Svolgimento esercitazione Data Mining B

Classificazione Multi-Classe con classificatori binari

Docente: Michela Antonelli

Alunno: Mario Canalella, matricola 00122833

Premesse preliminari

Per lo svolgimento della seguente esercitazione ho sviluppo un applicativo Java con interfaccia grafica. Essa permette di selezionare i path dei file CSV in modo da processarli ed analizzarli tramite WEKA.

Per compiere tutto il processo di sviluppo sono stati utilizzati i seguenti componenti software:

- Java 1.8
- Spring Boot 2.6.3
- Weka 3.8.6
- Weka SMOTE 1.0.3
- Maven

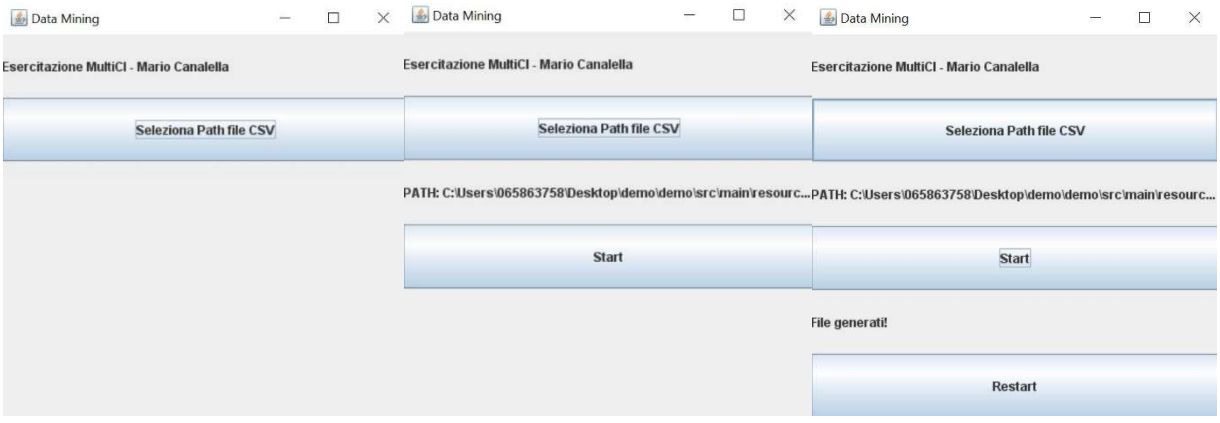
Per poter compilare ed eseguire nella propria macchina l'applicativo software occorre avere installato:

- Java 1.8
- Maven

Ed eseguire nel root della cartella del progetto Java i seguenti comandi nel terminale:

- mvn clean install -U
- ./mvnw -X spring-boot:run

In questo modo l'applicativo partirà senza intoppi.



1 2 3

Esso genererà i report e i risultati della classificazione dentro la cartella “resources/irisSingleCl”.

