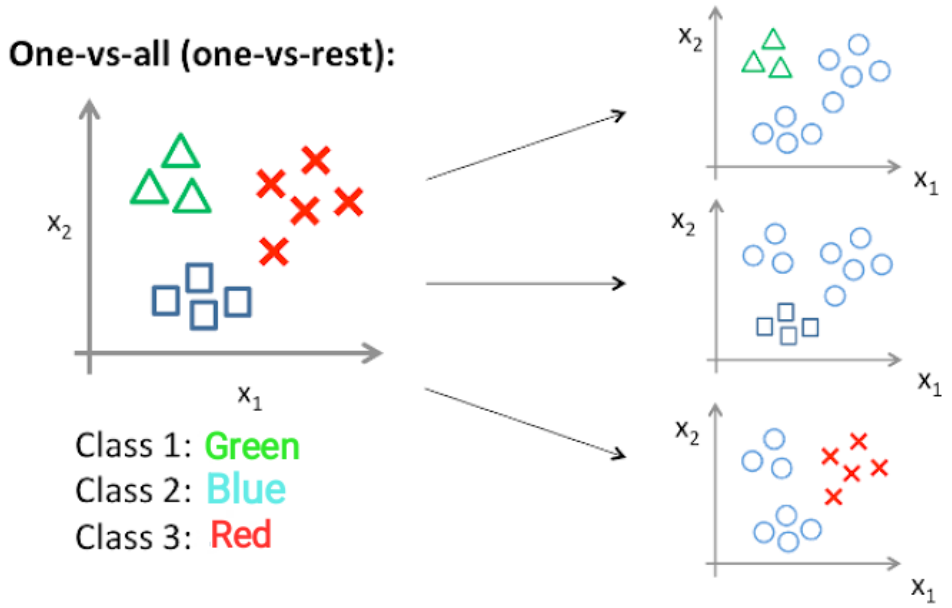
Importante salientar que o modelo SVM inicialmente foi projetado para o caso de **separações binárias**, mas com auxílio de estratégias de multiclasses consegue também fazer a classificação 2 ou mais classes. Para implementar o modelo para multiclasses basta passar um parâmetro em `decision_function_shape` da forma `ovo` ou `ovr` sendo as estratégias **One vs One** e **One vs Rest** respectivamente, descritas brevemente a seguir:

- **One vs One:** O método One vs One (OvO) consiste em basicamente pegar as classes em pares, e para cada um dos pares possíveis fazer as predições (por exemplo, pegar as classes quadrados e círculos, utilizar o modelo para definir se uma observação pertence ao quadrado ou ao círculo). Após avaliar todas as combinações de pares de classes, a predição da melhor classe é feita para cada uma das observações, olhando qual foi a classe mais votada para ela. Note que para este método, dado que um conjunto de dados tenha N classes, tem-se que o método irá rodar $\frac{N(N - 1)}{2}$ vezes, o que indica um ganho considerável no custo computacional;



Fonte: Medium

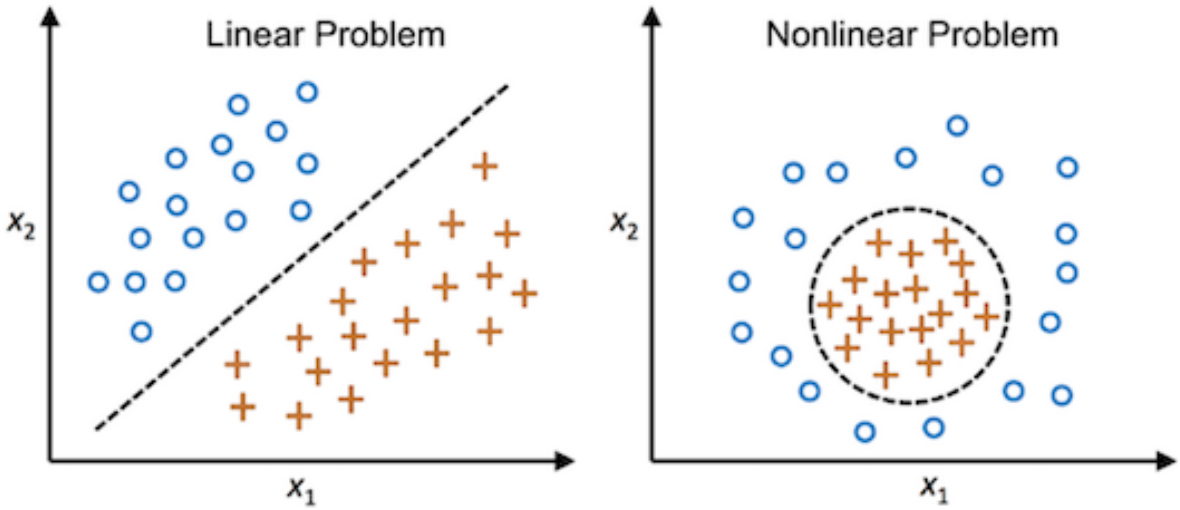
- **One vc Rest:** O método One vs Rest (OvR), diferente do anterior, consiste em fazer várias classificações olhando apenas para uma classe, ou seja, ele vai verificar se pertence ou não a uma classe e fazer esse tipo de avaliação para todas as demais. A predição da melhor classe é novamente feita para cada uma das observações, olhando qual foi a classe mais votada para ela. No caso One vs Rest será testado N vezes o modelo, para o mesmo caso de um conjunto de dados com N classes, mostrando uma diferença em questão de desempenho em termos de custo computacional.



