MACHINE LEARNING

Algoritmos de Classificação

cognitiza

Your Data & AI school

Prof. Altino Dantas

Agenda

- Conceito
- Problemas
- Algoritmos
- Exercícios

Conceito

- Aprendizado supervisionado, ou seja, existem as características (variáveis independentes) e o alvo (variável dependente);
- A variável dependente representa uma categoria para os dados, ou seja, uma classe para a amostra;
- Necessidade de separação de conjunto de treino e teste;
- Todos os exemplos destes slides usarão o *scikit learn.*

Problemas

- Identificação de notas falsas
- Seleção de frutas estragadas em esteiras
- Identificação de linhagem de espécie
- Identificação de origem de bebida
- Biometria
- Predição de inadimplência

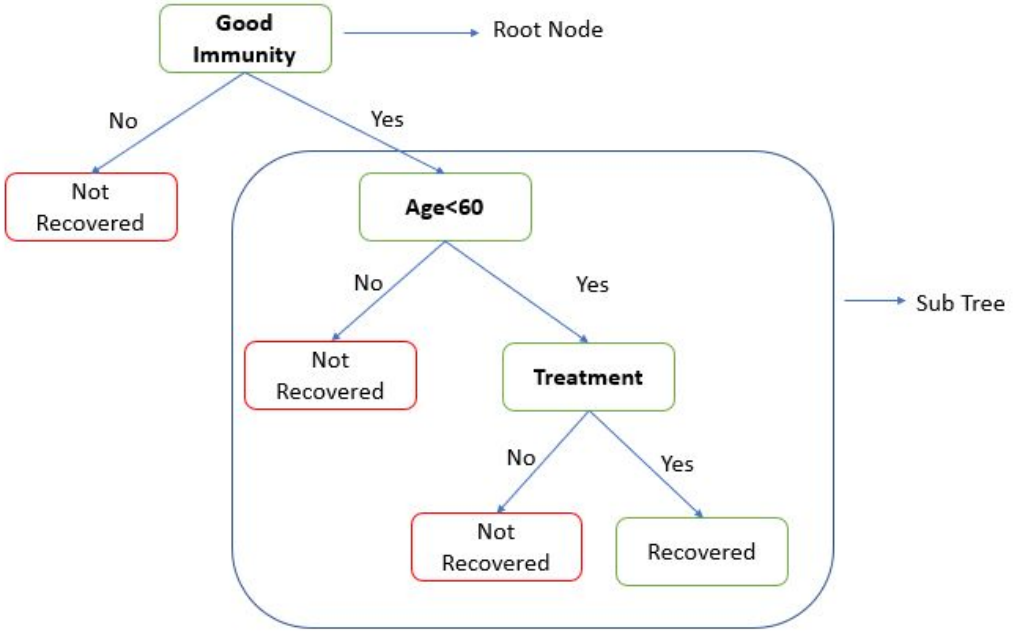
Alguns algoritmos

- Regressão logística
- Árvores de decisão
- Maquina de vetores de suporte (SVMs)
- Redes Neurais Artificiais - Perceptron Multicamadas
- K vizinhos mais próximos
- Muitos outros...

Árvore de decisão

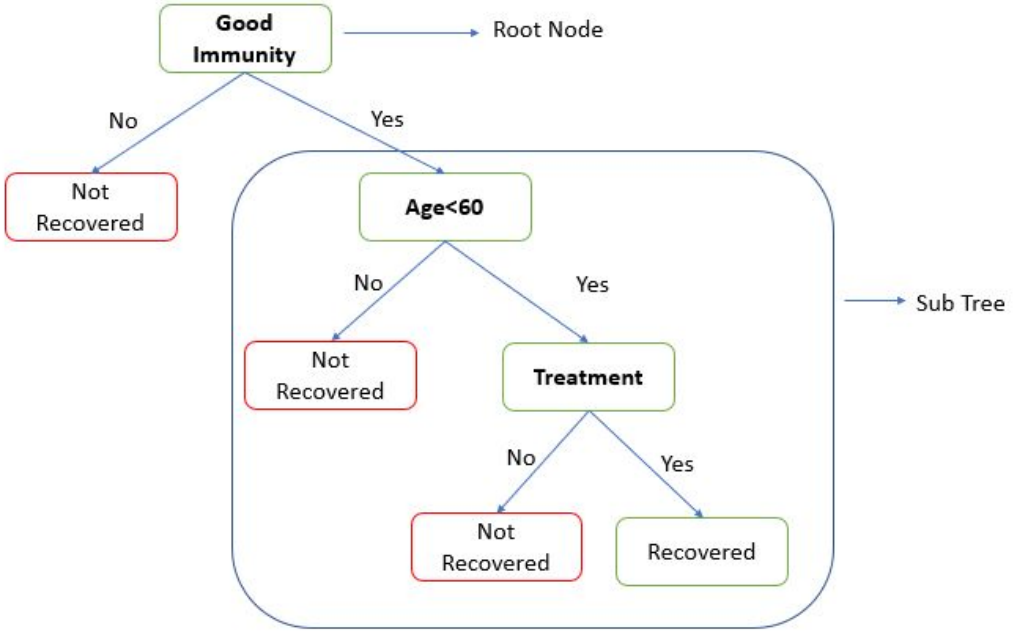
Árvore de decisão

- Técnica de aprendizado supervisionado;
- Realiza tarefas de regressão e classificação;
- Modela a relação entre as características (variáveis independentes) e a variável dependente, pelo conceito de árvore.



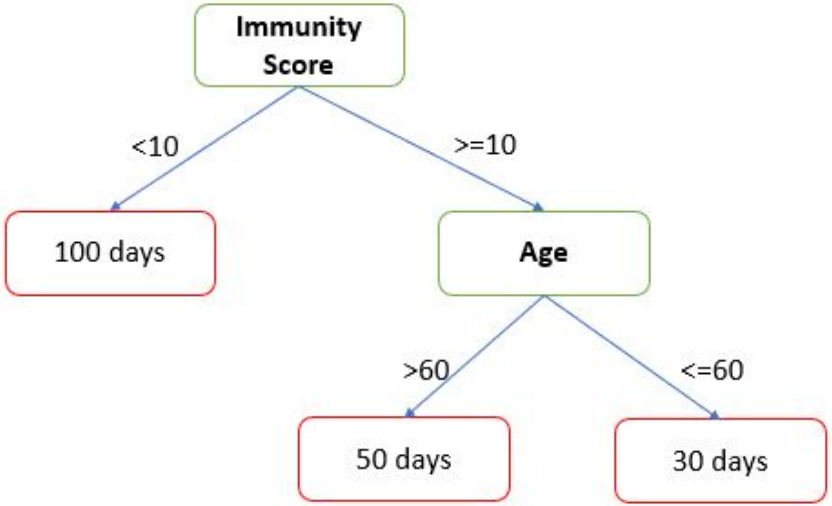
Árvore de decisão

- Nó raiz;
- Nó pai e nó filho;
- Nós folha;
- Nós de decisão.



Árvore de decisão

- Nó raiz;
- Nó pai e nó filho;
- Nós folha;
- Nós de decisão.



