Modeling and abstraction of the Machine Learning Domain

Angela Barriga

Department of Computing, Mathematics, and Physics Western Norway University of Applied Sciences, Bergen
abar@hvl.no

Abstract Machine Learning (ML) is a promising technological area, its broadness to solve many different types of problems makes it a highly desirable field for research and investment, both in academia and industry. Yet, to create a state of the art ML system, it is necessary to posses a complex skill set. The objective of this project is to reduce this complexity, abstracting the ML domain. In order to achieve said abstraction, it is proposed to apply Model Driven Software Engineering (MDSE) into ML, creating a metamodel and applying it into a concrete case study. Furthermore, it will be developed a graphical editor to protoype ML systems, translating said abstraction into a concrete syntax.

1 Introduction

ML is a field with quite a promising future [1] however, the skill set needed to build a state of the art system is complex and hard to reach [2]. In addition, the ML community has developed over the years a long list of algorithms for tackling different problems [2,3], so finding which algorithm is the best solution for a particular problem is added to the already high complexity of this technology.

At the moment, there are different frameworks, software tools and model-based solutions that make complex technology easier for all kind of users, even to those with low technical knowledge, hiding all unnecessary details in fields as: web development [4], mobile applications [5], basic programming [6,7] or robotics [8]. Still, and despite its growing importance, there is no tool like these for developing or learning how to build ML systems.

Some authors have already stated the vital importance of science democratization [9] and helping society with technology's growing skill demand, especially within the ML and artificial intelligence (AI) scope [10]: *"AI may play a role in augmenting or reducing the socio-economic impact of intelligence and wealth depending on whether it is sufficiently accessible for a wide population. Tools will be needed to help individuals cope with complexity."* Additionally, other authors [11] stress the relevance of performing research for machine learning that actually matters, having a real impact in society *"Current ML research suffers from a growing detachment from real problems"*. In order to overcome the obstacles to create real impact, they propose to simplify ML jargon and overall complexity *"Despite the proliferation of ML the field has not yet matured. Attempts often fail due to lack of knowledge. Simplifying can help to erode this obstacle."*

The aim of this project is to reduce the complexity of developing ML systems by creating a framework based on MDSE, abstracting irrelevant technical details.

It is already been widely proven in academia that MDSE [12,13,14] helps to reduce technology complexity in a broad range of fields, therefore it seems to be a good way to solve the stated problem.

In order to achieve these objectives, this project will focus on developing a metamodel for ML abstraction and its application into a case study. This will focus on the solution of a simple ML problem, such as prediction or classification. Applying abstraction into a concrete example will lead to gain more insight into the benefits of this project and make the metamodel more understandable.

Additionally, the modeled concepts will be translated into concrete syntax by the development of a graphical editor for prototyping ML systems.

2 Material and Method

In order to tackle a problem of such dimensions, it is of utmost importance to decide which is the best approach to start with. This section will cover the different subdomains within the ML domain and some of their algorithms, justifying which of them are chosen and how they are abstracted using MDSE.

Abstracting the whole ML domain at once would not be a good choice, since it has growth a lot during time; covering a great range of algorithms and techniques. A more feasible approximation would be to look into the usual division of this domain: supervised learning, unsupervised learning and deep learning.

1. Supervised learning is the most common form of ML [15,16]. In this kind of system, a set of examples (the training set) is submitted as input to the system during the training phase. Each input is labeled with a desired output value, "teaching" the system how would be the output for a specific input. An example of a supervised learning application could be a mail spam detector.
2. Unsupervised learning on the other hand provides an environment in where the training examples are not labelled with its belonging class. So the system develops and organizes the data, searching common characteristics among them and changing based on internal knowledge. Anomaly detection systems in security circuits are based on this technique.
3. Deep learning is based on the way the human brain process information and learns. It consists on a ML system composed by several levels of representation, in which every level uses information from the previous one to learn. Each level corresponds to a different area of the cerebral cortex. Every level abstracts more the information in the same way as the human brain. Image caption generation or automatic game playing could be two deep learning applications.

Since supervised learning is the most extended use of ML and the easiest to implement, it is decided to take this part as the starting point of this research.

2.1 Supervised Learning algorithms

Supervised learning (SL) domain includes algortihms able to perform regression (prediction) and classification tasks. The characteristics and relative performance of each algorithm can vary based upon the particulars of the data. Users should choose the algorithm that suits best their interests in terms of results and problem to solve.

