POLITECNICO DI TORINO

Master degree course in Computer Engineering

Master Degree Thesis

Defectiveness prediction in software development via machine learning

Artificial Intelligence applied to Software Engineering

Supervisors
prof. Maurizio Morisio

Candidates
Jacopo Nasi [255320]

Internship Tutor
dott. Davide Piagneri

Anno accademico 2019-2020

Summary

Everyday thousands line of code are committed, each of which contains a lot of informations, the file changed, the modifications, comments and even more. A structured and correctly managed sourcing control allow the extraction of useful data that can be analysed using neural networks.
The evaluation of the extracted data require different phases: the first step aim to study the structure of the data in order to correctly understand the data model and allow to use most of the data, next the data must be pre-processed to remove outliers and not useful informations, then more data is extracted, information like developer seniority, bag of words of component edited, linked version and potentially useful informations. The final step is the weekly aggregation by computing the difference between the closed and opened issue, traslating the priority label to a value extracted from the duration distribution of the different classes. Once the data is extracted will be passed to the machine learning models: Random Forest, Gradient Boosting and Neural Networks, to be forecasted with different time horizons. The generated data is then be used in different way: train-test over single extraction, cross version and even cross project.
The execution with the best performance is the cross project one, with a precision greater than 90% up to 4 weeks and still greater than 70% up to 20 weeks.

Acknowledgements

Un ringraziamento speciale a Smirnuff ed i suoi cavalieri, luce della mia battaglia.

Contents

Chapter 1

Introduction

1.1 General Problem

The software development is fundamental in the new world, how about using artificial intelligence to improve it?
The development of a software product is not different from any kind of hardware product development, after the first phase of design the production starts, during it problems emerge systematically and must be managed before the release in production.
Every day each a software house perform hundreds of commit, each one contains a lot of informations linked to an issue, with structured commit, driven issue report and other software engineering stuff, the quality of the sourcing improve, allowing data to be used in artificial intelligence analysis. Predicting defectiveness during the making of the software can drastically improve the process of development, allocating the correct number of developer could reduce the time required and avoid delay and problems before releases. With a correct preprocessing and evaluated aggregation, neural networks can predict, with high accuracy, the defectiveness trend. These forecast can be further used to improve the software engineering background of the related project.
Forecasting is one of the most critic part of a company, it could drive to easily success as well as drive to failure. A software project is not different from a manufacturing product, its development, infact, require analysis of different kind, from resources needed to costs and time required.
The software development experience shows that the process of analysis is really difficult, due to the nature of the problem, coding is a mind product and the time required to produce it can varying in accord to a lot of different factors.

1.2 Tools used

This work is mainly conducted using software tools, here a list of the tools used:

Python The main programming language of the thesis project. Used for data management, feature extraction, machine learning models and for interfaction with other softwares. The specific version used is the v3.7.0

Pandas Open source, BSD-licensed library providing high-performance, easy-to-use data structures and data analysis tools for the Python programming language.

NumPy Scientific computing with Python.

Matplotlib 2D Plotting library for Python.

Seaborn Another plotting library for Python.

Tensorflow Platform for machine learning.

Keras High level API for neural networks.

SciKit-Learn Tools and libraries for machine learning.

GitLab Sourcing platform based on Git. Used for the code of the project.

GitHub Sourcing platform based on Git. Used for the thesis and calendar sourcing:

- Thesis: `https://github.com/Jacopx/Thesis`
- Calendar: `https://github.com/Jacopx/ThesisCalendar`

JetBrains IDEs Student-free IDE for different language development, product used:

- PyCharm: `https://www.jetbrains.com/pycharm/`
- DataGrip: `https://www.jetbrains.com/datagrip/`

Chapter 2

State of art

2.1 Related works

Speaking about other works.

Chapter 3

Datasets

The following section illustrate the structure of the all the principal datasets used during this thesis project.

3.1 SEOSS33

The SEOSS33[1] is a dataset collecting bug, issue, reports, commit and lot of other information of 33 open source project, following their progress via sourcing platform. The dataset is enriched also with time stamps, release versions, component information and developer comments.
At today there are no other public research conducted over this datasets, this works seems to be first.
The data is retrived by the issue tracking system (ITS) and the version control system (VCS). To unify the project specific difference, the typed issue, e.g. *New Feature* or *Bug Report*, are mapped to five issue categories:

- Bug: A problem which impairs or prevents the functions of the product
- Feature: A new feature of the product
- Improvement: An enhancement to an existing feature
- Task: A task that needs to be done
- Other: Various

The study collects 33 projects that are using Atalassian Jira and git, for popularity reasons. Is also required that the projects in the dataset should be majorly written in one programming language, Java is choosen. Due to machine learning nature, the choosen project must have great number of issue, all project were in development for at least three years. Among these products we have selected five of them, because of size, as shown in table 3.1:

Table 3.1. Project data distribution

Project	Month	Issue
Hadoop	150	39086
Hbase	131	19247
Maven	183	18025
Cassandra	106	13965
Hive	113	18025

Figure 3.1 show the distribution of each issue category respect all the projects.