Classificação

- Contam-se as ocorrências de amostras em uma região ou folha;
- As condições são definidas pela minimização do erro:

$$E = 1 - \max_k(\hat{p}_{mk})$$

- $\hat{P}_{mk}$ é a proporção de observações na m-ésima região que são de uma k-ésima classe;

Classificação

- Contam-se as ocorrências de amostras em uma região ou folha;
- As condições são definidas pela minimização do erro:

$$E = 1 - \max_k(\hat{p}_{mk})$$

- $\hat{P}_{mk}$ é a proporção de observações na m-ésima região que são de uma k-ésima classe;
- Outras medidas usadas para cálculo de erro:

Índice Gini $G = \sum_{k=1}^{K} \hat{p}_{mk}(1 - \hat{p}_{mk})$

Varância total entre as classes

Entropia cruzada $D = -\sum_{k=1}^{K} \hat{p}_{mk} \log \hat{p}_{mk}$

Avalia a desordem de um grupo considerando as classes

Vantagens

- Fácil de entender e explicar;
- Funciona muito bem em conjuntos de dados menores e pode ser visualizado graficamente;
- Como não criam variáveis fictícias, podem ter um bom desempenho em problemas de classificação;
- Computacionalmente rápido na classificação de pontos de dados desconhecidos, baixo custo de inferência (rápida inferência);
- Baixa complexidade espacial.

Desvantagens

- Muito susceptíveis a superajuste nos dados de treino;
- Muito sensível aos dados, pequenas alterações nas entradas podem produzir grandes variações na predição;
- Quando grandes demais podem inviabilizar a predição ou não serem acuradas;
- Se um nó está tendo muitas divisões, então existe a possibilidade de dar mais importância a isso, resultando em previsões tendenciosas.

Exemplo de código

- Sklern em python possui uma implementação de árvore de decisão;
- Os principais parâmetros são:
  - **criterion**: define a função a ser usada para as divisões;
  - **max_depth**: profundidade máxima da árvore;
  - **min_sample_split**: quantidade mínima de exemplos para a divisão (padrão: 2);
- Cenário:
  - Um hospital que cuida de pacientes com diabetes deseja explorar os dados dos prontuários;
  - Depois de uma investigação, a área de dados monta um dataset contendo diversas características dos pacientes, sendo uma delas o diagnóstico.

```
import pandas as pd
from sklearn.tree import DecisionTreeClassifier
from sklearn.model_selection import train_test_split
from sklearn import metrics

dados = pd.read_csv("diabetes.csv")
```

```
feature_cols = ['pregnant', 'insulin', 'bmi', 'age','glucose','bp','pedigree']
X = dados[feature_cols]
y = dados.label

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=4)
```

```
clf = DecisionTreeClassifier()
clf = clf.fit(X_train,y_train)
y_pred = clf.predict(X_test)
```

```
print("Accuracy:",metrics.accuracy_score(y_test, y_pred))
print("\nConfusionMatrix:\n",metrics.confusion_matrix(y_test, y_pred))
```

```python
import pandas as pd
from sklearn.tree import DecisionTreeClassifier
from sklearn.model_selection import train_test_split
from sklearn import metrics

dados = pd.read_csv("diabetes.csv")
```

```python
feature_cols = ['pregnant', 'insulin', 'bmi', 'age','glucose','bp','pedigree']
X = dados[feature_cols]
y = dados.label

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=4)
```

```python
clf = DecisionTreeClassifier()
clf = clf.fit(X_train,y_train)
y_pred = clf.predict(X_test)
```

```python
print("Accuracy:",metrics.accuracy_score(y_test, y_pred))
print("\nConfusionMatrix:\n",metrics.confusion_matrix(y_test, y_pred))
```

```
Accuracy: 0.7012987012987013

ConfusionMatrix:
 [[115  37]
 [ 32  47]]
```

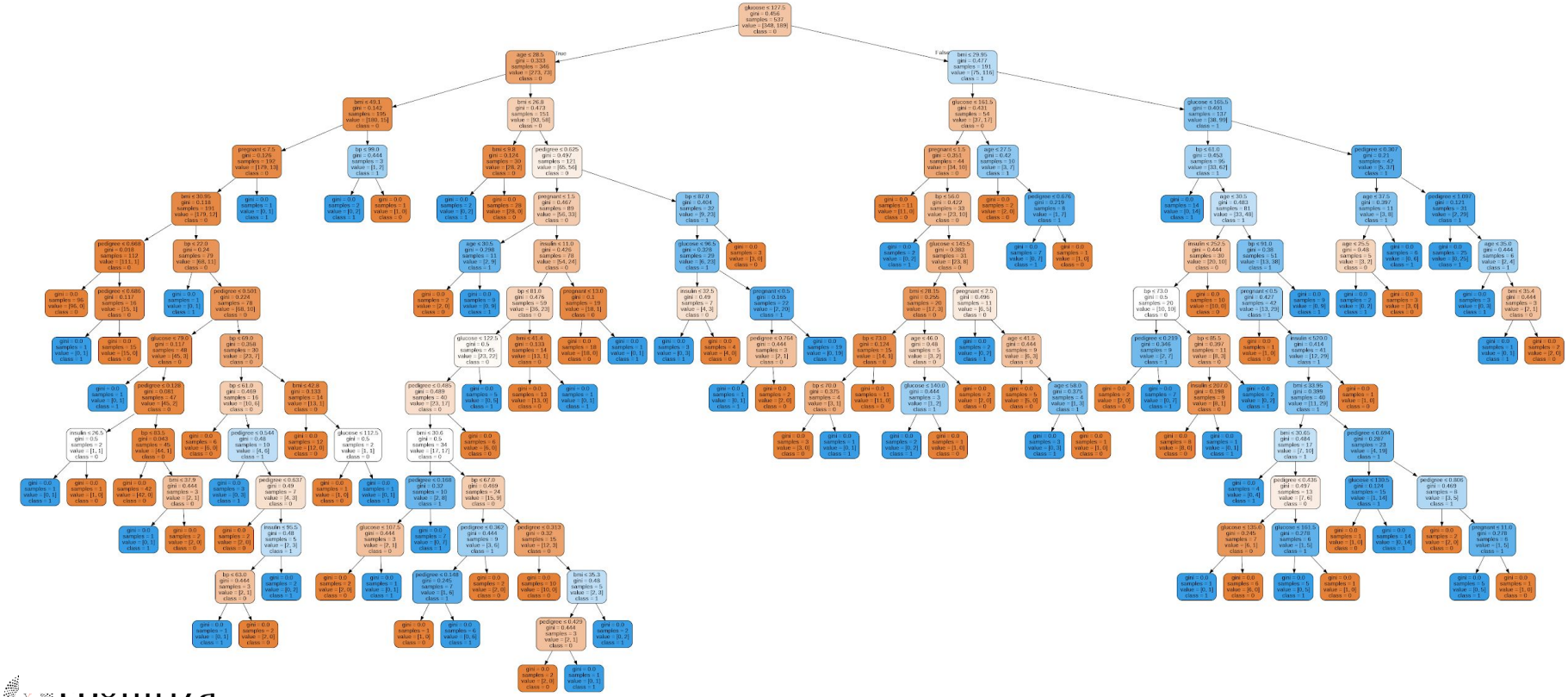
Exercício 1

- Replique o exemplo apresentado;
- O que poderia ser feito para melhorar a acurácia?
- Aumentar a profundidade da árvore necessariamente melhora a acurácia? Justifique.
- Atinja pelo menos 75% de acurácia.

