Código Java principal

```
	/** Se obtienen los conjuntos de entrenamiento y de testeo, a partir del conjunto de instancias.
	  * @param randomizar Entero que puede ser 0 o 1. Un 1 indica que hay que randomizar las ubicaciones.
	  * @param porc_split Double que debe ser mayor a 0 y menor 1. Indica qué porcentaje de las instancias es para ENTRENAMIENTO.
	  * @return No devuelve ningun valor.
	  * @throws No dispara ninguna excepcion.
	  */
public void trainTestInstancias(int randomizar, double porc_split){

		try{

			//Se ramdomiza la ubicacion de las instancias.
			if (randomizar == 1){

				this.instancias.randomize(new Random(this.seed));
			}

			//Se realiza el split, se obtienen dos subconjunto: train y test.
			if (porc_split > 0 && porc_split < 1){

				this.trainSize = (int) Math.round(this.instancias.numInstances() * porc_split);
			}else{

				this.trainSize = (int) Math.round(this.instancias.numInstances() * this.porcentaje_split);
			}

			//
```

```
                this.testSize = this.instancias.numInstances() - this.trainSize;
                this.train = new Instances(this.instancias, 0, this.trainSize);
                this.test = new Instances(this.instancias, this.trainSize, this.testSize);

                //this.test1 = new Instances(this.instancias, this.trainSize, this.extraSize);
                //this.test2 = new Instances(this.instancias, this.trainSize, this.extraSize);

                //Se configura el atributo class.
                this.instancias.setClassIndex(this.instancias.numAttributes() - 1);
                this.train.setClassIndex(this.train.numAttributes() - 1);
                this.test.setClassIndex(this.test.numAttributes() - 1);

        }catch(Exception e){
                System.err.println("Error en: " + this.filename);
                System.err.println("Exception: " + e);
                //e.printStackTrace();
        }
}



    /** Se obtienen los conjuntos de entrenamiento y de testeo, a partir del conjunto de instancias.
      * @param randomizar Entero que puede ser 0 o 1. Un 1 indica que hay que randomizar las ubicaciones.
      * @param porc_split Double que debe ser mayor a 0 y menor 1. Indica qué porcentaje de las instancias es para entrenamiento.
      * @return No devuelve ningun valor.
      * @throws No dispara ninguna excepcion.
      */
public List<LinkedHashMap<String,String>> construirYEvaluar(String tipoEvaluacion, int k){

        List<LinkedHashMap<String,String>> lista_diccionarios = new ArrayList<LinkedHashMap<String,String>>();
        String name = "";
```

```
StringWriter errores = null;
List<String> etiquetas_actuales = new ArrayList<String>();
boolean bandera = false;

try{

        this.fstream = new FileWriter("error_log",true);
        this.out = new BufferedWriter(fstream);

}catch(Exception e){}

try{

        Classifier modelo = null;
        Evaluation evaluar = null;
        Attribute atributo_de_clase = this.instancias.classAttribute();
        int cantidad_etiquetas = atributo_de_clase.numValues();
        String etiqueta;
        DecimalFormat df = new DecimalFormat("#.##");
        int indice_clase = 0;

        //
        double numero_instancias = 0;
        double cantidad_aciertos = 0;
        double porcentaje_aciertos = 0;
        double cantidad_desaciertos = 0;
        double porcentaje_desaciertos = 0;
        double kappa_statistic = 0;
        double mean_absolute_error = 0;
        double relative_absolute_error = 0;
        double root_mean_squared_error = 0;
```

```
double root_relative_squared_error = 0;

//
double true_positive_rate = 0;
double false_positive_rate = 0;
double true_negative_rate = 0;
double false_negative_rate = 0;
double precision = 0;
double recall = 0;
double f_measure = 0;
double mcc = 0;
double roc_area = 0;
double prc_area = 0;
double[][] confusion_matrix = null;

//Conjunto de clasificadores.
