Budapesti University of Technology and Economics
Faculty of Engineering and Informatics
Department of Telecommunication and Mediainformatics

Balázs Várhegyi

DEVELOPING OPERATORS IN RAPIDMINER TO MANAGE PYTHON BASED ANALYTICAL METHODS

ACADEMIC SUPERVISOR

Gábor Nagy

BUDAPEST, 2014

Table of contents

HALLGATÓI NYILATKOZAT

Alulírott **Várhegyi Balázs**, szigorló hallgató kijelentem, hogy ezt a diplomatervet meg nem engedett segítség nélkül, saját magam készítettem, csak a megadott forrásokat (szakirodalom, eszközök stb.) használtam fel. Minden olyan részt, melyet szó szerint, vagy azonos értelemben, de átfogalmazva más forrásból átvettem, egyértelműen, a forrás megadásával megjelöltem.

Hozzájárulok, hogy a jelen munkám alapadatait (szerző(k), cím, angol és magyar nyelvű tartalmi kivonat, készítés éve, konzulens(ek) neve) a BME VIK nyilvánosan hozzáférhető elektronikus formában, a munka teljes szövegét pedig az egyetem belső hálózatán keresztül (vagy hitelesített felhasználók számára) közzétegye. Kijelentem, hogy a benyújtott munka és annak elektronikus verziója megegyezik. Dékáni engedéllyel titkosított diplomatervek esetén a dolgozat szövege csak 3 év eltelte után válik hozzáférhetővé.

Kelt: Budapest, 2014. 05. 10.

...……

Várhegyi Balázs

Összefoglaló

Az adatbányászat területén alkalmazott két rendkívül széles körben használt technológia közül az egyik a RapidMiner szoftver, mely az adatbányászati funkcióit tekintve világvezető nyílt forráskódú rendszer, a másik a Python, egy általános célú magas szintű programozási nyelv, mely ugyancsak széleskörű adatbányászati moduljainak köszönhetően egy gyakran használt eszköz az adatok elemzésekor. A feladatom e két rendszer erősségének kombinálása volt, ezáltal új adatbányászati funkciókat nyújtva a RapidMinert használó adatelemző specialistáknak. A munkám során egy közös interfészt kellett találnom, mely egyfajta hidat képez a RapidMiner és Python között, ezáltal a RapidMiner képes arra, hogy a gyakran használt, megbízható python modulokat integráljon az adatbányászati folyamatokba.

A munkám részben a megfelelő technológia megtalálása volt, mely képes arra, hogy a két rendszer közötti kölcsönös kommunikációt megvalósítsa. A RapidMiner operátor fejlesztéséhez a Java programozási nyelvet, míg a kommunikációban résztvevő python objektumok létrehozásához a Python programozási nyelvet használtam. A munkám során fejlesztés kísérleti jellegéből fakadóan illetve az eredmények publikálása céljából számos alkalommal folytattam levelezést a RapidMiner és egyéb projektek fejlesztői csapatával. A fejlesztés során kihasználtam a RapidMiner által biztosított nyílt forráskód adta előnyöket, valamint egy másik nyílt forráskódú projekt, a Pyrolite funkcióit, továbbá nagyban építettem a mások által korábban implementált gépi tanulást megvalósító python könyvtárra, nevezetesen a scikit-learn-re. A munkám eredményeként a RapidMiner által nyújtott funkciók tovább bővültek, ezen kívül további hiányzó funkciók implementálhatók, valamint köszönhetően a magas teljesítményt nyújtó cpython könyvtáraknak a futási idő is csökkenthető. Mivel az operátor és a dokumentáció nyilvánosan is elérhető lesz, így a projekt dokumentációja értékes know-how-ként használható további, hasonló struktúrájú operátorok készítéséhez, az elkészült operátor pedig egy hasznos bővítménye a RapidMiner eddigi funkcióinak.

Abstract

There are two very popular data mining technologies, one of them is RapidMiner, the world-leading open source system for data mining and the other is Python, a general purpose high level programming language, which has a wide range of data mining modules. My task was combining these two technologies in order to provide a new approach of data mining with RapidMiner. I had to find an interface that creates a bridge between RapidMiner and Python, therefore RapidMiner is able to integrate the widely used python modules into its data mining process.

My work consists of looking for the right technology in order to create an appropriate interface between these technologies. Developing RapidMiner operator in Java programming language and developing python objects in Python programming language were also core parts of my tasks. In addition my work included conversations with foreign developers of RapidMiner and other projects' because of the new approach of the problem and because of the publishing of the results. During my work I took advantage of open source code of RapidMiner, and the source code of another open source project called Pyrolite and the analytical methods of a python machine learning library called scikits-learn. So consequently RapidMiner's data mining features has been expanded, and missing functions can be implemented and thanks to the high performance of these third party cpython modules the execution time may decrease. The documentation of the project can be used as a valuable know-how and the created project is a useful extension of RapidMiner's features, because the project and the documentation will be published.

1 Inroduction

1.1 Data mining technology

Data mining is an interdisciplinary technology, a subfield of computer science with the combination of artifical intelligence, machine learning, statistics and database systems. Due to the data that comes from everywhere: posts, shares and likes on social media sites, digital pictures and videos, purchase transaction records. We create a huge amount of data every day. Companies from a large scale of industries and governments can use their data as an asset to explore and understand phenomenons and general patterns to make fact-based decisions for future actions. The key factor of data mining is to detect or discover something new, a meaningful pattern that can be used in business decisions to manage future actions in a professional way. The applications of these techniques depend on the variation of different industries and governments. Healthcare and pharmaceuticals can take advantage of data mining to facilitate medicinie effectiveness. Banks can support their security by detecting fraud in their financial systems. Many kind of companies can use it for marketing campaigns to achieve enhanced responsiveness. In general, organizations can improve their business performance as a result of decisions that based on proven data mining models.[1]

There are many steps involved until we can make a predictive insight based on the initial data. The common steps in data mining includes sampling the big data into a smaller part that can be used for processing instead of alanyzing the whole data that is rather unfeasible and time-consuming. Also an important phase to explore the data and looking for relationship between the attributes and concieve a comprehensive understanding of the data. Then we need to modify the data, for example by looking for missing values and detecting outliers, before we can continue with the modeling phase where the analytical techniques used in order to search for a combination of attributes that solve our problem. In the last section of the process we validate the outcome of the modelling to avoid „overfitting”. This is a possible result of our data mining process, when we found a pattern that performs well on the sample data but doesn’t fit for the general dataset. To prove our results reliablity we test and evaluate our model against a test dataset that was not involved in the previous modeling process. Then we can use the proven data model to facilitate business decisions.[2] During this process analysts can

use a wide range of data mining tools. These data mining tools offer utilities to support data analysts and scientists during this progress.

1.2 Data mining tools

Creating effective data mining models without proper software tools is not viable in real life situations. These applications provide support for all the phases of the data mining process. These data mining tools can differ significantly on their usability. On one hand, there are complete solutions that fully cover all the mentioned steps, with full and immediate support for companies dealing with data mining. On the other hand there are libraries and modules that can cooperate to cover whole process flow. Finding the best solution might depend on our specific goals and requirements. Just to mention a few example, there are techniques especially for web crawling, processing texts, and some of them are especially for images. All these softwares can be categorized into two main groups.[3]

There are commercial data mining softwares and applications which require licenses to use them. Usually these applications has the advantage of handling specific requirements that are common in many real life situation. The continuous development ensured by a closed group of developers and secured by the paying clients. Clients also can expect reliable solutions and help for their specific needs. The largest market-share holder for advanced analytics currently is SAS (Statistical Analysis System), a software suit developed by SAS Institute.[4][5]

However, there are free open-source data mining softwares and applications. These projects has the advantages that they also have a community behind it, who take care of improving the projects but these developers are usually work only technically as a team. They might not be even in an office but stay in touch by emails and messages. People from all over the world can declare feature requests that might get implemented by the community. Developers can join to the community, that is much more open than teams of commercial applications, and can contribute to these project and the community merge the new feature into the core project. Users can benefit from the open source code since they can analyze the source code in case of a suspicious results or simply for better understanding. RapidMiner is a great example of this group but there are recent changes that made it a hibryd version of these two categories.[6][7] I will delve into the details of the recent changes in a later section (Chapter 8.) because this

topic is related to publishing my work. Rapidminer is an integrated environment used for statistical analysis, data mining predictive analytics. Succesfully used in business and industrial applications as well as for education and research reasons. At the time of this documentation RapidMiner latest stable version is 6.0 and was released on the 21st of November 2013.[7]

1.3 Commonly used python modules for data mining

Python is a widely-used programming language that is used by many data analysts during their development and modelling process. There are many libraries that are implemented in Python by developers and used by data analysists for data mining. The libraries are results of mutual work that consists of the users or data analysts and the developers of the libraries, that in some cases are the same people. Each library has its own community of developers that gives support in case of errors and implementation flaws that is usually reported by the data analysts. Also users of the libraries can define implementation requests that they would like to use in their data mining process or can implement their own libraries. Most part of the currently existing libraries were also started by only a few developers who shared their project on a site like sourceforge.net, then people could join to the community and get involved in the development process. Here is a list of some well-known python libraries that are used in data mining [8]:

1. Basics modules that are used in data mining as part of data modeling
    - Numpy: numerical library is the fundamental package for scientific computing [9]
    - Scipy: advanced math, signal processing, optimization, statistics [10]
    - Matplotlib: 2D plotting library for publication quality figures
2. Machine Learning and Data Mining
    - MDP: data pocessing framework
    - Mlpy: provides a wide range of machine learning methods that include a collection of supervised and unsupervised learning algorithms [11]
    - NetworkX: software package for creation, manipulation, and study of the structure, dynamics and functions of complex networks [12]

- Orange, Data Mining Fruitful & Fun: open source tool, used for data visualization and analysis for novice and experts including: components for machine learning and add-ons for bioinformatics and text mining. [13]
- Pandas: high-performance, easy-to-use data structures and data analysis tools [14]
- Pybrain, is short for **Py**thon-**B**ased **R**einforcement Learning, **A**rtificial **I**ntelligence and **N**eural Network Library [15]
- **Scikits-learn**: Python module that I integrated into RapidMiner, contains classic machine learning algorithms. Provides simple and efficient solutions to learning problems. Since it is the integrated python modul, there's a more detailed information in Chapter 6.

3. Natural Language

   - Nltk, Natural Language Toolkit: contains functionality for working with human language data. [16]

4. Web scraping

   - Scrapy, An open source web scraping framework for Python, used to crawl websites and extract structured data from their pages [17]
   - Urllib, a high-level interface for fetching data across the internet [18]

Most of these libraries are licensed by BSD license that is a permissive free license with minimal restriction regarding to the redistribution of the covered software. So they can be used as components of other projects.

1.4 New approach

In my thesis I will extend an open-source software and expand the toolkit of RapidMiner's data mining techniques with my custom-defined operator, which are implementations of our missed or thought-out analytical techniques. With this feature Rapid-Miner pushes the advantage of open-source projects against the profit-oriented program solutions, and increase the number of analytical technologies at the same time. During the process I will demonstrate the usage of the operator which is capable for using cpython libraries during its process. I will introduce the needed technologies and the phases of configuring them and the process of creating such an operator and finally

the usage of it. The operator core task is performing data mining analytical methods. During the analysis it connects to a python server, uses the python libraries to calculate the result for the operator and finally the operator provides the results for the user. I will show in depth how the operator works, the usage of the operator, for example how to set the parameters and how to provide the requested output. I will introduce a short data mining model to prove that how well the operator is integrated with other operators of RapidMiner and can be used in cooperation with them including all the RapidMiner features like plotting the results. I will also publish a part of the code to provide an example of the operator and the structure of the project to other developers. This way developers can analyze it and reproduce the achieved result or make their own implementations based on this idea.

