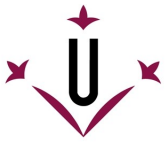

Universitat de Lleida

Escola Politècnica Superior

Aprendizaje automático de relaciones entre textos mediante Support Vector Machines

Dídac Sementé Fernández

Director: Ramon Bejár Torres

Treball Final de Grau, Juliol 2015

Contents

List of Figures

List of Tables

Listings

1 Introduction

Throughout the course of the years, machine involvement in what was traditionally considered "human activities" has been increasing considerably and steadily, one of those fields is what we call "machine learning" which, as its name suggest, stands for machines being capable of some level of reasoning and memorization in order to give answers to certain questions, like for example: how likely is a user to subscribe to a certain service based on his musical tastes?, this is one of the many things that in which machines have surpassed mankind in the course of the years, thanks to the implementation of the concepts of machine learning.

Now, about the purpose of this project, in the recent years there's been a certain service which has seen an incredible rise in popularity, that service is what we call social networks.

Social networking sites nowadays provide a lot of room for personal reference, discovery of stories and opinions, discussions and debate between different individuals apart from more other things. This project's aim is to cater this debates, trying to obtain the exact relation of each of the participant's arguments in them, whereas it is an attack, support or neither of both type of argument. In order to obtain this data the project makes use of some of the elements found in machine learning studies to determine the type of relation two arguments may have.

So basically what we wanted to do with this project could relate to some sort of auto-moderation system for debates in the way that if, for example, a given user's arguments are frequently attacked by other users that could mean that this user isn't really adding anything to the debates he enters into, so the system may decide to moderate his contributions or to set him some sort of "karma" value, with this system the other users would be able to know how much a user's argument should be taken into consideration or not when following a given debate, either in a social networking site or any general debate forum.

Finally, my personal motivation for accepting this project was the fact that I find all these concepts really interesting in the way that make machines look more "human", also there's the aspect that these technologies are growing more common and complex as you are reading this document so I was also interested in contributing a bit to these myriad new technologies and applications for computing that are currently on the rise.

1.1 Document Structure

This document is divided in multiple sections that cover different aspects of the project, from the original requisites to which improvements could be made in the future, below there's a little introduction to what each one will focus on:

Section	Description
Introduction	Project motivation and initial requirements
Learning with Support Vector Machines	Introduction to machine learning, neural networks and the applications of Support Vector Machines for learning
Project Design	General overview of the project structure and organization
Project Implementation	How was the design implemented and which were the choices made
Evaluation and Testing	Real use case for the project and analysis of its results
Conclusion and Future Work	Opinion on the project and possible improvements
Bibliography	Information sources used for the project

Table 1: Document organization

As you can see each section is well defined, although, if this is your first time reading this document you should follow the defined section order.

1.2 Project Requirements

The initial requirements for the project where to simply build an application which would receive an input in the form of an XML file using a defined encoding specified in the NoDE [6] website, that encoding defines a group of arguments belonging to a given debate, the arguments are grouped in pairs which represent the relation between the two, using this information provided by the file we try to classify the relations between those arguments into three different types: attack, support or unknown.

Once the classification was done, the application would return the result in one of the three following output formats:

- The same XML format as the input with the classification result as attribute.
- A format used for digraph description called DOT [3] used with tools such as GraphViz to visualize the digraph.
- A format used in the argument reasoning framework ASPARTIX [1].

Basically in almost everything regarding the specific implementation I had "charte blanche" to choose whatever language and/or set of tools I though would be more appropriate for the task, as long as the application was capable of at least satisfying the requirements the way I chose to implement it was completely up to me.

Below there's a basic representation of the what the project was expected to do based on the initial requirements:

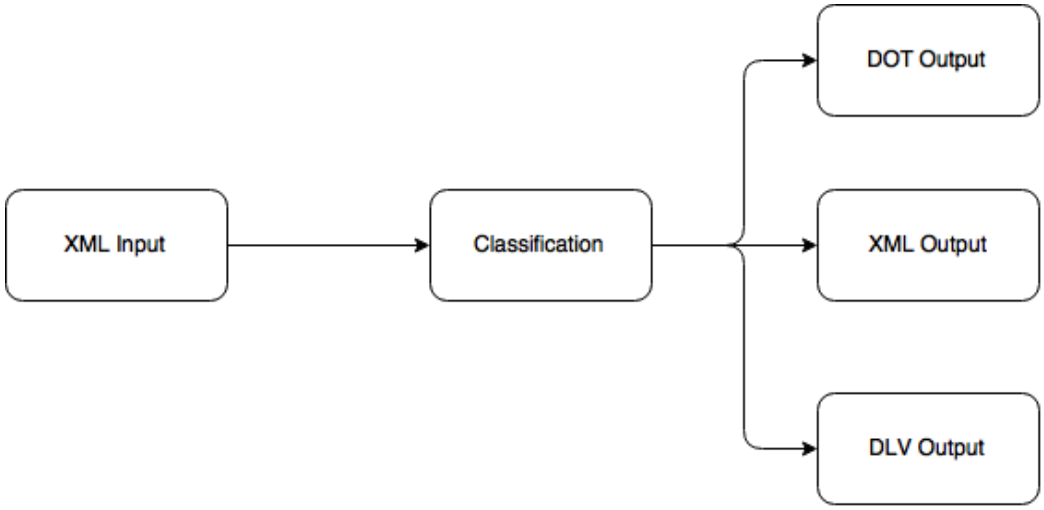

Figure 1: Initial Requirements Diagram

As you can see it's pretty simple isn't it? Now in the next section we'll explain some concepts of machine learning which will help you understand how the classification happens.

2 Learning with Support Vector Machines

Before going further into the project we'll dedicate this section in explaining a bit about what machine learning concepts and characteristics:

2.1 Machine Learning Concepts

So first of all we'll define the term in a concrete way and expand from there. From the Wikipedia [5] article we have that machine learning is "a sub-field of computer science that evolved from the study of pattern recognition and computational learning theory in artificial intelligence. Machine learning explores the construction and study of algorithms that can learn from and make predictions on data. Such algorithms operate by building a model from

example inputs in order to make data-driven predictions or decisions, rather than following strictly static program instructions."

In more common words that means that machine learning is a field which focuses on making applications capable of finding specific patterns in data, and used those patterns to make predictions on different data.

In machine learning we have three general types of tasks or categories depending on how the system learns:

- Supervised learning: The system is "trained" with valid inputs in order for it to learn the patterns.
- Unsupervised learning: The system isn't trained at all and has to figure the patterns by itself.
- Reinforced learning: The system is interacting constantly with the environment without previous knowledge of its goals or if it's getting closer to them or not.

In our project we use supervised learning for our system, so, we need valid input data to train or system before doing any real tasks.

There are multiple approaches in the way your system creates the pattern from your data, each approach is normally focused on a group of problems, or in a type of output you want to obtain from your system. Below there's a list of some of the algorithms used for machine learning:

- Decision tree learning
- Association rule learning
- Artificial neural networks
- Inductive logic programming
- Support vector machines
- Bayesian networks

We'll focus our attention in Artificial neural networks and support vector machines since these are the systems the library uses for machine learning.

2.2 Artificial Neural Networks

Artificial neural networks are a type of algorithm used in machine learning that tries to emulate the concept of biological neural networks, this systems have a number of inputs and outputs and process the data in a similar way a biological neural network would do.

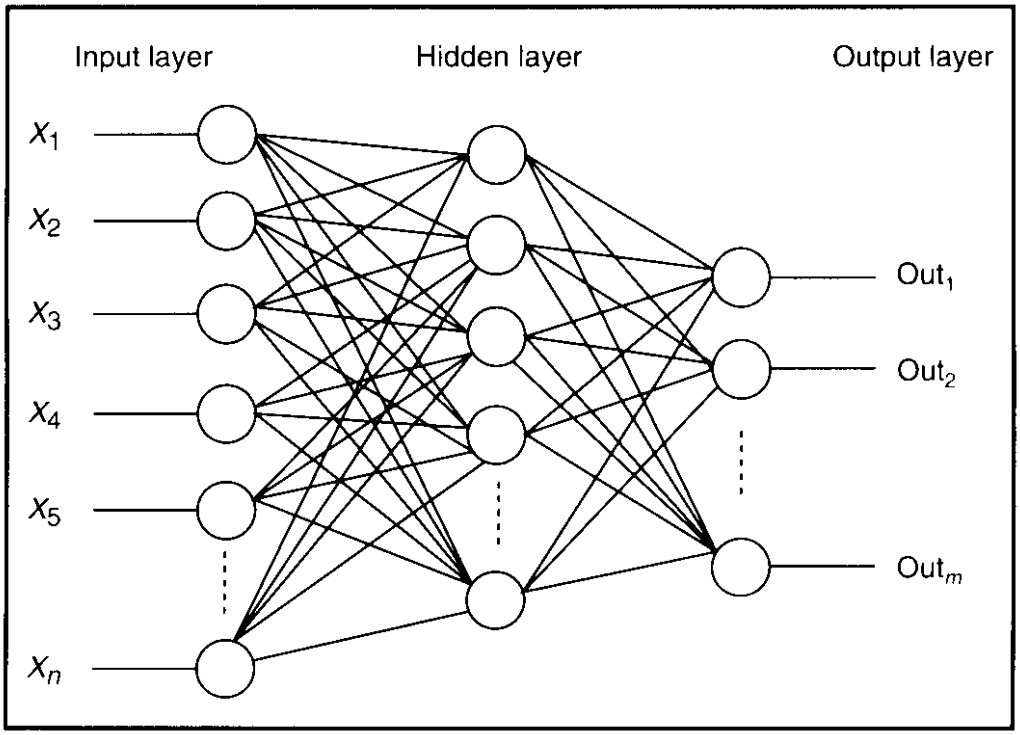

Figure 2: Neural network diagram

In this diagram, each node represents an artificial neuron which is connected to other neurons.

