
Aprendizagem Baseada em Instâncias

Fabrício Barth

Setembro de 2019

Sumário

- Problema (**Blue Flag Iris**)
- Espaço Euclidiano
- Aprendizagem Baseada em Instâncias (Modelos Baseados em Distâncias)
- Regra kNN (**k** vizinhos mais próximos)

Problema

Blue Flag Iris



- Considere uma base de dados sobre um determinado tipo de flor.
- Esta base de dados possui informações sobre o **comprimento** e **largura** das **sépalas** e das **pétalas** de várias flores parecidas (todas azuis).

-
- A *Blue Flag Iris* é classificada em três tipos:
 - ★ Iris Setosa.
 - ★ Iris Versicolor.
 - ★ Iris Virginica.

Blue Flag Iris - Dados

```
1  @ATTRIBUTE sepallength  REAL
2  @ATTRIBUTE sepalwidth  REAL
3  @ATTRIBUTE petallength  REAL
4  @ATTRIBUTE petalwidth  REAL
5  @ATTRIBUTE class        {Iris-setosa,Iris-versicolor,
6                           Iris-virginica}
7  @DATA
8  5.1,3.5,1.4,0.2,Iris-setosa
9  4.9,3.0,1.4,0.2,Iris-setosa
10 4.7,3.2,1.3,0.2,Iris-versicolor
11 5.0,3.6,1.4,0.2,Iris-versicolor
12 6.6,2.9,4.6,1.3,Iris-virginica
13 5.2,2.7,3.9,1.4,Iris-virginica
```

Todas as medidas são em cm.

Blue Flag Iris - Problema

- O que faz uma *Blue Flag Iris* ser do tipo *Iris Setosa*, *Iris Versicolor* ou *Iris Virginica*?
- Como extrair esta informação a partir dos dados existentes?

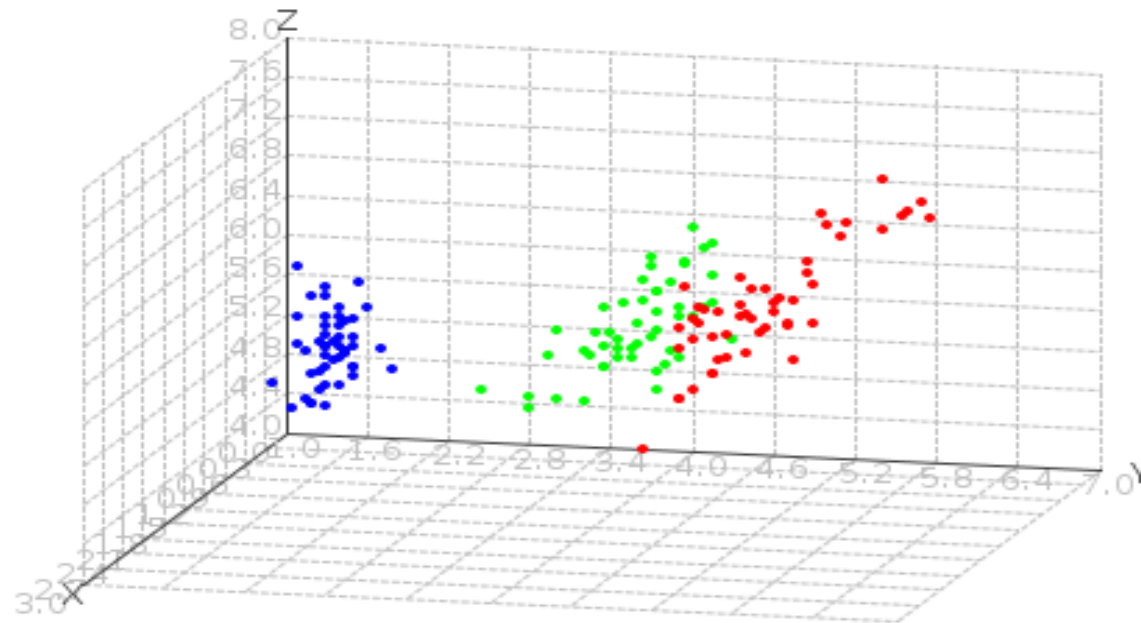
Aplicando uma abordagem baseada em instâncias

- Muitos métodos de aprendizagem constroem uma descrição geral e explícita da função alvo a partir de exemplos de treinamento.
- Os métodos de aprendizagem baseados em instâncias simplesmente **armazenam** os exemplos de treinamento.

-
- A generalização é feita somente quando uma nova instância é classificada.
 - Métodos de aprendizagem baseados em instâncias assumem que as instâncias podem ser representadas como pontos em um **espaço euclidiano**.

Espaço Euclidiano

■ Iris-setosa ■ Iris-versicolor ■ Iris-virginica



x = Petal Width, y = Petal Length, z = Sepal Length.

