T8-Árbol de decisión

Aarón Pérez Ontiveros – A01422524
Miguel Ángel Muñoz Vázquez - A01423629
ITESM CAMPUS CUERNAVACA

Parte 1

Introducción

Un árbol de decisión representa una función que toma como entrada un vector de valores de atributos y regresa una decisión como salida única. Los valores de entrada y salida pueden ser discretos o continuos. Un árbol de decisión alcanza su decisión mediante la realización de una secuencia de pruebas. Cada nodo interno corresponde a la prueba de un valor de los atributos de entrada A_i y las ramas del nodo están etiquetadas con los posibles valores del atributo, $A_i = V_{ik}$ Cada nodo hoja representa el valor retornado por la función. Para la construcción de estos árboles se usa la medida de la entropía.

Árbol de decisión

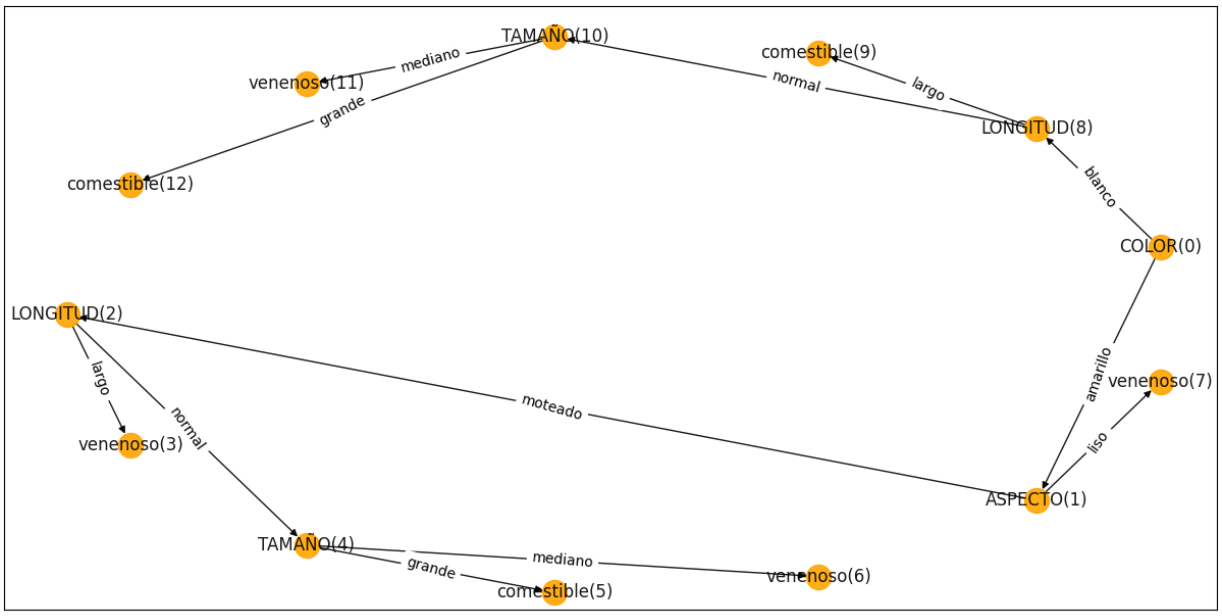
Se implementó en Python el algoritmo de árbol de decisión usando recursión para construir el modelo.

```
def build_model(data,attributes,my_number):
    global node_number
    class_frecuencies = {}
    # class_entropy = 0
    for row in data:
        class_name = row[1]
        if class_name not in class_frecuencies:
            class_frecuencies[class_name]=1
        else:
            class_frecuencies[class_name]+=1
    data_len = len(data)
    if len(class_frecuencies) <= 1:
        return Node(next(iter(class_frecuencies)),[])
    # for k,v in class_frecuencies.items():
    #     Pi = v / data_len
    #     class_entropy -= Pi * log2(Pi)
    best_attribute = float('inf'),''
    for attribute in attributes:
        sum_f = 0
        value_frecuencies,value_to_class_count = {},{}
        for row in data:
            value = row[0][attribute]
            if value not in value_frecuencies:
                value_frecuencies[value]=1
            else:
                value_frecuencies[value]+=1
            if value not in value_to_class_count:
                value_to_class_count[value]={}
            value_class = row[1]
            if value_class not in value_to_class_count[value]:
```

```
                value_to_class_count[value][value_class]=1
            else:
                value_to_class_count[value][value_class]+=1
        antigain = 0
        for k,f in value_frecuencies.items():
            value_entropy = 0
            for frecuency in value_to_class_count[k].values():
                Pi = frecuency  / f
                value_entropy -= Pi * log2(Pi)
            #print('value',k,'frecuency',f,'value_entropy based in class',value_e
ntropy)
            antigain +=  f/data_len * value_entropy
        best_attribute = min((antigain,attribute),best_attribute)
    best_attribute_name = best_attribute[1]
    attributes_copy = deepcopy(attributes)
    attributes_copy.remove(best_attribute_name)
    data_splitted = {}
    for row in data:
        bs_a_value = row[0][best_attribute_name]
        if bs_a_value not in data_splitted:
            data_splitted[bs_a_value]=[]
        row_copy = deepcopy(row)
        row_copy[0].pop(best_attribute_name)
        data_splitted[bs_a_value].append(row_copy)

    node = Node(best_attribute_name,[])
    for value,data in data_splitted.items():
        node_number+=1
        child_number = node_number
        child = build_model(data,deepcopy(attributes_copy),child_number)
        a = f'{node.attribute}({my_number})'
        b = f'{child.attribute}({child_number})'
        edges.append((a,b))
        edges_extended.append(((a,b),value))
        #print(edges)
        node.children.append((child,value))
    return node
```