The algorithms that compound the SL domain can be divided between parametric learners and non-parametric [17]. This research will focus on the first group, since they are easier to understand and implement than the second ones. These algorithms are:

1. Linear Regression: statistical method that allows to summarize and study relationships between two continuous (quantitative) variables, commonly used for permorming prediction tasks.
2. Logistic Regression: similar to the previous algorithm, it is the appropriate regression analysis to conduct when the dependent variable is dichotomous (binary). In this ocassion its best application are classification tasks.
3. Support Vector Machines (SVM): models with associated learning algorithms that analyze data used for classification and regression analysis. A SVM model is a representation of data as points in space, mapped so that the examples of separate categories are divided by a clear gap that is as wide as possible. They can be used both in regression and classification, specially useful if the data to analyze consists of multiple dimensions.
4. Neural Networks: computing systems inspired by the biological neural networks that constitute animal brains. Such systems learn (progressively improve performance on) tasks by considering examples, generally without task-specific programming. They are based on a collection of connected units or nodes called artificial neurons. Each connection (analogous to a synapse) between artificial neurons can transmit a signal from one to another until they produce a result. They are specially useful in complex classification problems like image recognition.

All of these algorithms share some features and parameters such as the use of labelled datasets, cost and accuracy functions, gradient descent... So the definition of an abstraction model covering of all them seems a feasible task.

With this goal in mind, it is decided to abstract first the regression algorithms (linear and logistic), since they are tightly related and share more similarities between them. The plan is to create a metamodel able to build instances for both of these algorithms.

2.2 Abstracting Regression algorithms

Once it is clear the algorithms that are going to be abstracted, it is necessary to identify which parts of them are shared, which not and how to model them.

During this step several examples of both algorithms are analyzed, written in Octave and Tensowflow. Some of the code analyzed can be found on this collection of Tensorflow examples [18].

The code examples were analyzed looking for similarities and differences. After its review it was clear that both regression algorithms:

1. Take as input labelled datasets, that may or not need preprocessing in order to be correctly interpreted by the system.
2. Have some initial parameters that are defined by the user (related to the learning phase and how to display the data). Most of these parameters are shared between both algorithms.
3. Perform a series of operations based on the previous parameters. These can be found always in differientated phases (training, evaluation and test) and in a similar order.
4. Display in plots information about these parameters, datasets, results or how internal features of the system are evolving (such as learning or accuracy).

Their main differences lay on the usage of diverse functions for the same purpose (for instance using two different functions to calculate the cost of the system). But since these functions use to appear always following the same order and using the parameters defined by the user, they can be easily abstracted and generate them depending on the algorithm chosen.

2.3 Regression metamodel

The next step for performing this abstraction is to define a metamodel taking into account the similarities extracted during the previous section.

So as to accomplish this goal, the tool selected is the Eclipse Modeling Framework (EMF) [19] due to its easiness of use and acceptance within the modeling community.

First of all, it is worth mentioning not only the successes of this phase but also the failures. Initial metamodel designs focused too much in modeling directly the regression algorithms, translating directly the code into UML notation, as can be seen in figure 1.

This metamodel was discarded as a bad approach towards the abstraction since it did not really improve the code notation, simply transformed it into a graphical way hiding some details.

In order to accomplish the ultimate goal of this research, making ML easier and more accessible, there is no use in following the path already defined by code.

Consequently, the metamodel definition was again tackled changing the design point of view, being the question to answer: *What would the user need to create a ML system without having ML specific knowledge?*

As a result a new metamodel is defined, as can be seen in figure 2. This new approach consists of the next elements:

1. MLSolution: the root of the metamodel.
2. Choice: the core of the system, the user will need to make a "choice" depending on what kind of ML problem want to tackle, prediction or classification (linear or logistic regression).
3. Parameter: a choice will count with 3 or 4 parameters depending on the algoritm chosen. Users will be able to edit the value of these attributes in order to fit the ML system.
4. Dataset: choices will also have at least one dataset from which learn and perform analysis. Users can define the type according to the dataset purpose (training o analysis) and also select if it needs to be preprocessed or not.