2 Configuration

When we implement operators in RapidMiner, first we have challenges with the configuration of the system. Basically there are three elements that we use in order to extend RapidMiner features.

1. the source code of RapidMiner
2. an extension package and documentation, that we can buy from RapidMiners web page (under Documentations/RapidMiner/Manuals/RapidMiner Extension Guide) [23]
3. Java Development Kit (Eclipse).

In this case I've used two extra elements, which are needed because of the special mechanism of the operator.

4. Python interpreter
5. PyroLite project

2.1 Configuring RapidMiner

Configuring RapidMiner from source code is usually a challange for me because of the lack of the information about it. First I had to download the RapidMiner source code, following the instructions on RapidMiner's web page. [25] I usually encountered with problems in this phase. My initial configuration was:

- OS: Ubuntu12.10
- Eclipse: window/preferencies/installed jre: /usr/lib/jvm/java-7-openjdk-i386/jre
- Eclipse project:
    - jre-system-library: java-7-openjdk-i386
    - compiler / compliance level:1.7
    - projectfacets / java:1.7
- antbuild: "clean, copy-resources, build, createJar"

After I checked out the SVN repository of RapidMiner, I had some warnings on the Eclipse's "Markers" tab like:

```
"/RapidMiner_Unuk/lib/blas.jar will not be exported or published. Runtime
ClassNotFoundExceptions may result."
```

Although I've already set the needed jar files on Java build path. When I have built the code I had 84 errors. I wrote about it in the RapidMiner forum and they pointed out, they are using Oracle Java7 (Sun JVM), and compile against java 6 (It set in the build.xml file). [24] Later I created a configuration with the JDK provided by Oracle as they suggested and all the errors were solved.

Part of the problem was the com.rapidminer.doc package, because I've deleted and 70 error disappeared, but I still had 14. Finally I was able to compile it. I had to put into comments further parts from SQLQueryBuilder.java. I don't want to use it, so it might not a problem. It said "cannot find symbol", (classes were not found in jar file) when I tried to attach source to them it tried to load it from rt.jar (from open-jdk's lib), so most likely the difference between the jdk-s (open-jdk and Oracle's jdk) caused the problem. If you run the build file with these targets: clean, build, createJar then the rapidminer.jar is created. This is what you need. Now you can start the program, if you run the Launcher.java file from src_launcher package.

Important notes

- The main advice is to keep solving errors, e.g. „cannot find symbol", until you can build the source.
- the svn link I used at the time of writing this documentation is: svn checkout http://svn.code.sf.net/p/rapidminer/code/Unuk RapidMiner_Unuk
- Maybe you have to set the main class"com.rapidminer.gui.RapidMinerGUI" in „Run configurations"

2.2 Configuring RapidMiner Extension

After I configured the RapidMiner successfully I imported the Extension template project. You should take the steps by following the White Paper Documentation. However it can be misleading because there's no documentation about the newer version (Unuk) only for the previous one (Vega), so do the next steps:

- Copy the Template Project next to the RapidMiner_Unuk Project.
- Import the template project, 'File/New/Java Project/Choose location where Template and Unuk Projects are and give the Project name Template Project. Then click Finish.

Because my extension is for Vega I replace all the occurrence of 'Vega' to 'Unuk' with 'Search/File search/ give the containing text: '*Vega*' and give the searching scope: Choose/New/Resource/ check the project and give a name for it'. Fortunately if you save this file or the whole extension project, you don't need to it again. About a half year ago, you would have needed to change the Unuk Projects's build extension file with the Vega's build extension file but now you don't have to change it because I think it's been fixed.

2.3 Configuring Python

At this point you should be able to build RapidMiner Extension project from Eclipse. I recommend to create a dummy operator in order to test that you can create a custom operator, that appears in RapidMiner's graphical user interface (GUI). The next step is going to be to create the appropriate python environment. The operator will use this python interpreter that we are going to setup now. The first crucial point is, if you have an installed python version on your system, you have to check its Universal Character Set is ucs4. On Linux, Python installations default to the unicode representation of the OS environment (ucs2 vs ucs4). Once you have installed it, a version of python with ucs4 will produce the following:

```
varh1i@varh1i-F5RL:~$ python
Python 2.7.3 (default, Sep 26 2012, 21:53:58)
[GCC 4.7.2] on linux2
>>>import sys
>>> print sys.maxunicode
1114111
```

And, the ucs2 version will produce the following:

```
>>>import sys
>>> print sys.maxunicode
65535
```

The ucs2 unicode representation might leads to character encoding errors, in this case you should force ucs4 when you install Python [19]:

```
./configure --enable-unicode=ucs4
make
make install
```

If you have the proper python installation, you can start installing python packages. For example:

```
cd my_python_modules/scikit-learn-0.13.1/
sudo python setup.py install
sudo apt-get install python-dev
cd ../numpy-1.7.0/
sudo python setup.py install
sudo apt-get install scipy
```

I installed these packages only and after this step you can import packages from python.

```
varh1i@varh1i-F5RL:~$ python
Python 2.7.3 (default, Sep 26 2012, 21:53:58)
[GCC 4.7.2] on linux2
>>>import sklearn
>>>import numpy
```

2.4 Configuring PyroLite

PyroLite is the project that is responsible for the communication interface between java and python. First of all, you have to install it for the python interpreter. It's very similar to installing numpy. You have to download pyrolite from its web page and install it with the setup.py file [20][21]. There's a jar file in the source that you could use in RapidMiner. However the pyrolite.jar compiled on java version (1.7), but I would like to use it with RapidMiner's source code which is compiled on 1.6, hence I needed the source code of pyrolite too. Otherwise I got an unsupported major.minor version 51.0. So after unzipping the source code, I imported it into the Eclipse and build it on 1.6 compliance level (Eclipse/Project Properties) with 1.6-jre system libraries. It created a jar file called „pyrolite.jar", which I can add as an external library to my Extension project. The built jar contains a Manifest file where you can check whether you have the right version number or not. Now we can run it in java 1.6 environment so we can use the functionalty of pyrolite from RapidMiner. You have to add it to the Extension Project as an external library and you have to add it to the RapidMiner's „lib" folder.

3 Implementation

3.1 Basic overview and terms

My main goal was creating an interface between RapidMiner and the python interpreter. Actually after I get through the configuration part it's getting quite easy. I had to create a custom operator as usual and use the API provided by pyrolite. Pyrolite provides a server where we are going to register our python objects. These python objects are wrapper objects of the python classes that contain the actual data mining algorithms but we need this wrapping to facilitate the communication. Every python object has an address in the sense of networking and we can connect to this address through the pyrolite's java API. After we connected to the address we can use the remote wrapper objects' methods that actually will forward the required actions to the inner classes.

3.2 Third party project

I had to search for the appropraite project that could resolve my initial requirements regarding the interface of Java and Python. One of the choices was jython, that is a great project but can't handle cpython libraries for example numpy and sklearn. CPython is the default, most-widely used implementation of the Python programming language. It is written in C, therefore the name. Handling python libraries written in C was a very important part of my project because of the performance and wide range of these libraries. The second option was Jepp. For the first look it seemed to be a right and reasonable choice. It has an API which can execute commands or a whole python files. Despite it was not good, I will cover the reasons of the failure shortly in the next chapter. Finally I found the pyrolite project that I chose finally because of its capability of using cpython libraries too.

3.2.1 Avoid using Jepp

The first idea was about using jepp. This is an open source project on sourceforge and this is handy for using python from java source. I have managed to run python from RapidMiner. I could pass variables with values and read them from python to java. That was really a good sign. My problem was only, that I couldn't import like:

'import statsmodels.api as sm' but I could successfully execute this: 'import statsmodels as sm'. So clearly there was a problem with submodules. I could even use numpy in this way:

```
import numpy as np
beta = np.array([1, 0.1, 10])
 print beta
```

My first thought was that import x.y.z as w is some kind of syntactic sugar and I can use references for example import x. and x.y.z so it might not be a problem. So I refactored my code but still didn't get the desired ouput but rather I faced with the following:

" X = statsmodels.api.add_constant(np.column_stack((x, x**2)))AttributeError: 'module' object has no attribute 'api'"

So in this case it couldn't find the module. Finally I reinstalled my operating system and tried it in windows. I hoped that it is only a configuration issue. I installed python and the sklearn library, but what worked from the python console, it does not worked from the jepp. I have tried with the following configurations:

- python 2.4: It hadn't got the appropriate '.exe'-files, and it didn't worked via python console too
- python 2.5: It worked via console but when I used jepp it gave: binary incompatibility for numpy
- python 2.6: It worked via console, but it doesn't in jepp because some dll file were missing, however there weren't any further information about it. From Linux the closest state to my desired functionality that I reached was that, I imported numpy succesfully and could use it but more sophisticated imports like import x.y.z as w didn't work. However jepp could easily handle things without import via Eclipse for example:
    - print
    - or even pass a Java object -> setting value from python -> read the value in java

However I simply couldn't put the pieces together. I asked about it on stackoverflow and even the leader developer of the jepp's project answered.[28] Finally I found a page which says, „Jepp doesn't seem to be able to import third-party libraries

like scipy, numpy or wx (pure Python modules can be imported, though)". [30] So I had to look for another solution.

3.2.2 Use PyroLite

On the website of Pyrolite there's a very good explanation about the project: „Pyrolite is a client library for Java and .NET. It allows your Java or .NET program to interface very easily with the Python world. It uses the Pyro protocol to call methods on remote objects. Pyrolite is a tiny library (~50Kb) that implements a part of the client side Pyro library, hence its name 'lite..'. Because Pyrolite has no dependencies, it is a much lighter way to use Pyro from Java/.NET than a solution with jython+pyro or IronPython+pyro would provide. Pyrolite contains an almost complete implementation of Python's pickle protocol (with fairly intelligent mapping of datatypes between Python and Java/.NET), and a small part of Pyro's client network protocol and proxy logic." [20][22]

3.3 Architecture of the operator

The two core programs that basically communicate with each other are RapidMiner and the Python server. We have to start the Python server so the RapidMiner can connect to it during the development process. I created a python class for LinearRegression called LR. We can create other classes for Naive Bayes and so on as much as we want. I will cover these objects in detail in the next chapter but now we should know that, I used the functionality of the instance of the LR class as a remote object in java. We can think of the architecture from two points of view.