Artificial neural networks are typically organized in layers. Layers are made up of a number of interconnected "nodes" which contain an "activation func-

tion". Patterns are presented to the network via the "input layer", which communicates to one or more "hidden layers" where the actual processing is done via a system of weighted "connections". The hidden layers then link to an "output layer" where the answer is output.

Most artificial neural networks contain some form of "learning rule" which modifies the weights of the connections according to the input patterns that it is presented with. In a sense, artificial neural networks learn by example as do their biological counterparts.

So, what are artificial neural networks good for? Well, artificial neural networks are universal approximators, and they work best if the system you are using them to model has a high tolerance to error. They work very well for:

- Capturing associations or discovering regularities within a set of patterns
- Tasks where the volume, number of variables or diversity of the data is very large
- Relations where the relationships between variables is vaguely understood
- Tasks in which the relationships are difficult to describe adequately with conventional approaches

As you can see the classification of relations between two arguments implies looking for specific patterns in each argument in order to determine the exact relation between them, so it's a perfect problem in which neural networks can be put to good use!.

2.3 Applications of Support Vector Machines

Support Vector Machines are based on the concept of decision planes that define decision boundaries. A decision plane is one that separates between a set of objects having different class memberships. A schematic example is shown in the illustration below. In this example, the objects belong either to

class GREEN or RED. The separating line defines a boundary on the right side of which all objects are GREEN and to the left of which all objects are RED. Any new object (white circle) falling to the right is labelled, i.e., classified, as GREEN (or classified as RED should it fall to the left of the separating line).

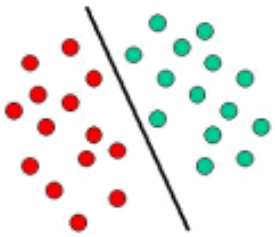

Figure 3: Linear classifier example

The above is a classic example of a linear classifier, i.e., a classifier that separates a set of objects into their respective groups (GREEN and RED in this case) with a line. Most classification tasks, however, are not that simple, and often more complex structures are needed in order to make an optimal separation, i.e., correctly classify new objects (test cases) on the basis of the examples that are available (train cases). This situation is depicted in the illustration below. Compared to the previous schematic, it is clear that a full separation of the GREEN and RED objects would require a curve (which is more complex than a line). Classification tasks based on drawing separating lines to distinguish between objects of different class memberships are known as hyperplane classifiers. Support Vector Machines are particularly suited to handle such tasks.

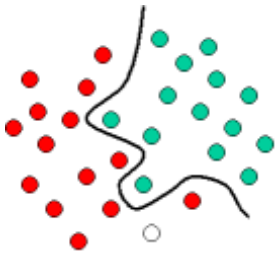

Figure 4: Complex curve example

The illustration below shows the basic idea behind Support Vector Machines. Here we see the original objects (left side of the schematic) mapped, i.e., rearranged, using a set of mathematical functions, known as kernels. The process of rearranging the objects is known as mapping (transformation). Note that in this new setting, the mapped objects (right side of the schematic) is linearly separable and, thus, instead of constructing the complex curve (left schematic), all we have to do is to find an optimal line that can separate the GREEN and the RED objects.

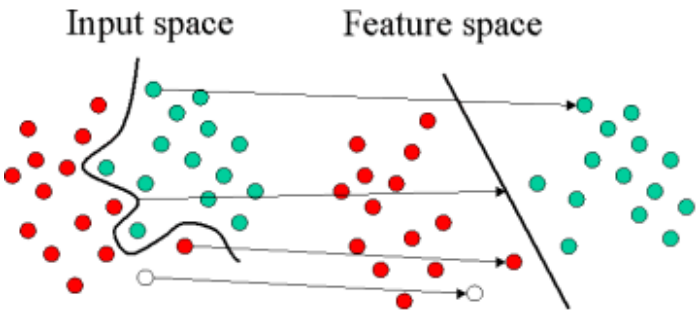

Figure 5: Element rearranging example

Support vector machines can be used to solve various real world problems:

- SVMs are helpful in text and hypertext categorization as their application can significantly reduce the need for labelled training instances in both the standard inductive and transductive settings.
- Classification of images can also be performed using SVMs.
- Experimental results show that SVMs achieve significantly higher search accuracy than traditional query refinement schemes after just three to four rounds of relevance feedback.
- SVMs are also useful in medical science to classify proteins with up to 90% of the compounds classified correctly.
- Hand-written characters can be recognized using SVM.

As you can see, the uses for support vector machines are mainly focused on pattern recognition which is exactly what we need for this project, since it

resembles a lot the way a human would perform the classification for the arguments, when you see two arguments you know if one is supporting or attacking the other because you find specific patterns in it like, for example, the way the text is written, the words used, the punctuation marks...

So, now that we have a more or less basic idea of what is machine learning and how it can help us solve real-world problems, let's go into the project itself.

3 Project Design

Continuing from diagram 1, I spent a couple of weeks analysing the options I had to build the application, finally I decided to expand the initial diagram into something a bit more closer to a possible implementation:

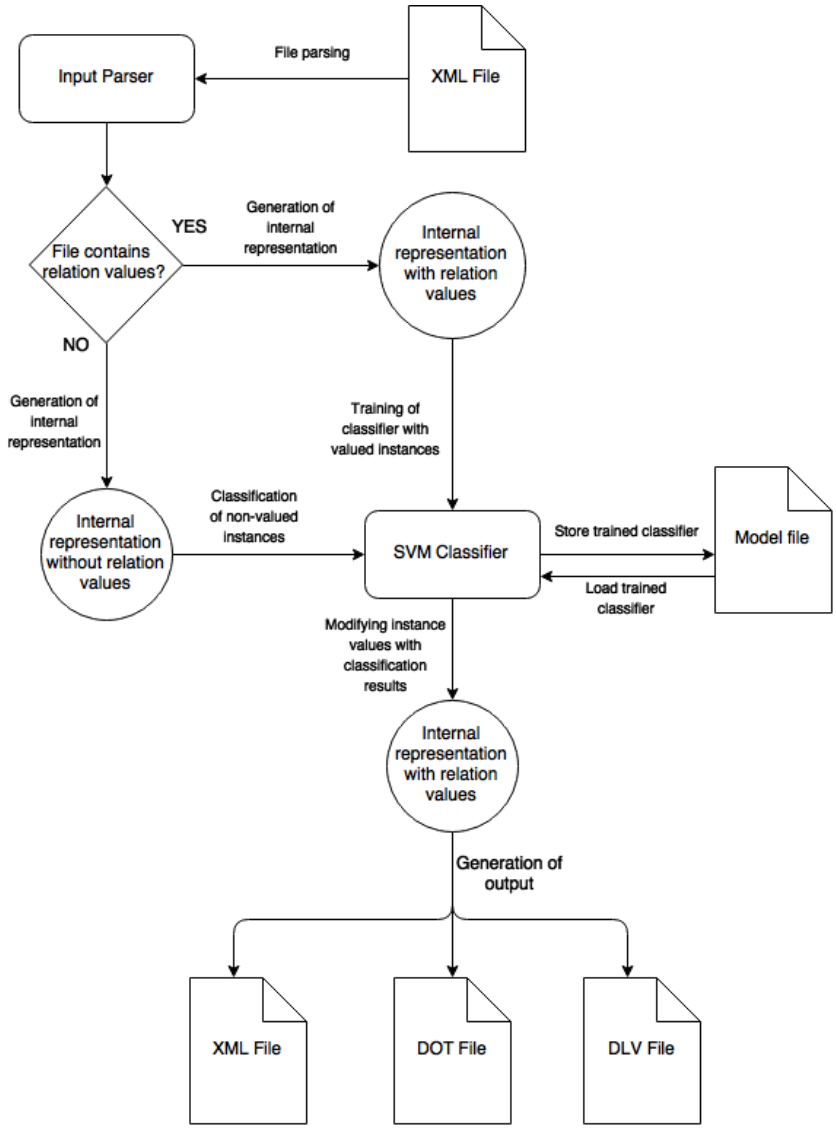

Figure 6: Advanced Requirements Diagram

As you can see this diagram is bit more complex than the initial, in here

there's already a clear sight of how would the application do in order to full-fill the initial requirements.