Espaço Euclidiano

- (Petal Width, Petal Length) **2-dimensional**
- (Petal Width, Petal Length, Sepal Length)
3-dimensional
- (Petal Width, Petal Length, Sepal Length, Sepal Width) **4-dimensional**

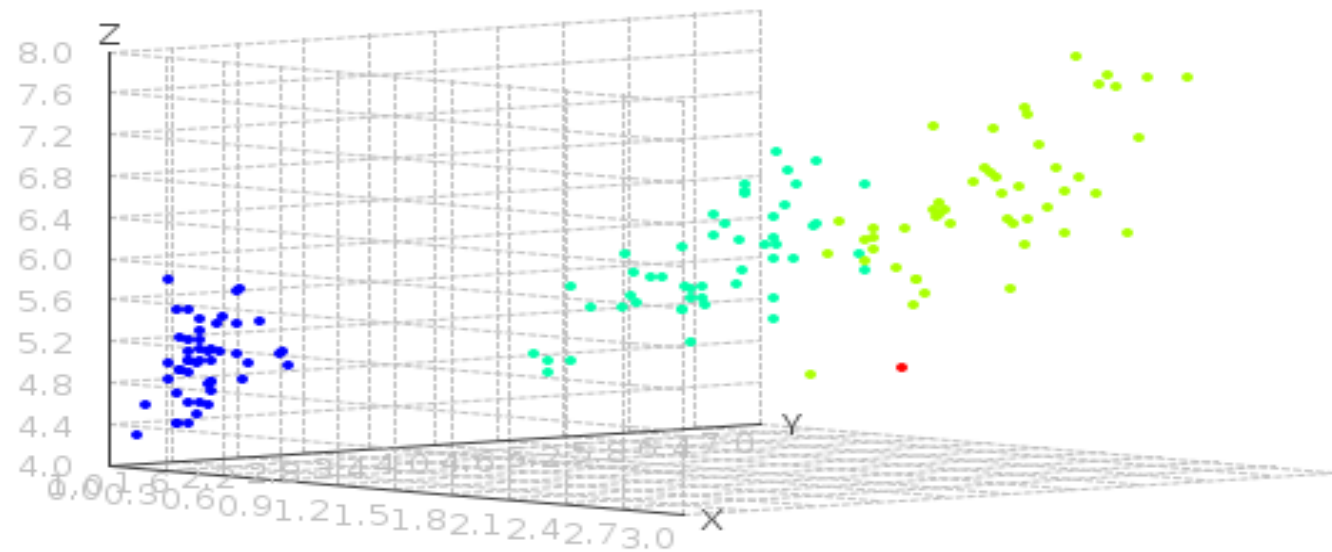
Aprendizagem Baseada em Instâncias

- A aprendizagem consiste somente em armazenar os exemplos de treinamento.
- Após a aprendizagem, para encontrar o valor do conceito alvo associado a uma nova instância, um conjunto de instâncias similares são buscadas na memória e utilizadas para classificar a nova instância.

-
- No final, teremos um conjunto de distâncias (medida de similaridade) entre a nova instância e todos os exemplos de treinamento.
 - Qual o valor do conceito alvo (classe) atribuímos à nova instância? **O conceito alvo associado ao exemplo de treinamento mais similar !!**

Exemplo com nova instância

■ Iris-setosa ■ Iris-versicolor ■ Iris-virginica ■ nova-instancia



x = Petal Width, y = Petal Length, z = Sepal Length.

Aprendizagem k -NN

- k -NN = *K Nearest Neighbor* = k vizinhos mais próximos.
- O algoritmo k -NN é o método de aprendizagem baseado em instâncias mais elementar.

