HUB IQ

Inteligencia Artificial & Machine Learning

Applicaciones en movilidad

Dr. Iván S. Razo Zapata

EngineeringX
Founded by the Royal Academy of Engineering and Lloyd's Register Foundation

Aprendizaje Supervisado

Random Forests

Métodos de ensamble - (ensemble methods)

Idea general – Wisdom of the crowd

Idea general

Algunas condiciones

- Los clasificadores son “weak classifiers” … pero
  - Tienen un desempeño “aceptable”, i.e. errores menores al 50%
  - Deben ser diversos, i.e. cometen diferentes errores (son independientes)

Métodos de ensamble comunes

- Bagging (voto mayoritario)
  - Random forest
- Boosting (voto ponderado)

Bagging - Bootstrap aggregation

- La idea básica es extraer conjuntos de datos de manera aleatoria y con reemplazo del conjunto de entrenamiento, cada muestra del mismo tamaño que el conjunto de entrenamiento original.
- D = {A,B,C,D,E,F}
  - D1 = {A,B,A,D,E,F}
  - D2 = {A,B,C,B,E,F}
- Entrenar un modelo en cada muestra

Código

Bagging - Bootstrap aggregation

```
from sklearn.ensemble import BaggingClassifier
from sklearn.neighbors import KNeighborsClassifier
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import train_test_split
from sklearn import datasets
import numpy as np

np.random.seed(2)

X, y = datasets.load_wine(return_X_y=True)

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2)
# Scaling data
scaler = StandardScaler()
scaler.fit(X_train)
X_train = scaler.transform(X_train)
X_test = scaler.transform(X_test)

n_neighbors = 5
modelo = KNeighborsClassifier(n_neighbors)
modelo.fit(X_train,y_train)

# Bagging
modeloB = BaggingClassifier(KNeighborsClassifier(n_neighbors), n_estimators=50)
modeloB.fit(X_train,y_train)

print(modelo.score(X_test,y_test))
print(modeloB.score(X_test,y_test))
```

Random forest

- Árboles CART
- Como con bagging, los árboles son entrenados con diferentes datasets
- A diferencia de bagging, para cada split se considera un subconjunto **m** del total de atributos **p**
  - **m < p**

Random forest

- ¿Cuántos arboles?
  - Sky is the limit
  - Computing power is the limit

- ¿Qué tamaño para m?

$$m = \sqrt{p}$$

Random forest

Algorithm 15.1 *Random Forest for Regression or Classification.*

1. For $b = 1$ to B:

   (a) Draw a bootstrap sample $\mathbf{Z}^*$ of size N from the training data.

   (b) Grow a random-forest tree T_b to the bootstrapped data, by recursively repeating the following steps for each terminal node of the tree, until the minimum node size n_{min} is reached.

      i. Select m variables at random from the p variables.

      ii. Pick the best variable/split-point among the m.

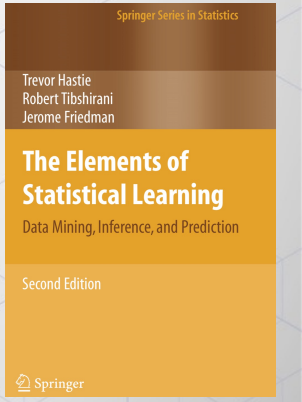
      iii. Split the node into two daughter nodes.

