Universidad de Ingeniería y Tecnología – UTEC

Departamento de Ciencia de la Computación
Prof. Cristian López del Álamo

Teoría y Algoritmos para el Aprendizaje Automático (CS 8113)

2021-II

Desarrollado por:

- Alarcón Delgado, Fernando.
- Juan Carlos Cruz Diaz
- Jhérsin García Solórzano

Colab:

https://colab.research.google.com/drive/1kQ9PILhxeabfJBBwAnicOBkzK7CXqn2f?authuser=1#scrollTo=ju_7oYW6mvMU&uniqifier=2

1. Introducción

El presente trabajo busca clasificar el tipo de flor iris utilizando los siguientes métodos: i) Decision tree, Bagging y Random Forest.

- **Conjunto de datos:** El conjunto de datos utilizado para los experimentos consta de 3 diferentes tipos de la flor iris (Setosa, Versicolor y Virginica), clasificados y etiquetados de acuerdo a 4 características (sepal_length, sepal_width, petal_length y petal width). En la figura a continuación se puede observar parte del conjunto de datos.

```
data.head(5)
```

	sepal_length	sepal_width	petal_length	petal_width	type
0	4.9	3.0	1.4	0.2	Iris-setosa
1	4.7	3.2	1.3	0.2	Iris-setosa
2	4.6	3.1	1.5	0.2	Iris-setosa
3	5.0	3.6	1.4	0.2	Iris-setosa
4	5.4	3.9	1.7	0.4	Iris-setosa

En lo relacionado a la cantidad y tipo de datos, podemos obtener la siguiente información resumen:

```
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 149 entries, 0 to 148
Data columns (total 5 columns):
 #   Column        Non-Null Count  Dtype
---  ------        --------------  -----
 0   sepal_length  149 non-null    float64
 1   sepal_width   149 non-null    float64
 2   petal_length  149 non-null    float64
 3   petal_width   149 non-null    float64
 4   type          149 non-null    object
dtypes: float64(4), object(1)
memory usage: 5.9+ KB
```

```
data['type'].value_counts()
```

```
Iris-virginica     50
Iris-versicolor    50
Iris-setosa        49
Name: type, dtype: int64
```

- **Métodos de Clasificación:** Según lo mencionado anteriormente los métodos de clasificación que se utilizarán en el presente trabajo son: *Decision Tree, Bagging y Random Forest.* Si bien se utilizarán tres métodos de clasificación, se debe observar que, tanto el método *Bagging* como el de *Random Forest*, son una generalización del *Decision Tree*. A continuación, se explican los métodos utilizados.

    1. ***Decision Tree*:** La clasificación utilizando este algoritmo consiste en determinar la variable y el punto de corte óptimo que genera una mejor clasificación de los datos. Para definir el punto de corte óptimo se definen unas métricas que permiten evaluar la mejor división, las cuales son i) Gini Impurity y ii) Entropy. Una vez obtenido el objetivo de optimización se procede a realizar cortes iterativamente hasta que todos los datos han sido clasificados correctamente.
    2. ***Bagging:*** Este método ejecuta varias veces el algoritmo *Decision Tree* utilizando diferentes muestras con reemplazo. Para ejecutar este algoritmo se debe definir el número de veces que se utilizará el algoritmo *Decision Tree.* En lo relacionado a la predicción, se escogerá el resultado según el tipo de variable que se quiere predecir. En caso de que sea una variable categórica, se puede utilizar votación (predicción que se repite más); mientras que, en caso sea numérica continua, podría ser un promedio.
    3. ***Random Forest:*** Este método es similar a Bagging, con la diferencia que cada vez que se utiliza el algoritmo *Decision Tree*, se selecciona aleatoriamente solo un sub-cojunto propio de las características disponibles. En general, se suelen seleccionar p^(0.5) o p/3, donde "p" es el número de características disponibles para el entrenamiento.