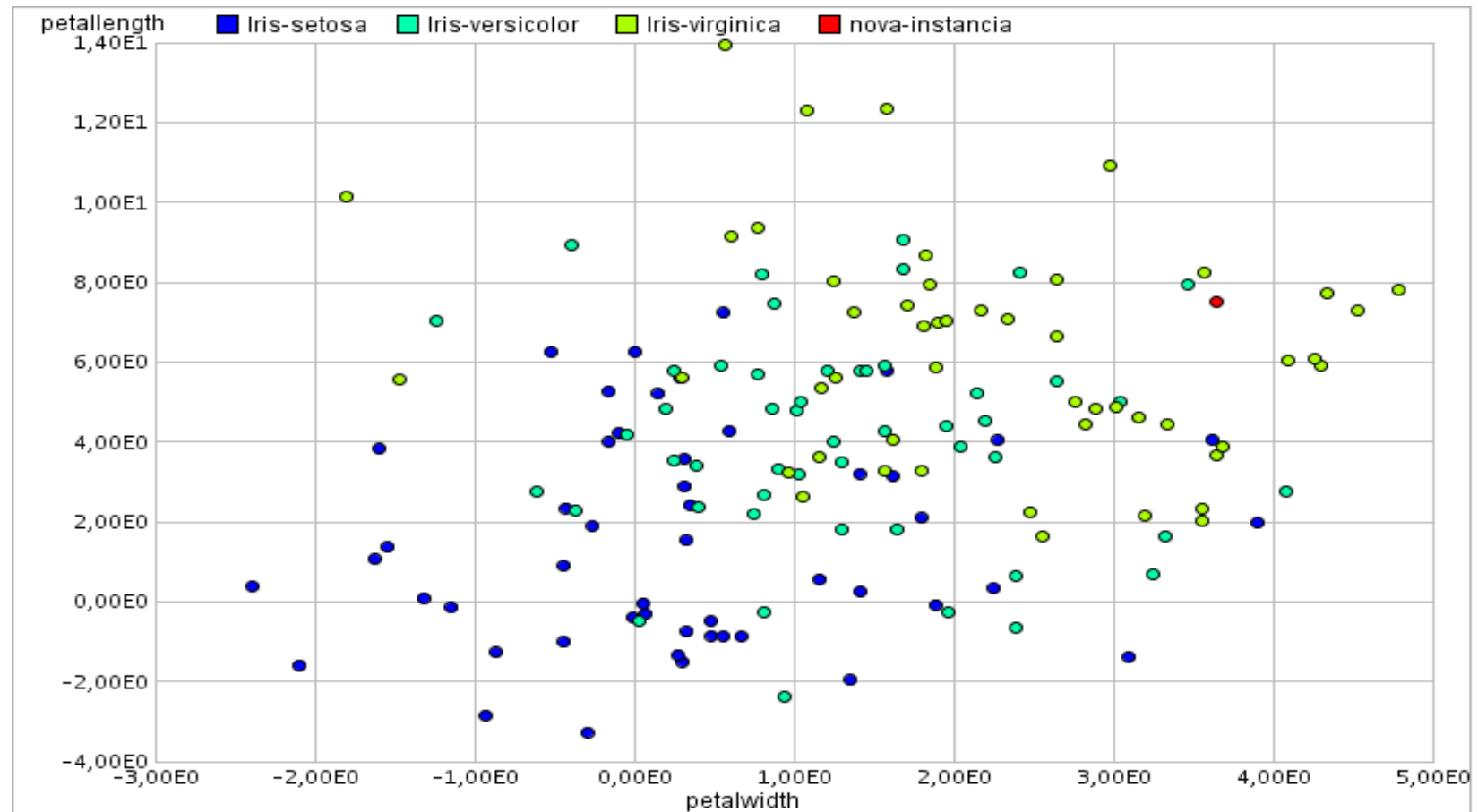
-
- O algoritmo k -NN assume que todas as instâncias correspondem a pontos em um espaço n -dimensional (\mathbb{R}^n).
 - Os "vizinhos mais próximos" de uma instância são definidos em termos da distância Euclidiana.