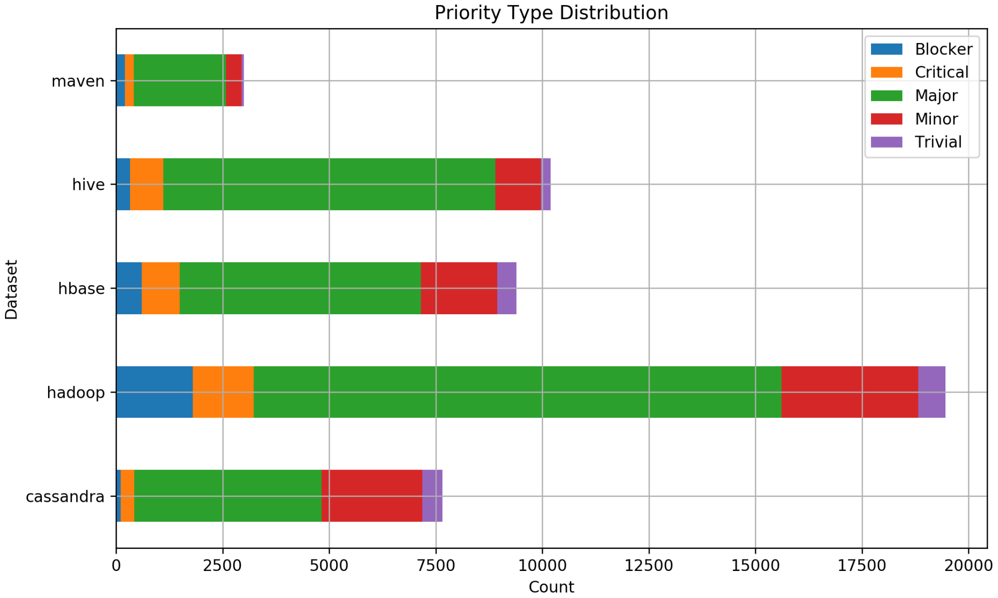
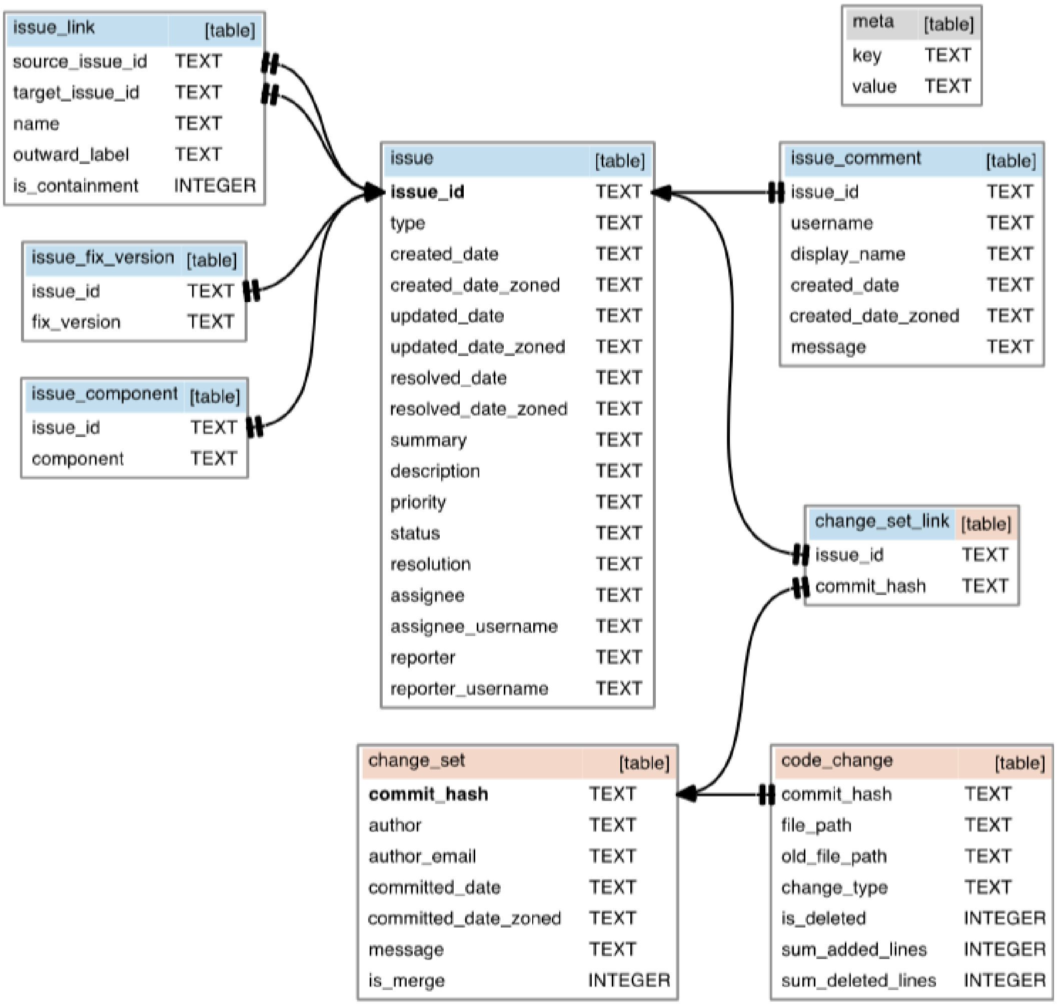