2. Explicación del modelo

El presente trabajo tiene el objetivo de presentar la implementación de los métodos: i) *Decision Tree, ii) Bagging y iii) Random Forest.* Para mostrar los resultados obtenidos por los diferentes métodos se utilizará la base de datos Iris, la misma que guarda información de tres especies de plantas: i) Setosa, ii) Virginica y iii) Versicolor. A continuación, se explica a detalle los modelos utilizados:

- *Decision Tree:* Este método busca encontrar la división óptima que permita clasificar correctamente la información. Para la utilización de este método se debe definir un objetivo de optimización, el mismo que se apoya en métricas de decisión. A continuación, se presentan las métricas utilizadas para comparar cuán óptima es una división realizada por este método.

  Gini Impurity:

  $$GI = \sum_{i=1}^{C} p(i) * (1 - p(i))$$

  En donde, "C" representa las clases que se quiere clasificar, p(i) es la probabilidad de seleccionar aleatoriamente un elemento de la clase "i". Como se observa, "GI" representa la suma de varianza de cada clase "i". En caso de que el conjunto de datos solo tenga elementos de la misma clase, el Gini Impurity será 0, es decir, sin varianza.

  Entropy:

  $$E = - \sum_{i=1}^{C} p(i) log_2(p(i))$$

  En donde, "C" representa las clases que se quiere clasificar, p(i) es la probabilidad de seleccionar aleatoriamente un elemento de la clase "i". Al igual que en el caso anterior, la métrica "Entropy" es una manera de caracterizar la variabilidad de los datos. Por ejemplo, en caso de que el conjunto de evaluación solo tenga una clase, entonces su Entropía o variabilidad será 0.

  Una vez definidas las métricas a optimizar, el algoritmo procede a buscar la variable y punto de corte de la misma, que da como resultado un menor valor en estas métricas.

  En lo que respecta a los puntos de cortes de las variables que permiten dividir las clases se debe diferenciar dos casos: i) variables categóricas y ii) variables numéricas continuas. En caso de que se tengan variables categóricas, se deben hacer divisiones por cada categoría. Por otro lado, si las variables sobre las cuales se van a realizar las divisiones son numéricas continuas, se debe establecer puntos de corte a lo largo del valor mínimo y máximo que toma dicha variable.

En el presente trabajo, la información de las características morfológicas de las flores se reportan como enteros; por lo cual, se decidió hacer cortes asociados a cada valor único que toma las características de las misma, siguiendo lo descrito por Fayyad (1992).

Una vez seleccionadas la variable y el corte óptimo se realiza el procedimiento iterativamente, hasta que los índices de Gini Impurity y Entropy se minimizan, lo cual equivale a decir que no la varianza es mínima.

- *Bagging:* Este método implementa múltiples veces el algoritmo de *Decision Tree,* con el objetivo de disminuir el ajuste del modelo a los datos de entrenamiento. La técnica consiste en seleccionar "t" muestras aleatorias con reemplazo de los datos de entrenamiento. Las muestras seleccionadas son del mismo tamaño del grupo de entrenamiento.

  Una vez estimados los múltiples *Decision Tree,* se utilizó la técnica de voto, la cual consistió en tomar la predicción que más se repite. Por ejemplo, si los resultados indicaron 3 predicciones de Setosa y dos de Virginica, entonces, se elegía como predicción de la clase Setosa.

- *Random Forest:* Este método, al igual que el método Bagging, implementa múltiples veces el algoritmo de *Decision Tree,* con el objetivo de disminuir el ajuste del modelo a los datos de entrenamiento. A diferencia del caso de Bagging, este método al momento de estimar cada *Decision Tree,* elimina aleatoriamente algunas características.

  En el caso del presente trabajo, se hicieron pruebas con la eliminación de 1 o 2 columnas, correspondientes a eliminar p^(0.5) o p/3, donde p es el número de características disponibles. Los resultados mostraron que eliminar 2 características tenía como consecuencia que los árboles sean muy profundos, debido a que el algoritmo, lo cual se debe a que el algoritmo no lograba diferenciar en algunas características un flor Virginica de una Versicolor (las más parecidas de la base de datos).