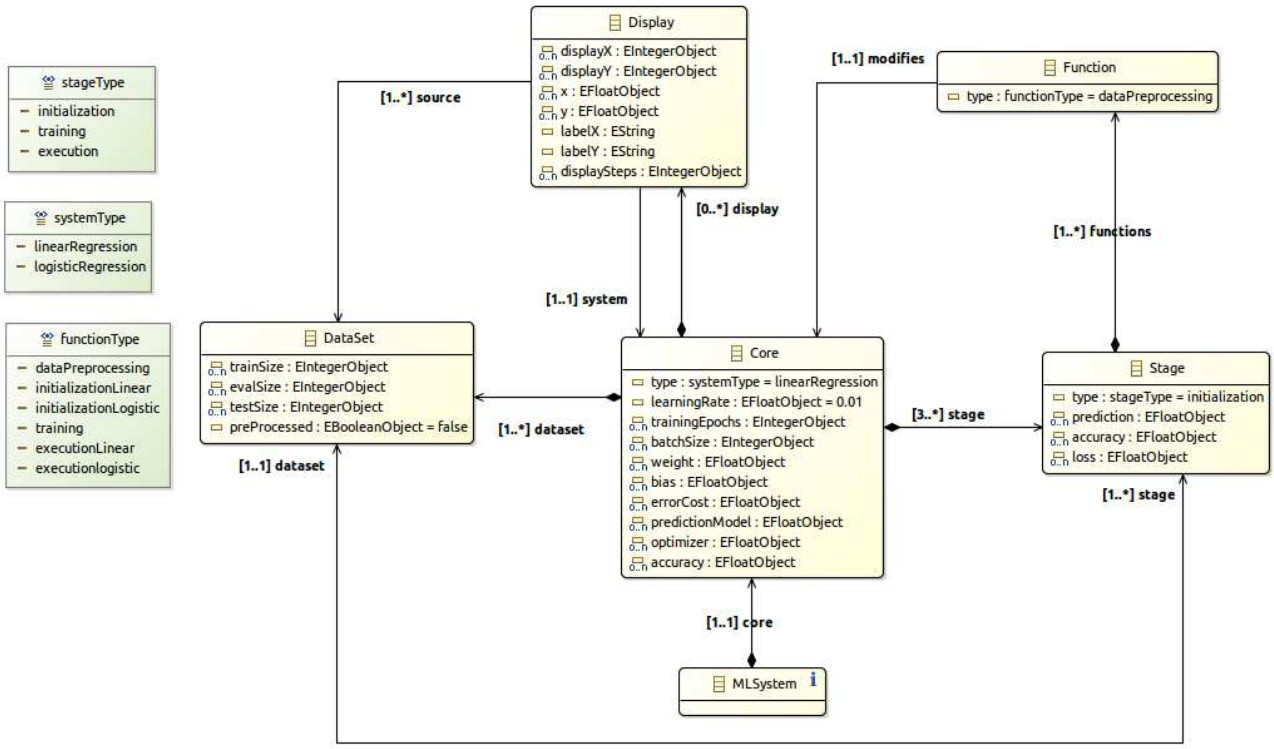
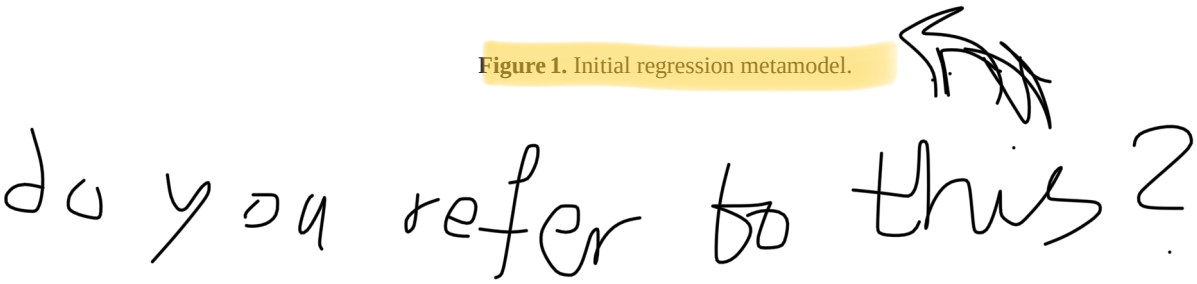


Figure 1. Initial regression metamodel.