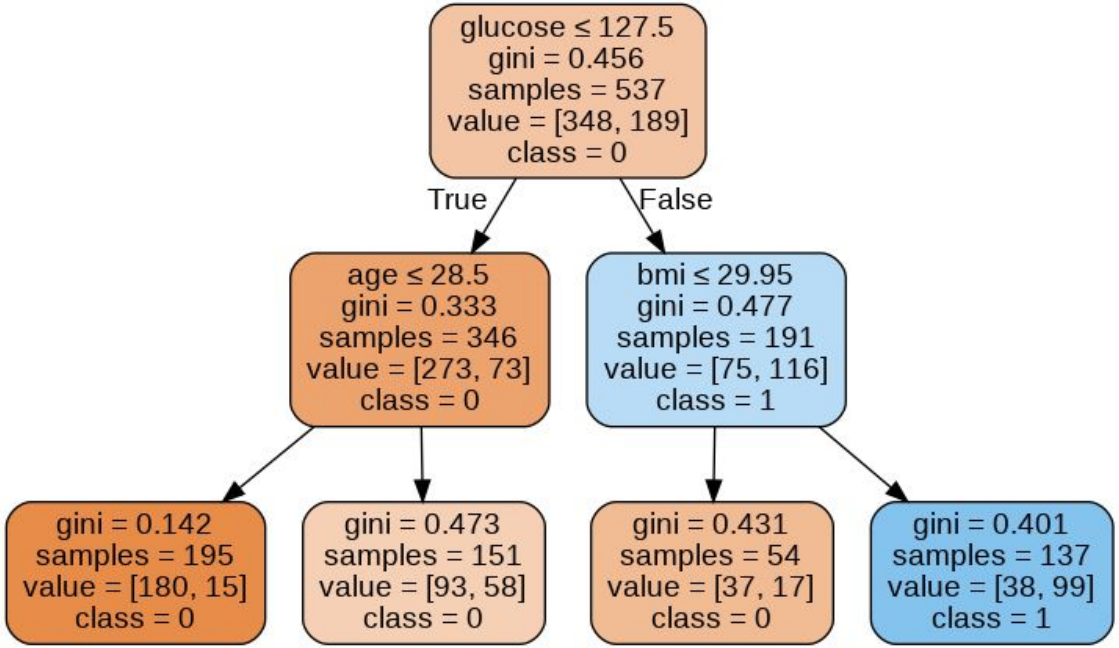
Visualizando

```
from sklearn.externals.six  import StringIO
from IPython.display  import Image
from sklearn.tree  import export_graphviz
import pydotplus
dot_data = StringIO()
export_graphviz(clf, out_file=dot_data,
                filled= True, rounded=True,
                special_characters= True,
               feature_names = feature_cols,
               class_names=['0','1'])
graph = pydotplus.graph_from_dot_data(dot_data.getvalue())
graph.write_png('diabetes.png')
Image(graph.create_png())
```

Árvore para configuração padrão



Profundidade máxima = 2



Exercício 2

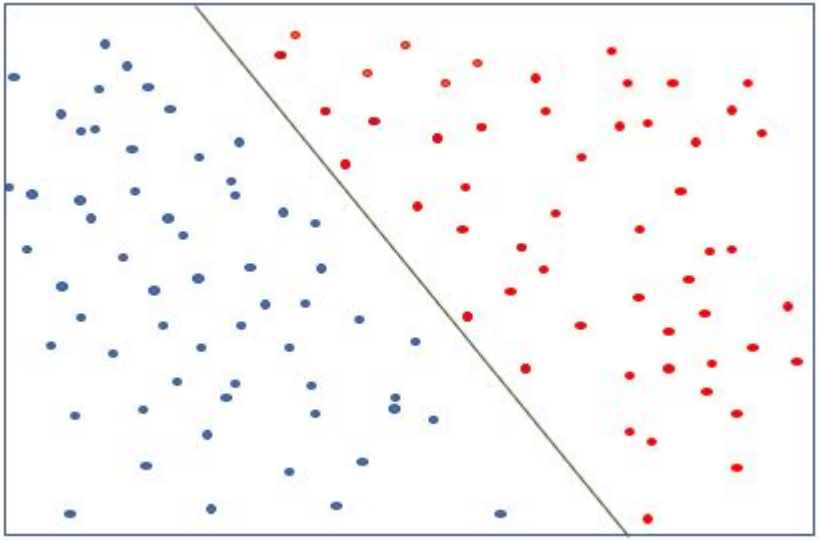
- Apresenta um gráfico de linhas, com valores de profundidade de árvore no eixo x e a acurácia no teste e no conjunto de treino y;
- Apresente o desenho da árvore que obteve maior acurácia no **treino**;
- Apresente o desenho da árvore que obteve maior acurácia no **teste**;

Máquina de Vetores de Suporte

Support Vector Machines (SVM)

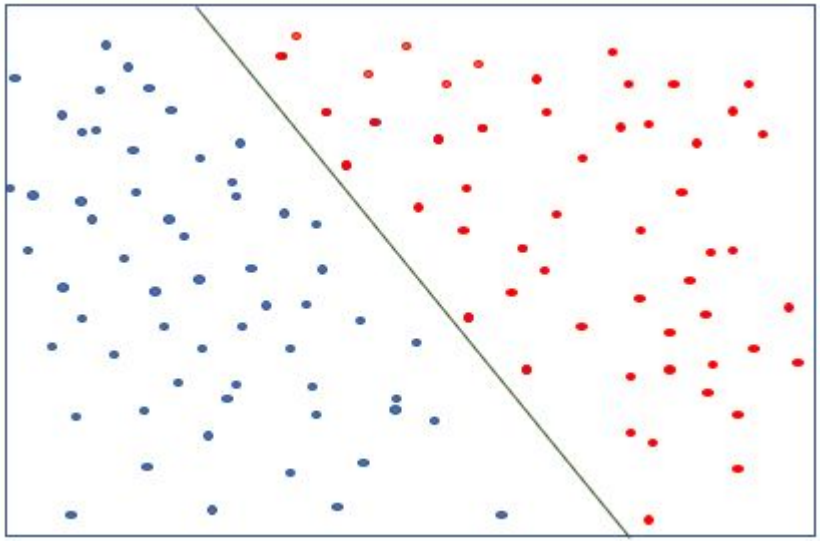
SVM

- Aprendizado supervisionado;
- Tarefas de regressão e classificação;
- Baseado nos conceitos de hiperplano, margem e núcleo (*kernel*);
- Tende a ter uma eficácia elevada, mas é computacionalmente complexo.