3. Experimentación y Resultados

- **Puntos de corte para la función *best_split*.**
  Inicialmente se utilizó los promedios de cada característica como puntos de corte para implementación del método *best_split*, sin embargo, se observa que los cortes resultantes no eran óptimos pues los árboles resultaban muy profundos, incrementando el costo computacional.

```
def best_split_mean(data, method='gini'):
  columns = data.columns[0:data.shape[1]-1]
  min_gini_feature = 2
  min_class = ""
  result = {}

  gini_pond = 0
  index_der = []
  index_izq = []
  threshold = 0

  for c in columns:
    #Se divide la muestra en dos partes
    train_izq = data[data[c] <= data[c].mean()]
    train_der = data[data[c] > data[c].mean()]
```

Código fuente usando las medias

```
[['NoIdent', ['root', 'petal_length', 3.739047619047619], 'L'],
 ['NoIdent', ['root', 'petal_length', 3.739047619047619], 'R'],
 ['NoIdent', ['rootR', 'petal_width', 1.6], 'L'],
 ['NoIdent', ['rootR', 'petal_width', 1.6], 'R'],
 ['NoIdent', ['rootRR', 'petal_length', 4.8], 'L'],
 ['virginica', ['rootRR', 'petal_length', 4.8], 'R'],
 ['virginica', ['rootRRL', 'sepal_width', 3.0], 'L'],
 ['versicolor', ['rootRRL', 'sepal_width', 3.0], 'R'],
 ['versicolor', ['rootRL', 'petal_length', 4.9], 'L'],
 ['NoIdent', ['rootRL', 'petal_length', 4.9], 'R'],
 ['versicolor', ['rootRLR', 'sepal_length', 6.0], 'L'],
 ['virginica', ['rootRLR', 'sepal_length', 6.0], 'R'],
 ['setosa', ['rootL', 'petal_length', 1.9], 'L'],
 ['versicolor', ['rootL', 'petal_length', 1.9], 'R']]
```

Árbol generado usando *best_split_mean.* No clasifica en las 5 primeras iteraciones

Luego de investigar y revisar la literatura (Fayyad, 1992), se implementó el *best_split* usando todos los puntos de cada característica, obteniendo resultados más precisos y ahorro de costo computacional en la predicción.

```python
def best_split(data, method='gini'):
  columns = data.columns[0:data.shape[1]-1]
  min_gini_feature = 2
  min_class = ""
  result = {}

  gini_pond = 0
  index_der = []
  index_izq = []
  threshold = 0

  for c in columns:
    #Se toman todos los puntos
    dots = np.unique(data[c])
    for d in dots: #probamos con todos los puntos para incrementar el accuracy
      #Se divide la muestra en dos partes
      train_izq = data[data[c] <= d]
      train_der = data[data[c] > d]
```

best_split evaluando todos los puntos posibles

```
[['setosa', ['root', 'petal_length', 1.9], 'L'],
 ['NoIdent', ['root', 'petal_length', 1.9], 'R'],
 ['NoIdent', ['rootR', 'petal_width', 1.6], 'L'],
 ['NoIdent', ['rootR', 'petal_width', 1.6], 'R'],
 ['NoIdent', ['rootRR', 'petal_length', 4.8], 'L'],
 ['virginica', ['rootRR', 'petal_length', 4.8], 'R'],
 ['virginica', ['rootRRL', 'sepal_width', 3.0], 'L'],
 ['versicolor', ['rootRRL', 'sepal_width', 3.0], 'R'],
 ['versicolor', ['rootRL', 'petal_length', 4.9], 'L'],
 ['NoIdent', ['rootRL', 'petal_length', 4.9], 'R'],
 ['versicolor', ['rootRLR', 'sepal_length', 6.0], 'L'],
 ['virginica', ['rootRLR', 'sepal_length', 6.0], 'R']]
```