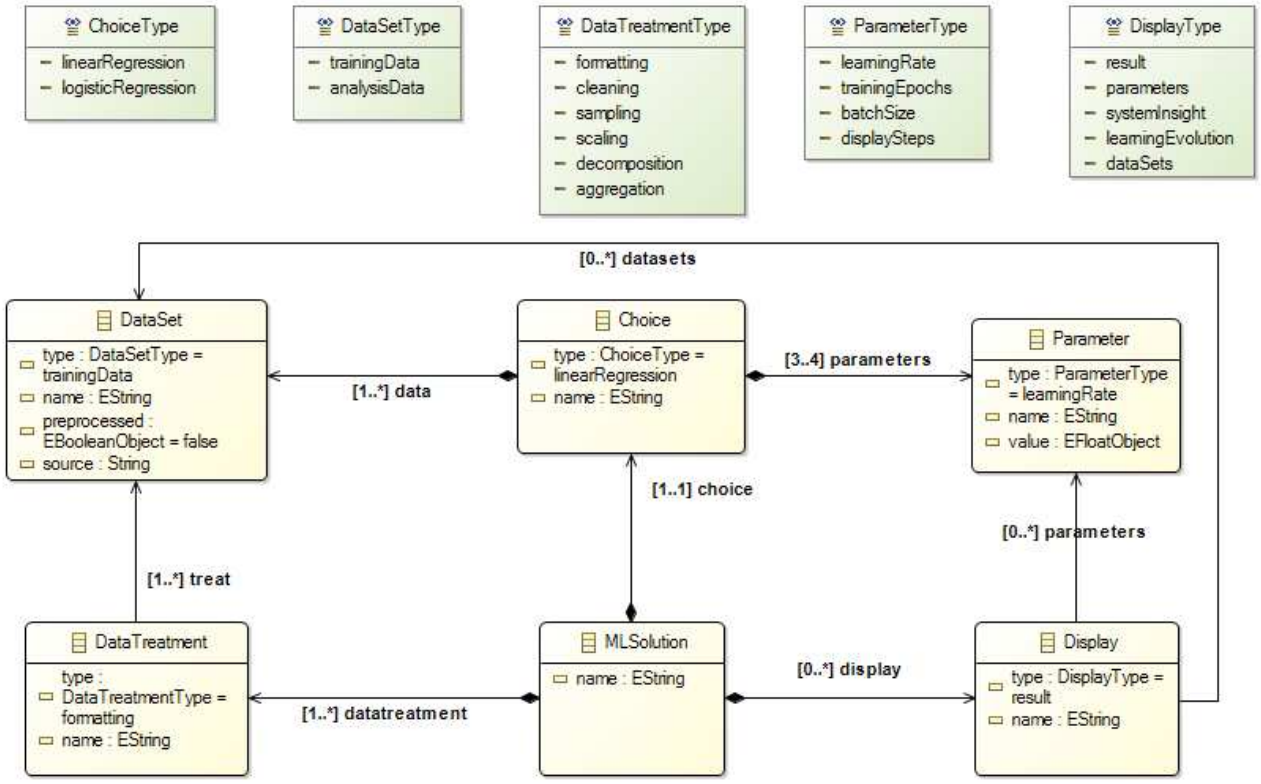


Figure 2. Final regression metamodel.

5. Data Treatment: in order to make datasets understandable for ML systems, they need to have a specific format, size... Users can select what operations need to be done on their datasets, choosing between the six functions needed to fully preprocess ML data [20].
6. Display: users can check graphically different information from the system; datasets, parameters, results or some predefined displays that will show insight into some system features such as the learning curve.

Given its features, this metamodel could help users to define a ML system in a more accessible way than traditional code, providing them with enough freedom to still being able to change parameters and configurations.

2.4 Regression graphical editor

The next step is to create a concrete syntax for defining instances (models) of this metamodel. Sirius [21] will be used to define a graphical concrete syntax. This tool is widely used in the MDSE community and integrates perfectly with EMF, easing the process to create a graphical editor based on a metamodel.

It has been chosen to build a graphical tool since its interface is generally easier to understand for most users, allowing them to comprehend the ML environment they are creating at a single glance.

The result of this step is a graphical editor for creating instances of the metamodel stated in the previous section. Figure 3 shows the available tools on the editor: all elements defined in the metamodel and arcs to configure the relantionships between some of them (Datasets-Data Treatment and Display-Parameters/Datasets).