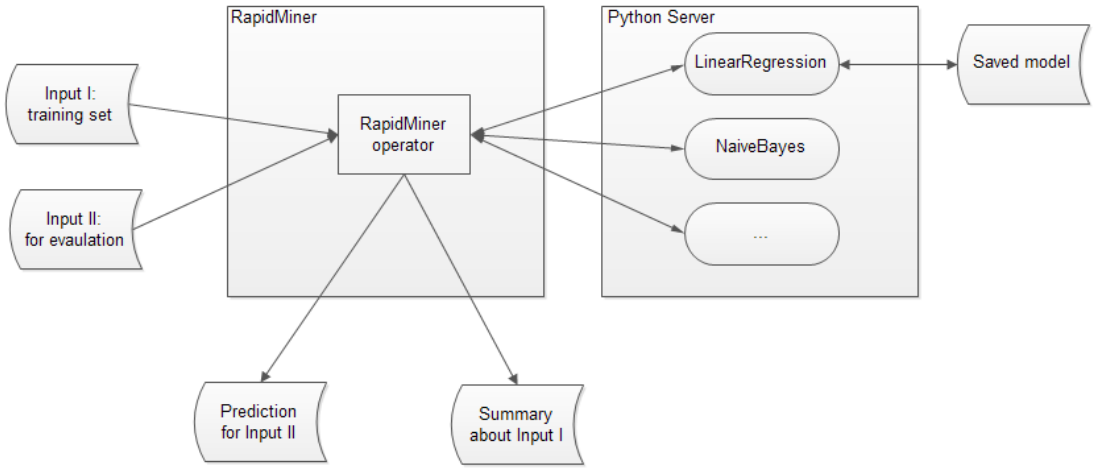
The python side

- create LR class
- create an instace of the LR class
- start the python server
- register the created object in the server

The RapidMiner side

- read the input of the operator
- connect to the instance of the LR class so it will initialize the remote object in java
- pass the input to the remote object's function
  (In this step cpython libraries are used)
- provide the result of the process to the user

The python operator in RapidMiner needs the training set to be passed and the set of values which should be predicted. The training set (Input I.) will be used as the input of the model. After the model is created it can be used to evaluate the other input (Input II.) or we can save the model to the hard disk. If we save it, then we can load it back and use it to predict the values later. In this case we don't need to rebuild the model, but we can use the loaded one. This way we can have many of the trained models saved in a folder and usable anytime. Or we can create different models, for example one for Linear Regression and one for Naive Bayes. The outputs of the operator are two spreadsheets. One of them is the result of the prediction and the other is a summary about the properties of the model. In case of LinearRegressionOperator the summary consist of values about intercept and coefficient.



3.1 RapidMiner and Python Server communication

3.3.1 Python object

The python objects are used by the java code. We can call the methods of the python object as they would be java objects with some sort of special syntax. A python object's functionality is API consists of the methods declared in it. The constructor defines the python library and class that the object uses. In the example below this is the LinearRegression. The fit method creates the model, where the parameters are X for the values of training set and y for the target variable of the training set. The predict function take X as a parameter which values must be predicted. The coef, the intercept and the predicted functions have expansive names. They provide the value of the coefficient, the intercept and the predicted values. There are two more methods, one for saving the model (pickle_out) and one for loading the model (pickle_in).

```
class LR:
     def __init__(self):
          self.lr = LinearRegression()
     def fit(self,X,y):
          self.lr.fit(X,y)
     def predict(self,X):
          self.lr.predicted = self.lr.predict(X)
     def coef(self):
          return self.lr.coef_.reshape(-1,).tolist()
     def predicted(self):
          return self.lr.predicted.reshape(-1,).tolist()
     def intercept(self):
          return float(self.lr.intercept_)
     def pickle_out(self, path):
          file_pi = open(path, 'wb')
          pickle.dump(self.lr,file_pi)
          file_pi.close()
     def pickle_in(self, path):
          file_pi = open(path, 'rb')
          self.lr = pickle.load(file_pi)
          file_pi.close()
```

3.3.2 Python server

When the python operator starts working in RapidMiner, first it connects to the Python server. There's an additional step included that I haven't told yet because I wanted the process to be understood in the first place. There is a nameserver which transform the address (uri) of registered object into a human readable format. Therefore the python object can be looked up with „LinearRegression" instead of a meaningless URI. After the nameserver is running, we can start the python server.

The structure of the server.py file is:

1. import of python modules
2. define python classes
3. instantiate the defined classes
   (the mentioned python objects in the previous chapter)
4. create server
5. register instances to the server supported by the nameserver
6. waiting for the requests…

So first we have to start this nameserver:

```
python -m Pyro4.naming
```

and then we can start the server with running this server file:

```
import Pyro4
import pickle
from sklearn.linear_model import LinearRegression
from sklearn.linear_model import Ridge

'''
… Here come the Python classes …
'''

lr = LR()
daemon = Pyro4.Daemon(port=59454)
ns = Pyro4.locateNS()
uri = daemon.register(lr)
ns.register("LinearRegression", uri)
daemon.requestLoop()
```

And now the registered objects are ready to use.

3.3.3 Java operator

Finally I had to create the RapidMiner operator. You can write the operator following the instruction in the White Paper Documentation from the RapidMiner's website.[23] However I will give exact details about it here too. You can look for examples in the source code of RapidMiner. All of the operators' source code is there. A good starting point can be the OperatorsCore.xml, it contains the operators and the corresponding classes.

In order to implement the operator, first I had to define the input and the output ports of the operator.

```
this.xInput = getInputPorts().createPort("X-input");
this.yInput = getInputPorts().createPort("y-input");
this.xPredictInput = getInputPorts().createPort("X-predict-input");

this.predictionOutput = getOutputPorts().createPort("prediction");
this.summaryOutput = getOutputPorts().createPort("summary");
….
```

The operator's 'doWork' method is responsible for the whole process of the operator. One input is represented by an ExampleSet. In the ExampleSet each row is an Example and it represents one record of the dataset. So I get the inputs from the Input ports and transform them to the appropriate format for sending them through the network.

```
x_exampleSet = xInput.getData(ExampleSet.class);
double[][] X = TransformDataUtil.getXInput(x_exampleSet);
```

I did the same for the Input II and for the target variable. The target variable is an array too but with only one dimension. The next step is connecting to the server with the operator and get a reference for the remote object. I used the nameserver provided by the pyrolite API to connect to the object. This is the wrapper object of the actual model class of scikit-learn modul that we registered previously in the python server.

```
try{
NameServerProxy ns = NameServerProxy.locateNS("localhost", 9090);
PyroProxy remoteobject = new PyroProxy(ns.lookup("LinearRegression"));
```

Now we can call methods on the remote object. The signature of the method is really straightforward. The first parameter is the name of the method, and the other parameters are given to that particular method. We call the remote object's „fit" method to create the model. It has no return value but it creates the model and the values that we might need, for example the coefficient.

Then we can call intercept and coef methods that return the corresponding values. We have to transform the variables on python side to get the values in the right format. More information about type conversions can be found in the appendix.

```
remoteobject.call("fit", X, y);
Double intercept = (Double) remoteobject.call("intercept");
ArrayList<Double> coef = (ArrayList<Double>)remoteobject.call("coef");
```

Another feature of the python class is the capability of saving the created model or loading it. The python objects pickle function provides us this feature. If we set on RapidMiner's parameter tab to save the model and we gave a filename and a place on the hard disk, then the python operator will use these parameters and „pickle out" the model to the given place with the given name. These files usually use the '.p' extensions. In case of loading the operator, it not necessary to use the training set (Input I) and rebuild the model, only the values need to be passed, that needs to be predicted (Input II).

```
remoteobject.call("pickle_out", path+filename);
…
remoteobject.call("pickle_in", path+filename);
…
```

After we decided whether we build a model or load a model we can do the prediction for the Input II. At this step I encountered with an interesting problem. I gave the multidimensional array to the remote object's predict function like this:

```
remoteobject.call("predict",Xpredict);
```

and I get the following exception:

```
net.razorvine.pyro.PyroException: remote exception occurred
at net.razorvine.pyro.PyroProxy.call(Unknown Source)
at net.razorvine.pyro.PyroProxy.call(Unknown Source)
at com.rapidminer.operator.python.PythonOperator2.doWork(
PythonOperator2.java:34)
at com.rapidminer.operator.Operator.execute(Operator.java:855)
at
... Comment: more from the stacktrace...
Caused by: net.razorvine.pickle.PythonException: predict() takes exactly 2
arguments (5 given)
... Comment: more from the stacktrace...
```

The reason is hiding in the PyroProxy.class. Inside this class there's the called method:

```
private Object call(String method, int flags, Object... parameters) throws
PickleException, PyroException, IOException {...}
```

The signatures are different because of the "int flags" parameter but it has a default value if we don't set this. The list of the parameters of the remote object's method is actually an Object array. If we set more than one parameter the 'Objects… parameters' will wrap it up in an Object array. In case of "fit" method the passed parameters look like: Object[Object1, Object2], where the Object[] is the wrapping box and Object1 is the „X" and Object2 is the „y" (because arrays are objects in Java).

When it will be used it will be unwrapped. But when I pass only one parameter Xpredict that is an array consist of arrays, it fits for the Object[Object1, Object2, ...] format. When it will be used it will be unwrapped and the number of the given parameters will be the number of the arrays in the Xpredict array. In the example I passed four array in the Xpredict and there is an extra parameter, the 'self' on the python side. In order to avoid this I had to wrap the parameter into an Object array, so when it will be unwrapped the method will get only one parameter and can handle it the correct way. The correct use of the function is:

```
remoteobject.call("predict",new Object[]{Xpredict});
```

After the prediction finished successfully, we can ask the remote object to provide the predicted target variables:

```
ArrayList<Double>prediction=(ArrayList<Double>)remoteobject.call("predicted")
```

From this point the python operator doesn't need the python server anymore, so it closes the connection. The operator closes the remote object and the nameserver too.

```
remoteobject.close();
ns.close();
```

Now we can print the result in two spreadsheets but before that, we have to transform the values and have to create two new ExampleSets. One for the prediction and another one for the summary. The summary will consist of the requested values on the RapidMiner's parameter tab. The TransformDataUtil.getSummaryExampleSet will create a spreadsheet where the columns are the values of the attribute. If the number of the rows are different from each other, the operator will create rows with the maximum number of the values and fill the gaps in the spreadsheet with missing values. This is the most convenient way that I found to provide the summary in one spreadsheet.

```
predictExampleSet=TransformDataUtil.getPredictionExampleSet(prediction);
double[] coefArray = TransformDataUtil.doubleArrayListToDoubleArray(coef);

summaryExampleSet = TransformDataUtil.getSummaryExampleSet(
new String[]{"intercept", "coef"},
new double[][]{new double[]{intercept},coefArray},
new boolean [] { getParameterAsBoolean(
 INTERCEPT ), getParameterAsBoolean( COEFFIENT)}
);
```

When the ExampleSets are ready we can provide them to the user by putting them to the output ports. If the user doesn't request any value about the model, only the predicted values will be delivered.

```
this.predictionOutput.deliver(predictionExampleSet);

if(y_exampleSet != null){
this.summaryOutput.deliver(summaryExampleSet);
}
```

Before I finally get my python operator working I had to define the implemantation class of the operator in order to make available the operator for RapidMiner. The definition of the operators is in the resources/com/rapidminer/ resources/OperatorsTemplate.xml file.

There are some graphical parameter that you can set from the configuration for example the color of the operator can be set in the groupsTemplate.properties. It's only one code of line: `group.python.color = #CCEEFF.` And you can set image for the operator too.

You just have to create three folder in the Extension Project: resources/com/ rapidminer/resources/icons 16, 24, 48 and each of them is the proper size of the image. These are minor modifications but the operators can be more distinguishable and it fits better in RapidMiner.