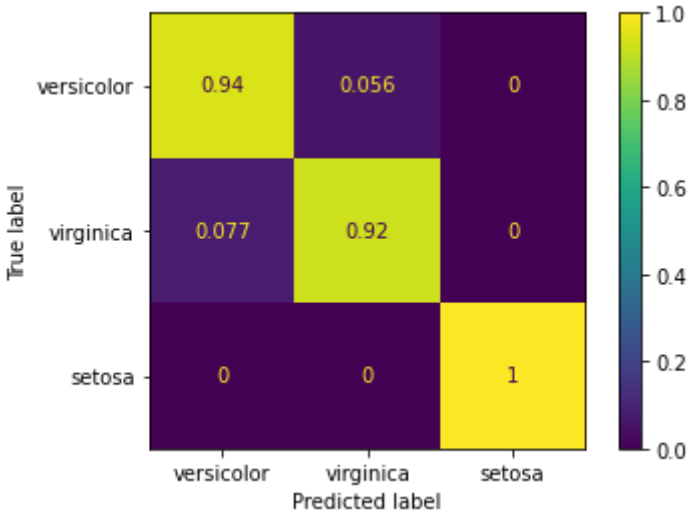
Árbol generado usando *best_split.* Clasifica en la primera iteración

- **Decision tree usando Gini**

```
tree = []
build_tree(train, train.index, tree, method = 'gini')

validation_dt = validation.copy()
validation_dt['predict'] = validation.apply(lambda row: clasificar(row, tree), axis=1)

print("Accuracy:", np.mean(validation_dt.type == validation_dt.predict) * 100)
validation_dt[validation_dt.type != validation_dt.predict]
```

```
Accuracy: 95.55555555555556
```

	sepal_length	sepal_width	petal_length	petal_width	type	predict
77	6.7	3.0	5.0	1.7	versicolor	virginica
119	6.0	2.2	5.0	1.5	virginica	versicolor



Matriz de confusión - *Decision tree* usando Gini

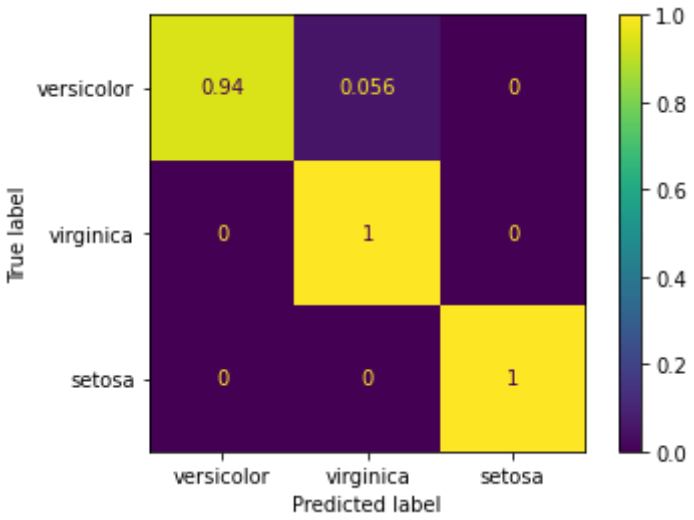
- **Bagging usando Gini**

```
[157] # Implementar Bagging
      random.seed(10)

      draws = 5
      valididation_bagging = validation.copy()
      samps = ['new1', 'new2','new3','new4','new5']

      for ix in range(0,draws):
          sample = pd.concat([train.sample(frac=0.5, replace=True, random_state=1),
                              train.sample(frac=0.5, replace=True, random_state=1)],
                              axis=0)
          tree = []
          build_tree(sample, sample.index, tree, method = 'gini')
          valididation_bagging[samps[ix]] = validation.apply(lambda row: clasificar(row, tree), axis=1)

      valididation_bagging["voting"] = valididation_bagging[samps].apply(lambda row: voting(row), axis=1)
```

```
print("Accuracy:", np.mean(valididation_bagging.type == valididation_bagging.voting) * 100)
valididation_bagging[valididation_bagging.type != valididation_bagging.voting]
```

```
Accuracy: 97.77777777777777
```

	sepal_length	sepal_width	petal_length	petal_width	type	new1	new2	new3	new4	new5	voting
77	6.7	3.0	5.0	1.7	versicolor	virginica	virginica	virginica	virginica	virginica	virginica