Classifier[] modelos = {
        //BAYES.
        new BayesNet(),//Primer testeo de algoritmos.
        new NaiveBayes(),//Primer testeo de algoritmos.
        new NaiveBayesUpdateable(),//Primer testeo de algoritmos.
        new NaiveBayesMultinomial(),
        new NaiveBayesMultinomialUpdateable(),
        //FUNCTIONS.
        new Logistic(),//Primer testeo de algoritmos.
        new MultilayerPerceptron(),//Primer testeo de algoritmos.
        new SimpleLogistic(),//Primer testeo de algoritmos.
        new SMO(),//Primer testeo de algoritmos.
        //LAZY.
        new IBk(),
        new KStar(),
```

```
new LWL(),
//META.
new AdaBoostM1(),
new AttributeSelectedClassifier(),
new Bagging(),
new ClassificationViaRegression(),
new CVParameterSelection(),
new FilteredClassifier(),
new IterativeClassifierOptimizer(),
new LogitBoost(),
new MultiClassClassifier(),
new MultiClassClassifierUpdateable(),
new MultiScheme(),
new RandomCommittee(),
new RandomizableFilteredClassifier(),
new MultiClassClassifier(),
new RandomSubSpace(),
new Stacking(),
new Vote(),
new Stacking(),
new WeightedInstancesHandlerWrapper(),
//MISC.
new InputMappedClassifier(),
//RULES.
new OneR(),//Primer testeo de algoritmos.
new DecisionTable(),//Primer testeo de algoritmos.
new JRip(),//Primer testeo de algoritmos.
new PART(),//Primer testeo de algoritmos.
new ZeroR(),//Primer testeo de algoritmos.
//TREES.
new DecisionStump(),//Primer testeo de algoritmos.
```

```
        new J48(),//Primer testeo de algoritmos.
        new LMT(),//Primer testeo de algoritmos.
        new RandomForest(),//Primer testeo de algoritmos.
        new RandomTree(),//Primer testeo de algoritmos.
        new REPTree(),//Primer testeo de algoritmos.
        new HoeffdingTree(),

};

if (k > 1){
        this.folds = k;
}

//Etiquetas.
for (int m=0; m < cantidad_etiquetas; m++){

        etiqueta = atributo_de_clase.value(m);
        etiquetas_actuales.add(etiqueta);
}

bandera = false;
LinkedHashMap<String,String> diccionario_incluir = new LinkedHashMap<>();
List<LinkedHashMap<String,String>> lista_diccionarios_temporales = new ArrayList<LinkedHashMap<String,String>>();
LinkedHashMap<String,String> diccionario_temporal = new LinkedHashMap<>();
double maxValue = 0.00;

//Se evalua el modelo.
if (tipoEvaluacion == "split"){

        //Por cada modelo, se construye su clasificador y se evalua.
        for (int j=0; j<modelos.length; j++){
```

```
//Se construye el clasificador.
modelo = modelos[j];
name = modelo.getClass().getName();

//Guardamos el modelo.
//SerializationHelper.write(name, modelo);

//Cargamos un modelo
//Classifier cargarModelo = (Classifier) SerializationHelper.read(nombre);

try{

        //ENTRENAMIENTO.
        modelo.buildClassifier(this.train);

        //TESTEO.
        //constructor Evaluation.Evaluation(Instances,CostMatrix)
        evaluar = new Evaluation(this.test);
        evaluar.evaluateModel(modelo, this.test);

}catch(Exception e){

        try{
                errores = new StringWriter();
                this.out.write("Archivo que dio error: " + this.filename + "\n");
                this.out.write("Modelo que dio error: " + name + "\n");
                this.out.write("Evaluacion que dio error: " + tipoEvaluacion + "\n");
                this.out.write("Exception: " + "\n");
                e.printStackTrace(new PrintWriter(errores));
                this.out.write(errores.toString() + "\n");
```

```
            }catch(Exception ex){}

            continue;
        }

        bandera = true;

        //EVALUACION.