Figure 3.1. Count of different priority for each project

More selection characteristics can be found in chapter 2.1 [1].
Is fundamental to understand the structure of this dataset, the majority of the forecasting operation tests are conducted using the data stored by this research.
Each project is stored in a SQLITE file, a SQL offline database, the structure is based on the entity of the *issue*, identified by an *issue_id*, the other tables are used to link additional information, like the number of commit, the version referred, comments and others features. The figure 3.2 show the database schema.

Each table contains different types of data that will be used to generate the data used to forecast.



issue_link	[table]
source_issue_id	TEXT
target_issue_id	TEXT
name	TEXT
outward_label	TEXT
is_containment	INTEGER

issue_fix_version	[table]
issue_id	TEXT
fix_version	TEXT

issue_component	[table]
issue_id	TEXT
component	TEXT

issue	[table]
issue_id	TEXT
type	TEXT
created_date	TEXT
created_date_zoned	TEXT
updated_date	TEXT
updated_date_zoned	TEXT
resolved_date	TEXT
resolved_date_zoned	TEXT
summary	TEXT
description	TEXT
priority	TEXT
status	TEXT
resolution	TEXT
assignee	TEXT
assignee_username	TEXT
reporter	TEXT
reporter_username	TEXT

meta	[table]
key	TEXT
value	TEXT

issue_comment	[table]
issue_id	TEXT
username	TEXT
display_name	TEXT
created_date	TEXT
created_date_zoned	TEXT
message	TEXT

change_set_link	[table]
issue_id	TEXT
commit_hash	TEXT

change_set	[table]
commit_hash	TEXT
author	TEXT
author_email	TEXT
committed_date	TEXT
committed_date_zoned	TEXT
message	TEXT
is_merge	INTEGER

code_change	[table]
commit_hash	TEXT
file_path	TEXT
old_file_path	TEXT
change_type	TEXT
is_deleted	INTEGER
sum_added_lines	INTEGER
sum_deleted_lines	INTEGER

Figure 3.2. SEOSS33 data model

Chapter 4

Machine Learning

4.1 Introduction

Machine learning (ML) is the scientific study of algorithms and statistical models that computer systems use to perform a specific task without using explicit instructions, relying on patterns and inference instead. The word ML is almost in the public domain now, in the last decades the usage of these kind of algorithms has drammatically raisen although most of it had already been developed for years. The main reason is the increase in the computational capacity of the systems.
There are two types of systems:

- Knowledge-based: Acquisition and modeling of common-sense knowledge and expert knowledge (from rules to facts)
- Learning: Extraction of knowledge and rules from example/experience (from facts to rules)

All the machine learning techniques try to develop systems similar to the second definitions. ML algorithms can be divided in three categories: supervised, unsupervised and reinforcement learning. The first will be used to predict class based on previous knowledge, the second tries to labeling data without a priori experience and the third bases its learning on action reward.
There a lot of different models available, the following chapters will focus on the models used in this project.