This diagram made me think that perhaps I should re-purpose the project not as a simple application but as a library that could be used by other developers since, in my opinion, the requisites of the project had a lot of potential in being expanded and made more "open" to any future developer that wanted to improve my original work.

So once I had spoken with my project tutor about the direction in which my implementation would go and had received the OK from him, I proceeded with it.

3.1 Project Structure

The project is divided in modules that are mostly independent from each other, each module is focused in a part of what diagram 6 shows, meaning that all modules are needed in order to comply with the initial requisites.

Below you have the modules in which the project is divided into:

- A module responsible for handling the parsing of the files containing the argumentation data and also used to generate output of the argumentation data in multiple output formats.
- A module tasked with providing an in-memory representation of the data parsed with ease of access and manipulation.
- A module responsible of filtering the data of the in-memory representation into a data usable for the classifier.
- A module that provides facilities for both classifier training and data classification.
- A module that contains various utilities for the other modules to use.

All of the elements in each module where designed thinking about scalability and future modifications or expansions by other developers in order to en-

hance or modify the functionality provided by the library, that's one of the reasons each module has a very specific task, because this way we can avoid dependences that would make future modifications more difficult.

3.2 Learning Work-flow

The process of learning implies training a classifier with a given dataset in which the type of all the relations is already defined, this is done because we are using supervised learning, and in supervised learning you need to train your classifiers first before trying to classify datasets where the class value (relation type in our case) is unknown. If you already read section 2 of the document then you should already know this.

So in order to create and train a given classifier this are the steps to follow:

1. Gather your training data and format it in the NoDE [6] XML format.
2. Using that data train the classifier.
3. Store the model generated for future use and/or proceed to classification with new data.

It's a very straightforward process, although it also needs a lot of tuning in the user side, since there are many ways in which classifiers can be configured that can alter dramatically the model obtained in the training. So, you'll probably spend quite more time testing different models with different classifier parameters than collecting your data for classification.

3.3 Labelling Work-flow

The process of labelling consists in classifying an instance with a trained classifier and assigning a "label" to its class value, which is the prediction the classifier has found for that particular instance based on its previous training of already labelled instances.

In order to classify a given set of instances this is the process to follow:

1. Collect the data you wish to classify and format it in the NoDE [6] XML format.
2. Using an already trained classifier, classify the data.
3. Write the classification results to the in-memory representation format.
4. Using any of the available output generators write the in-memory representation to an output source.

If your model in the previous step was well studied and tested the classification results will benefit from enhanced precision in determining the correct label for a particular instance, but once you have the data labelled, you can use it in a myriad ways not covered in this document I'm afraid since that subject alone would make enough room for multiple times the total length of this document. If you are really interested in the potential uses of this data, the Weka [8] contains a lot of useful information on the topic.

4 Project Implementation

In this section we'll talk about how everything explained in the previous sections was implemented, which were the decisions made and which technologies were used to make the project a reality.

But first I'll point you to the GitHub repository in which you'll find all the code for the project and also the Javadoc for it.

The repository is: relationship-learning. Licensed under the BSD license.

4.1 Technologies Used

The first decision before beginning the implementation, was to choose which machine learning library was more appropriate for the project requirements. Since I was already familiar with the Weka [8] library from previous assignments in the degree and it also seemed to be the most mature and well

documented of the different options I had, I decided that the project would be based on it for the learning aspect. That, in turn, meant that the project would be built in Java, a language in which I was also familiar with from the degree.

Also I made use of external libraries to help me with the generation of output and the processing of the arguments [4].

4.2 Code Structure

The project is divided into Java packages, each of them providing a specific functionality for the project, in most cases the packages contain a set of Java interfaces with also a default implementation for users that don't need or want to do any coding, and just want to get into classification of instances immediately.

All packages belong to the **org.relationlearn** main package, in the following sections we'll explain each package contents and the implementation decisions made in each on of them.

4.2.1 Package org.relationlearn.model

This package contains all the code used to manage the in-memory representation of the input in the form of a weighted digraph. As such this package contains classes and interfaces used the represent the concept of a weighted digraph. This package contains three Java interfaces:

ArgumentNode: This interface defines the behaviour any node in the digraph should have for storing the argument and the node specific weight, apart from having relations linked. Interface specification:

Listing 1: ArgumentNode specification

```
package org.relationlearn.model;

import java.util.List;

public interface ArgumentNode {

  public int getNodeId();
  public int getNodeWeight();
  public void addTargetRelation(ArgumentRelation aRel);
  public ArgumentRelation getTargetRelation();
  public ArgumentRelation getReplayRelation(int relationId);
  public List<ArgumentRelation> getReplyRelations();
  public void addReplyRelation(ArgumentRelation relation);
  public String getArgumentNodeText();

}
```

As you can see from the specification only one target relation may exist for each ArgumentNode, that's because in most debates you can only reply to one argument but your argument can receive multiple replies, because of that you have a list of reply relations but only one (or zero) target relations.

If the digraph is well-formed, only one node shouldn't have a target relation, this node is considered the original creator of the debate thread creating the first argument and, because of that, it is the only node that didn't reply to anyone.

ArgumentRelation: This interface defines the behaviour any relation between two ArgumentNode instances should have. Interface specification:

Listing 2: ArgumentRelation specification

```
package org.relationlearn.model;

import org.relationlearn.util.RelationType;

public interface ArgumentRelation {

  public int getArgumentRelationId();
  public ArgumentNode getArgumentator();
  public ArgumentNode getTarget();
```

```
  public RelationType getArgumentRelationType();
  public void changeRelationType(RelationType t);

}
```

As you can see this interfaces provides facilities for navigating the digraph's nodes since it provides with methods for accessing both sides of the relation, it also provides the only modifiable value in the whole digraph which is the type of relation this ArgumentRelation represents, **this is the value the classifier predicts when you use it for classification**.

RelationDigraph: This interface defines the behaviour any digraph used to store the arguments should have to match the project's expected behaviour. Interface specification:

Listing 3: RelationDigraph specification

```
package org.relationlearn.model;

import org.relationlearn.exception.AlreadyExistingNodeException;

public interface RelationDigraph extends Iterable<ArgumentNode> {

  public ArgumentNode getArgumentNode(int nodeId);
  public void addArgumentNode(ArgumentNode node)
          throws AlreadyExistingNodeException;
  public boolean containsNode(int nodeId);

}
```

As you can see by the specification the interface only provides facilities for adding and checking for a given ArgumentNode's existence but not for removing them from the digraph. This decision was made because the digraph is only used as an internal representation for data gathered with the parser and as such the digraph is basically used as a read-only structure, so the ability to remove nodes was unnecessary for it because it would make the resulting output different form the original input which could lead to inconsistencies. Of course any developer that needs that facility may implement its own extension of the interface with support for node removal, but in the

original implementation that capability is not necessary.

With these interfaces defined the library provides a default implementation for each of them which is the one used by the XML parser when reading a file. This three classes are:

NodeImpl: This class implements the ArgumentNode interface, the only remarkable element of this implementation is the use of a HashMap to store the relations that target this specific node in order to speed the search process when looking for siblings in the digraph.

RelationImpl: This class implements the ArgumentRelation interface, this class consists of basically getters for the attributes expected in the relation between two nodes in the digraph, the only value that can be modified is the type of the relation since when learning the initial value will be set to unknown and later set the correct value provided by the classifier.

DigraphImpl: This class implements the RelationDigraph interface and stores the nodes using a HashMap provided by the Java standard library using the node id as key and an ArgumentNode instance as the value. Additionally it provides an Iterator for the ArgumentNode instances which traverses the digraph in post-order, this Iterator was provided because most of the tasks the library does with the digraph consist in traversing its nodes either to gather information from them, or to set the value of the ArgumentRealtion instances once the classification has been made.

4.2.2 Package org.relationlearn.util

This package contains, as it's name implies, utility classes and enumerators that are used by other elements of the library, it also contains the two Java main classes used for training classifiers and for classifying new relations using trained classifiers.

InstanceGenerator: This class is used to generate instances that are then passed to a classifier, either to train it and get a model or to be classified. The generator needs two things in order to generate the instances: a collection of filters which implement the TextFilter interface grouped in a FilterGroup (explained in the next section) and a RelationDigraph from which the generator will extract the arguments from the ArgumentNode instances.