Ésta es la función recursiva que calcula el mejor atributo y llama recursivamente a sus hijos para cada valor posible del atributo escogido. En el caso de que ya todos los posibles valores pertenezcan a una sola clase se regresa el nodo como hoja.



Como se puede observar, el árbol generado tiene como raíz el atributo COLOR.

Conclusión

El árbol de decisión construido resultó tener 13 nodos en lugar de los 31 nodos que tendría si el árbol se construyera considerando absolutamente todas las posibilidades, logrando así una optimización del 58.06% con respecto a la solución exhaustiva.

Este árbol resultante puede ser utilizado para clasificar cualquier otra instancia en complejidad O(1), debido a que en el peor caso se tienen que bajar 4 niveles para retornar la clase que corresponda.

Bibliografía

Russell, S. and Norvig, P., 2016. *Artificial Intelligence: A Modern Appproach*. 3rd ed. Edinburgh Gate: Pearson Education Limited.

https://scikit-learn.org/stable/modules/naive_bayes.html

Parte 2

Introducción

Existen diferentes tipos de algoritmos útiles para la clasificación de datos. Cada uno de ellos tiene una implementación distinta, pero el objetivo siempre es aproximar la clasificación de las instancias de pruebas a las instancias que se usaron para el modelo de entrenamiento. Las pruebas que se muestran a continuación son con el conjunto de datos de Weka, *Iris.arff*.

Los algoritmos usados en esta comparación de algoritmos son.

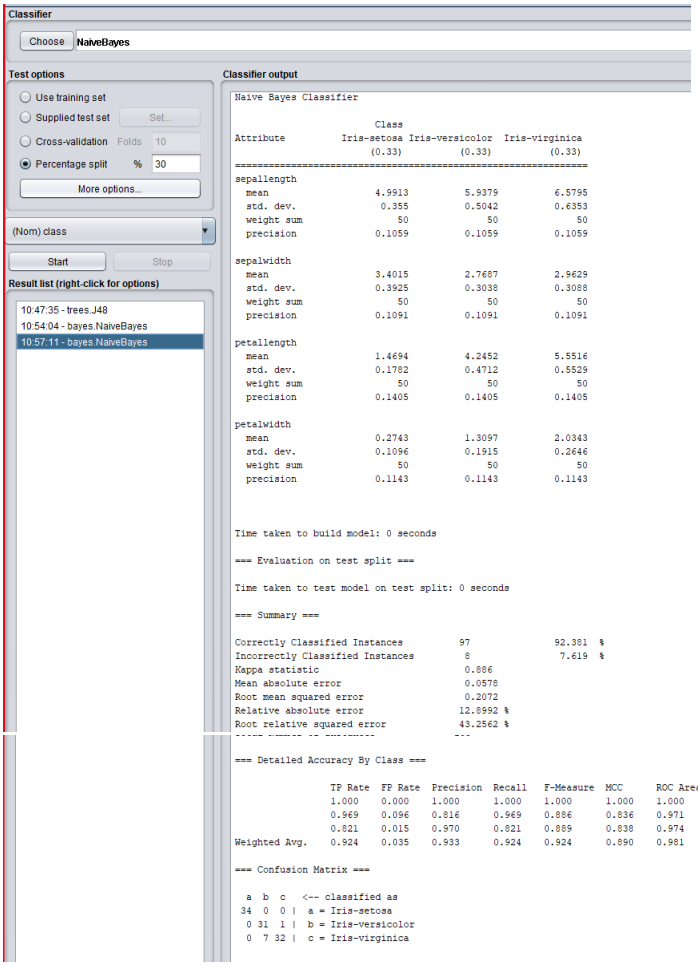
Naive Bayes: Es un clasificador probabilístico fundamentado en el teorema de Bayes y el aprendizaje es supervisado.