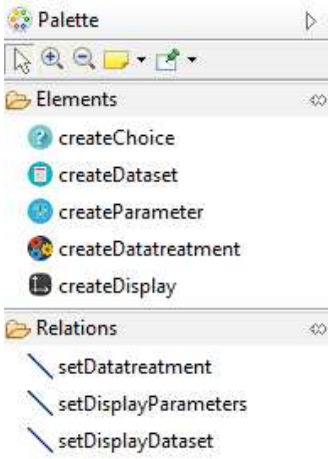


Figure 3. Regression editor tools.

Using this tool palette users can define an instance as the one showed in figure 4 just by dragging the elements and connecting them with arcs. The parameters edition can be done as showed in figure 5.

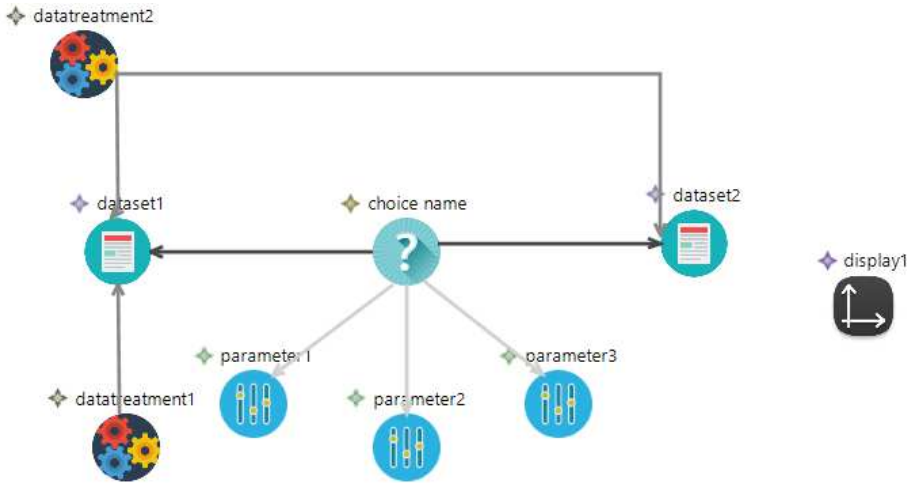


Figure 4. Regression instance created with the graphical editor.

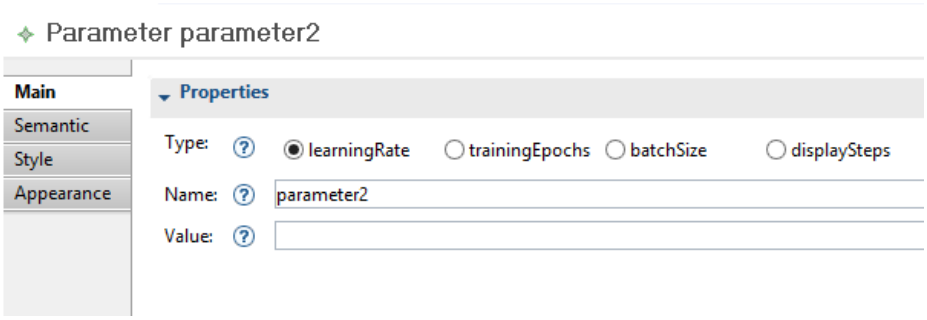


Figure 5. Attributes edition in the graphical editor.

As a result, the editor can create instances as the one showed in figure 6.

3 Results

The content of this section focuses on the results obtained with the regression metamodel and its graphical editor. In order to properly stress the benefits of this approach in

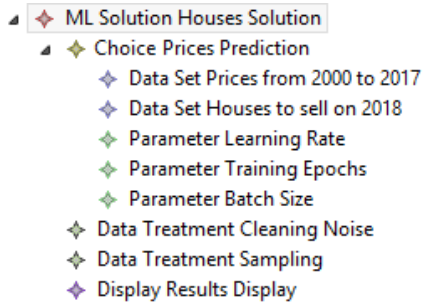


Figure 6. Instance created using the graphical editor.

contrast to generating a ML system as traditionally with code, a comparative analysis between both techniques is performed.

For this purpose it is chosen as a case study the well-known problem of house prices prediction, which can be solved using linear regression. The solution to this problem is the prediction of house prices given some variables (number of bedrooms and bathrooms, year built, square meters, condition, zipcode, latitude, longitude...), taking as training data information about other houses and their prices.