$$| \vec{x} - \vec{y} | = \sqrt{\sum_{i=1}^n (x_i - y_i)^2} \quad (1)$$

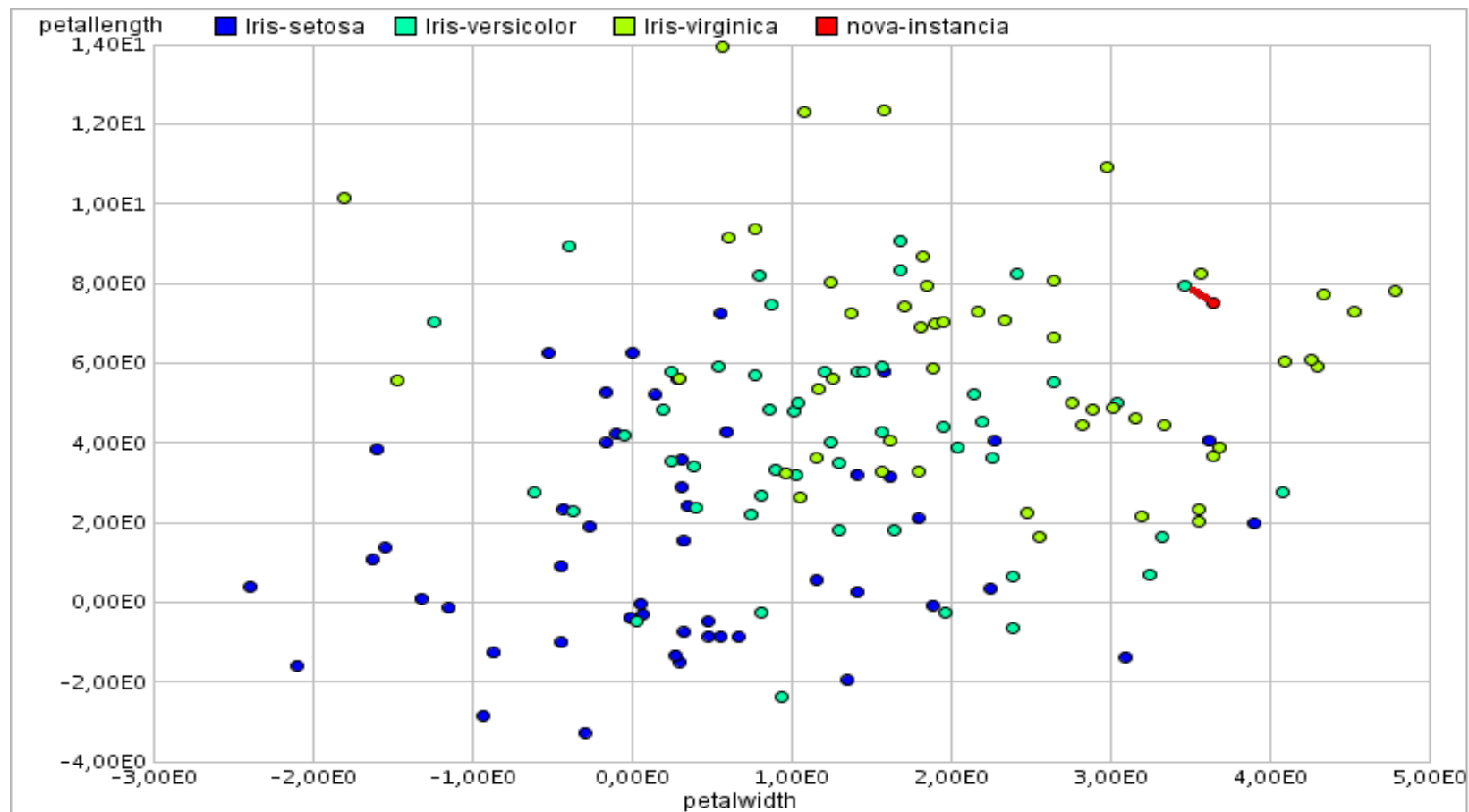
Aprendizagem k -NN

- **A regra dos vizinhos mais próximos:**
- Classificar a nova instância, atribuindo a ela o rótulo mais freqüente entre as k amostras mais próximas.