Fonte: Towards Data Science

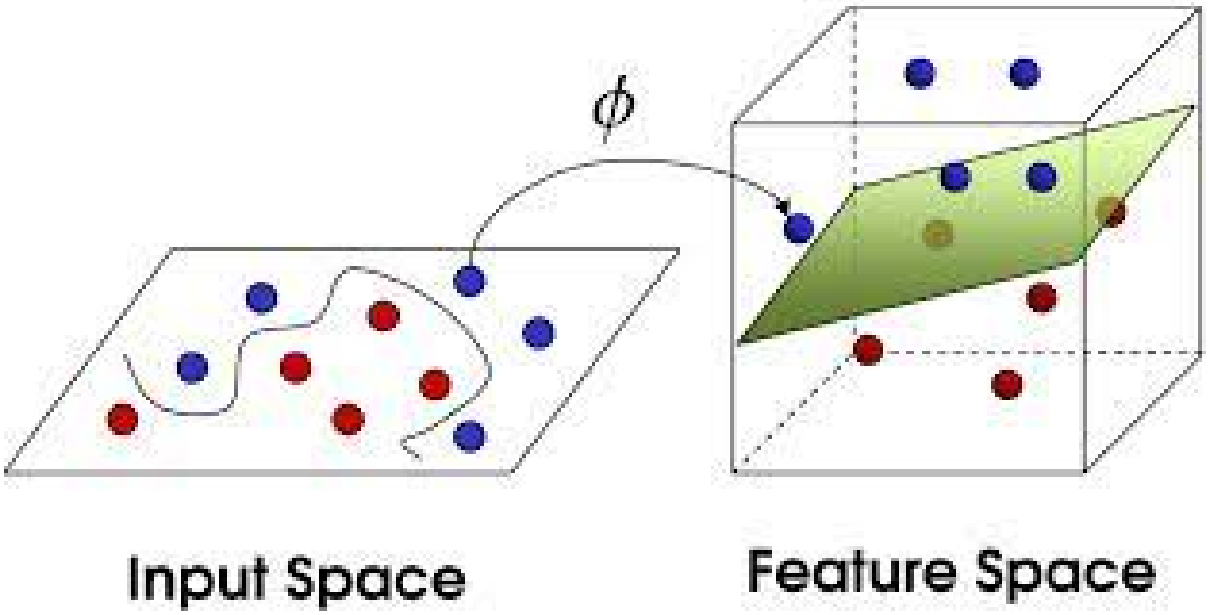
3. SVM Não Linear

Na prática, o SVM consegue fazer boas classificações lineares, mas devemos nos atentar ao fato de que muitos problemas do mundo real não são lineares, ou seja, linearmente separáveis!



Fonte: CMD Line Tips

Uma forma de contornar esses tipos de cuidados com os dados é transformando-os antes de passar pelo modelo, ou também conhecido como **truque de Kernel**:



Fonte: UTFPR

As transformações que podem ser aplicadas ao kernel na implementação do *Scikit-learn* para o SVM são `linear`, `radial`, `polynomial` e `sigmoid`.

Materiais Complementares

Canal *StatsQuest*, vídeo sobre SVM;

Documentação no Scikit-Learn sobre o SVM;

Artigo da *Machine Learning 101* publicado por Savan Patel no Medium - *Chapter 2: SVM*.

Referências

James, Gareth, et al. An Introduction to Statistical Learning: With Applications in R. Alemanha, Springer New York, 2013;

Talwalkar, Ameet, et al. Foundations of Machine Learning. Reino Unido, MIT Press, 2018.

Próximo Tópico