Listing 4: Graph instance generation code

```
package org.relationlearn.util;

import org.relationlearn.filters.FilterGroup;
import org.relationlearn.model.ArgumentNode;
import org.relationlearn.model.ArgumentRelation;
import org.relationlearn.model.RelationDigraph;
import weka.core.Attribute;
import weka.core.Instance;
import weka.core.Instances;

public class InstanceGenerator {

  //...

  public Instances getGraphInstances() {
    Instances ins = FILTER.getGroupDataset();
    Attribute classAttr = ins.classAttribute();
    for(ArgumentNode an : GRAPH) {
      ArgumentRelation relation = an.getTargetRelation();
      if(relation != null) {
        Instance in = generateInstace(an.getArgumentNodeText(),
          relation.getTarget().getArgumentNodeText());
        in.setClassValue(relation.getArgumentRelationType().toString());
        ins.add(in);
      }
    }
    return ins;
  }

  //...

}
```

As you can see the code is basically an iteration through the ArgumentNode instances contained in the digraph.

RelationClass: The RelationClass is used to define the nominal ARFF class attribute used by default when instances are generated using the SequentialFilterGroup class. Since this ARFF attribute is statically mapped we use a static initializer for it:

Listing 5: RelationClass code

```
package org.relationlearn.util;

import weka.core.Attribute;
import weka.core.FastVector;

public class RelationClass {

  public final static Attribute DEFAULT_CLASS;

  static {
    FastVector values = new FastVector(3);
    values.addElement(RelationType.ATTACK.toString());
    values.addElement(RelationType.SUPPORT.toString());
    values.addElement(RelationType.UNKNOWN.toString());
    DEFAULT_CLASS = new Attribute("default-class", values);
  }

  private RelationClass() {}
}
```

RelationType: The RelationType is an enumerator for the three different types of relations that can be found in an argumentation:

Listing 6: RelationType code

```
package org.relationlearn.util;

public enum RelationType {

  SUPPORT,

  ATTACK,

  UNKNOWN; /* Defaut value, can also mean that the relation is
              neither attack nor support */
}
```

TextUtils: The TextUtils class as it's name hints provides utility methods to parse and filter the text from the ArgumentNode instances into something more manageable by other elements of the library, because the text in the ArgumentNode instances is stored as a whole single String (which is not very useful for manipulation), with the aid of the LingPipe [4] library for natural language manipulation this class comes to aid with a couple utility methods:

Listing 7: TextUtils code

```
package org.relationlearn.util;

import com.aliasi.tokenizer.RegExTokenizerFactory;
import com.aliasi.tokenizer.TokenizerFactory;
import java.util.regex.Pattern;

public class TextUtils {

  public static final Pattern WORDS_PATTERN = Pattern.compile("\\w+");

  private static final TokenizerFactory TOKENIZER;

  static {
    TOKENIZER = new RegExTokenizerFactory(WORDS_PATTERN);
  }

  private TextUtils() {}

  public static String[] getWordsFromText(String text) {
    return getTokensFromTextUsingFactory(text, TOKENIZER);
  }

  public static String[] getTokensFromTextUsingRegex(String text, Pattern p)
      {
    TokenizerFactory tf = new RegExTokenizerFactory(p);
    return getTokensFromTextUsingFactory(text, tf);
  }

  public static String[] getTokensFromTextUsingFactory(String text,
      TokenizerFactory factory) {
    char[] ch = text.toCharArray();
    return factory.tokenizer(ch, 0, ch.length).tokenize();
  }

}
```

Using this methods it is possible to split the single String that contains the argument into an array of String using either a regular expression or using one of the TokenizerFactory classes provided by the LingPipe library. This makes for examples splitting the argument into words a piece of cake, which makes the work of the filters a much easier task.

ModelGenerator: The ModelGenerator class basically takes an input file and an output path and generates a trained model for each of the graphs found in the input file and stores them in the output path, it also requires of a filter configuration file for the process.

GraphClassifier: The GraphClassifier class takes an input file, a path to the models to be used for the classification, a path to the filter configuration files and a path in which to generate the output with a format selector and generates for each digraph in the input file that has both a valid model and a filter configuration file, the classified output in the selected format.

4.2.3 Package org.relationlearn.util.io

This specific package provides all the input-output capabilities of the library with two interfaces (one for input and one for output) and a couple of classes that provide support for different data formats.

First off we'll begin with the two interfaces contained in the package:

InputParser: This interface is meant to represent the action of parsing input data and returning in a Map of RelationDigraph instances which will represent the data contained in the input. The specification of the interface is very simple containing only one method:

Listing 8: InputParser specification

```
package org.relationlearn.util.io;

import java.util.Map;
import org.relationlearn.model.RelationDigraph;

public interface InputParser {

  public Map<String, RelationDigraph> parseInput(String uri);

}
```

As you can see the input parameter is a Java String and not a more specific object, that's because this interface is meant to represent the concept of input data and as such the best way to identify a source of data is using a combination of URL + URI which can be represented as a String, this makes the interface capable of supporting almost any input format in existence, from a single text file to a remote database URL for example.

OutputGenerator: This interface is meant to represent the action for gathering the data stored in a Map of RelationDigraph instances and storing it as persistent data. The specification is also quite simple with one method only:

Listing 9: OutputGenerator specification

```
package org.relationlearn.util.io;

import java.util.Map;
import org.relationlearn.model.RelationDigraph;

public interface OutputGenerator {

  public void generateOutput(Map<String, RelationDigraph> Map,
    String outputUri);

}
```

As you can see in the specification this interface also sets the output to a String which permits the same amount of output formats as the InputParser

interface.

Both interfaces are quite simple and at the same time allow for a great number of possible implementations making it easy for developers to define their own custom InputParser and OutputGenerator implementations. The library provides one implementation for the InputParser interface and three implementations for the OutputGenerator interface, which will be explained below:

XMLFileParser: This class which implements the InputParser interface receives the path to a XML file which is coded using the NoDE [6] format and generates a RelationDigraph for each different topic found in the file. In order to parse the file a SAX [7] parser is used instead of the more common DOM parser approach. The reason for it is that, in real-world environments, the XML file containing the data could have thousands of argument pairs making the file quite hefty, because of that a DOM parser would be too inefficient for parsing because, before being able to traverse the DOM tree, the parser would have to load the entire file in memory which would also take some time, as opposed to that, the SAX parser streams the file and when a tag is opened or closed it fires and event which is processed by the parser, making the problem of file size irrelevant both in memory consumption and unnecessary parsing time to load it.

The only problem with SAX parsers is that their implementation tends to require more coding which also is more complex to write than its DOM variants, but the benefits of a good SAX parser made it totally worth it in this specific scenario.

XMLFileGenerator: This class implements the OutputGenerator interface and it's the output counterpart of the XMLFileParser class, that is, it takes a Map of RelationDigraph instances and writes them in an XML file following the NoDE [6] format. The file is written using a streaming XML writer in order to be able to generate big files without memory issues.

DOTFileGenerator: This class also implements the OutputGenerator interface and generates a set of files in the DOT [3] format used for graphic representation of graphs. Because of some limitations in the library used to generate the output, this class will generate a file for each RelationDigraph found in the input parameter as opposed to the XML generator which generates a file that contains all the RelationDigraph instances found in the input parameter.

DLVFileGenerator: This class is the last provided implementation of the OutputGenerator interface, as with the previous generator this class also generates a file for each RelationDigraph instance found in the Map, because the output format is meant to work with one digraph per file. This format is used in argument reasoning systems such as DLV [2]. The specification of the format output can be found in the ASPARTIX [1] site.

4.2.4 Package org.relationlearn.classifiers

The classifiers package only contains one class: SVMClassifier, this class basically encapsulates the LibSVM class found in the Weka [8] library with a dataset and provides simplified methods for training, testing and classifying instances. Of course if a developer needs a broader scope for its machine learning application it can discard the use of this class and use whatever fits his needs (for example if he doesn't need to use a Support Vector Machine based classifier).

4.2.5 Package org.relationlearn.exception

This package contains wrapper classes that extend the Java Exception class, they are used basically, to provide the user with easier-to-understand information when some of the library classes encounter an error. The two Exception wrappers found in the package are:

- The **AlreadyExistingNodeException** which is thrown by the DigraphImpl class when you try to add a node with an already existing identifier to the digraph.
- The **ClassifierNotTrainedException** which is thrown by the SVMClassifier class when you try to classify an Instance with an untrained instance of SVMClassifier.

4.2.6 Package org.relationlearn.filters

The filters package contains all the interfaces and classes related to the modular filter system which will be explained in great detail in the next section, here we'll only name its two interfaces and what they represent.

FilterGroup: The FilterGroup interface defines the concept of a filter container in which you add filters that will be used when creating instances using that FilterGroup. The interface specification is the following:

Listing 10: FilterGroup specification

```
package org.relationlearn.filters;

import weka.core.Attribute;
import weka.core.FastVector;
import weka.core.Instance;
import weka.core.Instances;

public interface FilterGroup {

  public String getGroupDatasetName();
  public Instances getGroupDataset();
  public FastVector getGroupAttributes();
  public Instance createInstanceUsingFilters(String r, String h);
  public void addFilter(TextFilter filter);
  public void addClassAttribute(Attribute attr);

}
```

As you can see the interfaces provides methods for adding new filters and for setting the class attribute of the Instances that will be generated when

filtering text.