Matriz de confusión - *Bagging* usando Gini

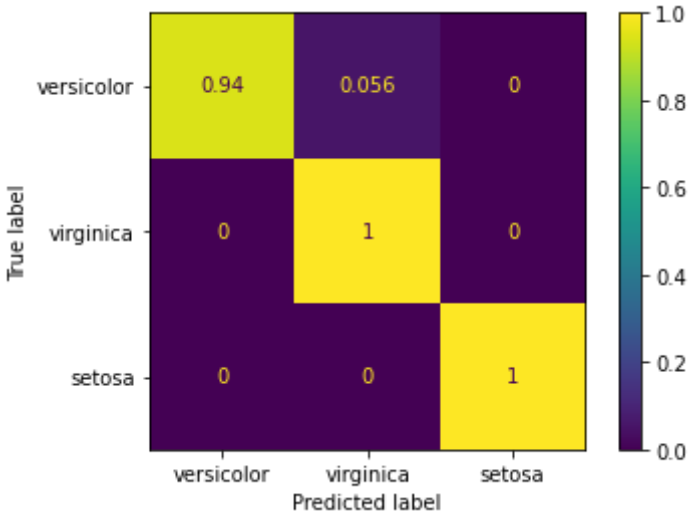
- **Random Forest usando Gini**

```
[160] # Implementar Random-Forest
      random.seed(100)

      validation_rf = validation.copy()
      for ix in range(0,draws) :
          sample = pd.concat([train.sample(frac=0.5, replace=True, random_state=1),
                              train.sample(frac=0.5, replace=True, random_state=1)],
                             axis=0)
          # Reduccion de caracteristicas
          features = random.sample(list(train.columns)[:-1], k=3)
          features.append('type')
          sample = sample.loc[:,features]
          tree = []
          build_tree(sample, sample.index, tree, method = 'gini')
          validation_rf[samps[ix]] = validation.apply(lambda row: clasificar(row, tree), axis=1)

      validation_rf["voting"] = validation_rf[samps].apply(lambda row: voting(row), axis=1)
```

```
print("Accuracy:", np.mean(validation_rf.type == validation_rf.voting) * 100)
validation_rf[validation_rf.type != validation_rf.voting]
```

```
Accuracy: 97.77777777777777
```

	sepal_length	sepal_width	petal_length	petal_width	type	new1	new2	new3	new4	new5	voting
77	6.7	3.0	5.0	1.7	versicolor	virginica	virginica	virginica	virginica	virginica	virginica



Matriz de confusión - *Random Forest* usando Gini

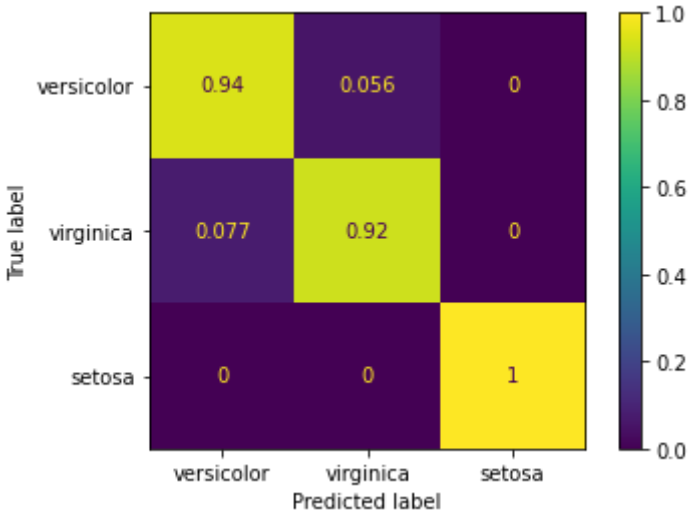
- **Decision tree usando *Entropy***

```
[162] tree = []
      build_tree(train, train.index, tree, method = 'entropy')

      validation_dt_e = validation.copy()
      validation_dt_e['predict'] = validation.apply(lambda row: clasificar(row, tree), axis=1)

      print("Accuracy:", np.mean(validation_dt_e.type == validation_dt_e.predict) * 100)
      validation_dt_e[validation_dt_e.type != validation_dt_e.predict]
```

```
Accuracy: 95.55555555555556
```