J48: Es un clasificador estadístico que genera un árbol de decisión.

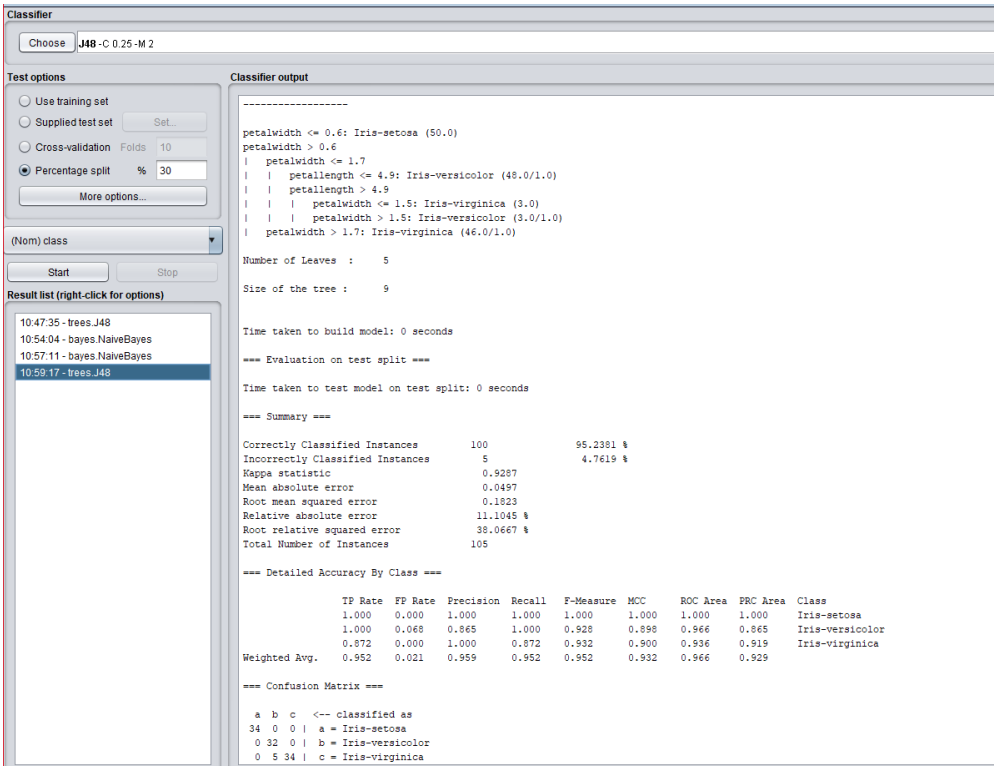
Random Forest: También llamado *random decision forests* es un método de clasificación que consiste en construir varios árboles de decisión durante el tiempo de entrenamiento y generando de salida la clase que es la moda de las clases. Los *random forests* arreglan el problema de *overfitting* que presentan los árboles decisión. Generalmente los *random forests* son mejores que los árboles de decisión.