Supervised Learning is a powerful technique to process labeled, which is a dataset with that has been classified, to infer a learning algorithm. These dataset is used as basis to predict, and learn how, unlabeled data. There are two types of supervised learning, classification and linear regression. The goal of classification models is to predict categorical class labels of new instances, based on a training set of past observations. The classification can be binary or multi-class.

Instead, regression, aim to predict continuos outcome, it tries to find mathematical relationships between variables to predict the target with a reasonable level of approximation. Our project is only focused on regression, classification will not be further discussed.

4.2 Ensable Methods

All the supervised learning method used are based on ensable, the basic idea is to merge multiple, different, hypotesis in order to improved the quality of the prediction, for example the random forest or gradient boosting are ensable of multiple decision trees. The following paragraphs will deal with the models used specifically.

Random Forest Random Forest (RF) is a supervised learning algorithm which uses ensable learning method for classification and regression. They are built by combining the predictions of several trees, each of which is trained independently and the prediction of the trees are combined through averaging [2]. A visualization of a RF is shown in figure 4.1.

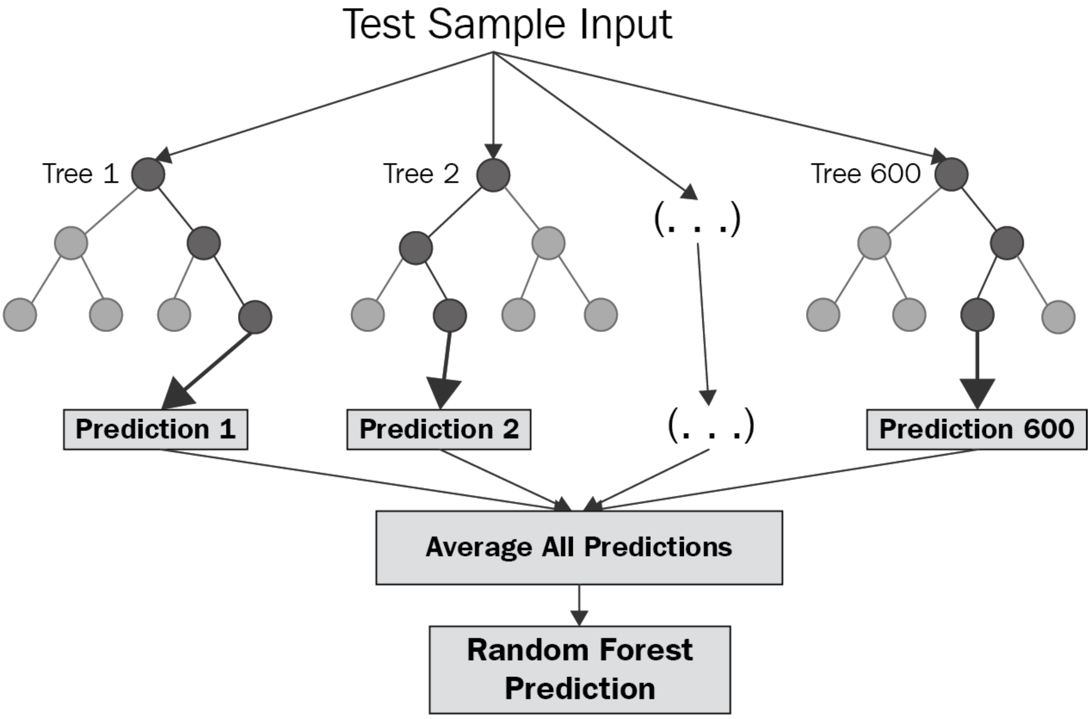


Figure 4.1. Random Forest simplified scheme [3]

Three parameters are important to define the RF: (1) the methods for splitting the leafs, (2) the type of predictor to use in each leaf, and (3) the method for injecting randomness. Splitting of require the selection of shape and a method for evaluating the quality of each candidate. Typical choices are axis aligned splits, where data is routed to sub-tree depending on a threshold. The threshold can be chosen randomly of by a leafs optimization function. After the generation of different candidate splits a choosing criterion must be defined in order to split the leaf. One of the simplest strategy is to choose among the candidates uniformly at random, otherwise is possible to select the candidate split which optimize a purity function (information gain for example) over the leaf that would be created. The most common predictors, for each leaf, is the average response over training points which fall in that leaf. The injection of randomness can happen in different ways, the size of candidate splits, coefficients for random combinations, etc...In any case the thresholds can be also defined randomly or by optimization over data. Another solution is build tree using bootstrapped or sub-sampled dataset.