Svolgimento Esercitazione

Punto 1

- Il seguente codice permette di convertire i file CSV presenti nel path selezionato tramite interfaccia nel formato ARFF creando 4 file: 3 file dove Ciascun dataset binario, contenuto nel file i-esimo, ha come istanze della classe positiva tutte le istanze della classe i-esima Ci e le istanze della classe negativa sono tutte quelle delle classi restanti e un quarto file dove sono messe assieme tutte le istanze presenti nei file csv forniti.

```
private List<String> stepOne(List<String> fileNames) throws Exception {
    List<String> fileNamesArff = new ArrayList<>();
    List<Instances> instances = new ArrayList<>();
    AtomicInteger index = new AtomicInteger();
    fileNames.forEach(
            file -> {
                CSVLoader loader = new CSVLoader();
                ArffSaver saver = new ArffSaver();
                try {
                    index.getAndIncrement();
                    loader.setSource(new File(file));
                    Instances data = loader.getDataSet();
                    //REMOVE
                    data = getInstances(data);
                    //ATTRIBUTO
                    FastVector values = new FastVector();
                    values.addElement("C" + index);
                    values.addElement("not_C" + index );
                    data.insertAttributeAt(new Attribute("Class", values),
data.numAttributes());
                    for (int i = 0; i < data.numInstances(); i++) {
                        data.instance(i).setValue(data.numAttributes() - 1,
"C" + index);
                    }
                    data.setRelationName("C" + index);
                    saver.setInstances(data);
                    String name = (file + ARFF).replace(".csv", "");
                    saver.setFile(new File(name));
                    saver.writeBatch();
                    instances.add(data);
                    fileNamesArff.add(name);
                } catch (Exception e) {
                    e.printStackTrace();
                    try {
                        throw e;
                    } catch (Exception ex) {}
                }
            }
    );


    //CREO IL DATASET METTENDO ASSIEME TUTTE LE ISTANZE PRESENTI NEI CSV
```

```
    int asize = instances.get(0).numAttributes();
    boolean[] strings_pos = new boolean[asize];
    for (int i = 0; i < asize; i++) {
        Attribute att = instances.get(0).attribute(i);
        strings_pos[i] = ((att.type() == Attribute.STRING) ||
                (att.type() == Attribute.NOMINAL));
    }
    Instances multiClasses = new Instances(instances.get(0));
    AtomicReference<String> relationName = new AtomicReference<>("");
    //ATTRIBUTO
    FastVector values = new FastVector();
    instances.forEach(instance -> {
        relationName.set(relationName + instance.relationName() + "_");
        values.addElement(instance.relationName());
    });
    multiClasses = getInstances(multiClasses);
    multiClasses.insertAttributeAt(new Attribute("Class", values),
multiClasses.numAttributes());
    for (int i = 0; i < multiClasses.numInstances(); i++) {
        multiClasses.instance(i).setValue(multiClasses.numAttributes() - 1,
"C1");
    }
    Instances finalMultiClasses = multiClasses;
    instances.stream().skip(1).forEach(
            istance -> {
                DataSource source = new DataSource(istance);
                Instances instancs = null;
                try {
                    instancs = source.getStructure();
                } catch (Exception e) {
                    e.printStackTrace();
                    try {
                        throw e;
                    } catch (Exception ex) {}
                }
                Instance instance;
                while (source.hasMoreElements(instancs)) {
                    instance = source.nextElement(instancs);
                    finalMultiClasses.add(instance);

                    // COPIA GLI ATTRIBUTI STRING
                    for (int j = 0; j < asize; j++) {
                        if (strings_pos[j]) {

finalMultiClasses.instance(finalMultiClasses.numInstances() - 1)
                                    .setValue(j, instance.stringValue(j));
                        }
                    }
                }
            });
    finalMultiClasses.setRelationName(relationName.toString().substring(0,
relationName.toString().length() - 1));
    ArffSaver saver = new ArffSaver();
    try {
        saver.setInstances(finalMultiClasses);
        saver.setFile(new File(PATH + "\\" + "multiClasses" + ARFF));
        saver.writeBatch();
```

```
        multiClassesFile = PATH + "\\" + "multiClasses" + ARFF;
    } catch (IOException e) {
        e.printStackTrace();
        throw e;
    }

    //CREO LE CLASSI I-ESIME DOVE LA CLASSE POSITIVA RAPPRESENTANO TUTTE LE
ISTANZE DELLA CLASSE I-ESIMA E
    //LE ISTANZE DELLA CLASSE NEGATIVA SONO TUTTE QUELLE DELLE CLASSI
RESTANTI
    index.set(0);
    List<String> finalFileNamesArff = new ArrayList<>();
    fileNamesArff.forEach(
            fileArff -> {
                Instances data;
                try {
                    data = new Instances(new BufferedReader(new
FileReader(fileArff)));
                    index.getAndIncrement();
                    instances.stream().filter(e->
!e.relationName().equals(data.relationName())).forEach(
                            istance -> {
                                DataSource source = new DataSource(istance);
                                Instances instants = null;
                                try {
                                    instants = source.getStructure();
                                } catch (Exception e) {
                                    e.printStackTrace();
                                    try {
                                        throw e;
                                    } catch (Exception ignored) {
                                    }
                                }
                                Instance instance;
                                while (source.hasMoreElements(instants)) {
                                    instance = source.nextElement(instants);
                                    data.add(instance);

                                    // COPIA GLI ATTRIBUTI STRING
                                    for (int j = 0; j < asize; j++) {
                                        if (strings_pos[j]) {
                                            data.instance(data.numInstances()
- 1)
                                                    .setValue(j, "not_C" +
index);
                                        }
                                    }
                                }
                            });
                    saver.setInstances(data);
                    saver.setFile(new File(fileArff + FINAL + ARFF));
                    saver.writeBatch();
                    finalFileNamesArff.add(fileArff + FINAL + ARFF);
                } catch (IOException e) {
                    e.printStackTrace();
                    try {
                        throw e;
```

```
                    } catch (IOException ignored) {}
                }
            }
    );
    return finalFileNamesArff;
}
```