4 Usage of the python operator

Even if you don't understand the components of the python operator exactly but you could configure it properly, then you can use this operator confidently. Before you start the process of the operator you have to make sure that you started the nameserver and the python server too. So the steps before you start the operator from RapidMiner:

1. open a terminal and start python nameserver with:

```
python -m Pyro4.naming
```

The output should look like this:

```
/usr/local/lib/python2.7/dist-packages/Pyro4-4.18-
py2.7.egg/Pyro4/core.py:163: UserWarning: HMAC_KEY not set, protocol data
may not be secure
  warnings.warn("HMAC_KEY not set, protocol data may not be secure")
Not starting broadcast server for localhost.
NS running on localhost:9090 (127.0.0.1)
URI = PYRO:Pyro.NameServer@localhost:9090
```

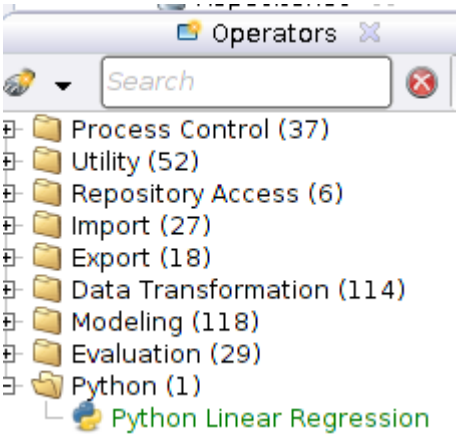
2. open a terminal and start python server with:

```
python python_server_file.py
```

The output should look like this:

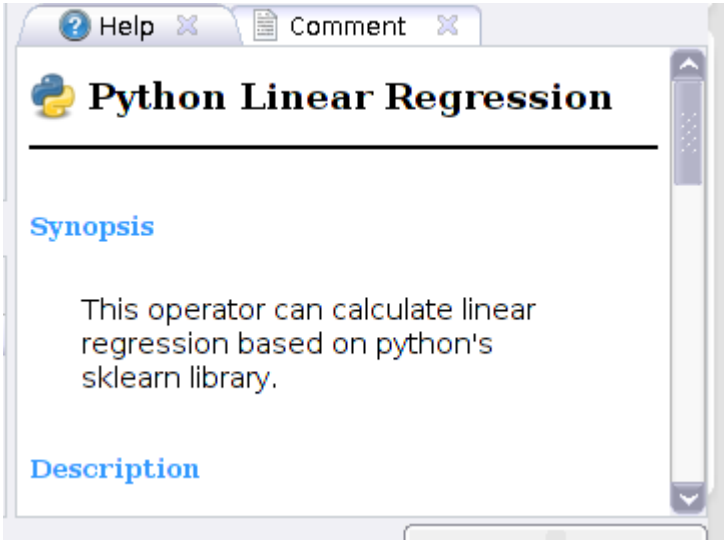
```
/usr/local/lib/python2.7/dist-packages/Pyro4-4.18-
py2.7.egg/Pyro4/core.py:163: UserWarning: HMAC_KEY not set, protocol data
may not be secure
  warnings.warn("HMAC_KEY not set, protocol data may not be secure")
PYRO:obj_6add75ccdf014d23961add506213ab30@localhost:59454
```

You just drag and drop the operator from the operators tab as usual.[26] It is under the Python folder. Later I add a check in the operators' contructor so the operator won't even appear in RapidMiner's operator tab if it's not available.



4.1. Operator's location

You can find detailed information about the operator in the help file. There's a description about the operator and the usage of it. In addition there's the information about the parameters and the input and the output of the operator. So it's more convenient because you can skip reading this long documentation.



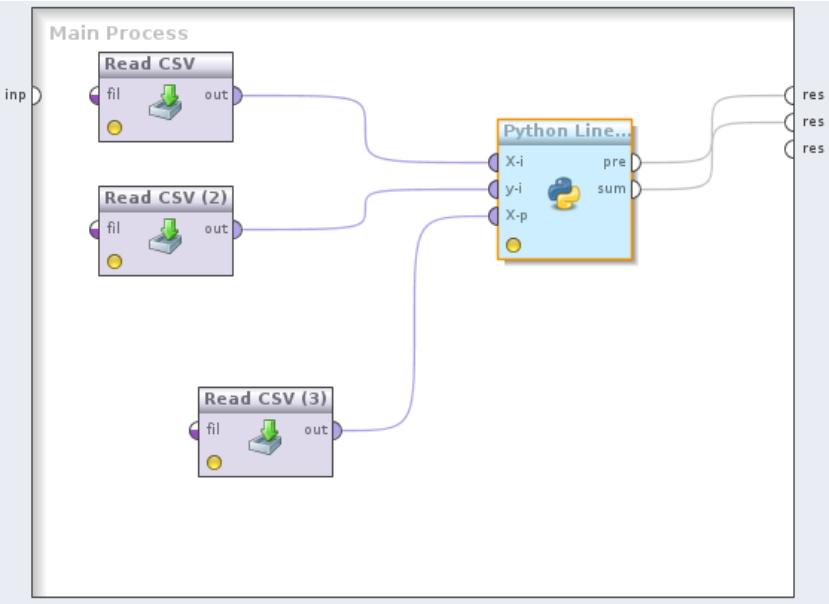
4.2. Operator's help info

The next step is providing the input for the operator. The Input I is the training set it consists of two components:

- the attributes with their values
- the target variables.

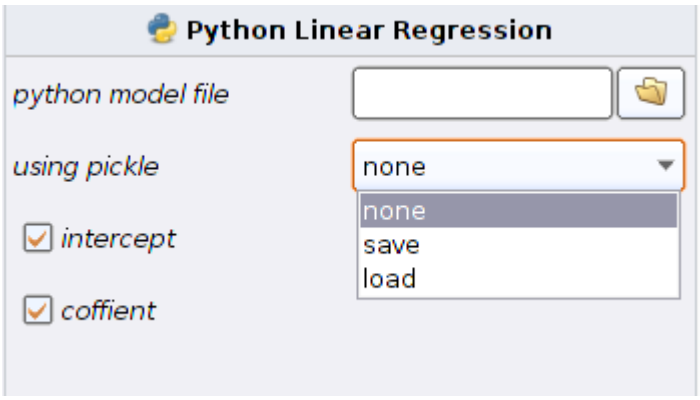
The Read CSV and Read CSV (2) provide these inputs. The Input II. is the values that must be predicted. The Read CSV (3) provides this input. If you are

confused the functions of the input port you can find detailed information in the Help documentation showed on figure 4.2.



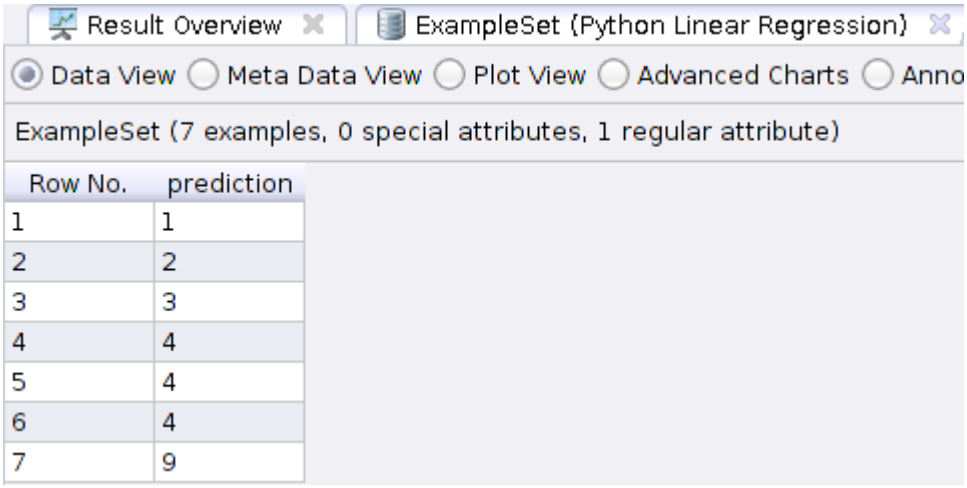
4.3. The operator in RapidMiner

The operator parameters can be set on the RapidMiner's parameter tab. You can create a python model by selecting a place on the hard disk and giving a model a name. The extension of the file is usually '.p'. This way you can also choose a model that you created previously. If you select the proper extension from the FileChooser, only the folders and the files with '.p' extension will be showed. The "using a pickle" parameter is a list of options and you can select how to use the python model. If you choose 'save', the model will be saved to the selected place. If you pick 'load', the model will be loaded and the operator will use it to predict the values. If you select 'none', the operator will build the model from input and uses this new model to predict the values and no model will be saved. This is the default value. The intercept and coefficient parameters are the values which will appear in the summary. If none of these checkboxes is selected, then the operator won't provide the summary ExampleSet, just the predicted values.



4.4. Operator's attributes

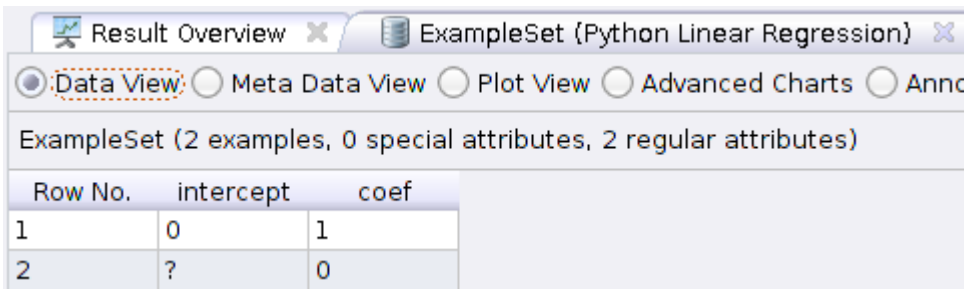
If we selected at least one attribute the operator will provide two ExampleSet. The first ExampleSet contains only the predicted values of the input and the second is the summary of the model.



Row No.	prediction
1	1
2	2
3	3
4	4
5	4
6	4
7	9

4.5. Operator's prediction ExampleSet

In the ExampleSet of the summary the coefficient has two values, and intercept has only one, so the summary has two rows (The maximum number of the values) The gaps are filled with missing values, so there is a question mark in the second row of intercept.



Row No.	intercept	coef
1	0	1
2	?	0

4.6. Operator's summary ExampleSet

5 Compare to RapidMiner's Linear Regression

I have tested RapidMiners operator to decide whether the python operator can perform better or not. I have used a yacht dataset [29]:

The attributes are:

1. Longitudinal position of the center of buoyancy, adimensional.
2. Prismatic coefficient, adimensional.
3. Length-displacement ratio, adimensional.
4. Beam-draught ratio, adimensional.
5. Length-beam ratio, adimensional.
6. Froude number, adimensional.

The measured variable is the residuary resistance per unit weight of displacement:

7. Residuary resistance per unit weight of displacement, adimensional.

I have built a model based on this data in RapidMiner with my PythonOperator and with RapidMiner's LinearRegression operator that wraps the LinearRegression model of scikit-learn.[31] After that, I compared the results and the performance of the two model.

The results:

Row No.	intercept	coef
1	-19.237	0.194
2	?	-6.419
3	?	4.233
4	?	-1.766
5	?	-4.516
6	?	121.668

5.1. Python operator linear regression

Attribute	Coefficient
x1	0.191
x2	-11.123
x3	3.962
x4	-1.578
x5	-3.770
x6	120.762
(Intercept)	-18.223

5.2. RapidMiner's linear regression

As we can see, the result are different, so I had to calculate the root-mean squared error for both model used by the predicted and the real values. The values for the RMSE:

RapidMiner's linear regression: 8.817

Python operator linear regression: 8.8572088390691608

So unfortunately the Python operator can not provide a better solution in this case. It can be because of the data, but it's probably because of the inner algorithm. Despite I think the operator is quite useful, because we can use it as a blueprint in order to extend RapidMiner's functionality with new features that are already implemented in Python. Furthermore we can find python methods that perform better than Rapidminer's operator so we can substitute them.

6 Integrating scikit-learn module

6.1 Scikit-learn

In this section I introduce the terminology and vocabulary that is related to usage of machine learning algorithms contained in the data mining python module called scikit-learn. Sometimes I refer to scikit-learn as „sklearn" that is the corresponding python package that developers can import in their code in order to use the algorithms of scikit-learn.[27] We can use sklearn to use it our data analysis process. During this data analysis process we consider a data of n number of samples. Usually each sample has more than one attribute. For example, if we think about a data of customers of an international supermarket chain, then each row contain one customer and most likely each customer has more than one attribute or so-called feature. These features contain the data of the customers that holds important information. These information can be analysed by the supermarket and can be used in focused marketing campaigns. So for example in the first phase the supermarket collects data of each purchase of the customers and in the next phase it hires data analysts in order to extract useful information for the supermarket about the customers' behaviour and purchasing habits. In general this process is continuous in that way, the supermarket iterates the aforementioned two phases and as a result they can give discounts in a form of tickets that is personalized based on previous purchases. This was just one example but many similar and other applications are available.