        //Numero de instancias.
        numero_instancias = evaluar.numInstances();

        //Aciertos.
        porcentaje_aciertos = evaluar.pctCorrect();

        //Kappa.
        kappa_statistic = evaluar.kappa();

        //Se guardan el modelo actual en un diccionario temporal.
        diccionario_temporal = new LinkedHashMap<>();

        //Nombre del modelo.
        diccionario_temporal.put("modelo", name);

        //Cantidad de instancias.
        diccionario_temporal.put("NINS", String.format("%.2f", numero_instancias));

        //Porcentaje de aciertos.
        diccionario_temporal.put("PACI", String.format("%.2f", porcentaje_aciertos));

        //Estadistica Kappa
```

```
                diccionario_temporal.put("KSTA", String.format("%.2f", kappa_statistic));

                //Todos los diccionarios temporales se guardan.
                lista_diccionarios_temporales.add(diccionario_temporal);
            }

            maxValue = 0.00;
            diccionario_incluir = new LinkedHashMap<>();

            //Se selecciona el mejor modelo segun el que tengo el mayor valor de Kappa y se guarda el modelo ZeroR.
            for (LinkedHashMap<String, String> entry : lista_diccionarios_temporales) {

                double f = Double.parseDouble(entry.get("KSTA"));

                if (f > maxValue){
                    maxValue = f;
                    diccionario_incluir = entry;
                }

                if (entry.get("modelo") == "weka.classifiers.rules.ZeroR"){
                    lista_diccionarios.add(entry);
                }
            }

            //Se guarda el mejor modelo segun el que tengo el mayor valor de Kappa.
            if (diccionario_incluir != null){
                lista_diccionarios.add(diccionario_incluir);
            }

        }else if (tipoEvaluacion == "cross"){
```

```
            //try{

            //      modelo.buildClassifier(this.train);
            //      evaluar = new Evaluation(this.test);
            //      evaluar.crossValidateModel(modelo, this.test, this.folds, new Random(this.seed));

            //}catch(Exception e){

            //      try{
            //            errores = new StringWriter();
            //            this.out.write("Archivo que dio error: " + this.filename + "\n");
            //            this.out.write("Modelo que dio error: " + name + "\n");
            //            this.out.write("Evaluacion que dio error: " + tipoEvaluacion + "\n");
            //            this.out.write("Exception: " + "\n");
            //            e.printStackTrace(new PrintWriter(errores));
            //            this.out.write(errores.toString() + "\n");
            //      }catch(Exception ex){}

            //      continue;
            //}

      }else if (tipoEvaluacion == "crossEstratificado"){

            maxValue = 0.00;
            diccionario_incluir = new LinkedHashMap<>();
            Classifier modelo_ganador = null;

            //Por cada modelo, se construye su clasificador y se evalua.
            for (int j=0; j<modelos.length; j++){

                  //Se construye el clasificador.
```

```
modelo = modelos[j];
name = modelo.getClass().getName();

//Guardamos el modelo.
//SerializationHelper.write(name, modelo);

//Cargamos un modelo
//Classifier cargarModelo = (Classifier) SerializationHelper.read(nombre);

try{

	//ENTRENAMIENTO.
	modelo.buildClassifier(this.train);
	evaluar = new Evaluation(this.train);
	evaluar.crossValidateModel(modelo, this.train, this.folds, new Random(this.seed));

}catch(Exception e){

	try{
		errores = new StringWriter();
		this.out.write("Archivo que dio error: " + this.filename + "\n");
		this.out.write("Modelo que dio error: " + name + "\n");
		this.out.write("Evaluacion que dio error: " + tipoEvaluacion + "\n");
		this.out.write("Exception: " + "\n");
		e.printStackTrace(new PrintWriter(errores));
		this.out.write(errores.toString() + "\n");
	}catch(Exception ex){}

	continue;
}
```

```
bandera = true;

//EVALUACION.

//Numero de instancias.
numero_instancias = evaluar.numInstances();

//Aciertos.
porcentaje_aciertos = evaluar.pctCorrect();

//Kappa.
kappa_statistic = evaluar.kappa();

//Se guardan el modelo actual en un diccionario temporal.
diccionario_temporal = new LinkedHashMap<>();

//Nombre del modelo.
diccionario_temporal.put("modelo", name + "_entrenamiento");

//Cantidad de instancias.
diccionario_temporal.put("NINS", String.format("%.2f", numero_instancias));

//Porcentaje de aciertos.
diccionario_temporal.put("PACI", String.format("%.2f", porcentaje_aciertos));

//Estadistica Kappa
diccionario_temporal.put("KSTA", String.format("%.2f", kappa_statistic));

double f = Double.parseDouble(diccionario_temporal.get("KSTA"));

//Aqui se selecciona el mejor algoritmo del entrenamiento.