Punto 2

- Si costruisce il classificatore J48 e i modelli vengono valutati in cross validation (5 fold)

```
private void stepTwo(List<String> fileNamesArff) {
    J48 j48 = new J48();
    fileNamesArff.forEach(
            fileArff -> {
                try {
                    J48Filter(j48, fileArff);
                } catch (Exception e) {
                    e.printStackTrace();
                    try {
                        throw e;
                    } catch (Exception ignored) {}
                }
            });
}

private void J48Filter(J48 j48, String multiClassesFile) throws Exception {
    try {
        Instances data = prepareData(multiClassesFile);
        evaluation(j48, data, false);
    } catch (Exception ex) {
        ex.printStackTrace();
        throw ex;
    }
}

private Instances prepareData(String instanceFile) throws Exception {
    Instances data = new Instances(new BufferedReader(new
FileReader(instanceFile)));
    DataSource dataSource = new DataSource(data);
    Instances instance = dataSource.getDataSet();
    if (instance.classIndex() == -1) {
        System.out.println("reset index...");
        instance.setClassIndex(data.numAttributes() - 1);
    }
    return data;
}

private void evaluation(J48 j48, Instances instance, Boolean withSmote)
throws Exception {
    Evaluation evaluation = new Evaluation(instance);
    evaluation.crossValidateModel(j48, instance, 5, new Random(1));
    String result = evaluation.toSummaryString("Results J48 for " +
```

```
instance.relationName() + "\n\n", true);
    if (withSmote) {
        writer = new BufferedWriter(new
FileWriter("C:\\Users\\065863758\\Desktop\\demo\\demo\\src\\main\\resources\\
irisSingleCl\\resultWithSMOTE" + instance.relationName() + ".txt"));
    } else {
        writer = new BufferedWriter(new
FileWriter("C:\\Users\\065863758\\Desktop\\demo\\demo\\src\\main\\resources\\
irisSingleCl\\result" + instance.relationName() + ".txt"));
    }
    writer.write(result);
    writer.close();
}
```

Risultati ottenuti

Classe C1

```
=== Detailed Accuracy By Class ===
```

	TP Rate	FP Rate	Precision	Recall	F-Measure	MCC	ROC Area	PRC Area	Class
	0,898	0,041	0,917	0,898	0,907	0,862	0,946	0,874	C2
	0,959	0,102	0,949	0,959	0,954	0,862	0,946	0,964	not_C2
Weighted Avg.	0,939	0,082	0,939	0,939	0,939	0,862	0,946	0,934	

Classe C2

```
=== Detailed Accuracy By Class ===
```

	TP Rate	FP Rate	Precision	Recall	F-Measure	MCC	ROC Area	PRC Area	Class
	0,959	0,000	1,000	0,959	0,979	0,970	0,980	0,973	C1
	1,000	0,041	0,980	1,000	0,990	0,970	0,980	0,980	not_C1
Weighted Avg.	0,986	0,027	0,987	0,986	0,986	0,970	0,980	0,978	

Classe C3

```
=== Detailed Accuracy By Class ===
```

	TP Rate	FP Rate	Precision	Recall	F-Measure	MCC	ROC Area	PRC Area	Class
	0,959	0,031	0,940	0,959	0,949	0,924	0,965	0,916	C3
	0,969	0,041	0,979	0,969	0,974	0,924	0,965	0,976	not_C3
Weighted Avg.	0,966	0,037	0,966	0,966	0,966	0,924	0,965	0,956	

Dai dati analizzati si nota una precisione non variabile per l'affinità tra dati rilevati e dati recuperati e dall'abbassamento di precisione della classe positiva ne consegue un aumento di quella negativa. Inoltre dall'osservazione dei valori delle classi negative emerge che quest'ultime presentano un'elevata recall, confermata anche dai valori del TPR.

Punto 3

Analizziamo il classificatore J48 utilizzando il dataset che si ottiene mettendo assieme tutte le istanze presenti nei file csv forniti.

```
private void stepThree() throws Exception {
    J48 j48 = new J48();
    J48Filter(j48, multiClassesFile);
}

private void J48Filter(J48 j48, String multiClassesFile) throws Exception {
    try {
        Instances data = prepareData(multiClassesFile);
        evaluation(j48, data, false);
    } catch (Exception ex) {
        ex.printStackTrace();
        throw ex;
    }
}
private void evaluation(J48 j48, Instances instance, Boolean withSmote)
throws Exception {
    Evaluation evaluation = new Evaluation(instance);
    evaluation.crossValidateModel(j48, instance, 5, new Random(1));
    String result = evaluation.toSummaryString("Results J48 for " +
instance.relationName() + "\n\n", true);
    if (withSmote) {
        writer = new BufferedWriter(new
FileWriter("C:\\Users\\065863758\\Desktop\\demo\\demo\\src\\main\\resources\\
irisSingleCl\\resultWithSMOTE" + instance.relationName() + ".txt"));
    } else {
        writer = new BufferedWriter(new
FileWriter("C:\\Users\\065863758\\Desktop\\demo\\demo\\src\\main\\resources\\
irisSingleCl\\result" + instance.relationName() + ".txt"));
    }
    writer.write(result);
    writer.close();
}

private Instances prepareData(String instanceFile) throws Exception {
    Instances data = new Instances(new BufferedReader(new
FileReader(instanceFile)));
    DataSource dataSource = new DataSource(data);
    Instances instance = dataSource.getDataSet();
    if (instance.classIndex() == -1) {
        System.out.println("reset index...");
        instance.setClassIndex(data.numAttributes() - 1);
    }
    return data;
}
```