TextFilter: The TextFilter interface defines an element which receives to texts as a single String and returns a value mapped to the ARFF attribute the interface defines. Based on that the specification for the interface is pretty simple:

Listing 11: TextFilter specification

```
package org.relationlearn.filters;

import weka.core.Attribute;

public interface TextFilter {

  public Attribute getMappedAttribute();
  public double filter(String r, String h);

}
```

As you can see we basically have two methods, the first one should be used by the FilterGroup implementation to obtain the ARFF attribute this interface defines and add it to its internal ARFF header. The other method returns the value for that Attribute based on the input parameters.

4.3 The Filter System

In this section we'll explain the modular system used to convert the arguments of two ArgumentNode instances into a Weka Instance ready to be used either for training a classifier or to be classified by a trained classifier. This system has a handful of strengths compared to a more "direct" approach, which we'll enumerate below:

- Modularity: You can have thousands of different filters, but you can choose the ones that fit better your needs for a particular dataset.
- Scalability: You can easily define your own filters and filter groups and integrate them with the rest of the library elements without any hassle.

- Usability: You can use a configuration file to specify which filters you want to use for a given execution, those filters could be on your local drive, or in a remote server saving both space and complexity in any project that used the library since you could simply have a filter server from which your client applications would get the filters when they needed them.

In short, the main strong point of this system resides in its ability to adapt your needs and provide an easy way to expand its functionality, below you have a digram that explains the process of converting two argument texts to a Weka Instance:

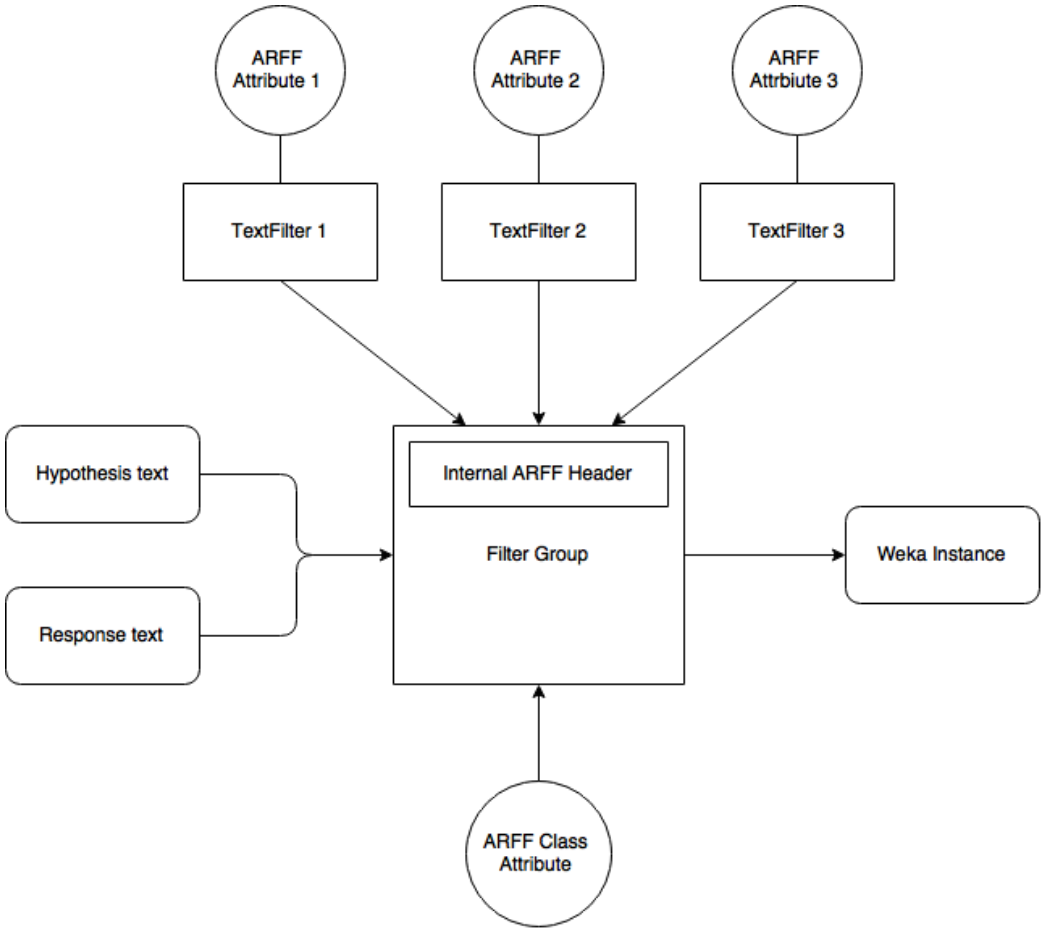

Figure 7: Filtering System Diagram

In order to better understand this diagram we'll have a very thorough look at the found in the SequentialFilterGroup class which is the implementation of the FilterGroup interface provided by the library, the reason why it is called SequentialFilterGroup will be explained later in this section.

So first we'll take a look at the class constructors and analyse what happens in them:

Listing 12: SequentiaFilterGroup constructors code

```
public class SequentialFilterGroup implements FilterGroup {

  //...

  private final String GROUP_DATASET;
  private final FastVector FILTER_ATTRS;
  private final List<TextFilter> FILTERS;

  private Attribute classAttr;

  private Instances filterDataset;

  private boolean changedFilter;

  public SequentialFilterGroup() {
    this("test-dataset");
  }

  public SequentialFilterGroup(String dataset) {
    this.GROUP_DATASET = dataset;
    this.FILTER_ATTRS = new FastVector();
    this.FILTERS = new ArrayList<>();
    this.filterDataset = new Instances(dataset, FILTER_ATTRS, 0);
    this.changedFilter = true;
  }

  public SequentialFilterGroup(String dataset, File configFile) {
    this(dataset);
    loadFiltersFromFile(configFile);
  }

  //...

}
```

The first three class attributes are in order, the dataset name which is the equivalent to the @RELATION attribute in an ARFF file, the next is the vector where the attributes from the filters are added, and the third is the filters container where the TextFilter instances are added.

Next we have the class attribute for the header, then we have the ARFF header and finally a value used when generating instances which tells us if the header needs to be rebuild (because we added more filters) before generating a new instance.

Now moving to the constructors, the first two are pretty straightforward since the just initialize the class attributes, one thing to note is the use of the ArrayList class for the FILTERS attribute, because we are going to iterate a lot through this list the ArrayList class offers superior performance compared to the LinkedList class, specially with big lists.

Finally the third constructor is the more interesting part of this code, this constructor is used when we have a filter configuration file and we want to instantiate the class and load the filters at the same time (which in real world uses is probably what everyone is going to do), to do this the SequentialFilterGroup has an internal XML parser that reads the file and tires to load the classes by its name and cast them into an instance of TextFilter with the appropriate constructor. We are not going dive into more details about how the parser works but we are going explain the format in which this filter configuration files have to be written in the following subsection.

4.3.1 Filter configuration file

Filter configuration files are basically XML files with tags and attributes that inform the XML parser of a couple things when trying to load a filter. First we'll have a look at an example file with all the possible configurations:

Listing 13: Filter configuration file format

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<filters>
  <filter id="1" local="yes" relative="no" classpath="file://your/absolute/
      path/here">
    class.fully.qualified.name
  </filter>
  <filter id="4" local="yes" relative="yes" classpath="your/relative/path/
      here">
    class.fully.qualified.name
    <filter_params>
      <param>your_param_value_here</param>
    <filter_params>
  </filter>
  <filter id="3" local="no" classpath="your_url_here">
    class.fully.qualified.name
  </filter>
</filters>
```

As you can see there are three different types of filters you can specify with the configuration file: local filters with absolute path, local filters with relative path and remote filters which require an URL. The filter cannot surpass directories or URLs that require some sort of authentication, this is obviously a disadvantage but some compromises had to be made because of time constrictions and excessive complexity that would've come to in the code.

Apart from the location of the file, you also have to specify its priority in the ARFF header, that is, the order in which the filter's attributes will be put in the ARFF header in the SequentialFilterGroup, the priority is set from negative numbers towards positive so, for example -23 has more priority than -12 which also has more priority than 4, although this can be an advantage when editing configuration files it is advisable to order the filters correctly from the beginning for clarity's sake (so begin with for example 1, 2, 3...).

Also if you want to call a constructor different from the default one, you can do it with the **filter_params** and **param** tags, keep in mind that the only parameters you can pass through this method are Strings so check the TextFilter implementation constructors before adding it to a filter configuration file to avoid problems.

Now that we have explained the constructors and the filter configuration file, lets move to the process of adding a filter the SequentialFilterGroup:

Listing 14: SequentialFilterGroup filter addition code

```
public class SequentialFilterGroup implements FilterGroup {

  //...

  @Override
  public void addFilter(TextFilter filter) {
    changedFilter = true;
    FILTER_ATTRS.addElement(filter.getMappedAttribute());
    FILTERS.add(filter);
  }

  //...