Scikit-learn separates the techniques that is used by data analysts in the following few large categories:

- supervised learning, in this case the data comes with additional features that we would like to predict.
    - classification: the samples belong to two or more classes and we would like to learn from a data, that has the features and a label of the class or so-called target variable, and predict the labels of an unlabeled new dataset. Using the previous example it means, that the supermarket wants to sell a specific product and they already know about a set of customers who bought this product or not. Based on this knowledge they can

predict that which of the customers of the new dataset has similar attributes to the customers that bought this product, so they can try to target these customers more efficiently.

    - regression: if the desired output consists of one or more continuous variable then this process is called regression. A simple example of regression problem would be the prediction of the views of a website based on the number of the referrer links to the website and the referrer websites' view count.

- unsupervised learning, in this case the data comes without any additional label of the desired class. The goal in such a situation is to define newly discovered groups that contains samples with similar attributes. This process known as clustering. An other concept to determine the distribution of data within the input space is called density estimation. In some situation we would like to visualize the data but this is not possible immediately when we are dealing with high-dimensional data. We need to project the data from the high-dimensions to two or at leat 3 dimensions.

Furthermore scikit-learn module also contains useful submodules that can be used for model selection and evaluation. We can find the implemention of the tools that can be used for evaluating estimatior performances, for example cross-validation, or quantifying the quality of predictions, like the efficiency and goodness of the learned models.

In addition scikit-learn also contains dataset loading utilities. With this subpackage scikit-learn provides some „toy dataset" that is used in some example covered on the website of sklearn. We can also evaluate the impact of the scale of dataset while controlling the statistical properties of the dataset. There are sample images embedded that can be used to test algorithms too. Scikit includes sample generators that can build artificial datasets of controlled size and complexity. For instance generating a random n-class classification problem or generate the "Friedman #1" regression problem, etc.

Because of the „nature" of the package, being under BSD license, it allows the developers to contribute to the package. It can either submitting bug report, posting

feature requests or you can start implementing your own estimator anytime that can be merged into the package by the authors of the package. However they advise you to contact the developers on the mailing list before you would start implementing a non-trivial feature.

6.2 Contained machine learning algorithms

As I mentioned in section 6.1. machine learning algorithms are divided into two main groups, one of them is supervised learning and the other one is unsupervised learning. These two groups can be splitted into more other subcategories:

1. Supervised learning
    - 1.1. Generalized Linear Models
    - 1.2. Support Vector Machines
    - 1.3. Stochastic Gradient Descent
    - 1.4. Nearest Neighbors
    - 1.5. Gaussian Processes
    - 1.6. Cross decomposition
    - 1.7. Naive Bayes
    - 1.8. Decision Trees
    - 1.9. Ensemble methods
    - 1.10. Multiclass and multilabel algorithms
    - 1.11. Feature selection
    - 1.12. Semi-Supervised
    - 1.13. Linear and quadratic discriminant analysis
    - 1.14. Isotonic regression

2. Unsupervised learning
    2.1. Gaussian mixture models
    2.2. Manifold learning
    2.3. Clustering
    2.4. Biclustering
    2.5. Decomposing signals in components (matrix factorization problems)
    2.6. Covariance estimation
    2.7. Novelty and Outlier Detection
    2.8. Hidden Markov Models
    2.9. Density Estimation
    2.10. Neural network models (unsupervised)

It's a huge amount of machine learning model. In total, it's more than one hundred different models. It was a real challange to handle so much models. In the next section I'll describe how I did that.

6.3 Implement operators with common interface

In order to handle such amount of models I realized I would need some extra utility. First of all the most important was, that I should have done much earlier, to find a proper python integrated development environment (IDE), instead of using simple text editor. I chose PyCharm 3, that is an intelligent software tool with unique code assistance and analysis, for productive Python development on all levels. Some of the features that I could use during the development are: code completion, on-the-fly error highlighting, integrated console in the running view.

The other main change was that, I had to use a version control system in order to implement operators efficiently. I already had some experience with git. Git is a distributed revision control and source code management tool with complete history of the changes and and full version tracking capabilities. It's not dependent on any network access or a central server, so it was reasonable to use that.

However during the development I still had to use four different windows at the same time:

- Terminal: for the nameserver and checking registered objects, also for using git
- RapidMiner graphical user interface: for tesing the operators
- Pycharm: for implementing python side of the operator
- Eclipse: for implementing and debugging java side of the operator

Implementing more than one hundred operators is a huge task even if the base interface is quite similar in many cases. The smallest difference can cause troubles when trying to take advantage of the similarities. Obviously my main problem was the huge amount of operators that I had to implement. Just to express it better, Rapidminer has 118 operators for models and the other operators are related to importing, exporting, evaulation, data transformation and utilities. The python operators that I was trying to integrate into RapidMiner would add 109 extra operators. My main idea was to find the best structure so I can avoid code duplication. In order to do that, I had to find all the similar components in all operators, put them in a common place or into a parent class that can be used for inheritance by the child classes and try to generate as many code as I can based on slightly different properties. Also an important part of this general structure to use naming conventions that I'll describe in details in the next section. Here are the steps that I went through.

1 Create python files for the operators

In this step I created one python package for each subcategory from the list. So „Generalized Linear Models" has a corresponding python module: „linear_model" and this module contains all the operators from the sklearn's linear_model module. The operator introduced in Chapter 4. also resides in this package with other 26 operators. The naming conventions that I applied here: each model class's name is the original class name from the sklearn modul prefixed with „J" (e.g., JLinearRegression) and the python package name is also the original package from sklearn prefixed with „j_" (e.g., j_linear_model). After this step I had only the empty python files that contained literally nothing, but these files are already grouped by their corresponding sklearn modul.

2 Python base class

In order to avoid code duplication I created one base class for each package therefore j_linear_model package has a class JBaseModel that holds the methods that are used in many classes, for example: fit(X), predict(X), score(X), pickle_in(filepath), pickle_out(filepath). My first idea was to use only one class as a base class. Finally using one base class in each package proved to be a better approach because it gave a less verbose API for the base class because methods that are used in one package but not in any other are only in the package's base class.

3 Creating the content of the python model classes

After I created the python files I realized that the basic implementation of these model specific classes contain only one method that sets two fields: the name of the class (for debugging purposes) and the class that actually comes from the sklearn library (for wrapping the sklearn model). So I created a bash script that is capable of creating a list of the files that I created manually and based on the name conventions that I used in the previous steps I could use a template to generate all the operators. I will show a more interesting template than this when I go into details of generating the Java side. Another generalization that I could use was, importing and registering the operators for the python server. This is also just a few lines of code however for this amount of operators it would have taken quite a lot of time but can be easily generated based on the name conventions. Unfortunately I couldn't do any further generalization on the python side. It means that the model specific details, e.g., getting the attrbitues that are specific to the models (coefficient in case of LinearRegression) or initializing the model with parameters that can set by the user on RapidMiner parameter tab (cluster numbers for KMeans), still need to be manully coded.

4 Generating java side

Then I had to define a similar structure for the java side as well. The approach was quite similar: creating a base class for each java package and generate the classes that extend this base class. This time I already had the list of the necessary operator names so I didn't even have to create the empty files, creating the java files was also handled by the templates that I programmed. First I created the folders for the java classes. I could take advantage of the previously used naming conventions again. Each

python package corresponds to a java package and each sklearn model needed an own operator. These templates are files, that have some fields, like the name of the class or the name of the contructor, which are replaced with the actual parameters during the generation. An example of that is the following template that was used for generating the python operators is shown here:

```
package com.rapidminer.operator.python.PACKAGE;
public class MODELOperator extends BasePythonOperator {
private static final Logger logger
= Logger.getLogger( MODELOperator.class.getName());
private String pObject = "MODEL";
private PyroProxy model;

public MODELOperator(OperatorDescription description) {
	super(description);
	try {
	logger.info("Looking for " + pObject + "registered object");
		model = new PyroProxy(ns.lookup(pObject));
	//Handling exceptions
	}
	protected void output(){//Creating output…}
	protected void reset(){//Creating output…}
}
```

With this contructor only those Python operators will appear in the list of operators of RapidMiner that are registered in the server and the others simply will show an error in the log. Note the MODEL and PACKAGE fields in the template that need to be subtituted. I got the values of these fields with a smart bash script that I will show soon. This template is the common implementation of an operator of an sklearn model that can not be refactored to a base class but yet would take a huge amount of time to create manually. The rest of the code that is specific to each operators still needed to be implemented manually. It includes also the doWork method that is the core of every operator.

And a corresponding bash script that uses the template above is:

```
models=(`find . -regex '.*j_.*py' | grep -v '.*__init__.*' | grep -v
'.*BaseModel.*'`)
i=0
for model in "${models[@]}"
do
    OLD_IFS="$IFS"
    IFS="/"
    tmpModelName=( $model )
    modelName="${tmpModelName[2]}"
    modulName="${tmpModelName[1]}"
    length=${#modelName}
    length=$((length-3))
    pLength=${#modulName}
    finalOrigModelName=${modelName:1:$((length-1))}
    finalModulName=${modulName:2:$((pLength-2))}
    sed -e "s/PACKAGE/${finalModulName}/g" -e
"s/MODEL/${finalOrigModelName}/g" OperatorTemplate.java >
"/home/balazs/Develop/workspace/RapidMiner_Extension_Template/src/com/rapidmi
ner/operator/python/${finalModulName}/${finalOrigModelName}Operator.java"
    IFS="$OLD_IFS"
done
```

The following code creates an array of the python files which correspond to model files So the __init__.py file and the files that are the python base classes are excluded by the grep command. Then iterate over the list in a for loop and extract the necessary fields from the file path. In case of AffinityPropagation it looks like this:

„./j_cluster/JAffinityPropagation” → PACKAGE = cluster,

MODEL = AffinityPropagation

Finally a stream editor program called „sed” replaces the parameters from the templates to the actual value in the template and creates the java file with the specific fields. Note that the java files are also generated by the sed command when the output is redirected into the right java package (that is a variable depending on the sklearn module) and with the proper java class files (that is also a variable based on the name of the model of the sklearn module). Furthermore, I needed another script for defining the python operators in the xml file that is responsible for registering the operators to RapidMiner that is also about 400 lines but could be safely generated saving a lot of time.

5 Testing

After these steps I had more than one hundred operators in RapidMiner that I had to test. I didn't create automated integration testing because there are many steps that are involved and also wanted to write down the error messages that appeared. However there might be some tools which can be used for such a task. So I ended up doing smoke tests for each operator manually. Then I faced many different error messages that was needed to be solved.

- Some errors originated from the violation of the expected API that should be common in many sklearn API: RandomizedLasso from linear_model has only the fit method because of the goal of that model it doesn't have predict and score.
- Others were because of the incorrect input data: „Can't handle mix of continuous and binary" that can be easily solved.
- Some of them I couldn't resolve because for example neighbors's KDTree doesn't have a documentation on scikit-learns website and I got the following error: „object has no attribute fit".
- Some errors came from the differences how each model is built: „'tuple' object has no attribute shape".

My first attempt was to integrate all the models into RapidMiner, however I couldn't implement completely all the operators because of the amount of them and the wide range of differrent errors but still implemented 50 of them using this common interface.

In order to help the users of the operators, I checked how could I notify them about any kind of error that is related to invalid input or wrong usage of the operator. Finally I figured out how can I write to the log of RapidMiner, so when someone doesn't get the expected results, he can check the log and modify the input accordingly. Another concept of helping the end-users is also provided by RapidMiner. We can write a synopsis of the operator, the details about the operator input ports and what is expected and also we can give information about the parameters. In addition it's useful to provide a link to the page of the API defined by scikit-learn.