	sepal_length	sepal_width	petal_length	petal_width	type	predict
77	6.7	3.0	5.0	1.7	versicolor	virginica
119	6.0	2.2	5.0	1.5	virginica	versicolor

```
[163] # Generando la matriz de confución
      y_true = np.array(validation_dt_e["type"])
      y_pred = np.array(validation_dt_e["predict"])
      print_confusion_matrix(y_true, y_pred)
```



Matriz de confusión - Decision tree usando *Entropy*

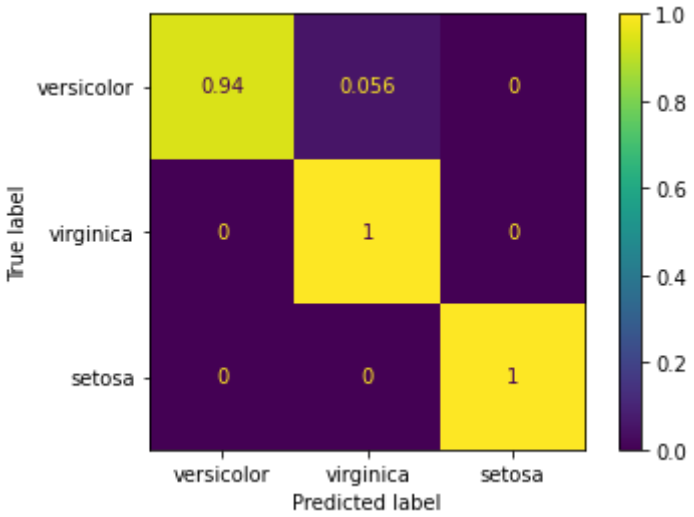
- ***Bagging* usando *Entropy***

```
[20] # Implementar Bagging
     random.seed(10)

     draws = 5
     perc = 0.6
     valididation_bagging = validation.copy()
     samps = ['new1', 'new2','new3','new4','new5']

     for ix in range(0,draws):
         sample = train.sample(frac=perc, replace=True, random_state=1).copy()
         tree = []
         build_tree(sample, sample.index, tree, method = 'entropy')
         valididation_bagging[samps[ix]] = validation.apply(lambda row: clasificar(row, tree), axis=1)

     valididation_bagging["voting"] = valididation_bagging[samps].apply(lambda row: voting(row), axis=1)
```

```
[21] print("Accuracy:", np.mean(valididation_bagging.type == valididation_bagging.voting) * 100)
     valididation_bagging[valididation_bagging.type != valididation_bagging.voting]
```

```
Accuracy: 97.77777777777777
```

	sepal_length	sepal_width	petal_length	petal_width	type	new1	new2	new3	new4	new5	voting
77	6.7	3.0	5.0	1.7	versicolor	virginica	virginica	virginica	virginica	virginica	virginica