Iris con 30 % de datos para entrenamiento

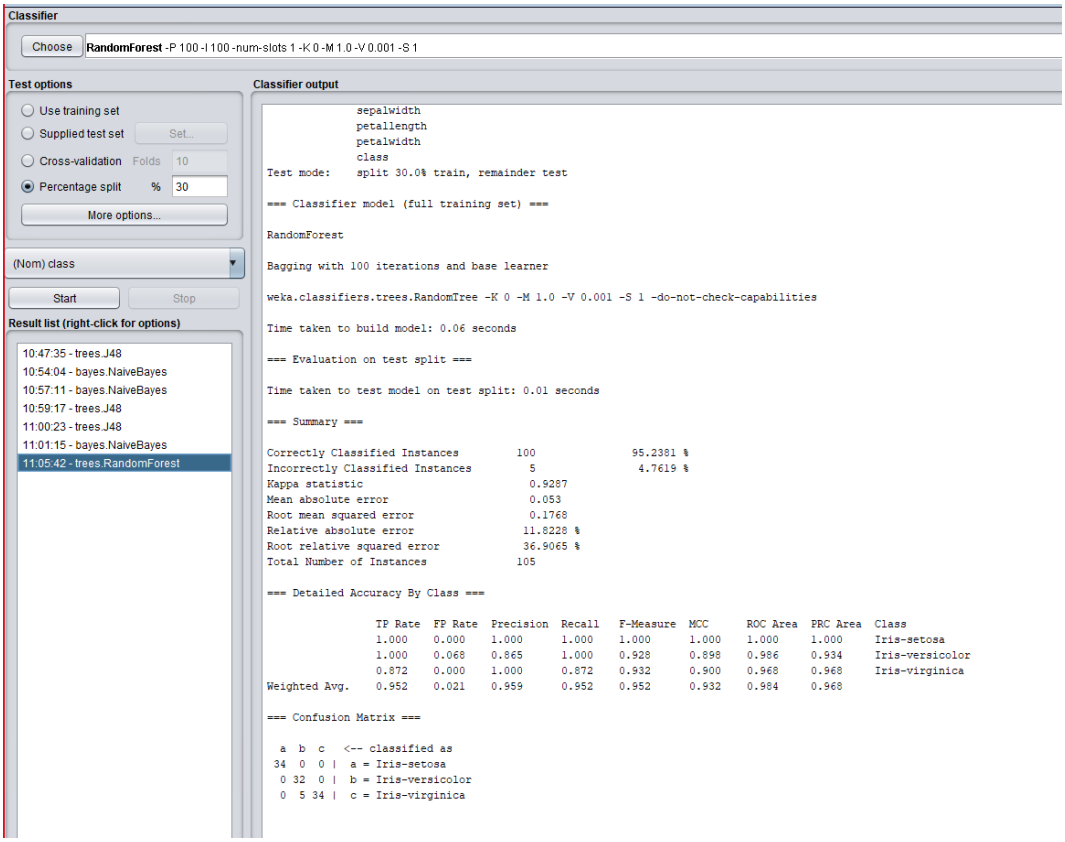
- Naïve Bayes 30 %


Classifier
Choose
NaiveBayes
Test options
Use training set
Supplied test set
Set...
Cross-validation
Folds
10
Percentage split
%
30
More options...
(Nom) class
Start
Stop
Result list (right-click for options)
10:47:35 - trees.J48
10:54:04 - bayes.NaiveBayes
10:57:11 - bayes.NaiveBayes
Classifier output
Naive Bayes Classifier
Class
Attribute Iris-setosa Iris-versicolor Iris-virginica
(0.33) (0.33) (0.33)
sepallength
mean 4.9913 5.9379 6.5795
std. dev. 0.355 0.5042 0.6353
weight sum 50 50 50
precision 0.1059 0.1059 0.1059
sepalwidth
mean 3.4015 2.7687 2.9629
std. dev. 0.3925 0.3038 0.3088
weight sum 50 50 50
precision 0.1091 0.1091 0.1091
petallength
mean 1.4694 4.2452 5.5516
std. dev. 0.1782 0.4712 0.5529
weight sum 50 50 50
precision 0.1405 0.1405 0.1405
petalwidth
mean 0.2743 1.3097 2.0343
std. dev. 0.1096 0.1915 0.2646
weight sum 50 50 50
precision 0.1143 0.1143 0.1143
Time taken to build model: 0 seconds
=== Evaluation on test split ===
Time taken to test model on test split: 0 seconds
=== Summary ===
Correctly Classified Instances 97 92.381 %
Incorrectly Classified Instances 8 7.619 %
Kappa statistic 0.886
Mean absolute error 0.0578
Root mean squared error 0.2072
Relative absolute error 12.8992 %
Root relative squared error 43.2562 %
=== Detailed Accuracy By Class ===
TP Rate FP Rate Precision Recall F-Measure MCC ROC Area PRC Area Class
1.000 0.000 1.000 1.000 1.000 1.000 1.000 1.000 Iris-setosa
0.969 0.096 0.816 0.969 0.886 0.836 0.971 0.907 Iris-versicolor
0.821 0.015 0.970 0.821 0.889 0.838 0.974 0.967 Iris-virginica
Weighted Avg. 0.924 0.035 0.933 0.924 0.924 0.890 0.981 0.960
=== Confusion Matrix ===
a b c <-- classified as
34 0 0 | a = Iris-setosa
0 31 1 | b = Iris-versicolor
0 7 32 | c = Iris-virginica