SVM - hiperplano

- Assumindo um espaço 2D e alguns pontos, a linha [*ax+by+c=0*] é capaz de separar esse espaço;
- Existe uma quantidade infinita de hiperplanos para qualquer espaço *p*-dimensional, mas a visualização é complexa;
- X = (X1,X2,…,X*p*), *p* dimensões, pode ser avaliado pela seguinte equação:

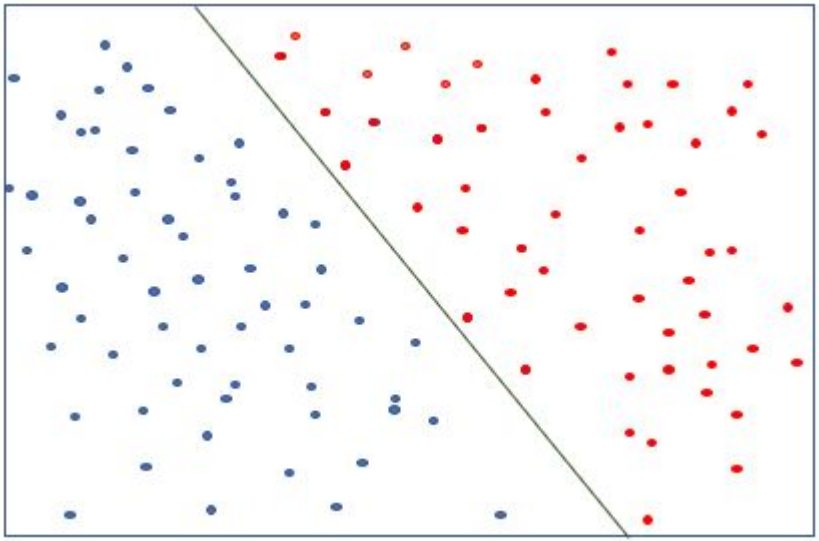


$$\beta_0 + \beta_1 X_1 + \beta_2 X_2 + \ldots + \beta_p X_p = 0$$

SVM - hiperplano

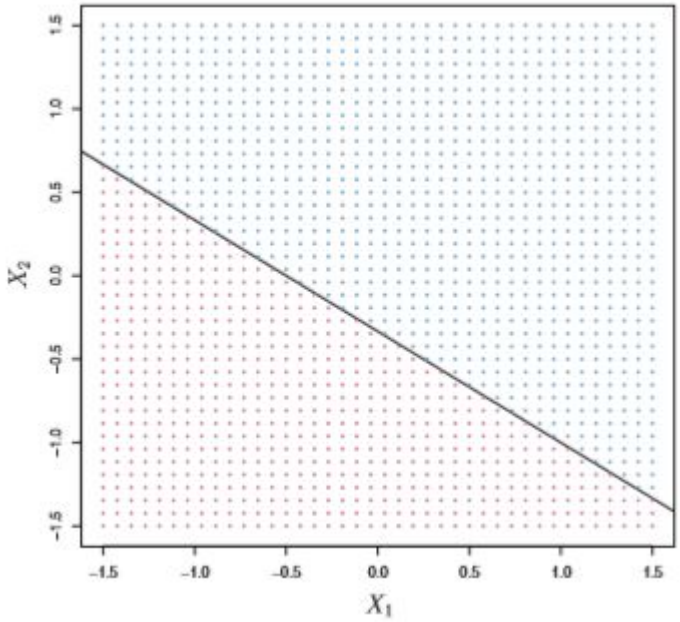
$$\beta_0 + \beta_1 X_1 + \beta_2 X_2 + \ldots + \beta_p X_p = 0$$

- X = (X1,X2,…,X*p*)
- Se X satisfaz a inequação X > 0, então X está de um dos lados do hiperplano;
- Caso contrário, se X pode estar sobre o hiperplano, ou ser menor do que 0, e, neste caso, está do outro lado;



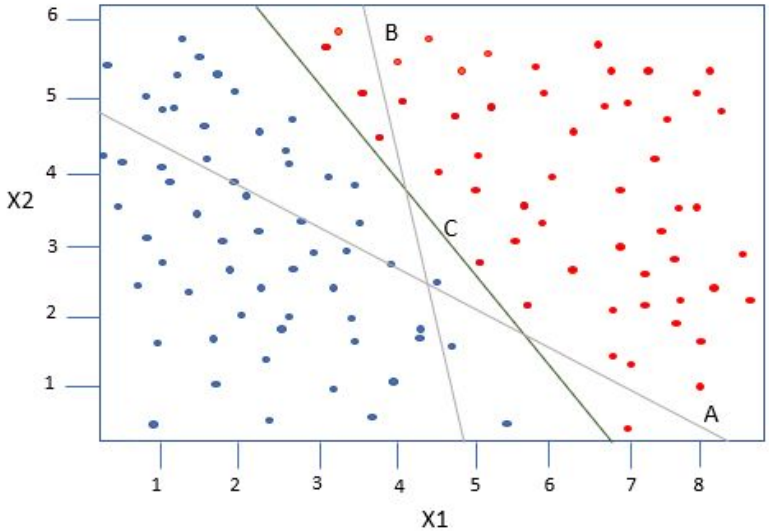
SVM - hiperplano

- ***1+2X_1+3X_2=0***, por exemplo, seria como a figura ao lado;
- Qualquer ponto que satisfaça ***1 + 2X1 + 3X2 < 0*** está na região vermelha e ***1 + 2X1 + 3X2> 0*** está na região azul.



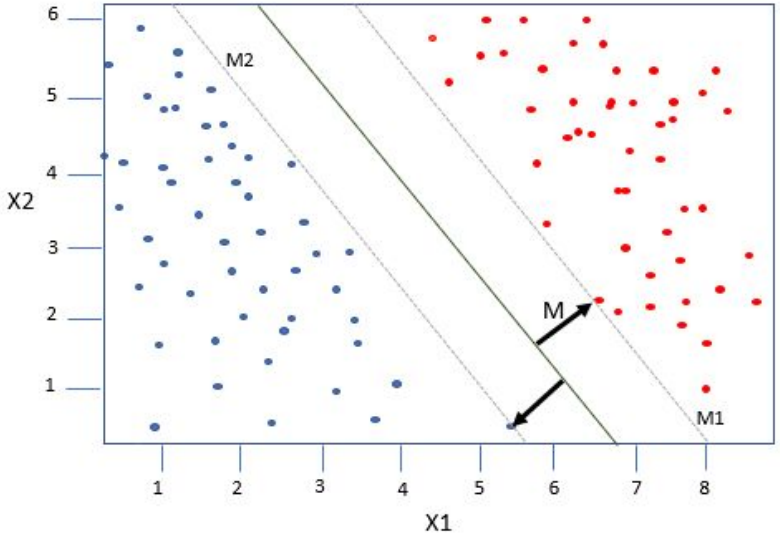
SVM - margem

- Existem infinitos possíveis hiperplanos para um espaço contínuo;
- Qual dele é o melhor? No exemplo ao lado, seria o hyperplano A, B ou C?;
- SVM usa o conceito de **margens**.