2. Output the ensemble of trees $\{T_b\}_1^B$.

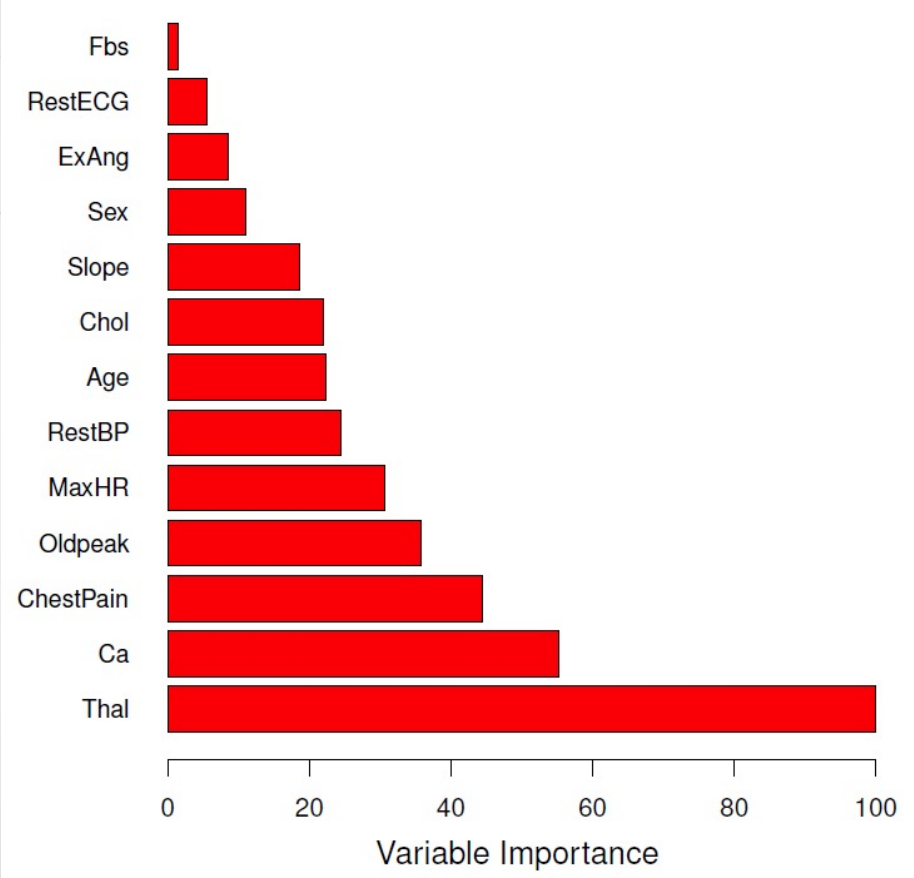
To make a prediction at a new point x:

Regression: $\hat{f}_{\text{rf}}^B(x) = \frac{1}{B}\sum_{b=1}^{B} T_b(x)$.

Classification: Let $\hat{C}_b(x)$ be the class prediction of the bth random-forest tree. Then $\hat{C}_{\text{rf}}^B(x) = \textit{majority vote}\ \{\hat{C}_b(x)\}_1^B$.



Random forest and variable importance

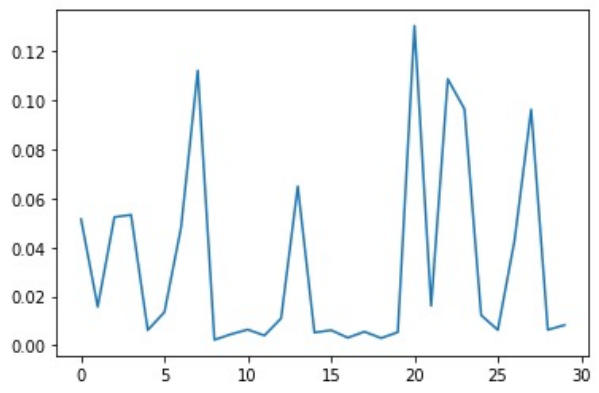


Código

Random forest

```
from sklearn import tree
from sklearn.ensemble import RandomForestClassifier
from sklearn.neighbors import KNeighborsClassifier
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import train_test_split
from sklearn import datasets
import numpy as np

np.random.seed(123)

X, y = datasets.load_breast_cancer(return_X_y=True)

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2)
# Scaling data
scaler = StandardScaler()
scaler.fit(X_train)
X_train = scaler.transform(X_train)
X_test = scaler.transform(X_test)

# K-neighbors
n_neighbors = 5
modelo = KNeighborsClassifier(n_neighbors)
modelo.fit(X_train,y_train)

# Decision tree
modeloDT = tree.DecisionTreeClassifier(criterion='gini')
modeloDT.fit(X_train,y_train)

# Random Forest
modeloRF = RandomForestClassifier(n_estimators=100, max_depth=5)
modeloRF.fit(X_train,y_train)

print("K-neighbors  : ", modelo.score(X_test,y_test))
print("Decision tree: ", modeloDT.score(X_test,y_test))
print("Random forest: ", modeloRF.score(X_test,y_test))
```

```
import matplotlib.pyplot as plt

plt.plot(modeloRF.feature_importances_)
```