- J48 30%



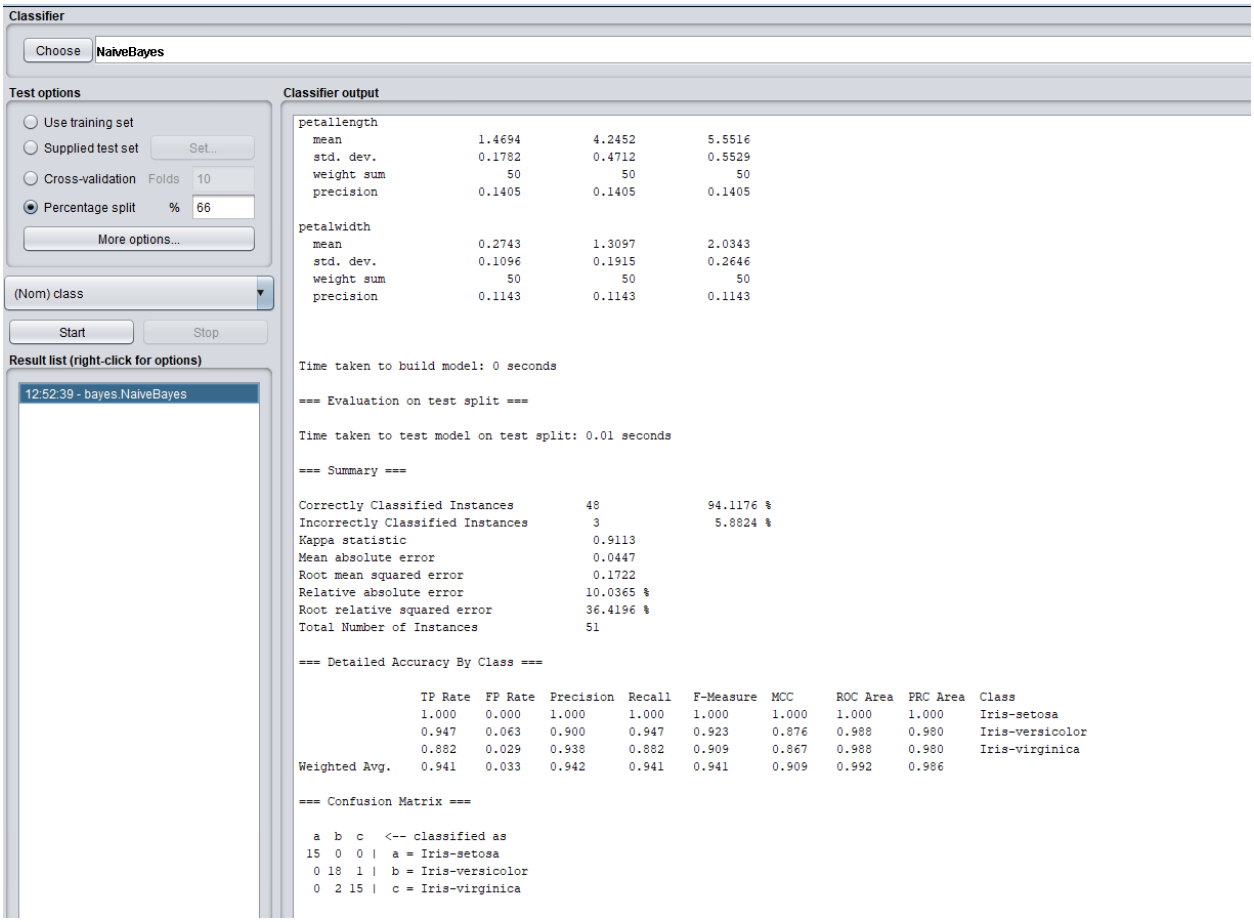
- Random Forest 30 %



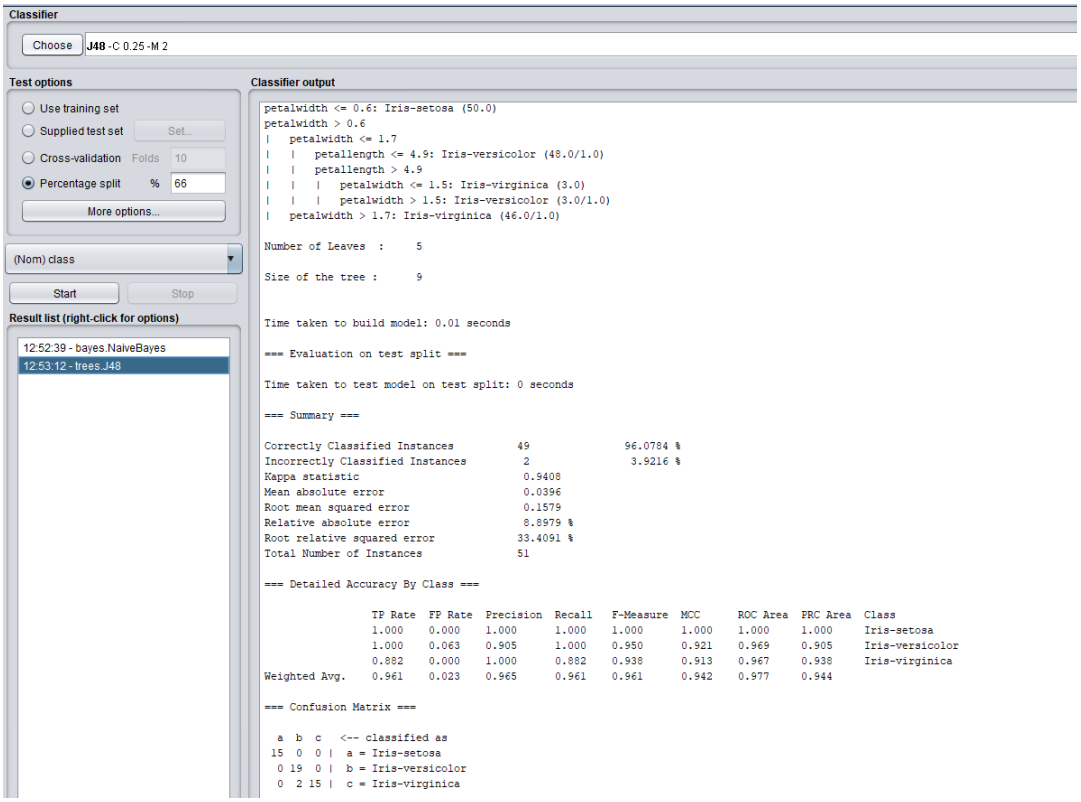
Con el 30% de los datos de entramiento y con los tres diferentes algoritmos podemos observar que Los algoritmos J48 y Random Forest obtienen el mismo resultado, con 105 instancias para pruebas logran acertar con 100 instancias. Naive Bayes acertó solo con 97.