}
```

So as you can see, the first thing that happens when we add a filter is setting the changedFilter variable to true, this tells the SequentialFilterInstance that the internal ARFF header needs to be rebuild before generating an instance since the total number of attributes has changed.

After that we add the filter Attribute to the attribute vector using the TextFilter method getMappedAttributes, which return the Attribute this filter gives a value for.

Finally we add the TextFilter object to the TextFilter list in order to use it when generating an Instance.

We have the filters and their attributes added now, but what about the class attribute?, below we have the code for adding it to the SequentialFilterGroup:

Listing 15: SequentialFilterGroup class attribute code

```
public class SequentialFilterGroup implements FilterGroup {

  //...

  @Override
  public void addClassAttribute(Attribute attr) {
    if(attr == null) {
      this.classAttr = RelationClass.DEFAULT_CLASS;
    } else {
      this.classAttr = attr;
    }
  }

  //...

}
```

So basically you can add your own class Attribute to the filter group or use the one already provided by the library code passing null as the argument for the method call, pretty simple isn't it? Now we'll go to the process of creating an Instance from two text arguments:

Listing 16: SequentialFilterGroup instance creation code

```
public class SequentialFilterGroup implements FilterGroup {

  //...

  @Override
  public Instance createInstanceUsingFilters(String r, String h) {
    buildInstances();
    Instance instance = new Instance(filterDataset.numAttributes());
    for(TextFilter filter : FILTERS) {
      instance.setValue(filter.getMappedAttribute(), filter.filter(r, h));
    }
    instance.setDataset(filterDataset);
    return instance;
  }

  //...

}
```

So the first thing that happens when creating a new instance is the call to the private method buildInstances:

Listing 17: SequentialFilterGroup ARFF header creation code

```
public class SequentialFilterGroup implements FilterGroup {

  //...

  private void buildInstances() {
    if(changedFilter) {
      FastVector completeAttr = new FastVector(FILTER_ATTRS.size() + 1);
      completeAttr.appendElements(FILTER_ATTRS);
      completeAttr.addElement(classAttr);
      filterDataset = new Instances(GROUP_DATASET, completeAttr, 0);
      filterDataset.setClassIndex(completeAttr.size() - 1);
      changedFilter = false;
    }
  }

  //...

}
```

This method is the responsible for building the internal ARFF Header for the filter group, as you can see the first thing the method does is check if the group filter has changed (by adding a new filter) since the last time the method was called, if that's the case then we create a FastVector containing all the Attributes from the filters plus the class attribute, the we create a new Instances object (this is the object that represents the ARFF Header in the Weka library), set the class index to the last attribute and mark the filter as unchanged (so further calls won't repeat the process unless a new filter is added).

Now, back to the instance generation, we simply create a new Instance object with the number of attributes equal to the ARFF header (the filterDataset object), and then we iterate through the filter list filling each attribute with the value the filter gives us, once that process is completed we link the Instance with the ARFF Header (this is necessary for classification), and finally, we return the instance.

So as you have seen the filter group makes it easy to translate from arguments in a debate to instances that can be used either for classification or training later. Since you have this system of modular filters which at the same time create a modular ARFF header which changes according to the filters you've set previously this class is one of the most powerful tools of the library in terms of functionality provided to the user.

One thing I wish to discuss before ending this section is the name of the class: SequentialFilterGroup; why did I choose that name instead of, for example GroupFilterImpl?, well if you've seen how the system works when creating an instance from two text arguments, you know that the process is strictly sequential: first we gather the value for the first filter, then once that is done the value for the second an so on. One of the things I though would greatly improve the system, specially in large environments (think about hundreds of very complex filters), would be to simultaneously gather more than just one value, because each filter is independent from the others and Java strings are immutable so the input parameters wouldn't be affected by simultaneous manipulation, I thought of calling that class ConcurrentFilterGroup, as opposed to the current SequentialFilterGroup I didn't implement it (although if the project sees some real use I probably will), because of time constrictions basically and in that moment it wasn't a priority but anyway that's the reason why this class is called SequentialFilterGroup, because it was supposed to have a "brother" which would do things a bit differently.

One more thing you might have noticed is that, although we define the class attribute for the header and for any instance created, we never set a value to that attribute (which means that when an instance is created this value is marked as undefined), this is because, if you were using an already defined relation value for that pair of arguments you wanted to convert to an instance once you received the instance from the method call you would simply set its class value to whatever value you had in the relation, alternatively if you didn't have a value for that relation you would just mark the value as missing (since that value is the one the classifier will give you later).

In the following section we'll discuss and example of usage for the library.

4.4 Example Usage

Here we'll take a look at how all code explained in the previous sections comes together to either train a classifier with an already labelled argumentation file, or to classify an argumentation file with unknown values for the argument relations. First we'll go with the training code:

Listing 18: Example model generation code

```
// We create and XML parser and get a Map of digraphs
InputParser parser = new XMLFileParser();
Map<String, RelationDigraph> in_digraphs = parser.parseInput("
    your_xml_file.xml");
// We get the digraph form which we want to train a classifier
RelationDigraph digraph = in_digraphs.get("digraph_name");
// We instantiate a FilterGroup and an InstanceGenerator to get the
// attributes we want from the digraph relations
FilterGroup filter =  new SequentialFilterGroup("digraph_name");
// Setting class Attribute to default value
filter.addClassAttribute(null);
// Adding filters to the filter group
filter.addFilter(new WordRatioFilter("word-ratio"));
// ...
InstanceGenerator generator = new InstanceGenerator(digraph, filter);
// We generate the instances and train an SVM classifier with them
Instances instances = generator.getGraphInstances();
SVMClassifier svm = new SVMClassifier();
svm.trainClassifier(instances);
// Finally we store both the classifier and the dataset in a file
svm.storeModel("digraph_classifier.model");
```

This code is used to train a model for a given digraph. Next we'll take a look at the labelling code:

Listing 19: Example instance classification code

```
// We create again and XML parser and get a Map of digraphs
InputParser parser = new XMLFileParser();
Map<String, RelationDigraph> in_digraphs = parser.parseInput("
    another_xml_file.xml");
// We get the digraph form which we want to classify its relations
// with an already trained classifier
RelationDigraph digraph = in_digraphs.get("digrahp_name");
// We instantiate a FilterGroup and an InstanceGenerator to get the
// attributes we want from the digraph relations
```