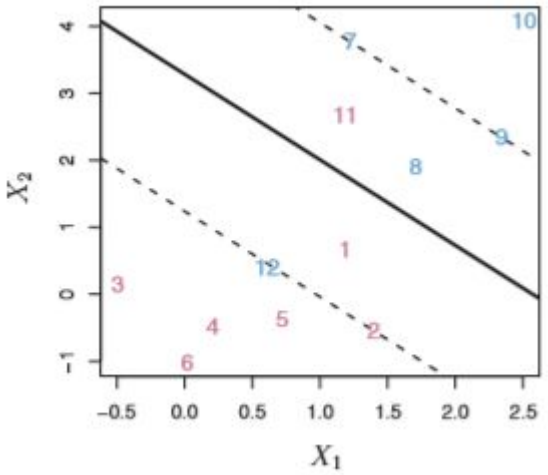
SVM - margem

- As linhas paralelas ao hiperplano mais próximas dos pontos limites são M1 e M2;
- A distância entre o hiperplano e os M1 e M2 é chamada de margem;
- Os pontos mais próximos de M são chamados de vetores de suporte;
- Para diferentes hiperplanos há diferentes tamanhos de margem;
- A margem máxima é obtida pelo hiperplano de separação ótimo.



SVM - margem

- Os pontos 1 e 8 estão em lados errados de acordo com suas respectivas margens, mas estão corretos em relação ao hiperplano;
- Os pontos 11 e 12 estão errados tanto quanto a margem como o hiperplano;
- O que acontece quando os vetores de suporte estão próximos do hiperplano?



SVM - margem

- O objetivo do algoritmo é maximizar M:

$$\underset{\beta_0,\beta_1,\ldots,\beta_p,\epsilon_1,\ldots,\epsilon_n}{\text{maximize}} \quad M$$

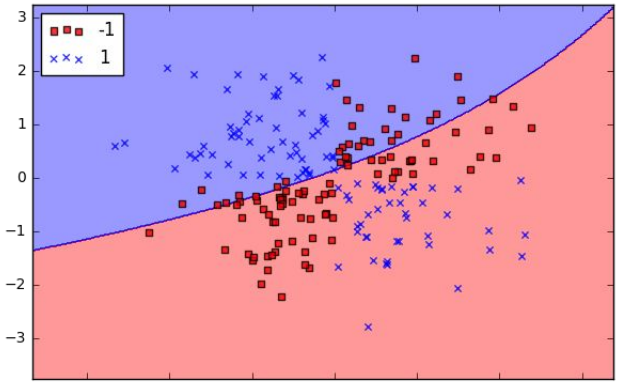
$$\text{subject to} \quad \sum_{j=1}^{p} \beta_j^2 = 1,$$

$$y_i(\beta_0 + \beta_1 x_{i1} + \beta_2 x_{i2} + \ldots + \beta_p x_{ip}) \geq M(1 - \epsilon_i),$$

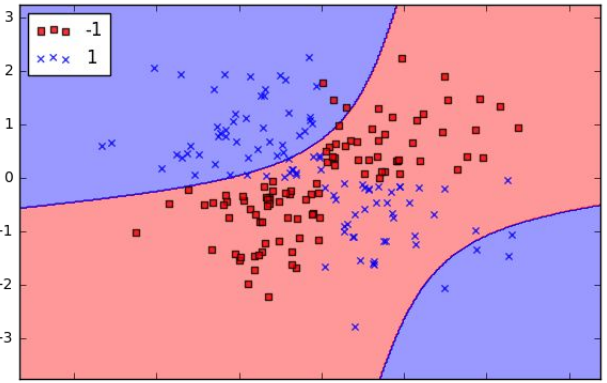
$$\epsilon_i \geq 0, \quad \sum_{i=1}^{n} \epsilon_i \leq C,$$

- $B_o,\beta_1,\ldots,\beta_p$ são os coeficientes que produzem o hiperplano de margem máxima;
- ε_i são variáveis de folga que permitem observações do lado errado da margem;
- C é um parâmetro que balanceia a quantidade e intensidade das violações às margens.
- $C = o$ indica que apenas a margem será levada em conta, sem violações.

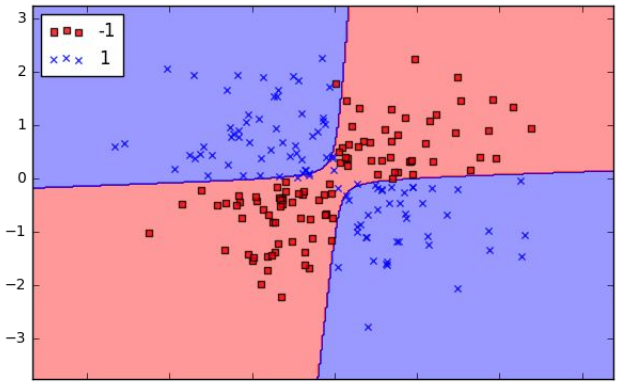
C=1



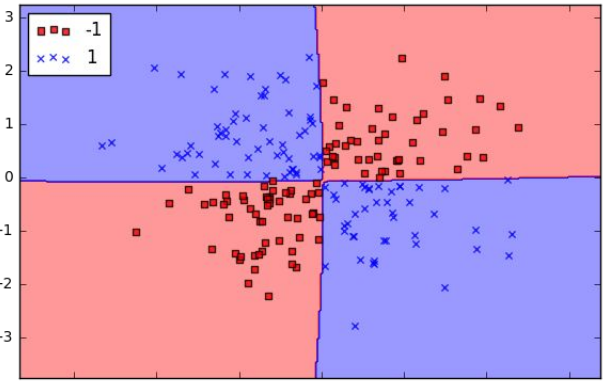
C=10



C=100

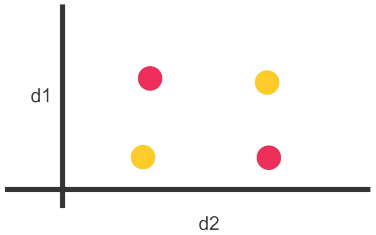


C=100000

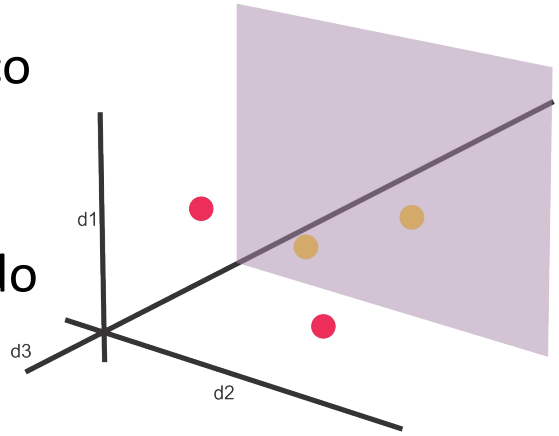


Source: https://scikit-learn.org/stable/auto_examples/svm/plot_rbf_parameters.html

SVM - Núcleo (Kernels)



- No mundo real, dificilmente os dados estão dispostos de forma a serem linearmente separáveis;
- Ou seja, é necessário separar dados que estão dispostos de qualquer forma no espaço;
- Os dados podem não estar separados em um espaço *p*-D, mas podem estar se o espaço tiver *p + n* dimensões;
- É possível mudar o espaço das características usando funções polinomiais, por exemplo.