Iris con 66 % de datos para entrenamiento

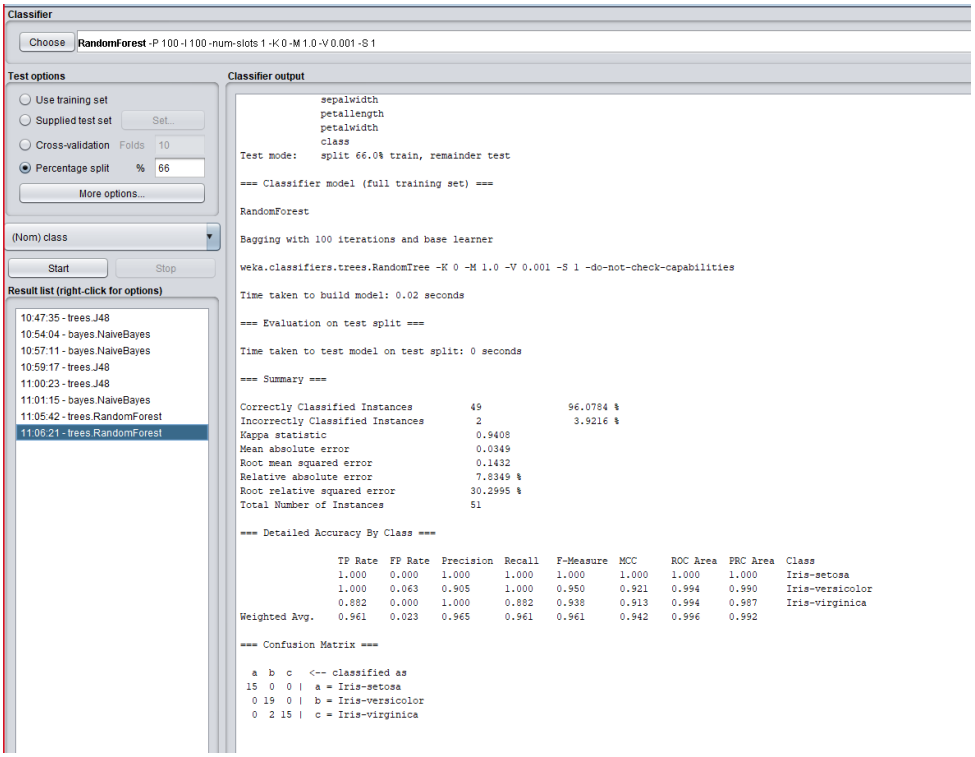
- Naïve Bayes 66%



- J48 66%



- Random Forest 66 %

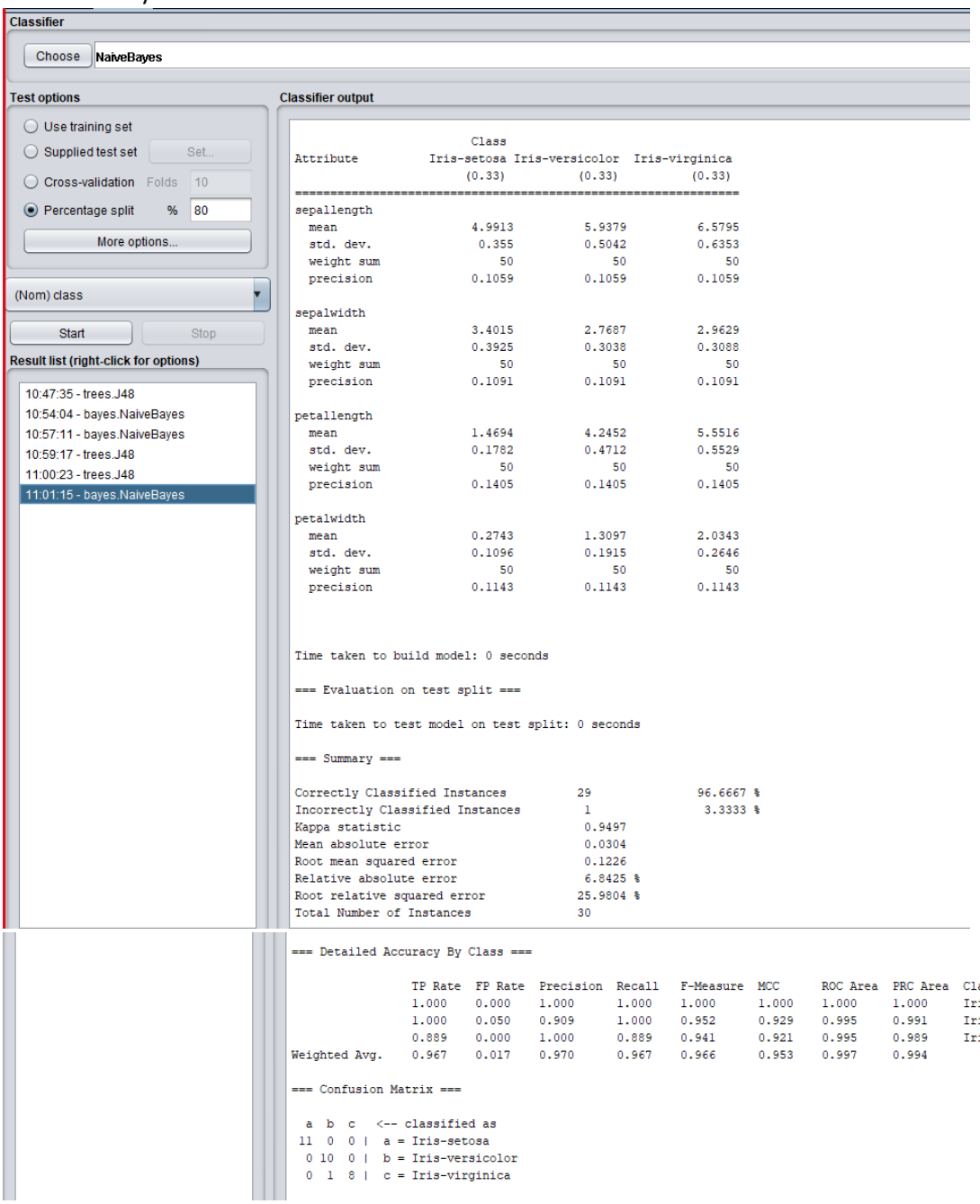


Con el 66% de los datos de entramiento y con los tres diferentes algoritmos podemos observar que Los