```
FilterGroup filter = new SequentialFilterGroup("digraph_name");
// Setting class Attribute to default value
filter.addClassAttribute(null);
// Adding filters to the filter group
filter.addFilter(new WordRatioFilter("word-ratio"));
// ...
InstanceGenerator generator = new InstanceGenerator(digraph, filter);
// We generate the instances and initialize a SVM classifier with a
// previously trained model
Instances instances = generator.getGraphInstances();
File model = new File("path_to_your_model.model");
SVMClassifier svm = new SVMClassifier(model);
// We classifiy each instance using the SVM classifier
for(int i = 0; i < instances.numInstances(); i++) {
  double result = svm.classifyInstance(instances.instace(i));
  // Do something with the result value...
}
```

In this code you use a model previously trained and label the instances from the digraph.

5 Evaluation and Testing

In this section we'll see an example of real world data training and classification and discuss how different classifier parameters affect the results.

First we have the training file which we'll use to train different classifier models using different parameters for the classifier. Once we have those models we'll classify the test file with each of them and compare the results in order to figure out which is the best parametrization of the classifier for that given data set.

5.1 The training file

First we'll take a peek at the training file which is coded in the NoDE [6] format:

Listing 20: Evaluation training file

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<entailment-corpus>
  <pair task="ARG" id="1" topic="Singlesexschools" entailment="NO">
    <t id="2">Having schools seperated by biological sex from an early age
        is a terrible idea.  It absolutely ignores the reality of indivduals
         who are transgendered or intersex, and creates another issue on top
         of what is alreay a difficult life.  By enforcing the "gender
        binary", the House is trying to eliminate the rest of the gender
        spectrum, which is at best done out of ignorance, at worst out of
        malice or even hatred.  This motion cannot stand, because it
        eliminates the ability of gender expression and discovery only
        capable by being around a wide variety of individuals. Learning
        within schools can be corrected without eliminating the learning
        that comes from interaction with your peers.</t>
    <h id="1">Single-sex schools are good for education.</h>
  </pair>
  <pair task="ARG" id="2" topic="Singlesexschools" entailment="NO">
    <t id="3">I do agree with your point, however I do have some counter
        agruements. Alot of people of whom i have talked to believe that
        teachers favor studnets over their sex. For example feamale teachers
         favor male students and male teachers favor feamale students. I am
        NOT stating that all teachers believe this way, but it does make
        sense. The cases between teachers and students having realtions
        outside of school. So against my better judgement I would have to
        believe that placing in students in single-sex schools rather than
        co-ed schools may be benifical for SOME students. Another point is
        how easy it is for my generation to be pressured into things such as
         drugs or sexual relations with another sex. This can be very
        helpful into decreasing the teen pregancy numbers.</t>
    <h id="2">Having schools seperated by biological sex from an early age
        is a terrible idea.  It absolutely ignores the reality of indivduals
         who are transgendered or intersex, and creates another issue on top
         of what is alreay a difficult life.  By enforcing the "gender
        binary", the House is trying to eliminate the rest of the gender
        spectrum, which is at best done out of ignorance, at worst out of
        malice or even hatred.  This motion cannot stand, because it
        eliminates the ability of gender expression and discovery only
        capable by being around a wide variety of individuals.  Learning
        within schools can be corrected without eliminating the learning
        that comes from interaction with your peers.</h>
  </pair>
  <pair task="ARG" id="3" topic="Singlesexschools" entailment="YES">
    <t id="4">Hi, This is a very intresting point. However, I believe it
        deals more with sexual abuse of the young students. Abuse of anykind
         - sexual or emotional affects the victim to some degree of mental
        trauma and affects their performance in studies, I agree. However,
        we have to understand that sexual tension between two individuals
        can occur within the same gender as well. Male teachers, in your
```

```
      example, may be interested in male students. And can abuse if they
      are such kind of a person. Hence, separating two genders is not a
      solution to end abuse. we have to get out of the mindset of gender
      binary.</t>
  <h id="2">Having schools seperated by biological sex from an early age
      is a terrible idea.  It absolutely ignores the reality of indivduals
       who are transgendered or intersex, and creates another issue on top
       of what is alreay a difficult life.  By enforcing the "gender
      binary", the House is trying to eliminate the rest of the gender
      spectrum, which is at best done out of ignorance, at worst out of
      malice or even hatred.  This motion cannot stand, because it
      eliminates the ability of gender expression and discovery only
      capable by being around a wide variety of individuals.  Learning
      within schools can be corrected without eliminating the learning
      that comes from interaction with your peers.</h>
</pair>
<pair task="ARG" id="4" topic="Singlesexschools" entailment="NO">
  <t id="5">They are not good for education because, most of our "fellow
      guys" would turn homosexual. Girls the same. Guys and girls have
      some disadvantages but mainly advantages.</t>
  <h id="1">Single-sex schools are good for education.</h>
</pair>
<pair task="ARG" id="5" topic="Singlesexschools" entailment="NO">
  <t id="6">They wouldn't turn homosexual. I'm from a single-sex school
      and I don't know anyone that is homosexual. I strongly agree with
      single-sex schools, I always found the boys at my primary mixed
      school were a big distraction from my work.</t>
  <h id="5">They are not good for education because, most of our "fellow
      guys" would turn homosexual. Girls the same. Guys and girls have
      some disadvantages but mainly advantages.</h>
</pair>
<pair task="ARG" id="6" topic="Singlesexschools" entailment="NO">
  <t id="7">I do agree with you, homosexuality is something else that has
      nothing to do with boys studying together seperatly from girls.  I
      am perfectly aware of the positive and negative aspects of single
      and eco education. but i would opte for eco-education in so far as a
       real education is the one that is supposed to teach us how to live
      together in harmony and built our society. Under no circumstances
      can we built a long standing society without both sexes. </t>
  <h id="6">They wouldn't turn homosexual. I'm from a single-sex school
      and I don't know anyone that is homosexual. I strongly agree with
      single-sex schools, I always found the boys at my primary mixed
      school were a big distraction from my work.</h>
</pair>
<pair task="ARG" id="7" topic="Singlesexschools" entailment="YES">
  <t id="8">I am strongly in for single-sex school as all the horrible
      nightmares that you can dream a student will never happen in a
      single-sex school , as single-sex school prevents students from
      sexual assualt , harrassment and etc. It also helps prevents anyone
      from dating that will there fore ensure that their results is stable
```

```
      </t>
  <h id="1">Single-sex schools are good for education.</h>
</pair>
<pair task="ARG" id="8" topic="Singlesexschools" entailment="NO">
  <t id="9">First point, which many people already mentioned is that co-ed
       schools are required for comprehensive development. True! Children
      are like moulds. They are not forced by confounding stereotypes and
      infectious attitudes as ideas and thoughts are still budding in them
      . They have a free will and are open to all sorts of possibilities.
      Aren't women going to be around men in future and the vice versa? So
      , why shield them now? Let them study together, learn, explore their
       lives together. I have seen multitudes of people who feel shy in
      talking to the person of opposite sex. The thought process, behavior
      , etc. of the opposite sex seems alien to them. How will society
      function like this? If they perceive each other wrongly in early
      stages, it is simpler to correct them. Moreover, what is imparted
      early stays till late. So, it is better to let children and
      adolescents study together and develop holistically.</t>
  <h id="1">Single-sex schools are good for education.</h>
</pair>
<pair task="ARG" id="9" topic="Singlesexschools" entailment="YES">
  <t id="10">I agree, in addtion to your argument, shilding opposite sexes
       from each other may lead to a drustic increase in potential sexual
      violences, for we human beings are sexually greedy. Restraining
      opposite sexes from each other may cause excessive desire for
      another, which ultimately leads to raping. </t>
  <h id="9">First point, which many people already mentioned is that co-ed
       schools are required for comprehensive development. True! Children
      are like moulds. They are not forced by confounding stereotypes and
      infectious attitudes as ideas and thoughts are still budding in them
      . They have a free will and are open to all sorts of possibilities.
      Aren't women going to be around men in future and the vice versa? So
      , why shield them now? Let them study together, learn, explore their
       lives together. I have seen multitudes of people who feel shy in
      talking to the person of opposite sex. The thought process, behavior
      , etc. of the opposite sex seems alien to them. How will society
      function like this? If they perceive each other wrongly in early
      stages, it is simpler to correct them. Moreover, what is imparted
      early stays till late. So, it is better to let children and
      adolescents study together and develop holistically.</h>
</pair>
<pair task="ARG" id="10" topic="Singlesexschools" entailment="YES">
  <t id="11">Single sex education is the best form of providing education
      as it does not let any one to get distracted from its study due to
      the attraction towards the opposite gender . when the gender of same
       category will study together , there is a negligible possibility of
       the student to get distracted from its studies and induldge in such
       activities which are not productive or fruitful not for him as well
       as society.if any one is arguing that it will not work bcz  if the
      same gender study together then they will hesitate in talking to the
```

```
        opposite gender when they reach proffessional level  but statistics
        say that single sex education system is more succesful than co-ed
       education system as they are more educated which make them more
       confident in dealing with proffessional difficulties. there was a
       time when co-ed schools were not there that education system was
       awesome when the students will only do thier studies and the teacher
        instill in them a feeling of  doing eveything for the development
       of the country then they will only think about how to contribute to
       the wellfare of thier society leading to a happy and a prosperous
       country with satisfied and devoted individuals towards their country
       . Just look at the statistics and ask how many harrasment cases you
       have heard in the 1970s and 1980s , almost negligible as compared to
        today . </t>
    <h id="1">Single-sex schools are good for education.</h>
  </pair>
</entailment-corpus>
```

The file contains eleven arguments forming ten relations about the topic of single sex schools, it's not a very large file but it will suffice for our testing purposes.

5.2 The test file

The test file follows the same pattern as the training one:

Listing 21: Evaluation test file

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<entailment-corpus>
  <pair task="ARG" id="1" topic="Singlesexschools" entailment="NO">
    <t id="2">Idea of single-sex school is not going to help students in
        their education. I have experienced both co-educational as well as
        single schools and I always repent "why i went to single school".
        The confidence I had in co-educational school unfortunately I did
        not continue to have in single school. Education is not about making
         anyone bookwarm or its scope is not confined to schools lectures.
        Education is to build our individuality and the presence of opposite
         sex around us help us to build our interpersonal skills also it
        helps to groom one mentally and physically.Therefore I certainly
        beleive that co-educational schools are good for education.</t>
    <h id="1">Single-sex schools are good for education.</h>
  </pair>
  <pair task="ARG" id="2" topic="Singlesexschools" entailment="NO">
    <t id="3">I think they are not good for education.</t>
    <h id="1">Single-sex schools are good for education.</h>
```