6.4 General behaviour of the operators

During the development I had two different concepts of handling the models. One of them was a 3-node structure and another 1-node structure. The 3-node structure could handle the three following features separatly: training a model, applying a model to predict a new dataset and a read/write operator to pickle in and pickle out the models using python pickle modul. This is theoretically feasible to pass the pickled version of the model accross the operators but finally I didn't test it. This would be more a RapidMiner-style because operators from RapidMiner usually have an output port that provides the model. In my case however I have the model created on the python server which needs to be serialized in order to pass the model through the network and then passed between the operators. So I decided I will go with the 1-node structure and implement a fairly intelligent operator that check the conditions of each input port and behave as it is desired.

So for example in case of the LinearRegression operator there are four input ports:

- X-train-input, the training set
- y-train-input, the labels for the training set
- X-predict-input, the training set
- y-true, the labels for the predict input

Based on the data provided on each input port and the properties defined on the parameters tab the operator contains a logic to decide which output does it need to provide. For example if nothing is specified, then obviously the operator doesn't do anything. If and only if *X-train-input*, *y-train-input* and *X-predict-input* are provided to the operator, then the model is fitted using *X-train-input* and *y-train-input* and the prediction of *X-predict* is calculated by the operator and provided on „prediction" output port. If the *y-true* input is also provided then the operator generates the output of the summary that varies depending on the type of the operator. Usually the summary contains attributes of the model class that is defined on the official website of the sklearn model and result of the score method. Each model has it's own API documentation on this website where these attributes and parameters of the operator can be found.

In contrast in case of KMeans operator the operator has only two input ports:

- X-train-input, the training set
- y-true, the labels of the training set

In this case the operator generates the prediction of the labels on the output port even if only *X-train-input* is provided and creates the summary if *y-true* input is provided. Put in short the operator has an eager approach of generating the output, in other words it creates every output as soon as the necessary data is available on the input ports.

7 Case study of KMeansOperator

In this chapter I will show a case-study of using the iris dataset that is widely known as a dataset that is used by data analyst to test many machine learning algorithms. I will use the complete implementation of KMeansOperator in order to show the features of a python operator and present its compatibily with other RapidMiner capabilities like scatter plots and other operators implemented by the community of RapidMiner.

7.1 Iris dataset

The dataset that I used is the famous iris dataset that consists of 50 samples of different variations of Iris flower: Iris setosa, Iris vergonica, and Iris versicolor. Each sample was measured and they collected data about the length and the width of sepals and petals.[33] The dataset usually used to set a good example to explain the differences of supervised and unsupervised techniques in data mining. (These two concepts of data mining is reflected in the libraries of scikit-learn as well.) As we will see, unsupervised techniques like clustering the dataset does not provide a sufficient result for separating the three distinct species since the separation rahter contains two well distinguishable clusters. One of them contains Iris setosa and the other one contains mixed samples of Iris virginica and Iris versicolor. However the three species can be separeted with other techniques: „Nevertheless, all three species of Iris are separable in the projection on the nonlinear branching principal component”.[32] „On a so-called "metro map" for the Iris data set. Only a small fraction of Iris-virginica is mixed with Iris-versicolor. All other samples of the different Iris species belong to the different nodes.”[32]

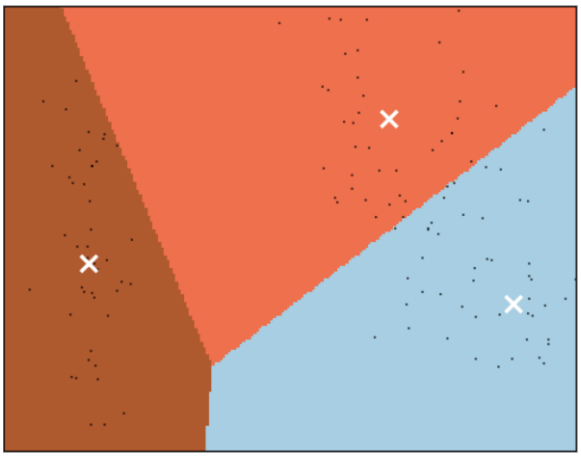
7.2 KMeans algorithm

KMeans algorithm, or often referred as Lloyds algorithm, is used to cluster data by defining groups from the input dataset based on the attributes. The advantage of the algorithm is, it scales well to large number of samples and can be used for general-purpose, thus it is used across a large range of application areas. The algorithm iterates two steps. In the very first step it requires to define the n number of initial centroids of the clusters. In scikit-learn it can be accomplished in three ways: by selecting random

data samples to use as centroids or using uses sklearn internal implementation that selects initial clusters in a smart way to speed up convergence, or user can pass an array of shape (n_clusters, n_features) that select the n_cluster number of initial centers from the n_features-dimensional space. In the second step it creates new centroids by taking the mean value of all of the samples assigned to each previous centroid. The objective function minimizes the criterion known as the „inertia" of the groups:

$$J(X, C) = \sum_{i=0}^{n} \min_{\mu_j \in C} (\|x_j - \mu_i\|^2)$$

These two steps iterate while the objective function relative decrease is less than a given threshold. This iteration always will converge - if it has enough time - and stop but it may found a local minimum. Therefore the iteration usually repeated several times in order to provide better result.[34] The process can be explained with a Voronoi diagram. In the first step mentioned above, each segment has its own centroids that are define the clusters of the points on the plane. In the next step the new centroids are calculated by the mean of the points and clusters are updated according to the new cluster centers, then the process is continued with the iteration of these steps until the changes of boundaries of clusters are within the defined threshold. On the following figure (7.1.) we can see the final clusters of the iris dataset on a Voronoi diagram that is defined by the result of Kmeans algorithm from scikit-learn. The white crosses are the final centroids of the clusters and the black dots are the samples of the iris dataset. In order to plot the result I reduced the dimensions of features with PCA from scikit-learn library and displayed the results.



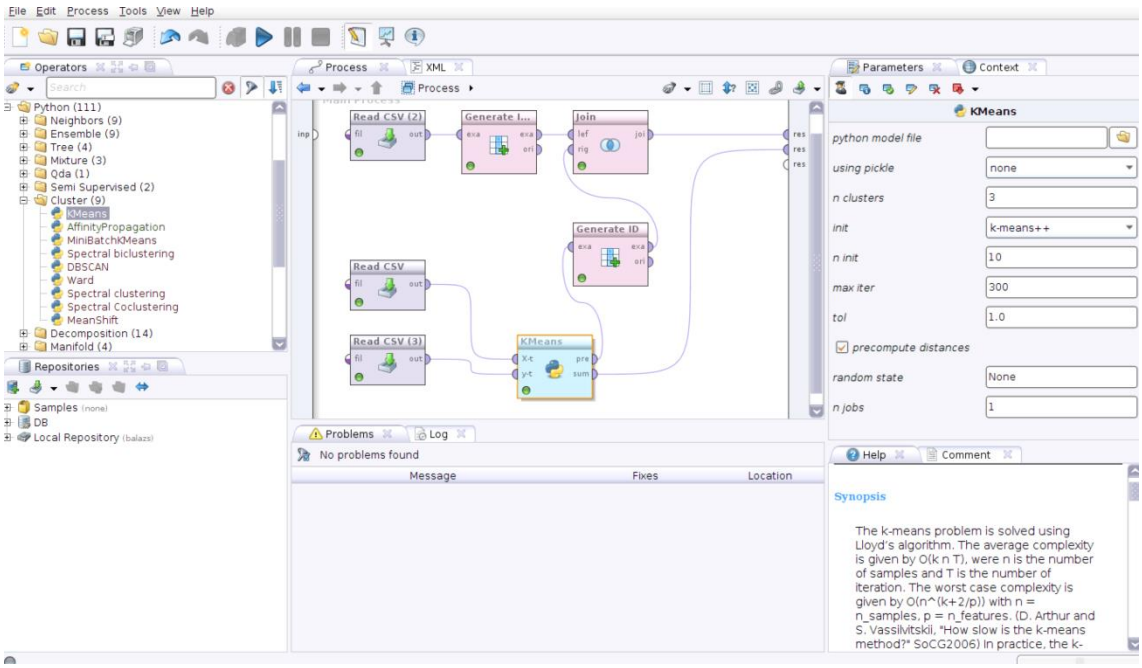
7.1. Result of KMeans algortihm on Iris dataset with pure python

During the usage of KMeans alogrithm I continously used the API documentation of Kmeans and in general the scikit-learn's webpage also provides many helpful examples of different algorithms usually with code implementation examples and explanations. For creating the figure above I could use a modified version of one of the examples provided on the website. [35]

7.3 Using RapidMiner with KMeans python operator

In this section I will describe the usage of the created KMeans python operator, what features it provides and how does it work together with other built-in RapidMiner and finally will plot the result using RapidMiner the colored version of 3D scatter plot.

Since I already used the iris dataset that is an embedded dataset in sklearn and can be loaded after a simple import, in order to get the dataset I wrote a simple python code that wrote the dataset into a comma separated file. Similarly I wrote the target values of the iris dataset into a different datafile. Kmeans has two input ports, the first one will receive data that contains the features of the iris dataset and provide the prediction of the target variables on one of its output ports, the second input port will get the real target variables and the prediction provided by the operator and it creates a summary of evalutation of clustering. The described process is shown on the image below.



7.2. RapdiMiner GUI with the integrated python operators

On the left we can see the list of the integrated operators and we can see that they are categorized and grouped in different folders according to their functionality, therefore can be found easier. On the right we can see the parameters that can be set for the KMeans operator.

On the right bottom of the picture we can see that in order to help the user I used the features that RapidMiner provides for the users, so I added the synopsis of Kmeans, that was available on scikit-learns website, to the help documentation. It contains information about the underlying algorithm in details like the complexity of the algorithm. Running time is absolutely not a concern in case of iris dataset because it contains only 150 samples and four features but these equations can be used to calculate expected running time for high dimensional data with high number of samples. Furthermore the help documentation also contains data about the parameters and input and output ports.

On the right part of the image we can see the Parameters tab that corresponds to the parameters that can be set from python. Each operator can have many parameters that can be set to customize the algorithm based on our needs. The python class sklearn.cluster.KMeans' constructor has the following API:

```
class sklearn.cluster.KMeans(n_clusters=8, init='k-means++', n_init=10,
 max_iter=300, tol=0.0001,precompute_distances=True, verbose=0,
random_state=None, copy_x=True, n_jobs=1)
```

In order to use these parameters I needed to modify the python class that is registered in the python server to handle all these parameters. My solution was to declare „None" for the model field of the KMeans python class when it's created and registered and when the operator runs it actually initialzes the model with the parameters provided through the RapidMiner by the user.

In some cases, handling the parameters are not straightforward. If we look at the parameter called „init". The documentation says:

„{'k-means++', 'random' or an ndarray}
Method for initialization, defaults to 'k-means++':
'k-means++' : selects initial cluster centers for k-mean clustering in a smart way to speed up convergence. See section Notes in k_init for more details.
'random': choose k observations (rows) at random from data for the initial centroids.

If an ndarray is passed, it should be of shape (n_clusters, n_features) and gives the initial centers.”

The first two options can be easily integrated into the python operator but passing an n dimensional array is just not feasible on the parameter tab. However it could still be accomplished via a new input port.