SVM - Núcleo (Kernel)

$$y_i = \beta_0 + \beta_1 X_1 + \beta_2 X_1^2 + \beta_3 X_2 + \beta_4 X_2^2 + \beta_5 X_2^3 \ldots = 0$$

- Existem diversas funções núcleo, algumas:

$$K(x, y) = (1 + \sum_{j=1}^{p} x_{ij} . y_{ij})^d$$

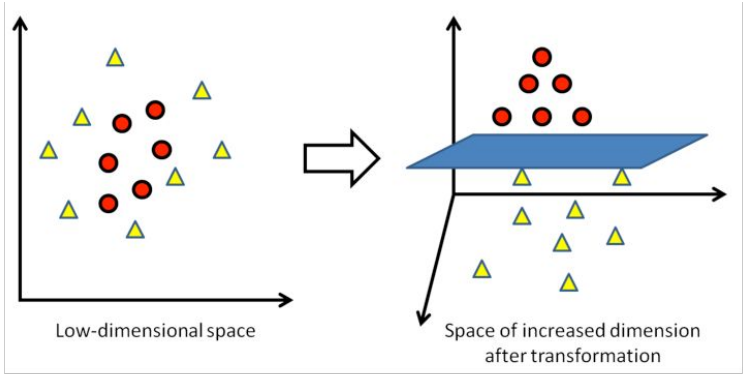
Polinomial

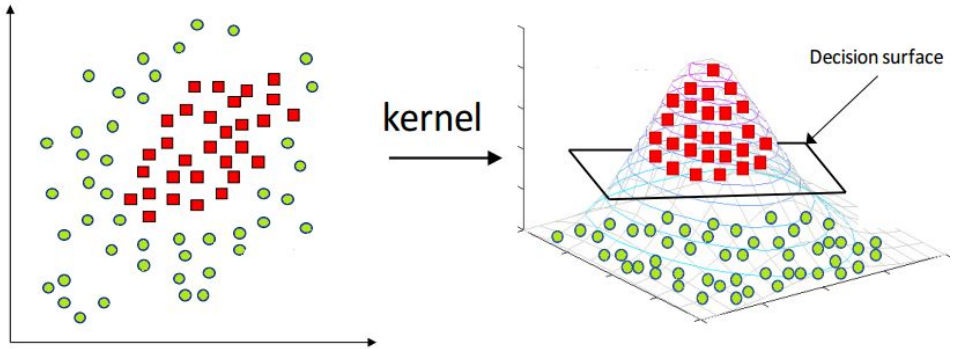
$$k(x, y) = \exp(-\gamma \sum_{j=1}^{p} (x_{ij} - y_{ij})^2)$$

Radial

- Finalmente, teríamos: $f(x) = \beta_0 + \sum_{i \in \mathcal{S}} \alpha_i K(x, y)$

SVM - Núcleo (Kernel)





Fuente: Google Images

Principais parâmetros

- C = parâmetro de regularização;
- kernel = **‘linear’, ‘poly’, ‘rbf’, ‘sigmoid’, ‘precomputed’,** default**=’rbf’**
- degree **=** para kernel polinomial especifica o grau;
- Gamma = mais um parâmetro de regularização para os kernels radial, polinomial e sigmoidais.

Exemplo de código

```python
import pandas as pd
import seaborn as sns
from sklearn.tree import DecisionTreeClassifier
from sklearn.model_selection import train_test_split
from sklearn import metrics
from sklearn.svm import SVC, LinearSVC

dados = pd.read_csv("diabetes.csv")
```

```python
feature_cols = ['pregnant', 'insulin', 'bmi', 'age','glucose','bp','pedigree']
X = dados[feature_cols]
y = dados.label

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=1)
```

```python
clf = SVC()
clf = clf.fit(X_train,y_train)
y_pred = clf.predict(X_test)
```

```python
print("Accuracy:",metrics.accuracy_score(y_test, y_pred))
print("ConfusionMatrix:\n",metrics.confusion_matrix(y_test, y_pred))
```

```
Accuracy: 0.7662337662337663
ConfusionMatrix:
 [[138   8]
 [ 46  39]]
```

Exercício 2

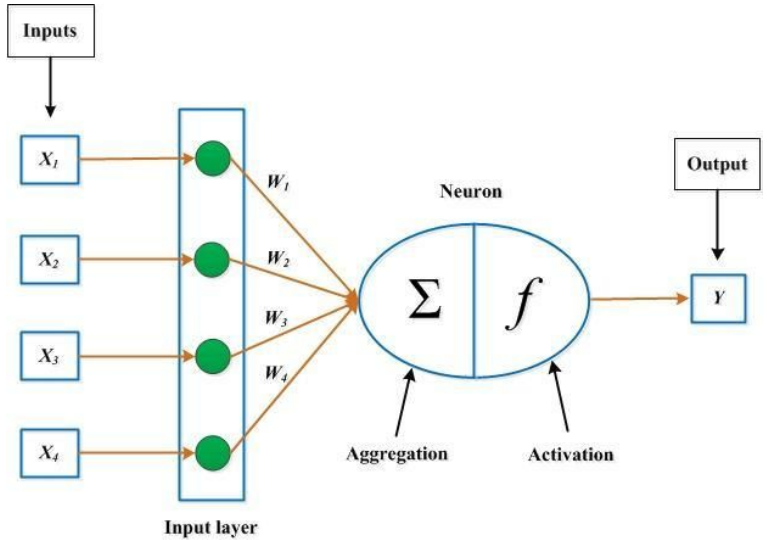
- Replique o exemplo apresentado;
- Considerando os parâmetros estudados na aula, melhore a acurácia para pelo menos 80%, no conjunto de teste;
- Monitore os resultados de predição no treino e no teste;
- Faça alguns testes e responda: qual é a técnica mais suscetível a *overfitting*, SVM ou Árvore de decisão? Justifique sua resposta ou apresente dados que a embase.