```
</pair>
<pair task="ARG" id="3" topic="Singlesexschools" entailment="YES">
  <t id="4">Hi All, I believe, single sex school or co-ed does not impact
      the personality or growth of any person in anyway. its totally
      irrelevant. They neither help, or stop the learning experience. The
      students always get exposure to the people of opposite sex outside
      the school boundaries. I think the system of same sex school was
      invented by the instituitions who wanted to shy away from their
      responsibilities and issues that accompany a diverse student group.
      An instituition that is not confident enough that they can handle
      sexual diversity in the students are essentially demonstrating their
       lack of interset in taking accountability if any issues happens to
      anyone. And hence they have chosen the easier way. If I were a
      parent, I would trust that instituition more which has confidence in
       itself and is ready for accountability.</t>
  <h id="1">Single-sex schools are good for education.</h>
</pair>
<pair task="ARG" id="4" topic="Singlesexschools" entailment="NO">
  <t id="5">So what would happen to transgendered children? exactly.
      HORRIBLE idea.</t>
  <h id="1">Single-sex schools are good for education.</h>
</pair>
<pair task="ARG" id="5" topic="Singlesexschools" entailment="NO">
  <t id="6">Well... I rather think that single-sex education is not bad
      but I really wouldn't recommend it. That's because uni-sex schools
      or education teaches also about the opposite sex and gives you
      experience on how later society will be when there is no uni or
      single sex education but a working place which is in most cases
      going to be with the opposite sex. And I don't think experience
      would be bad even if it could have a bit of trouble here and there
      but it probably would be better than not having any idea what the
      opposite sex would be like before one goes into society.</t>
  <h id="1">Single-sex schools are good for education.</h>
</pair>
<pair task="ARG" id="6" topic="Singlesexschools" entailment="YES">
  <t id="7">Surely then co-ed schools are good because they educate you
      for later life when you are in a co-ed society. Also maturing faster
       but in the wrong type of society is often worse for you than
      maturing maybe a bit slower but still with the other sex. There is
      no point making barriers when they will be destroyed in later life.
      Also when sexes have none or little contact with the other sex they
      find it hard to interact with the other sex, there is a stereo type
      about pupils in Eton being very strange around women this stereotype
       is not because evryone hated people in Eton but after they left the
       school lots of the pupils had a hard time interacting with women
      becasue the were in a co-ed school, which went against them in later
       life.</t>
  <h id="6">Well... I rather think that single-sex education is not bad
      but I really wouldn't recommend it. That's because uni-sex schools
      or education teaches also about the opposite sex and gives you
```

```
        experience on how later society will be when there is no uni or
        single sex education but a working place which is in most cases
        going to be with the opposite sex. And I don't think experience
        would be bad even if it could have a bit of trouble here and there
        but it probably would be better than not having any idea what the
        opposite sex would be like before one goes into society.</h>
  </pair>
</entailment-corpus>
```

This file contains seven arguments which form six relations, again the topic is the same as the training file, or else the model would be pretty much useless for classification if the topics where too different.

5.3 The filter file

In order to obtain a model we need to use filter to gather information form the arguments found in the files, below there's the configuration file used for the filters when generating models and creating instances for classification:

Listing 22: Evaluation filter configuration file

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<filters>
  <filter id="1" local="yes" relative="yes" classpath="">
    org.relationlearn.filters.WordRatioFilter
    <filter_params>
      <param>word-ratio</param>
    </filter_params>
  </filter>
  <filter id="2" local="yes" relative="yes" classpath="">
    org.relationlearn.filters.PunctuationCounterFilter
    <filter_params>
      <param>punctuation-count</param>
    </filter_params>
  </filter>
  <filter id="3" local="yes" relative="yes" classpath="">
    org.relationlearn.filters.WordSearchFilter
    <filter_params>
      <param>word-search-no</param>
      <param>no</param>
    </filter_params>
  </filter>
  <filter id="4" local="yes" relative="yes" classpath="">
    org.relationlearn.filters.CommonWordsFilter
```

```
    <filter_params>
      <param>comm-words</param>
    </filter_params>
  </filter>
  <filter id="5" local="yes" relative="yes" classpath="">
    org.relationlearn.filters.WordOcurrenceFilter
    <filter_params>
    <param>word-occurrence</param>
      <param>words.lst</param>
    </filter_params>
  </filter>
</filters>
```

We use all the available filters from the library with the parameters set to match the topic of the debate in order to maximise the efficiency of the information gathered from the arguments.

The final filter has as parameter a serialized Java List instance containing relevant words to search in the arguments.

5.4 The classifier

As we explained in section 3, the Support Vector Machines have multiple ways of operating, everyone of them is suited to a given task, so we will try different settings and see which one adjusts better to the test set. From the Javadoc page of LibSVM:

Listing 23: LibSVM configuration parameters

```
Valid options are:

-S <int>
 Set type of SVM (default: 0)
   0 = C-SVC
   1 = nu-SVC
   2 = one-class SVM
   3 = epsilon-SVR
   4 = nu-SVR


-K <int>
 Set type of kernel function (default: 2)
   0 = linear: u'*v
```

```
    1 = polynomial: (gamma*u'*v + coef0)^degree
    2 = radial basis function: exp(-gamma*|u-v|^2)
    3 = sigmoid: tanh(gamma*u'*v + coef0)


-D <int>
 Set degree in kernel function (default: 3)


-G <double>
 Set gamma in kernel function (default: 1/k)


-R <double>
 Set coef0 in kernel function (default: 0)


-C <double>
 Set the parameter C of C-SVC, epsilon-SVR, and nu-SVR
  (default: 1)


-N <double>
 Set the parameter nu of nu-SVC, one-class SVM, and nu-SVR
  (default: 0.5)


-Z
 Turns on normalization of input data (default: off)


-J
 Turn off nominal to binary conversion.
 WARNING: use only if your data is all numeric!


-V
 Turn off missing value replacement.
 WARNING: use only if your data has no missing values.


-P <double>
 Set the epsilon in loss function of epsilon-SVR (default: 0.1)


-M <double>
 Set cache memory size in MB (default: 40)


-E <double>
```

```
  Set tolerance of termination criterion (default: 0.001)


-H
  Turns the shrinking heuristics off (default: on)


-W <double>
  Set the parameters C of class i to weight[i]*C, for C-SVC
  E.g., for a 3-class problem, you could use "1 1 1" for equally
  weighted classes.
  (default: 1 for all classes)


-B
  Generate probability estimates for classification


-seed <num>
  Random seed
  (default = 1)
```

As you can see these parameters allow you to do a very extensive fine tuning of the classifier which can help a lot in boosting the accuracy of trained models.

Of course not all the parameters have the same impact in the results but nevertheless all of them have their importance in expert hands.

5.5 Test results

Once we got all data gathered we began the testing with different parameters for the classifier:

Results					
Test	Parameters	#args	#yes	#no	accuracy %
Original		6	2	4	100
Test 1	-S 0 -K 0	6	1	5	50
Test 2	-S 1 -K 0	6	1	5	50
Test 3	-S 0 -K 1	6	0	5	66.67
Test 4	-S 1 -K 1	6	2	4	33.34
Test 5	-S 1 -K 1 -D 2	6	1	5	50

Table 2: Testing results

THe first line doesn't represent a test, it represents the results obtained directly from the test file which are obviously 100% correct. The rest of the lines are tests made with different parameters for the classifier.

As you can see even with this tiny data set we've used we still have some variation with different classifier parameters. The objective of this test was not to show how to fine tune the classifier for a given dataset, but to prove you, the reader, that this sort of classification requires a lot of testing time before deploying the trained classifier to work with real data in a production environment.

6 Conclusion and Future Work

For the conclusion I'll like to ask you, the reader, a question. Do you think this kind of technologies represent the future of the interaction between man and machine? If you ask me for my opinion then my answer is a clear yes. In the future this kind of technologies that make the machines seem more "human" we'll just keep increasing in both number and quality, I just hope that this project has opened your mind at least a bit about what a computer could do for you in the near future.

Regarding the current state of the project, I'm quite satisfied with the results, although I'll admit that there's still room for improvements (there always is),

so I'll take a bit more of your time detailing possible improvements that could be made to the project:

- Implement the ConcurrentFilterGroup idea from section 4.3.
- Improve the efficiency and capabilities of the DigraphImpl iterator.
- Expand the capabilities of the filer configuration file syntax.
- Move the code to the last version of Java and use some of its new features (like lambdas) to improve code efficiency and readability.

This are just some of the possible improvements in the project, and since the project is licensed under the GPL license, you could be the one making those changes a reality.

Finally, my only hope for this project is that it seems some use (not much, maybe as a test of concept) but that someone one day tries to use it and helps make it better.

That's all I have to say, thank you for taking your time reading this document for my project!, hope you enjoyed your read as much as I enjoyed making the project a reality.

References

[1] ASPARTIX Documentation. http://www.dbai.tuwien.ac.at/research/project/argumentation/systempage/docu.htm. Accessed: 25 June 2015.

[2] DLVSYSTEM S.R.L. http://www.dlvsystem.com/. Accessed: 25 June 2015.

[3] DOT (graph description language). https://en.wikipedia.org/wiki/DOT_%28graph_description_language%29. Accessed: 25 June 2015.

[4] LingPipe - A toolkit for text processing. http://alias-i.com/lingpipe/. Accessed: 25 June 2015.

[5] Machine learning. https://en.wikipedia.org/wiki/Machine_learning. Accessed: 23 June 2015.

[6] NoDE - A Benchmark of Natural Argumentation. http://www-sop.inria.fr/NoDE/index.html. Accessed: 25 June 2015.

[7] Simple API for XML. https://en.wikipedia.org/wiki/Simple_API_for_XML. Accessed: 25 June 2015.

[8] Weka 3: Data Mining Software in Java. http://www.cs.waikato.ac.nz/ml/weka/. Accessed: 25 June 2015.