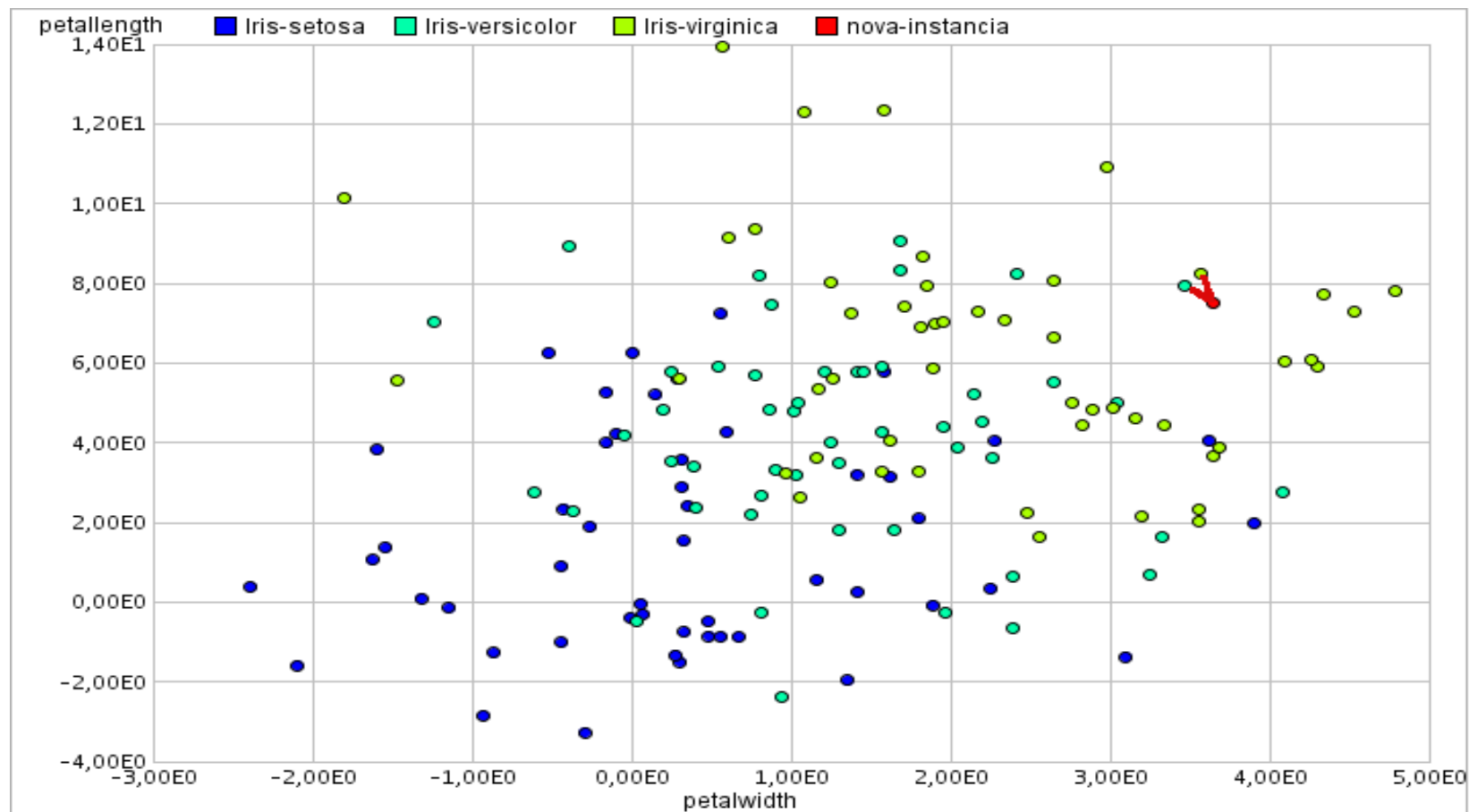
Classificando a nova instância



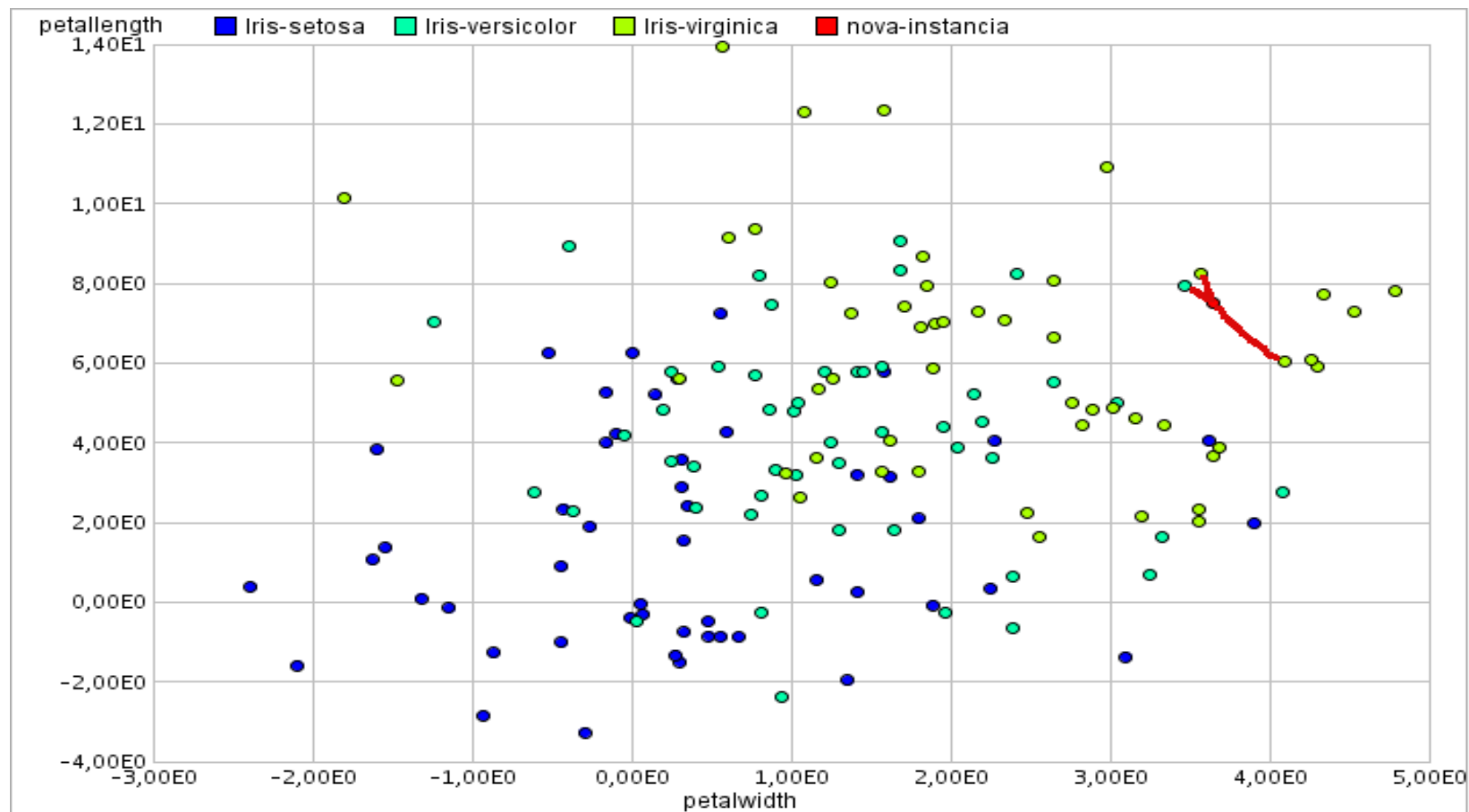
Classificando a nova instância (k=1)