Rede Neural Artificial

Perceptron Multicamadas

Caracterização

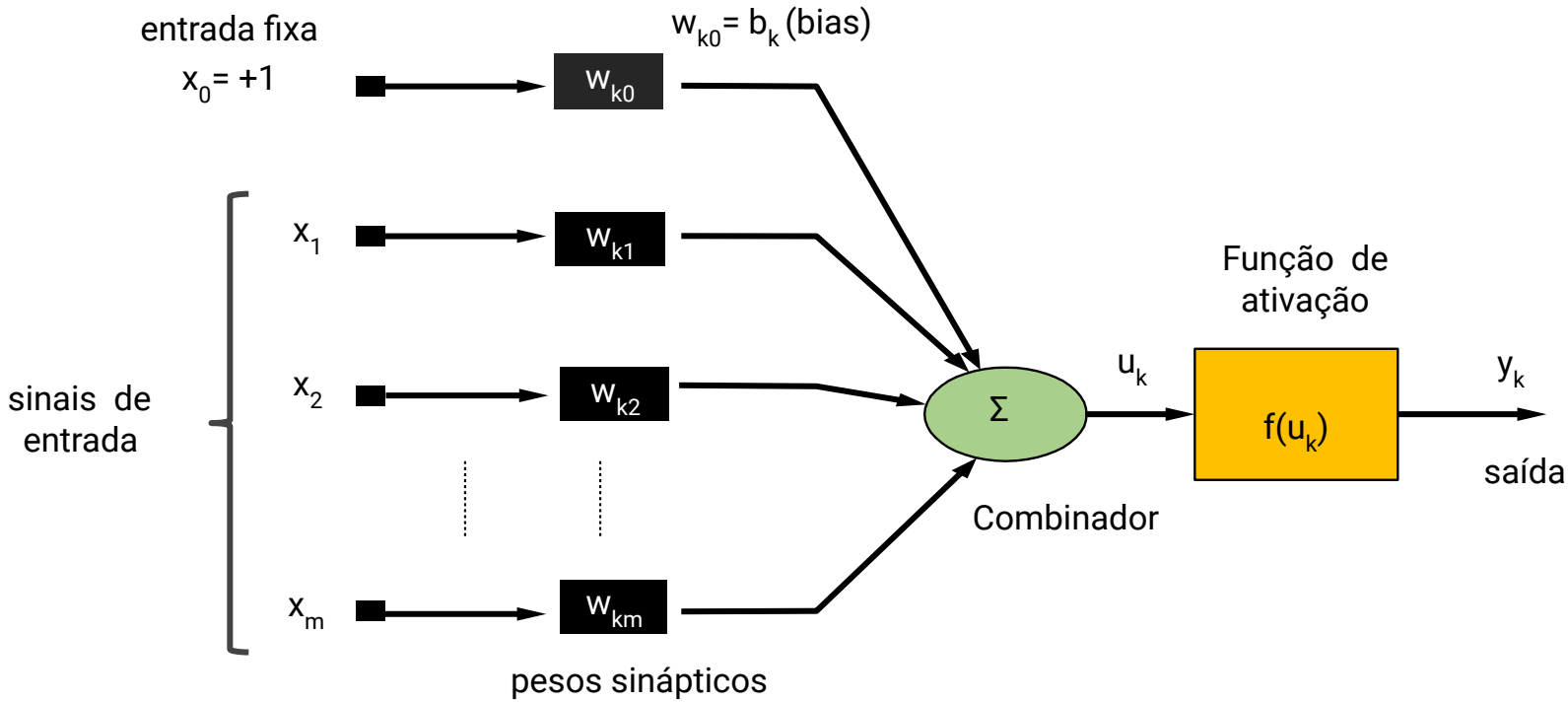
- Redes neurais artificiais são inspiradas nas redes neurais reais;
- Conhecidas como aproximadores universais de função;
- Podem ser usadas para qualquer tarefa de machine learning;
- Têm alta complexidade computacional;
- Boa desempenho em problemas numéricos.



Caracterização

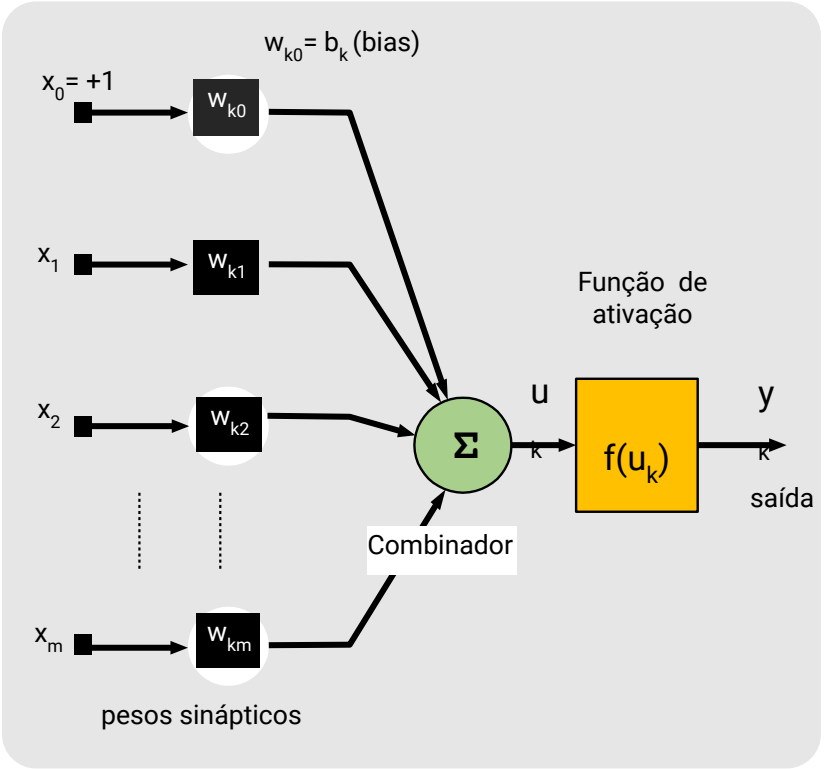
- Rede de **camada única**, como Hopfield;
- Redes neurais **multicamadas** com retropropagação, como backpropagation padrão;
- Redes neurais **temporais**, como redes neurais recorrentes;
- Redes neurais **auto organizadas**, como Kohonen;
- Redes neurais **supervisionadas** e **não supervisionadas**, como redes de funções de base radial;