The code to analyze is written in Tensorflow, since this ML framework is one of the most extended, used and robust at the moment. This concrete example aims to predict house prices based on the number of room and total size of the place [22]. In the example below some parts of the code are omitted since they are quite generic, these sections are explained in comments.

```
# Imports and dependencies

rng = np.random

# Datapath and columns declaration

learning_rate = 0.01
training_epochs = 1000
display_step = 10

# Data initilization (data, x1, x2, y, train_variables, n_samples, X1, X2, Y)

W1 = tf.Variable(rng.randn(), name="weight1")
W2 = tf.Variable(rng.randn(), name="weight2")
b = tf.Variable(rng.randn(), name="bias")

sum_list = [tf.multiply(X1,W1),tf.multiply(X2,W2)]
pred_X = tf.add_n(sum_list)
pred = tf.add(pred_X,b)

cost = tf.reduce_sum(tf.pow(pred–Y, 2))/(2*n_samples)
```

```
optimizer = tf.train.GradientDescentOptimizer(learning_rate).minimize(cost)

init = tf.global_variables_initializer()

with tf.Session() as sess:
  sess.run(init)

  for epoch in range(training_epochs):
        for (x1, x2, y) in zip(train_X1, train_X2, train_Y):
          sess.run(optimizer, feed_dict={X1: x1, X2:x2, Y: y})

    if (epoch+1) % display_step == 0:
          c = sess.run(cost, feed_dict={X1:train_X1 , X2:train_X2, Y: train_Y})
          print( "Epoch:", '%04d' % (epoch+1), "cost=", "{:.9f}".format(c),"W1=",
          sess.run(W1), "W2=", sess.run(W2), "b=", sess.run(b))


  training_cost = sess.run(cost, feed_dict={X1:train_X1 , X2:train_X2, Y: train_Y})
  print("Training cost=", training_cost, "W1=", sess.run(W1), "W2=",
  sess.run(W2), "b=", sess.run(b), '\n')

# Print several values

# Graphic result display
```

Tensorflow is a high level language that already abstracts many parts of ML, for example it provides a gradient descent function and does not ask users to build it themselves. However, for most developers without knowledge of ML this is not a code understandable at a first attempt, for several reasons: its specific jargon and its background (math and statistics).

In order to get the ability to reproduce this code for themselves it is required to learn or at least possess a foundation on maths an statistics applied to data science, as well as knowledge of the Tensorflow library.

Still, taking a closer look at this code and analyzing more examples, it is easy to find a pattern between them and how some of the tasks defined could be easily automated and structured in blocks. For instance in the example it is necessary that users indicate which are their data and paramater values, while the rest of the code is based on consecutive functions making use of them in a concrete order. This statement can also be applied to logistic regression and probably to the rest of parametric SL algorithms.

Getting this idea into the editor, users would be able to create models as in figure 6 in a graphical way as showed in figure 3. This way they would define the value of their data, parameters, algorithm to use, data treatment if needed and displays as convenience. Having this prototype, it could be translated into code just by taking users values and generating functions in the same order as the code example shows.

Currently, the editor still uses some ML jargon although less than in Tensorflow. Next versions of the tool will abstract this syntax further.

In conclusion, the editor allows users to solve regression problems in a easier and more accessible way than traditional code. They can build their own ML system just by knowing how to use the editor and each element purpose.

4 Future work

There are several lines to explore as future work. Firstly, in order to validate the metamodel approach, it is planned to perform several interviews with ML experts in order to know their opinion and integrate their feedback into the framework.

Then, it is expected to integrate SVMs and neural networks into the metamodel, covering a big part of the SL domain.

As the metamodel evolves the editor will do the same. Ultimately, it is planned to transform the editor into a more intuitive interface in which users can select different options and introduce their inputs (data, parameters, options...). All technical jargon will be hidden in the future.

Finally, the tool should be able to generate code automatically, so that users can create the totality of a ML system without the need to write any code by themselves. The language chosen for this automated code generation will be subject of study, being one of the options Python with Tensorflow.

5 Conclusions

This project aims to improve and make more accessible ML development, creating a software framework that allow users to work with this kind of technology without having technical knowledge about it.

In order to build said tool, the first step goes through indentifying the best way to start ML domain's abstraction. During this paper it has been explained the reasons behind the decission to start said abstraction using the SL subdomain, specifically regression algorithms.

The abstraction of these algorithms has lead to the design of a regression metamodel and a concrete graphical syntax for it, thus developing a graphical editor that allows accessible regression ML systems prototyping.