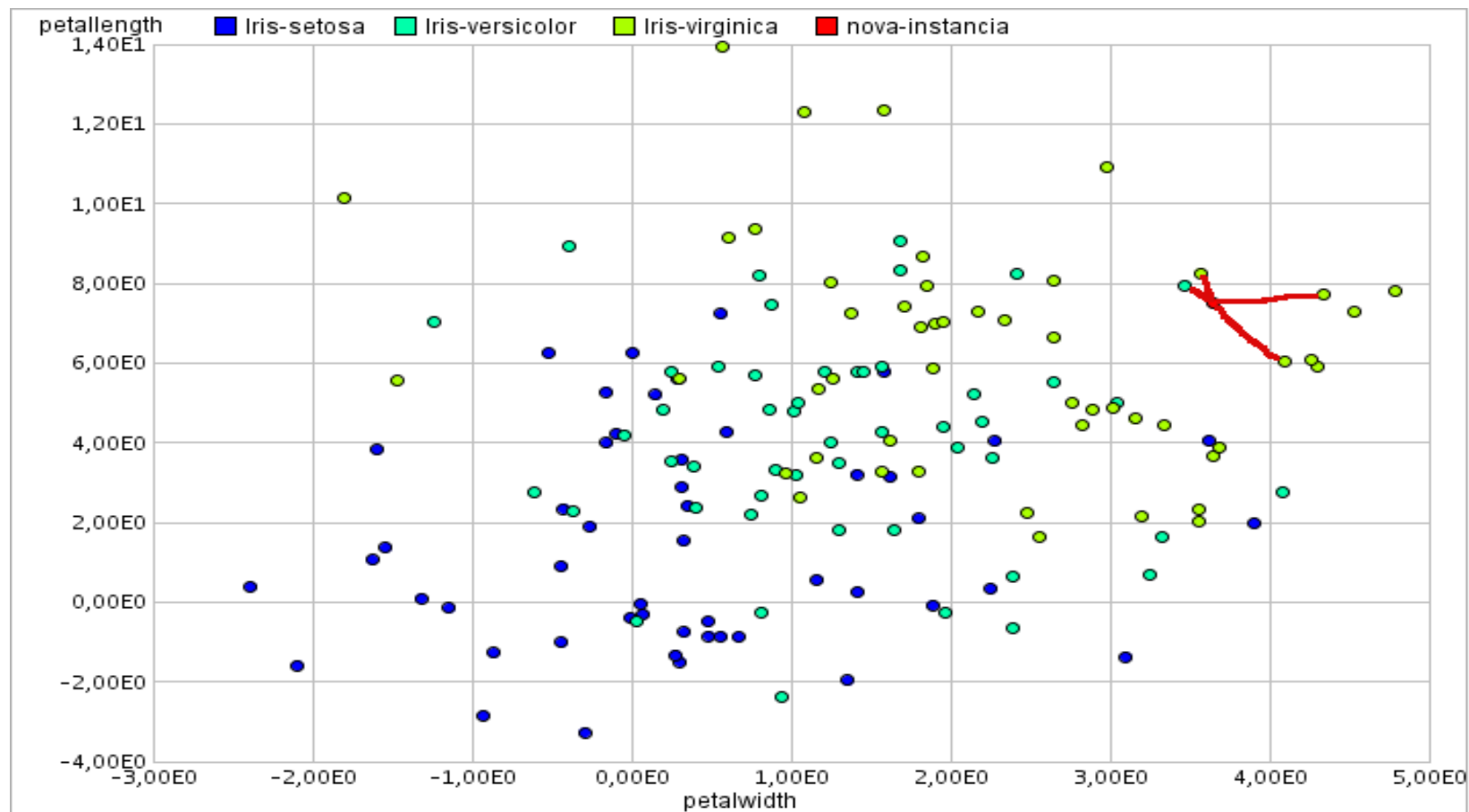
Classificando a nova instância (k=2)



Classificando a nova instância (k=3)



Classificando a nova instância (k=4)



Exemplo k -NN

- $k=3$ (valor ímpar) e $e_i = (0.10, 0.25)$
- Exemplos de treinamento:
 - ★ $(0.15, 0.35, c_1)$
 - ★ $(0.10, 0.28, c_2)$
 - ★ $(0.09, 0.30, c_5)$
 - ★ $(0.12, 0.20, c_2)$

-
- Os vetores mais próximos a e_i , com suas classes, são:
 - ★ $(0.10, 0.28, c_2)$
 - ★ $(0.12, 0.20, c_2)$
 - ★ $(0.15, 0.35, c_1)$
 - Uma votação atribui a classe c_2 a e_i , pois c_2 é a classe representada com mais frequência.

Knn no Python

```
from sklearn.neighbors import KNeighborsClassifier

knn = KNeighborsClassifier(n_neighbors = k)
knn.fit(X_train, y_train)
y_pred = knn.predict(X_test)
```

Como escolher o melhor k ?

- Escolher o valor de k é crítico.
- Um k muito pequeno resulta em uma solução que não tolera ruído.
- Um k muito grande vai contra a filosofia do **KNN**.
- Regra genérica para escolha de k :

$$k = n^{(1/2)} \quad (2)$$

-
- Executar a primeira validação com

$$k = n^{(1/2)} \quad (3)$$

- Medir a acurácia e executar outra validação com $k + 1$ e assim sucessivamente até a nova acurácia não for melhor que a acurácia anterior.

Outra forma para encontrar K

```
from sklearn.neighbors import KNeighborsClassifier
from sklearn import metrics
```

```
k_range = range(1,15)
```

```
scores = {}
```

```
scores_list = []
```