This is when the power of the usage of inheritance of the operators in java comes into play. The base class, located in com.rapidminer.operator.python.cluster package, is responsible for establishing a skeleton of all the cluster operators. I defined all the ports that are used by all the operators defined in this cluster java package in the base class. In general I needed to find all the functionalty that can be put together in the base class. However, all the operator specific implementations can be defined in the class of operator. These operator specific attributes are the parameters that can be passed to the model and differences like defining an extra input port to support the n dimensional array as centroids for the Kmeans model. One additional common thing in each java package, I created two enums, one of them holds the parameters, that are used in the communication called ClusterParameter and the other enum contains the executable methods of the registered python objects called ClusterMethod. I found that putting these variables in the actual python operator would result in code duplication but creating only one for all the operators (so not for each package) would create unmaintainable enums because of the hundreds of variables.

However there are parameters that are different types and in the API of RapidMiner we can set only one type for each parameter. An example for such a parameter in Kmean is „random state”. It can be either „None” or an Integer value. These parameter issues can be handled fairly easily by getting the parameter as a String in every case then try to parse the value and decide which type it is actually. The parameter I finally changed for clustering the iris dataset is only the number of clusters that needs to be three, other parameters remained the default. One another concern regarding the parameters is that, some of the data needs to be transformed before I could pass it to the wrapped model. In case of KMeans it meant that I had to transfer the multidimensional array to a matrix.

```
def fit(self,X):
        self.data = X
        self.model.fit(numpy.asmatrix(X))
```

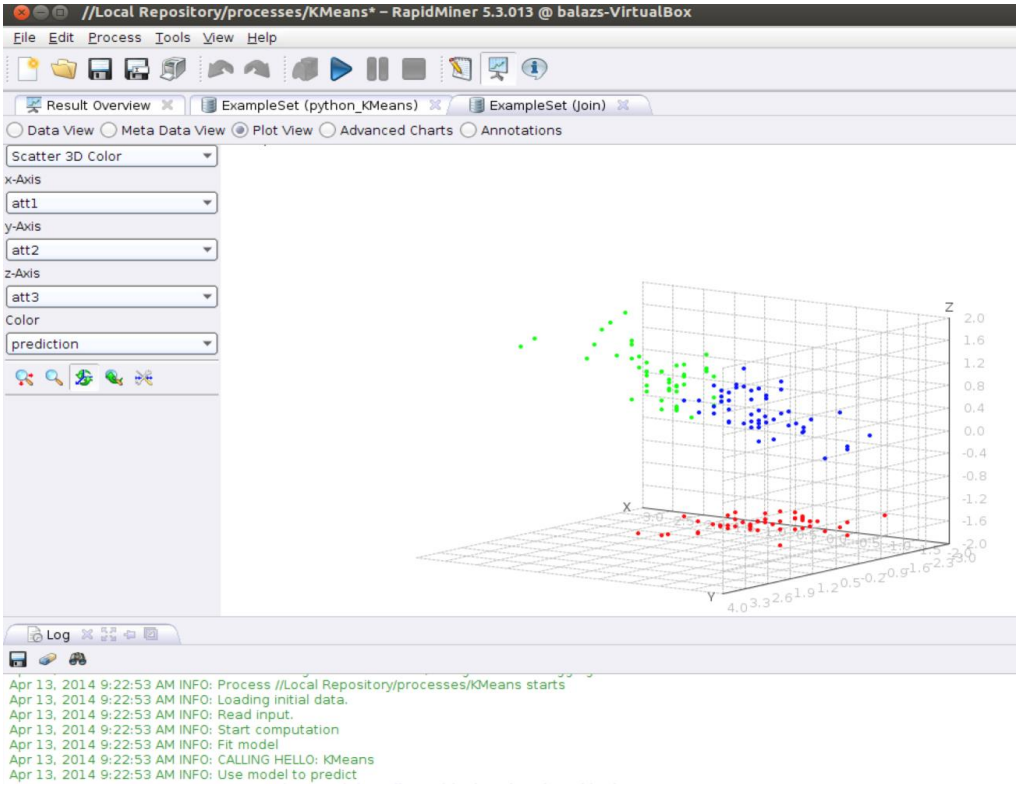
As always we can see the process flow for the clustering in the middle of the picture. As I mentioned the input ports has the iris dataset (for the clustering) and the target labels (for the evaluation summary). After the KMeans python operator ran it delivers the prediction on the output port that is connected to „Generate ID" node. I needed to add another Read CSV node to provide the iris dataset again. The iris dataset and the predicted values are merged together by their ids with the „Join" operator that is also built-in operator in RapidMiner. Due to this transformations finally I had the original dataset merged together with it's predicted values.

As we would expect, after these steps further operators can be used on the outputted dataset. It can be exported with one of the RapidMiner's export operator or can be displayed with the wide range of RapidMiner features like scatter plots or advanced charts and so on. A colored 3D Scatter plot is shown on the image below (7.3.), the predicted values in different colors shows the cluster labels that was assigned to the samples by the KMeans operator. The result proves our presumption about the clusters. We can see that on the figure, the KMeans operator separated one well defined cluster and the other one is effectively splitted in half. In spite of that, it might seems that the Kmeans separated the values properly, but the actual values are mixed in the bigger cluster (green and blue marks on the picture).

On the bottom of the picture we can see the log of the steps of the process:

- Fit model
- Predict

And an additional message: „Calling hello: KMeans" that is just an extra intermediate step that shows the operator works properly after the model was fitted. During the development these log messages were also printed in Eclipse console log and Pycharm console printed the log messages of python objects registered in the python server. There's no need to explain how helpful can be these kind of messages during debugging.

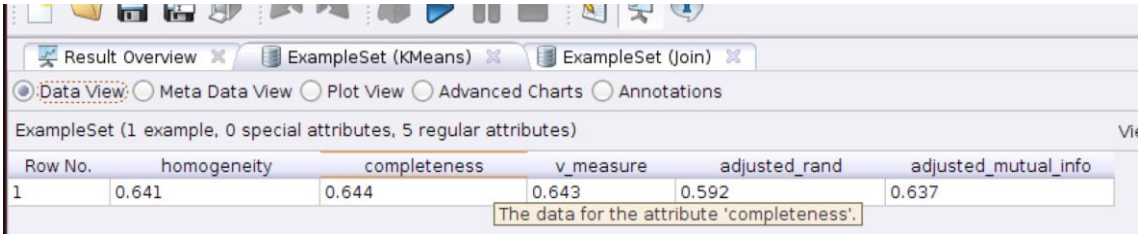


7.3. Result of KMeans algortihm on Iris dataset in RapidMiner with the python operator

There is one another detail that I have to mention. RapidMiner handles metadata of the received and delivered data. Some of the operators checks some conditions of the metadata in order to validate the input of the operators. After the operator finished its duty it delivers the data and usually it also need to provide metadata for the connected nodes' input ports. In case of the process above GenerateID also validates that the connected node provides an ExampleSet as an input of the GenerateID operator. Without defining the metadata the GenerateID operator and the subsequent operators shows an error about the missing mandatory input source. We can define really sophisticated metadata handling. The following line defines a simple rule for the output of the operators therefore the connected nodes can recognize that the provided data is an ExampleSet.

```
getTransformer().addGenerationRule(predictionOutput, ExampleSet.class);
```

We also provided the target variables of the dataset that are the actual values of the classes of iris dataset. So as it is expected the summary is outputted on the summary output port. We can see the results of the evalutaion of the clustering on the image below.



7.4. Result of KMeans algortihm on Iris dataset with pure python

The following evaluation metrics are calculated by the operator:

Homogenity, Completeness, V-measure

„In particular Rosenberg and Hirschberg (2007) define the following two desirable objectives for any cluster assignment:

- homogeneity: each cluster contains only members of a single class.
- completeness: all members of a given class are assigned to the same cluster.

We can turn those concept as scores homogeneity_score and completeness_score. Both are bounded below by 0.0 and above by 1.0 (higher is better)

Their harmonic mean called V-measure.”[36]

Adjusted Rand index

„Given the knowledge of the ground truth class assignments labels_true and our clustering algorithm assignments of the same samples labels_pred, the adjusted Rand index is a function that measures the similarity of the two assignments, ignoring permutations and with chance normalization” [36]

Adjusted mutual information

„Given the knowledge of the ground truth class assignments labels_true and our clustering algorithm assignments of the same samples labels_pred, the Mutual Information is a function that measures the agreement of the two assignments, ignoring permutations. Two different normalized versions of this measure are available, Normalized Mutual Information (NMI) and Adjusted Mutual Information (AMI). NMI is often used in the literature while AMI was proposed more recently and is normalized against chance. Perfect labeling is scored 1.0” [36]

8 Publishing the operator with new RapidMiner license

When I started to write my thesis RapidMiner was an open source project with all its advantages but during my work, at the end of 2013 they changed their business model and defined RapidMiner as „business source". This is a huge change from many point of view and some members of the community considered it as the frailty of open source since RapidMiner always stressed the „community" nature of the software. In this section I'll describe what are these changes and how they influenced the publishing of the extensions. In the end I'll reveal how did I publish my source code to provide insight for others developers who are interested in it.

In general changing the licenses from open source to business source means that as soon as a major version is released i.e. RapidMiner 7, then the current version (6.0) will be unlocked and released as an open source community edition. In fact until the version 7.0 is released, using RapidMiner 5 is also a possibility. They consider this model superior to their previous open core model for all parties because the community is getting a more feature-rich platform and an open source version with a time-delay (in previous versions some features was available only for customers), and customers are getting easy to understand benefits immediately. Due to this differentiation RapidMiner aims to support better both, their paying customers as well as their user community.[37] The current version is categorized in 5 different edition from Starter to Enterprise based on 5 key factors [38]:

- RAM
- File based data sources
- Database systems
- Support
- Period

Using RapidMiner in education was also a concern because professors gladly used it in their data analysis courses. For education RapidMiner team has promised special offer for the Professional edition of RapidMiner.

As a rule of thumb, they wanted to offer a feature-rich platform for evaluation, or proving that the analysis is feasible in RapidMiner, especially for smaller projects but

as soon as the software is used in „production mode", where the business party gains profit, then there should be stronger relationship and they need to establish a commercial agreement between the parties. The team of RapidMiner found that the best differentiators between „production" and „evaluation" modes are the memory usage and data sources. These are also part of the key factors mentioned in the list above.[37]

In regard to extensions created for Community editions they promised compatibility. So developers still can use the community editions as a testing environment and publish their extensions as it was for the latest release version of RapidMiner. So currently the released version is 6.0 but developers can use the previous (5.0) for developing their own extensions and operators.[37] Therefore the python operator should be also compatible with the newer versions.

My first attempt was to release a version of my python extension on the RapidMiner Marketplace. RapidMiner Marketplace is the site where developers of the community can share their submissions with the world of RapidMiner users. It is fully controlled by the team of RapidMiner, they need to approve all the extensions submitted there. The first step is to send a submission request. I did so and a member of the team of RapidMiner, who was responsible for publishing extension, contacted me and asked for further information. As I provided part of this document his response was really promising. His professional opinion was that the project is really interesting and implemented in a reasonable way. However I should provide an extension that is fully understandable for persons without any python knowledge. As soon as I reviewed other extensions on the Marketplace I could figure out what he meant. One example for that, extensions are using wizards for installing the necessary dependency projects.[39] In case of my extension probably installing python and necessary modules via wizard would be necessary as well as starting and registering objects without the command line interface. I made some tests about it and I think these are all possible. In a latter email he also confirmed these assumptions. In spite of this he agreed and encouraged to publish it. A part of his response message:

„thanks for sending me your draft. I think I got a good understanding about the work you have done. Very interesting. I wrote the R extension a few years back, so I know to value the combination of native code with Java. I think you did it in a reasonable way.