Risultati

Classe C1_C2_C3

```
=== Detailed Accuracy By Class ===

                 TP Rate   FP Rate   Precision   Recall   F-Measure   MCC      ROC Area  PRC Area  Class
                 0,980     0,000     1,000       0,980    0,990       0,985    0,990     0,986     C1
                 0,939     0,051     0,902       0,939    0,920       0,879    0,953     0,876     C2
                 0,918     0,031     0,938       0,918    0,928       0,892    0,948     0,887     C3
Weighted Avg.    0,946     0,027     0,946       0,946    0,946       0,919    0,964     0,916
```

Possiamo notare come nel caso di C1 abbiamo un miglioramento dei valori mentre C2 e C3 un leggero peggioramento per il TRP e la Recall

Punto 4

Aggiungiamo il filtro SMOTE alle classi

```
private void stepFour(List<String> fileNamesArff) {
    J48 j48 = new J48();
    fileNamesArff.forEach(
            fileArff -> {
                try {
                    J48FilterWithSMOTE(j48, fileArff);
                } catch (Exception e) {
                    e.printStackTrace();
                    try {
                        throw e;
                    } catch (Exception ignored) {}
                }
            });
}

private void J48FilterWithSMOTE(J48 j48, String multiClassesFile) throws
Exception {
    try {
        Instances data = prepareData(multiClassesFile);
        SMOTE smote = new SMOTE();
        smote.setInputFormat(data);
        Instances dataWithSmote = Filter.useFilter(data, smote);
        evaluation(j48, dataWithSmote, true);
    } catch (Exception ex) {
        ex.printStackTrace();
        throw ex;
    }
}

private void evaluation(J48 j48, Instances instance, Boolean withSmote)
throws Exception {
    Evaluation evaluation = new Evaluation(instance);
    evaluation.crossValidateModel(j48, instance, 5, new Random(1));
    String result = evaluation.toSummaryString("Results J48 for " +
instance.relationName() + "\n\n", true);
    if (withSmote) {
        writer = new BufferedWriter(new
FileWriter("C:\\Users\\065863758\\Desktop\\demo\\demo\\src\\main\\resources\\
irisSingleCl\\resultWithSMOTE" + instance.relationName() + ".txt"));
```

```
    } else {
        writer = new BufferedWriter(new
FileWriter("C:\\Users\\065863758\\Desktop\\demo\\demo\\src\\main\\resources\\
irisSingleCl\\result" + instance.relationName() + ".txt"));
    }
    writer.write(result);
    writer.close();
}

private Instances prepareData(String instanceFile) throws Exception {
    Instances data = new Instances(new BufferedReader(new
FileReader(instanceFile)));
    DataSource dataSource = new DataSource(data);
    Instances instance = dataSource.getDataSet();
    if (instance.classIndex() == -1) {
        System.out.println("reset index...");
        instance.setClassIndex(data.numAttributes() - 1);
    }
    return data;
}
```

Ottieniamo i seguenti risultati:

C1

```
=== Detailed Accuracy By Class ===

                TP Rate  FP Rate  Precision  Recall   F-Measure  MCC      ROC Area  PRC Area  Class
                0,990    0,000    1,000      0,990    0,995      0,990    0,995     0,995     C1
                1,000    0,010    0,990      1,000    0,995      0,990    0,995     0,990     not_C1
Weighted Avg.   0,995    0,005    0,995      0,995    0,995      0,990    0,995     0,992
```

C2

```
=== Detailed Accuracy By Class ===

                TP Rate  FP Rate  Precision  Recall   F-Measure  MCC      ROC Area  PRC Area  Class
                0,939    0,041    0,958      0,939    0,948      0,898    0,949     0,954     C2
                0,959    0,061    0,940      0,959    0,949      0,898    0,949     0,917     not_C2
Weighted Avg.   0,949    0,051    0,949      0,949    0,949      0,898    0,949     0,936
```

C3

```
=== Detailed Accuracy By Class ===

                TP Rate  FP Rate  Precision  Recall   F-Measure  MCC      ROC Area  PRC Area  Class
                0,949    0,041    0,959      0,949    0,954      0,908    0,946     0,912     C3
                0,959    0,051    0,949      0,959    0,954      0,908    0,946     0,933     not_C3
Weighted Avg.   0,954    0,046    0,954      0,954    0,954      0,908    0,946     0,923
```

Di cui notiamo che utilizzando il filtro SMOTE e ribilanciando la classe positiva si ha miglioramento della precisione solo nel caso della classe positiva C1.