Matriz de confusión - *Bagging* usando *Entropy*

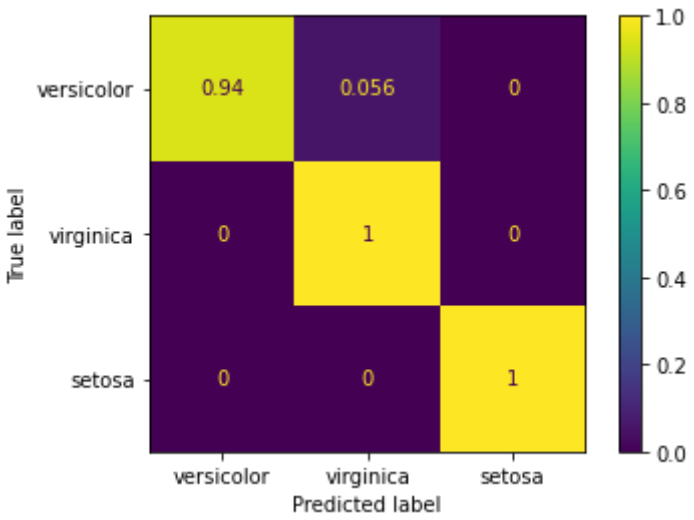
- ***Random Forest* usando *Entropy***

```
# Implementar Random-Forest
random.seed(10)

validation_rf_e = validation.copy()
for ix in range(0,draws) :
    sample = pd.concat([train.sample(frac=0.5, replace=True, random_state=1),
                        train.sample(frac=0.5, replace=True, random_state=1)],
                       axis=0)
    # Reduccion de caracteristicas
    features = random.sample(list(train.columns)[:-1], k=3)
    features.append('type')
    sample = sample.loc[:,features]
    tree = []
    build_tree(sample, sample.index, tree, method = 'entropy')
    validation_rf_e[samps[ix]] = validation.apply(lambda row: clasificar(row, tree), axis=1)

validation_rf_e["voting"] = validation_rf_e[samps].apply(lambda row: voting(row), axis=1)
```

```
[170] print("Accuracy:", np.mean(validation_rf_e.type == validation_rf_e.voting) * 100)
      validation_rf_e[validation_rf_e.type != validation_rf_e.voting]
```

```
Accuracy: 97.77777777777777
```

	sepal_length	sepal_width	petal_length	petal_width	type	new1	new2	new3	new4	new5	voting
77	6.7	3.0	5.0	1.7	versicolor	virginica	virginica	virginica	virginica	virginica	virginica



Matriz de confusión - *Random Forest* usando *Entropy*

4. CONCLUSIONES

- El método de clasificación basado en árboles de decisión resulta eficiente, logrando un *accuracy* de hasta 95.5% para el *dataset* Iris usando tanto las métricas *Gini gain* como *Entropy* para la identificación de los mejores puntos de corte
- En cuanto a los puntos de corte, inicialmente, se tomó los promedios de cada característica para identificar los puntos de corte, obteniendo un árbol de mayor profundidad, haciendo que la predicción tenga un mayor costo computacional, dado este escenario, optamos por tomar todos los puntos posibles del dataset sin repetición, obteniendo un árbol con menor profundidad y capaz de etiquetar desde el primer corte
- No hay ninguna diferencia perceptible entre usar Gini y *Entropy*, esto en cierta medida por el tamaño y distribución del *dataset*
- Fijar la semilla de la función random ayuda a tener valores reproducibles y poder validar de manera sencilla los experimentos
- Al aplicar *Bagging*, mientras menos features se utilice, más tiempo toma el cálculo de los cortes ya que podría darse el caso de no encontrar una división óptima, esto podría ser manejado mediante un hiper parámetro de máxima profundidad en la recursión
- El mecanismo de *voting*, ayuda a determinar las predicciones más recurrentes. La predicción más recurrente es tomada como la predicción final en los métodos *Bagging* y *Randon Forest*. En el caso presentado, se utilizaron cinco predicciones para que el resultado del voting sea único. Este método ayudó a incrementar el *accuracy* sobre el *dataset* de validación.

5. Bibliografía

Fayyad, U. M., & Irani, K. B. (1992). On the handling of continuous-valued attributes in decision tree generation. *Machine learning*, *8*(1), 87-102.

Matrices de confusión

Gini

Decision tree