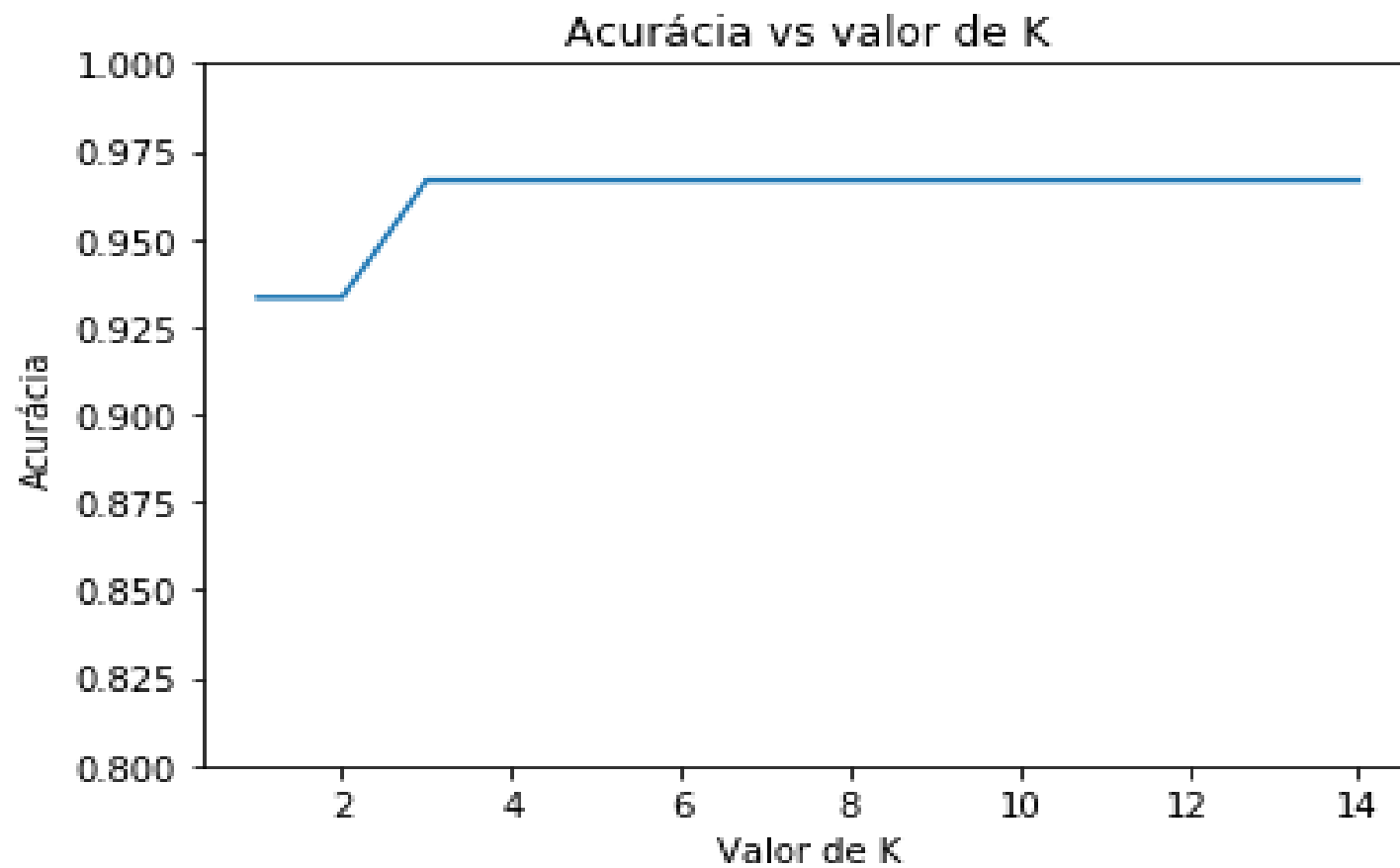
```
for k in k_range:
```

```
    knn = KNeighborsClassifier(n_neighbors = k)
```

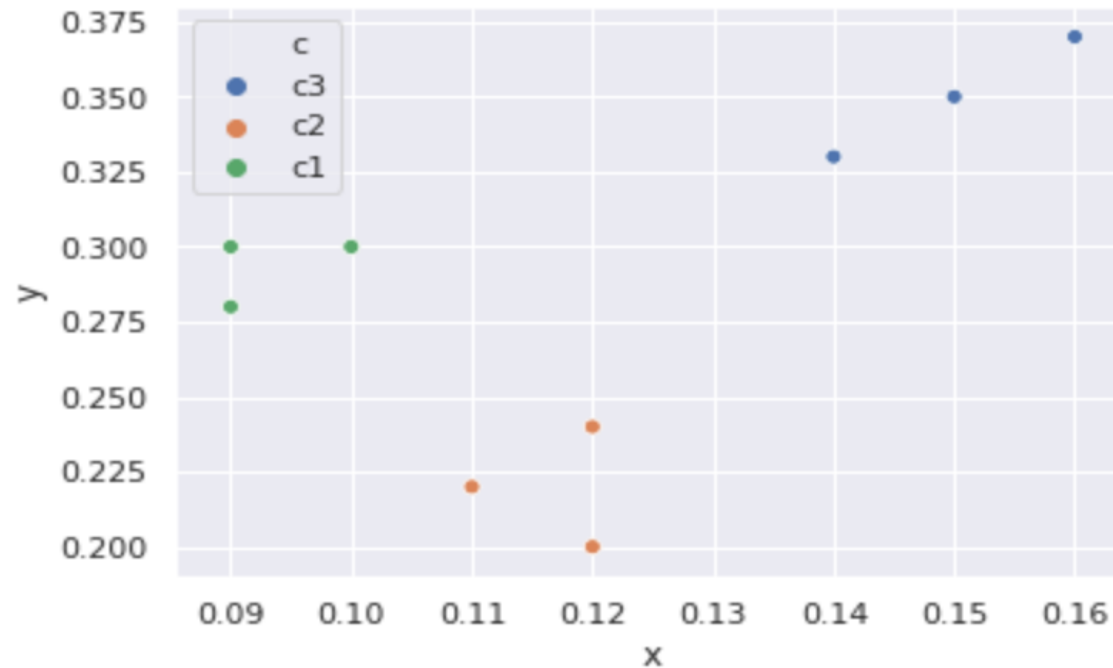
```
    knn.fit(X_train, y_train)
```

```
    y_pred = knn.predict(X_test)
```

```
scores[k] = metrics.accuracy_score(y_test ,  
                                     y_pred)  
scores_list.append(scores[k])
```