Punto 5

Costruiamo 2 metaclassificatori che inglobino ciascuno un metodo diverso di attribute selecion (infogain+ ranker (valore infogain >0) e cfsSubsetEval+ best first) applicando a ciascuno il filtro SMOTE.

```
private void stepFive(List<String> fileNamesArff) {
    fileNamesArff.forEach(
            fileArff -> {
                try {
                    metaClassifiersCfsSubsetEvalBestFirst(fileArff);
                    metaClassifiersInfogainRanker(fileArff);
                } catch (Exception e) {
                    e.printStackTrace();
                    try {
                        throw e;
                    } catch (Exception ignored) {}
                }
            }
    );
}

private void metaClassifiersInfogainRanker(String classFile) throws Exception
{
    // CfsSubsetEval + best first
    Instances data = prepareData(classFile);
    //APPLICO SMOTE
    SMOTE smote = new SMOTE();
    smote.setInputFormat(data);
    data = Filter.useFilter(data, smote);
    //AGGIUNGO ATTRIBUTE SELECTION DEL TIPO cfsSubsetEval + best first
    AttributeSelectedClassifier classifier = new
AttributeSelectedClassifier();
    InfoGainAttributeEval eval = new InfoGainAttributeEval();
    Ranker ranker = new Ranker();
    ranker.setNumToSelect(Math.min(1, data.numAttributes() - 1));
    NaiveBayes nb = new NaiveBayes();
    classifier.setClassifier(nb);
    classifier.setEvaluator(eval);
    classifier.setSearch(ranker);
    Evaluation evaluation = new Evaluation(data);
    evaluation.crossValidateModel(classifier, data, 5, new Random(1));
    String result = evaluation.toSummaryString("Results Infogain + Ranker for
" + data.relationName() + "\n\n", true);
    writer = new BufferedWriter(new
FileWriter("C:\\Users\\065863758\\Desktop\\demo\\demo\\src\\main\\resources\\
irisSingleCl\\resultInfogain+Ranker" + data.relationName() + ".txt"));
    writer.write(result);
    writer.close();
    ArffSaver saver = new ArffSaver();
    saver.setInstances(data);
    saver.setFile(new
File("C:\\Users\\065863758\\Desktop\\demo\\demo\\src\\main\\resources\\irisSi
ngleCl\\resultInfogain+Ranker" + data.relationName() + ARFF));
    saver.writeBatch();
```

```
    getAttributeSelectorInfogainRanker(eval, ranker, data);
}

private void getAttributeSelectorCfsSubsetEvalBestFirst(CfsSubsetEval
evaluator, BestFirst search, Instances data) throws Exception {
    AttributeSelection selector = new AttributeSelection();
    selector.setEvaluator(evaluator);
    selector.setSearch(search);
    selector.SelectAttributes(data);
    writer = new BufferedWriter(new
FileWriter("C:\\Users\\065863758\\Desktop\\demo\\demo\\src\\main\\resources\\
irisSingleCl\\selettoriCfsSubsetEval" + data.relationName() + ".txt"));
    writer.write("Selettori CfsSubsetEval + BestFirst data " +
data.relationName() + "\n\n");
    writer.write(Arrays.toString(selector.selectedAttributes()));
    writer.close();
}
private void getAttributeSelectorInfogainRanker(InfoGainAttributeEval
evaluator, Ranker ranker, Instances data) throws Exception {
    AttributeSelection selector = new AttributeSelection();
    selector.setEvaluator(evaluator);
    selector.setSearch(ranker);
    selector.SelectAttributes(data);
    writer = new BufferedWriter(new
FileWriter("C:\\Users\\065863758\\Desktop\\demo\\demo\\src\\main\\resources\\
irisSingleCl\\selettoriInfogainRanker" + data.relationName() + ".txt"));
    writer.write("Selettori Infogain + Ranker data " + data.relationName() +
"\n\n");
    writer.write(Arrays.toString(selector.selectedAttributes()));
    writer.close();
}
```