Finally, the results obtained from said editor have been compared with a traditional code example, showing the reasoning behind the metamodel design and how it could improve users experience towards the ML field.

There is a wide range for future work, interdisciplinary collaborations and evolution relating this work yet, so the results presented in this paper are still a work in progress.

References

1. Pedro Domingos. A few useful things to know about machine learning. *Communications of the ACM*, 55(10):78–87, 2012.
2. Dominic Breuker. Towards model-driven engineering for big data analytics–an exploratory analysis of domain-specific languages for machine learning. In *System Sciences (HICSS), 2014 47th Hawaii International Conference on*, pages 758–767. IEEE, 2014.

3. Christopher M Bishop. Model-based machine learning. *Phil. Trans. R. Soc. A*, 371(1984):20120222, 2013.
4. Gustavo Rossi and Daniel Schwabe. Model-based web application development. *Web engineering*, pages 303–333, 2006.
5. Mohamed Lachgar and Abdelmounaïm Abdali. Generating android graphical user interfaces using an mda approach. In *Information Science and Technology (CIST), 2014 Third IEEE International Colloquium in*, pages 80–85. IEEE, 2014.
6. Mitchel Resnick, John Maloney, Andrés Monroy-Hernández, Natalie Rusk, Evelyn Eastmond, Karen Brennan, Amon Millner, Eric Rosenbaum, Jay Silver, Brian Silverman, et al. Scratch: programming for all. *Communications of the ACM*, 52(11):60–67, 2009.
7. John Maloney, Mitchel Resnick, Natalie Rusk, Brian Silverman, and Evelyn Eastmond. The scratch programming language and environment. *ACM Transactions on Computing Education (TOCE)*, 10(4):16, 2010.
8. Frank Klassner. A case study of lego mindstorms suitability for artificial intelligence and robotics courses at the college level. In *ACM SIGCSE Bulletin*, volume 34, pages 8–12. ACM, 2002.
9. David H Guston. Forget politicizing science. let's democratize science! *Issues in Science and Technology*, 21(1):25–28, 2004.
10. Miles Brundage. Economic possibilities for our children: Artificial intelligence and the future of work, education, and leisure. In *AAAI Workshop: AI and Ethics*, 2015.
11. Kiri Wagstaff. Machine learning that matters. *arXiv preprint arXiv:1206.4656*, 2012.
12. Jon Whittle, John Hutchinson, and Mark Rouncefield. The state of practice in model-driven engineering. *IEEE software*, 31(3):79–85, 2014.
13. Thomas Weigert and Frank Weil. Practical experiences in using model-driven engineering to develop trustworthy computing systems. In *Sensor Networks, Ubiquitous, and Trustworthy Computing, 2006. IEEE International Conference on*, volume 1, pages 8–pp. IEEE, 2006.
14. Davide Brugali. Model-driven software engineering in robotics: Models are designed to use the relevant things, thereby reducing the complexity and cost in the field of robotics. *IEEE Robotics & Automation Magazine*, 22(3):155–166, 2015.
15. Peter Harrington. *Machine learning in action*, volume 5. Manning Greenwich, CT, 2012.
16. Stuart Russell, Peter Norvig, and Artificial Intelligence. A modern approach. *Artificial Intelligence. Prentice-Hall, Egnlewood Cliffs*, 25:27, 1995.
17. Sabri Maini. *Machine Learning for Humans*. Edited by Sachin Maini, 2017.
18. Github - aymericdamien/tensorflow-examples: Tensorflow tutorial and examples for beginners with latest apis. `https://github.com/aymericdamien/TensorFlow-Examples`. (Accessed on 01/17/2018).
19. Eclipse modeling project. `https://www.eclipse.org/modeling/emf/`. (Accessed on 01/17/2018).
20. How to prepare data for machine learning - machine learning mastery. `https://machinelearningmastery.com/how-to-prepare-data-for-machine-learning/`. (Accessed on 01/17/2018).
21. Sirius - the easiest way to get your own modeling tool. `http://www.eclipse.org/sirius/`. (Accessed on 01/17/2018).
22. tensorflow_with_2variables/tf_with_2variables.py at master · natarslan/tensorflow_with_2variables · github. `https://github.com/natarslan/tensorflow_with_2variables/blob/master/TF_with_2variables.py`. (Accessed on 01/18/2018).