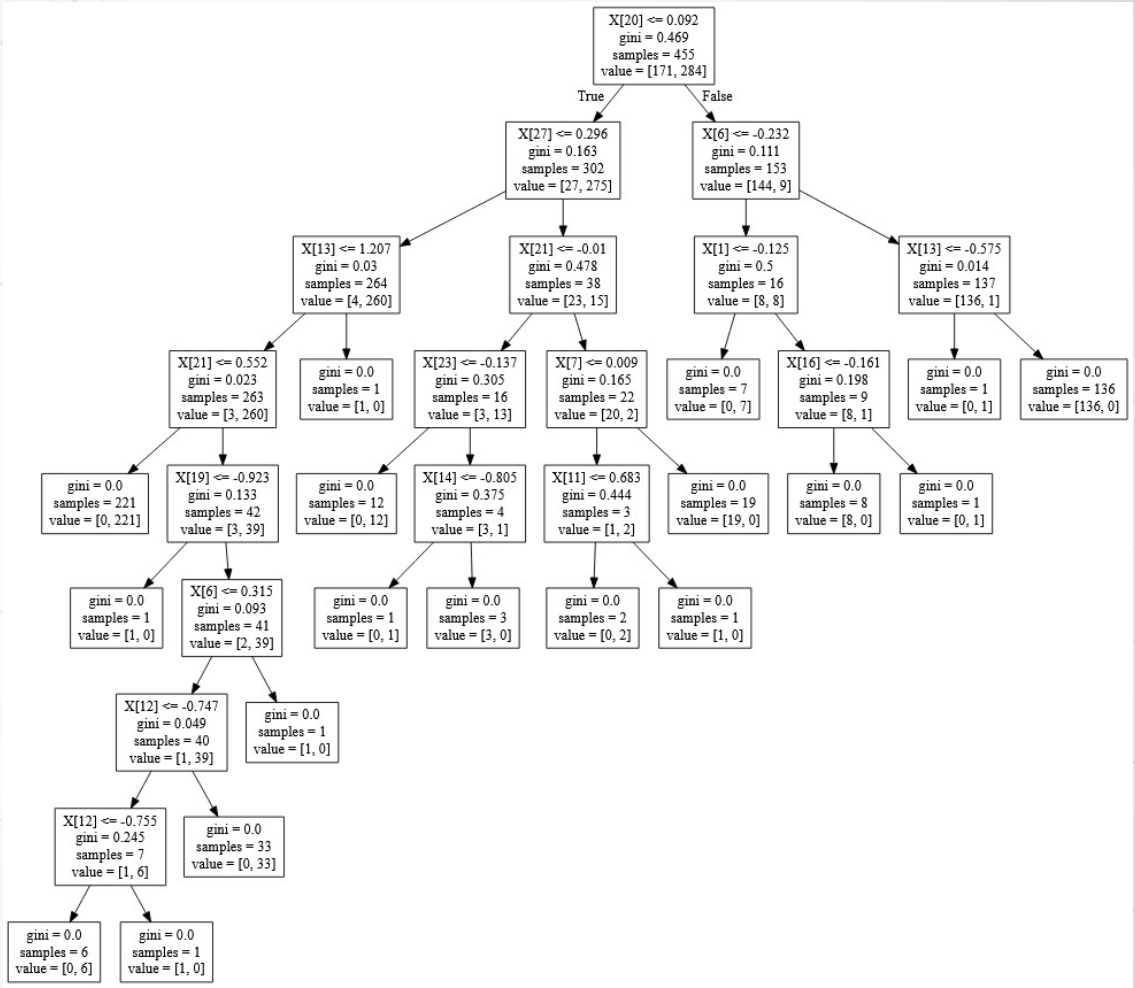


max_features : *{"auto", "sqrt", "log2"}, int or float, default="auto"*

The number of features to consider when looking for the best split:

- If int, then consider `max_features` features at each split.
- If float, then `max_features` is a fraction and `round(max_features * n_features)` features are considered at each split.
- If "auto", then `max_features=sqrt(n_features)`.
- If "sqrt", then `max_features=sqrt(n_features)` (same as "auto").
- If "log2", then `max_features=log2(n_features)`.
- If None, then `max_features=n_features`.

Random forest vs DT



Código

EngineeringX

Founded by the Royal Academy of Engineering
and Lloyd's Register Foundation

GRACIAS

https://hubiq.mx/

HUBIQRO HUBIQ HUBIQRO