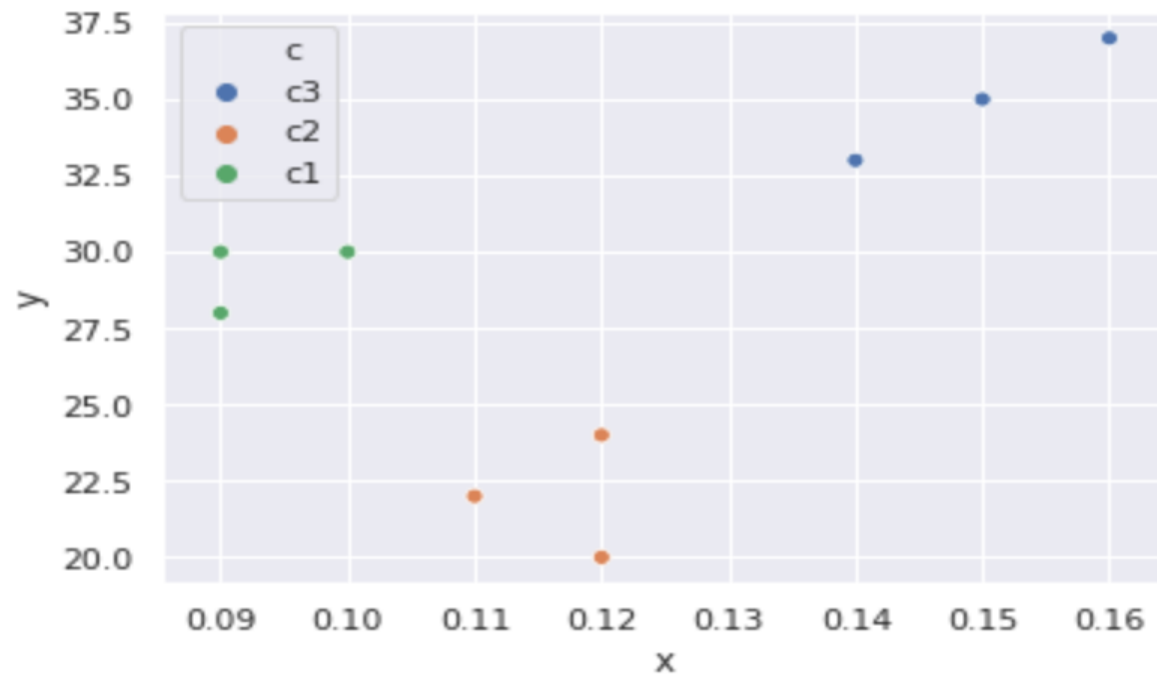


Cálculo da distância e identificação do exemplo mais próximo



```
new_point = [[0.14, 0.3]]
from sklearn.neighbors import NearestNeighbors
neigh = NearestNeighbors(n_neighbors=1)
neigh.fit(df[['x', 'y']], df['c'])
print(neigh.kneighbors(new_point))

(array([[0.03]]), array([[7]]))
```

```
new_point = [[0.14, 30]]

neigh = NearestNeighbors(n_neighbors=1)
neigh.fit(df[['x', 'y']], df['c'])
print(neigh.kneighbors(new_point))

(array([[0.04]]), array([[4]]))
```

Cuidado: normalizar dados

- Objetivo: Manter todos os valores entre 0 e 1.
- Ao utilizar o pacote *sklearn.preprocessing.MinMaxScaler*, a transformação implementada é:

```
X_std = (X - X.min(axis=0)) / (X.max(axis=0) - X.min(axis=0))  
X_scaled = X_std * (max - min) + min
```

onde, *max* e *min* são fornecidos por parâmetro. Os valores *default* são 1 e 0, respectivamente.

- Ver https://github.com/fbarth/ml-espmm/blob/master/scripts/python/04_02_knn.ipynb

Knn no R

```
data(iris)

normalize <- function(x){
  return ((x - min(x)) / (max(x) - min(x)))
}

iris_norm <- as.data.frame(lapply(iris[1:4], normalize))
summary(iris_norm)

set.seed(1234)
ind <- sample(2, nrow(iris), replace=TRUE, prob=c(0.67, 0.33))

iris.training <- iris_norm[ind==1, 1:4]
iris.test <- iris_norm[ind==2, 1:4]

iris.trainLabels <- iris[ind==1, 5]
iris.testLabels <- iris[ind==2, 5]

library(class)
iris_pred <- knn(train = iris.training, test = iris.test, cl = iris.trainLabels, k=3)

t <- table(iris_pred, iris.testLabels)
t
```

Considerações

- **Vantagens:**

- ★ A informação presente nos exemplos de treinamento nunca é perdida.

- **Desvantagens:**

- ★ Toda a computação ocorre no momento da classificação!!!
- ★ A computação aumenta com a quantidade de exemplos de treinamento.

Referências

- https://github.com/fbarth/ml-espm/blob/master/scripts/python/04_01_knn.ipynb
- https://github.com/fbarth/ml-espm/blob/master/scripts/python/04_02_knn.ipynb

References

[Mitchell, 1997] Mitchell, T. M. (1997). *Machine Learning*.
McGraw-Hill.