So far it is a nice proof of concept, but not yet in a state to be useful for someone, who is not a Python geek :)”

„I think we should consider an alternative to put it on the marketplace. Because I see most users just installing any extension, but if it is just meant for developers that doesn’t make sense. It will only clutter up their machines. You can of course upload that to sourceforge and create a project about it. If you don’t find the time to maintain that yourself, you could also contribute that to us and if we think it is useful, we will continue to develop it.”

Finally I rather chose github to publish my results, that is also a popular website for sharing sources of projects with others and collecting contributors for the projects. Github is a very useful website where developers can register and publish the source code of their public or private projects on a central server provided by github. Publishing on github the developers need to use the popular version control system called git. I needed to publish a code with a simple structure that is easily readable and understandable for other developers so I decided to publish only one of the operators, namely the KMeansOperator that has the complete functionality of handling metadata, documentation of the operator, passing parameters and attibutes for creating the model in the registered python object. The published version already reflects the architecture of using many operators based on scikit-learn, so the architecture specific details like ClusterMethod and ClusterParameter enums are already there. Also published a small server-client program to test the usage of the registered python objects. I added a „Readme” file about this, and will also add a link to the published version of this document to help people understand the whole project.

I wanted to have a version that is easily readable for others and one version that is the „sandbox” of the developing process. Git provides an elegant way of handling these kind of purposes. We can create „branches” for different versions of the same project. So on one branch I have the source code that is currently under development with all the different operators and with a source code that contains prototypes and experimental implementations and on the other branch I have the published version of my project. Whenever I want I can switch between the two branches and can implement things separately. Obviously implementing the same code twice wouldn’t make any sense but git has a solution for this too. I used git „patch” and „checkout” commands’ functionality. Git’s patch command is capable of copying the changes of only one file

from branch *A* to branch *B*. This way I can easily add new implementation to the published version. Git's checkout can add new file from a different branch.

The published code is available on github and readable for everyone without any registration.[40] Developers can look around and analyze the source code and can download and try the extension itself. Can even contribute and ask me to add their changes to the source code or simply just contact me with any kind of question.

9 Overview of the project

Connecting the two worlds of data mining techniques was really interesting and challanging at once. I was aware of the project can be interesting for many data analysts and developer as well because if the project could proves its usability it would be a huge achievement. The RapidMiner could benefit from the huge amount of already implemented algorithms and the developers of the python moduls could earn a large amount of new users for their libraries. From my professional point of view, I had new experience with many development tools and programs like RapidMiner, git, PyCharm, ant, sed, eclipse, etc. and data mining libraries like scikit-learn that are used by data analysts. It certainly helped deepen my programming skills of Java and Python as well. Furthermore discussing interesting topics with developers of such technologies is always intriguing.

During the development I created a „proof of concept" project that shows, it is feasible to call any kind python modules from RapidMiner regardless it is a python or cpython module. Therefore the functionality of the software can be highly expanded regarding to data mining techniques provided by the wide range of python libraries. I used the scikit-learn library as an example but many other python moduls can be integrated similarly. I successfully created the interface for the python classes that can be called by the RapidMiner operator, also implemented by me. This interface also shows how can it handle more than one operator up to any number. After the connection is established between the python object registered in the python server and the RapidMiner operator the python operator is capable of the same functionality as all the other built-in RapidMiner operator. It can be connected to nodes that provides input data for the Python operator and output data can be delivered to many other operators as an input. It proves the fact that RapidMiner is an extendable product despite there were changes in their business approach. The created project also proves that the RapidMiner community is a great support and asset of the RapidMiner group and, as I experineced, the relation is mutual in a way of the employees of the RapidMiner also give help for the members of the developer community.

In general I’m really glad that I could go through with this interesting project and with this topic despite the many obstacles during the development process. The project can be continued in many ways beyond the achieved results that is currently documented in this thesis. Just to mention a few: testing the application on different server machines, since the python server can run anywhere and the operator should be able to connect to it. Furthermore other data mining moduls can be also tested against the current architecture. Hopefully other developers will recognize this architecture’s capability and may use this approach in their projects too.

Bibliography

[1] Wikipedia: Data mining. http://en.wikipedia.org/wiki/Data_mining (revision 17:11, 12 February 2014)

[2] Tapan Patel and Wayne Thompson: Data Mining from A to Z: Better Insights, New Opportunities. http://www.sas.com/content/dam/SAS/en_us/doc/whitepaper1/data-mining-from-a-z-104937.pdf (revision 18:05, 12 February 2014)

[3] Wikipedia: Free open-source data mining software and applications. http://en.wikipedia.org/wiki/Data_mining#Free_open-source_data_mining_software_and_applications (revision 08:45, 15 February 2014)

[4] SAS Institute: Company Information. http://www.sas.com/en_us/company-information.html (revision 14:24, 15 February 2014)

[5] Wikipedia: SAS Components. http://en.wikipedia.org/wiki/SAS_(software)#Components (revision 15:15, 15 February 2014)

[6] RapidMiner: About us. http://rapidminer.com/about-us/ (revision 10:53, 16 February 2014)

[7] Wikipedia: RapidMiner. http://en.wikipedia.org/wiki/RapidMiner (revision 11:24, 16 February 2014)

[8] KDNuggets: Best Python modules for data mining. http://www.kdnuggets.com/2012/11/best-python-modules-for-data-mining.html (revision 17:41, 18 February 2014)

[9] Numpy: Numpy homepage. http://numpy.scipy.org/ (revision 23:12, 18 February 2014)

[10] Scipy: Scipy homepage. http://www.scipy.org/ (revision 20:23, 18 February 2014)

[11] Mlpy: Mlpy homepage. http://mlpy.sourceforge.net (revision 10:11, 19 February 2014)

[12] NetworkX: NetworkX homepage. http://networkx.github.io/ (revision 18:06, 18 February 2014)

[13] Orage: Orange homepage. http://orange.biolab.si/ (revision 17:53, 20 February 2014)

[14] Pandas: Pandas homepage. http://pandas.pydata.org/ (revision 08:20, 23 February 2014)

[15] Pybrain: Pybrain homepage. http://pybrain.org/ (revision 13:32, 23 Februray 2014)

[16] Nltk: Nltk homepage. http://www.nltk.org/ (revision 14:43, 23 Februray 2014)

[17] Scrapy: Scrapy homepage. http://scrapy.org/ (revision 15:30, 23 February 2014)

[18] Python: Open Arbitrary resources by URL. https://docs.python.org/2/library/urllib.html (revision 16:52, 23 February 2014)

[19] Dev Notes: Configure new python instructions, http://dev-notes.com/code.php?q=91 (revision 18:11, 25 March 2013)

[20] PyroLite: Pyro 4.18 documentation, http://pythonhosted.org/Pyro4/pyrolite.html (revision 16:43, 07 April 2013)

[21] PyroLite: Pyro 4.18 Installing pyrolite, http://pythonhosted.org/Pyro4/install.html (revision 18:11, 25 March 2013)

[22] PyroLite: Pyro 4.18 readme, http://irmen.home.xs4all.nl/pyrolite/README.txt (revision 16:48, 07 April 2013)

[23] Rapid-I GmbH: How to extend RapidMiner 5, http://www.rapid-i.com/ (revision 19:00, 25 March 2013)

[24] Rapid-I GmbH: RapidMiner Forum about configuration. http://rapid-i.com/rapidforum/index.php/topic,6575.0.html (revision 09:38, 19 March 2013)

[25] Rapid-I GmbH: RapidMiner Tutorial for Eclipse. http://rapid-i.com/content/view/25/27/lang,en/ (revision 14:12, 15 April 2013)

[26] Rapid-I GmbH: RapidMiner User Manual. http://www.rapid-i.com (revision 08:32, 12 March 2013)

[27] Scikit-learn: User guide. http://scikit-learn.org/stable/ (revision 17:50, 06 April 2013)

[28] Stackoverflow: How to use python modules in jepp. http://stackoverflow.com/questions/15734259/how-to-use-python-modules-in-jepp (revision 18:15, 25 April 2013)

[29] UCI Machine Learning Repository. http://archive.ics.uci.edu/ml/datasets/Yacht+Hydrodynamics# (revision 18:11, 15 May 2013)

[30] Weka: Using weka via Jepp. http://weka.wikispaces.com/Using+WEKA+via+Jepp (revision 18:11, 25 April 2013)

[31] Scikit-learn: Linear regression. http://scikit-learn.org/stable/modules/generated/sklearn.linear_model.LinearRegression.html (revision 18:43, 12 March 2014)

[32] Wikipedia: Iris flower dataset. http://en.wikipedia.org/wiki/Iris_flower_data_set (revision 12:48, 13 March 2014)

[33] UCI Machine learning repository. http://archive.ics.uci.edu/ml/datasets/Iris (revision 12:53, 13 March 2014)

[34] Scikit learn: K-means. http://scikit-learn.org/stable/modules/clustering.html#k-means (revision 09:25, 24 March 2014)

[35] Scikit-learn: A demo of K-Means clustering on the handwritten digits data. http://scikit-learn.org/stable/auto_examples/cluster/plot_kmeans_digits.html (revision 14:42, 24 March 2014)

[36] Scikit-learn: Clustering performance evaluation. http://scikit-learn.org/stable/modules/clustering.html#clustering-evaluation (revision 15:13, 25 March 2014)

[37] RapidMiner: Version 6: Community Edition? http://rapid-i.com/rapidforum/index.php?&topic=7397.15 (revision 22:43, 15 April 2014)

[38] RapidMiner: Comparing RapidMiner Editions. http://rapidminer.com/products/rapidminer-studio/#compare (revision 20:57, 12 April 2014)

[39] e-Lico: Intelligent Discovery Assistant. http://www.e-lico.eu/ida-extension.html (revision 01:05, 17 April 2014)

[40] Balazs Varhegyi: RMPyro project. https://github.com/varh1i/RMPyro (revision 01:05, 20 April 2014)

Appendix

Converting python types to java types

PYTHON	JAVA
None	null
bool	boolean
int	int
long	long or BigInteger (depending on size)
string	String
unicode	String
complex	net.razorvine.pickle.objects.ComplexNumber
datetime.date	java.util.Calendar
datetime.datetime	java.util.Calendar
datetime.time	java.util.Calendar
datetime.timedelta	net.razorvine.pickle.objects.TimeDelta
float	double (float isn't used)
array.array	array of appropriate primitive type
list	java.util.List<Object>
tuple	Object[]
set	java.util.Set
dict	java.util.Map
bytes	byte[]
bytearray	byte[]
decimal	BigDecimal
custom class	Map<String, Object> (dict with class attributes including its name in "__class__")
Pyro4.core.URI	net.razorvine.pyro.PyroURI
Pyro4.core.Proxy	net.razorvine.pyro.PyroProxy
Pyro4.errors.*	net.razorvine.pyro.PyroException
Pyro4.utils.flame.FlameBuiltin	net.razorvine.pyro.FlameBuiltin
Pyro4.utils.flame.FlameModule	net.razorvine.pyro.FlameModule
Pyro4.utils.flame.RemoteInteractiveConsole	net.razorvine.pyro.FlameRemoteConsole

Converting java types to python types

JAVA	PYTHON
null	None
boolean	bool
byte	int
char	str/unicode (length 1)
String	str/unicode
double	float
float	float
int	int
short	int
BigDecimal	decimal
BigInteger	long
any array	array if elements are primitive type (else tuple)
Object[]	tuple
byte[]	bytearray
java.util.Date	datetime.datetime
java.util.Calendar	datetime.datetime
Enum	the enum value as string
java.util.Set	set
Map, Hashtable	dict
Vector, Collection	list
Serializable	treated as a JavaBean, see below
JavaBean	dict of the bean's public properties + __class__ for the bean's type
net.razorvine.pyro.PyroURI	Pyro4.core.URI
net.razorvine.pyro.PyroProxy	cannot be pickled