algoritmos J48 y Random Forest obtienen el mismo resultado. Del conjunto de datos restan 51

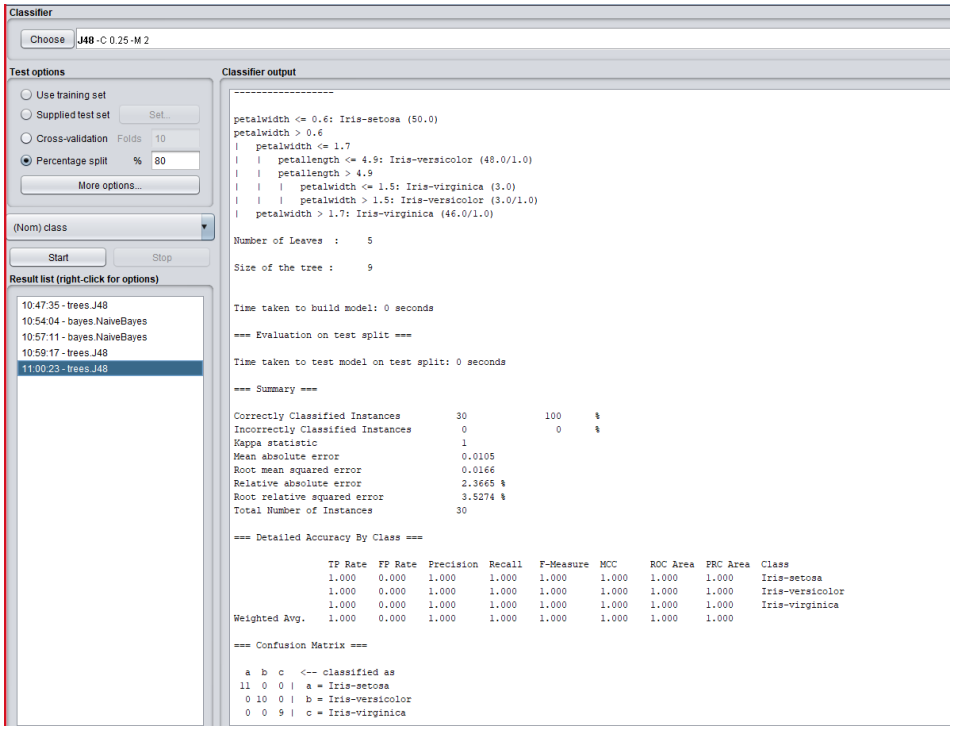
instancias y las pruebas arrojan que 49 instancias fueron clasificadas de forma correcta y 2 de forma incorrecta. Naive Bayes acertó solo con 48.