Caracterização - Neurônio artificial



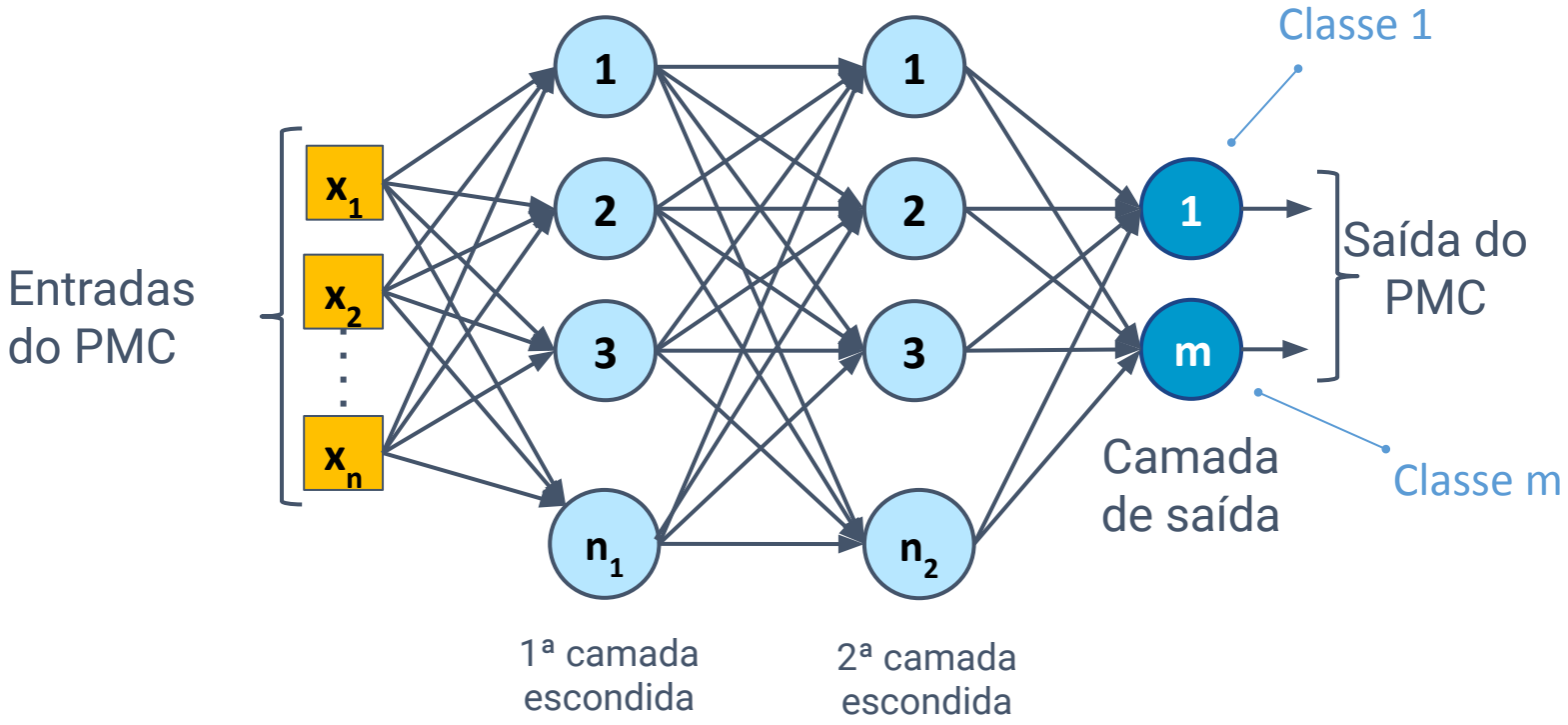
Funcionamento - Neurônio artificial

- Considerando um **neurônio k:**
- **Sinais** são apresentados à entrada **(x_1 a x_m)**;
- Cada sinal é **multiplicado por um peso (w_k)** que indica **influência do sinal** na saída;
- É feita a **soma ponderada** dos sinais produzindo o potencial de ativação (**u_k**);
- O potencial de ativação é submetido à **Função de Ativação** que produz a saída **y_k**;
- Processo conhecido como propagação e é usado tanto para **treino** quando para **inferência**.



Treinamento - Perceptron e Adaline

Funcionamento - Perceptron Multicamadas



Funcionamento - Perceptron Multicamadas

- Uma rede neural artificial composta por pelo menos uma camada de neurônios escondidos e uma camada com neurônios de saída;
- Utiliza regra delta generalizada no processo de treinamento;
- Possui uma fase de propagação, na qual se projetam os dados da camada entrada até a camada de saída;
- Possui uma fase de retropropagação, na qual se atualizam os pesos com base na regra delta.

Exemplo de código

```python
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.neural_network import MLPClassifier
from sklearn.metrics import accuracy_score, confusion_matrix

X = pd.read_csv("diabetes.csv")
```

```python
y = X.pop('Outcome').values
```

```python
X_train, X_test, y_train, y_test = train_test_split(X, y, stratify=y,
                                                    random_state=1)
```

```python
clf = MLPClassifier(random_state=1, max_iter=300).fit(X_train, y_train)
y_pred = clf.predict(X_test)
```

```python
print("Acurácia do modelo: ", accuracy_score(y_test,y_pred))
print("Matriz de confusão: \n", confusion_matrix(y_test,y_pred))
```

```
Acurácia do modelo:  0.640625
Matriz de confusão:
 [[108  17]
 [ 52  15]]
```

Principais parâmetros

- **hidden_layer_sizes -** número de neurônios por camada, em formato de tupla (60,120);
- **activation : *{'identity', 'logistic', 'tanh', 'relu'}, default='relu'} -**função de ativação;
- **solver: *{'lbfgs', 'sgd', 'adam'}, default='adam'* -** tipo de otimizador, algoritmo de treinamento;
- **learning_rate: *{'constant', 'invscaling', 'adaptive'},*** default='constant' - tipo da taxa de aprendizagem;
- **learning_rate_init: *double, default=0.001*** - valor de início da taxa de aprendizagem***;***
- **max_iter: *int, default=200* -** quantidade máxima de épocas;

Modificando alguns parâmetros

```
1 import pandas as pd
2 from sklearn.model_selection import train_test_split
3 from sklearn.neural_network import MLPClassifier
4 from sklearn.metrics import accuracy_score, confusion_matrix
5
6 X = pd.read_csv("diabetes.csv")
```

```
1 y = X.pop('Outcome').values
```

```
1 X_train, X_test, y_train, y_test = train_test_split(X, y, stratify=y,
2                                                     random_state=1)
```

```
1 clf = MLPClassifier(random_state=1,
2                     max_iter=3000,
3                     hidden_layer_sizes=(50),
4                     activation='tanh',
5                     learning_rate = 'adaptive',
6                     learning_rate_init = .001
7                     ).fit(X_train, y_train)
8 y_pred = clf.predict(X_test)
```

```
1 print("Acurácia do modelo: ", accuracy_score(y_test,y_pred))
2 print("Matriz de confusão: \n", confusion_matrix(y_test,y_pred))
```

```
Acurácia do modelo:  0.7135416666666666
Matriz de confusão:
 [[111  14]
 [ 41  26]]
```

Exercício

- Resolva o problema de classificação de diabetes com um MLP;
- Utilize o recurso `sklearn.preprocessing`.MinMaxScaler para normalizar os dados de entrada;
- Execute novos experimentos e veja se o desempenho da rede melhorou.

```
from sklearn.preprocessing import Normalizer


scaler = MinMaxScaler()
X_scaled = pd.DataFrame(scaler.fit_transform(X), columns=X.columns)
```

Salvando e carregando modelos

- Em produção, deseja-se obter um modelo e a partir de então executa-se várias inferências com este;
- Todos os frameworks disponibilizam formas de se salvar e posteriormente carregar um modelo treinado;
- No sklearn, podem ser usados arquivos .pickle ou .joblib, através dos métodos dump e load

Salvando e carregando modelos

pickle

```
model = LogisticRegression(max_iter=5000)
model.fit(x_train,y_train_labels)
pickle.dump(model, open("modelo_salvo.pkl", 'wb'))
```

```
loaded_model = load(open("modelo_salvo.pkl", 'rb'))
preds = loaded_model.predict(x_test)
```

joblib

```
model = LogisticRegression(max_iter=500)
model.fit(x_train,y_train_labels)
dump(model, "modelo_salvo.joblib")
```

```
loaded_model = load("modelo_salvo.joblib")
preds = loaded_model.predict(x_test)
```

Obrigado.