```

```
            if (f > maxValue){
                        maxValue = f;
                        diccionario_incluir = diccionario_temporal;
                        modelo_ganador = modelo;
            }

            //Se incluye los resultados del algoritmo ZeroR.
            if (name == "weka.classifiers.rules.ZeroR"){
                        lista_diccionarios.add(diccionario_temporal);
            }
}

//Se guarda el mejor modelo segun el que tengo el mayor valor de Kappa.
if (diccionario_incluir != null){

            //Se guardan los resultados del mejor algoritmo del entrenamiento.
            //El modelo ganador se utiliza para el testeo.
            lista_diccionarios.add(diccionario_incluir);
            name = modelo_ganador.getClass().getName();

            try{

                        //TESTEO.
                        evaluar = new Evaluation(this.test);
                        evaluar.evaluateModel(modelo_ganador, this.test);

            }catch(Exception e){

                        try{
                                    errores = new StringWriter();
                                    this.out.write("Archivo que dio error: " + this.filename + "\n");
```

```
                    this.out.write("Modelo que dio error: " + name + "\n");
                    this.out.write("Evaluacion que dio error: " + tipoEvaluacion + "\n");
                    this.out.write("Exception: " + "\n");
                    e.printStackTrace(new PrintWriter(errores));
                    this.out.write(errores.toString() + "\n");
            }catch(Exception ex){}
    }

    //EVALUACION.

    //Numero de instancias.
    numero_instancias = evaluar.numInstances();

    //Aciertos.
    porcentaje_aciertos = evaluar.pctCorrect();

    //Kappa.
    kappa_statistic = evaluar.kappa();

    //Se guardan el modelo actual en un diccionario temporal.
    diccionario_temporal = new LinkedHashMap<>();

    //Nombre del modelo.
    diccionario_temporal.put("modelo", name + "_testeo");

    //Cantidad de instancias.
    diccionario_temporal.put("NINS", String.format("%.2f", numero_instancias));

    //Porcentaje de aciertos.
    diccionario_temporal.put("PACI", String.format("%.2f", porcentaje_aciertos));
```

```
                        //Estadistica Kappa
                        diccionario_temporal.put("KSTA", String.format("%.2f", kappa_statistic));

                        //Todos los diccionarios temporales se guardan.
                        lista_diccionarios.add(diccionario_temporal);
                }

        }else{

                System.out.println("Tipo de evaluacion desconocida!");
                return null;
        }

}catch(Exception e){

        try{
                errores = new StringWriter();
                this.out.write("Archivo no procesado: " + this.filename + "\n");
                this.out.write("Modelo que dio error: " + name + "\n");
                this.out.write("Evaluacion que dio error: " + tipoEvaluacion + "\n");
                this.out.write("Exception: " + "\n");
                e.printStackTrace(new PrintWriter(errores));
                this.out.write(errores.toString() + "\n");

        }catch(Exception ex){}

        return null;
}

//Se carga la cabecera actual.
this.extra_cabecera = "tipo_evaluacion" + ";" + "producto_evaluado" + ";";
```

```
            LinkedHashMap<String,String> diccionario_keys = lista_diccionarios.get(1);
            this.lista_claves = new ArrayList<String>();

            //Se obtienen las claves.
            for (String key:diccionario_keys.keySet()){

                this.lista_claves.add(key);
                this.cabecera = this.cabecera + key + ";";
            }

            this.cabecera = this.cabecera + this.extra_cabecera;
            this.extra_datos = tipoEvaluacion + ";" + this.filename + ";";

            try{

                this.out.close();

            }catch(Exception e){}


            if (bandera == false){

                return null;
            }

            return lista_diccionarios;
    }//Hasta aqui construirYEvaluar.
```