Iris con 80 % de datos para entrenamiento

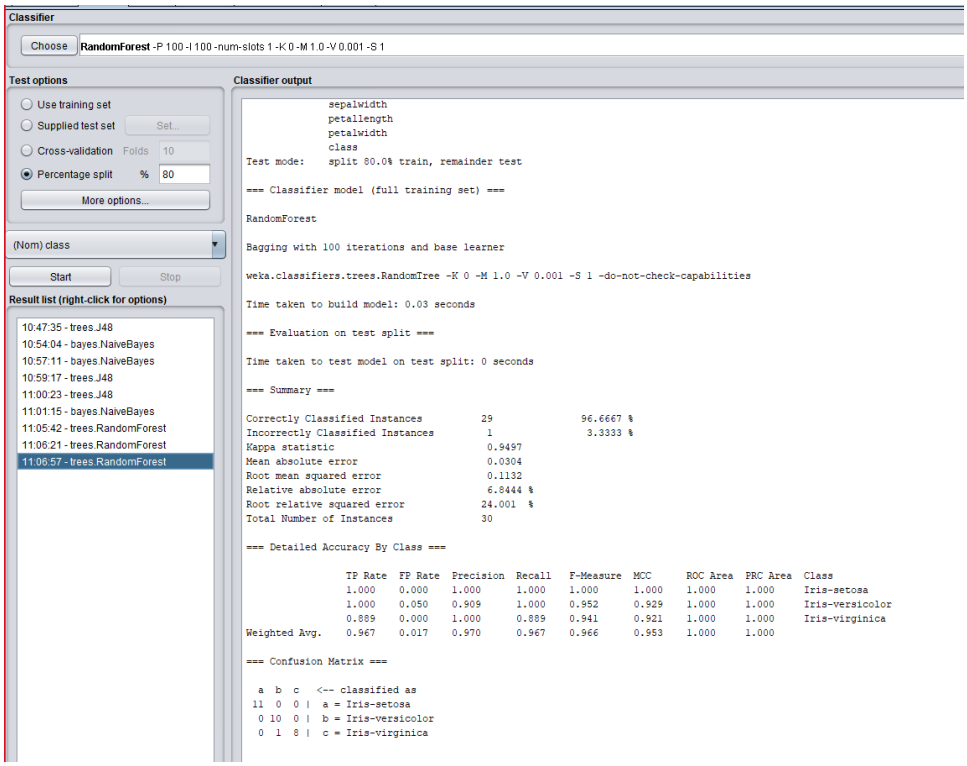
- Naïve Bayes 80 %



- J48 80 %



- Random Forest 80 %



Con el 80% de los datos de entramiento y con los tres diferentes algoritmos podemos observar una diferencia con respecto a los anteriores. La cantidad de instancias para pruebas son solamente 30. La mejor precisión fue con el algoritmo J48, ya que pudo clasificar de forma correcta todas las instancias de prueba. Después, con los algoritmos Random Forest y Naive Bayes se tuvieron 29 clasificaciones correctas y una incorrecta.

Conclusión

En general, los tres algoritmos tuvieron un buen grado de clasificación en todos los casos. Pero podemos destacar que el algoritmo con mejores resultados es el J48. El algoritmo J48, a diferencia de los otros dos, siempre se mantuvo con el mejor porcentaje de clasificación. Probablemente el J48 sea mejor debido a que no son muchos atributos y porque del dominio de cada atributo no es tan grande.

Bibliografía

Russell, S. and Norvig, P., 2016. *Artificial Intelligence: A Modern Appproach*. 3rd ed. Edinburgh Gate: Pearson Education Limited.

https://scikit-learn.org/stable/modules/naive_bayes.html