The training phase is performed independently by each tree by exploiting an assignement in structure and estimation points to respectively change the shape of the tree and to improve the estimator fit.

Once the forest has been trained it can be used to make predictions for unlabeled data points. In the prediction phase, each single tree, independently make its own prediction than an average of all the trees is computed to make a single outcome value. The contribution of each tree to the final value is the same.

Out implementation use Random Forest developed by SciKit-Learn v0.21:

```
from sklearn.ensemble import RandomForestRegressor
```

The specific parameters of the model will be explained during the chapter 5 about forecasting.

Gradient Boosting Machines Gradient Boosting Machines (GBM) are a family of powerful machine learning techniques that have shown considerable success in a wide range of pratical application. They are higly customizable to the particular needs of the application, line being learned with respect to different loss functions [4]. Techniques like RF rely on simple averaging of model in the ensable. The family of boosting methods is based on a different constructive strategy of ensemble formation. Boosting add, sequentially, new models to the ensable. In GBM the learning phase consecutively fits new models to improve the accuracy of the estimations. Ideally they construct new basic layers, decision trees of fixed size, to be maximally correlated with the negative gradient of the loss function of the ensable. The loss function should be chosen by the developer, even ad-hoc loss function could be implemented. The attempt is to solve this minimization problem numerically via steepest descent [5].

This high flexibility of GBM makes them really customizable to any kind of task. Given its simplicity a lot of experimentation can be performed over the model.

Our project implement Gradient Boosting Decision Tree (GBDT) developed by SciKit-Learn v0.21:

```
from sklearn.ensemble import GradientBoostingRegressor
```

4.3 Neural Networks

A Neural Network (NN) is a machine learning approach inspired by the way in which the brain performs a particular learning task, network of simple computational units (neuros) connected by links (synapses). The knowledge about the learning task is given in the form of training examples, the inter neuros connection strengths (weights) are used to store the acquired information. During the learning process the weights are modified in order to model the particular learning task correctly on the examples.

The learning phase can be both supervised or unsupervised. Supervised is used for pattern recognition, regression and similar, is trained using data with both input and desired output. Unsupervised instead is mainly used for clustering and the learning is made using unlabeled training examples. Only supervised one will be evaluated.

There are three main classes of network architectures:

- Single-layer feed-forward
- Multi-layer feed-forward
- Recurrent

A standard architecture is composed of three different layers, *input units*, *hidden units* and *output units*, all the layers are linked with the learning algorithms used to train. An example of a complete multi-layer network in 4.2.

The neuron is the basic processing unit of the network, which received the inputs. Each input is cobined with its internal state and an optional activation function, it produce an output value that will be passed to the next layers. The weights, defined during the training phase, correspond to the importance of the connection. The different inputs are summed using a weighted sum:

$$u = \sum_{j=1}^{m} w_j x_j \tag{4.1}$$

The computed value is than scaled using an activation function φ for limiting the amplitude of the output of the neuron by applying the function:

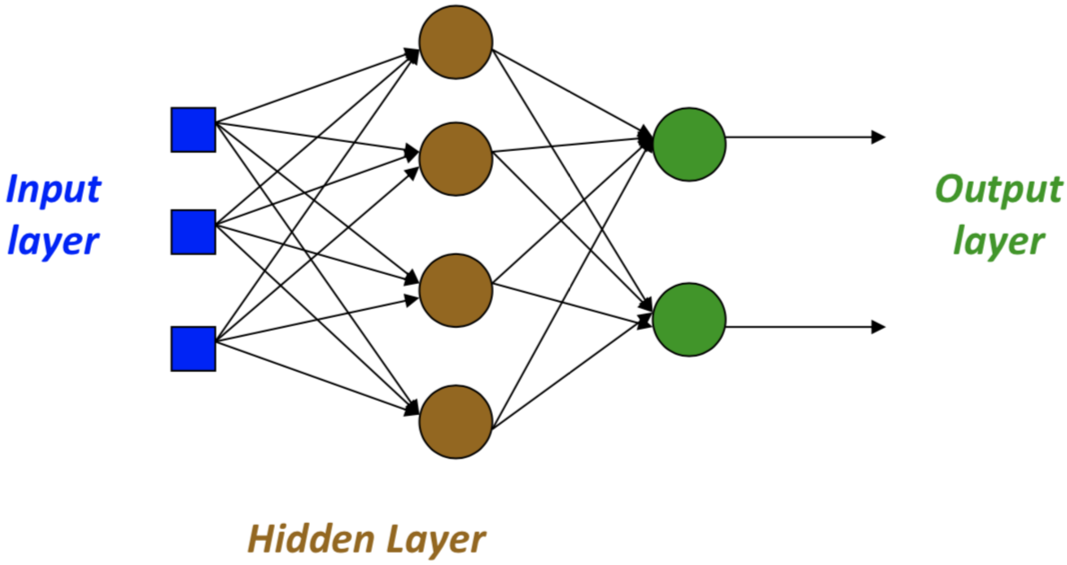


Figure 4.2. Multi-layer feed-forward NN

$$y = \varphi(u + b) \tag{4.2}$$

Where b represent the bias, an external parameter of the neuron, y represent than the output value for the next level. An example of the structure can be found in figure 4.3.

There are several different functions that can be used to activate the neurons, they are trying to emulate the typical response of a biological neuron and its different activation methods. The most common are: linear, step, relu and sigmoid. Each functions behave in a different way with different peculiarities and critical issues, there are two main type, linear and non-linear.

Step is a binary, linear and threshold-based activativation function, figure 4.4. When the input is above or below a defined threshold, the neuron is activated and pass the exact same input value to the next layer. The main drawback is that not allow multiple-value output.

Linear is of course a linear activation function that take the form:

$$f(x) = x \tag{4.3}$$

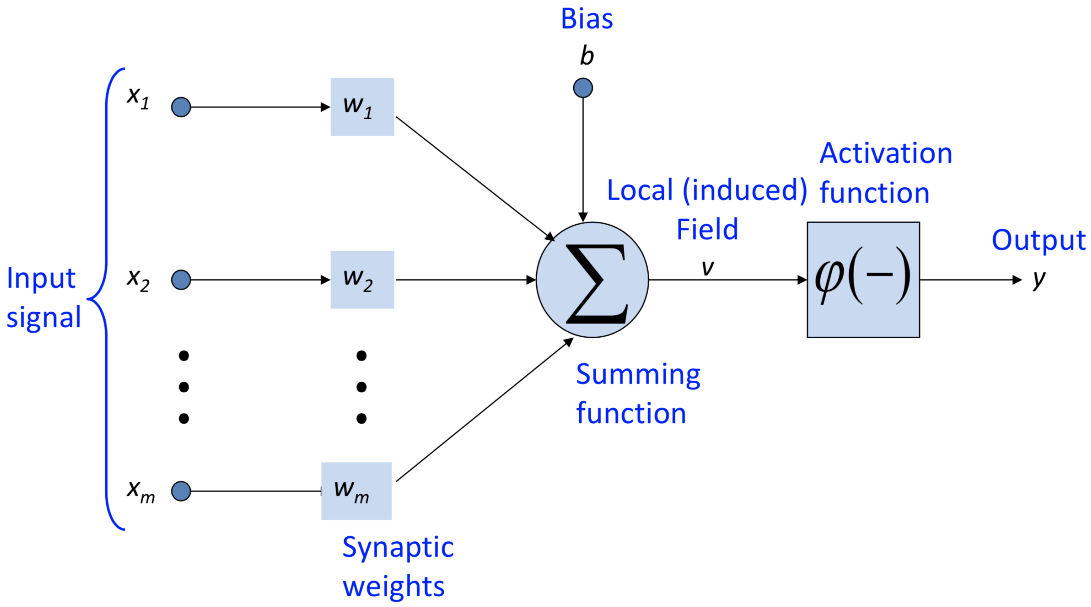


Figure 4.3. Neuron view

This function take the input and, by multiply it by the weight of the neuron, it calculate the output. Respect the step function allows multiple-value output, but there are two major problem: is not possible to use backpropagation (treated later) to train the network because the derivative function is constant and is not related to the input. The other problem is that linear function collapse all the layer into once, because the last layer will be a linear combination of the first in any case. Figure 4.5 show the curve.

Sigmoid is a non-linear function defined in the following way:

$$f(x) = \frac{1}{1 + e^x} \tag{4.4}$$

The advantages are the smooth gradient that prevents out-of-scale output values, the implicit normalization bound from $[0, +1]$ and the cleaning of the forecast. The main drawback is the vanishing of the gradient, for very high or low input value there is no change in the prediction output, the sigmoid is also computationally expensive. The figure 4.6 show the curve.

ReLU that stand for Rectified Linear Unit and is defined in the following way:

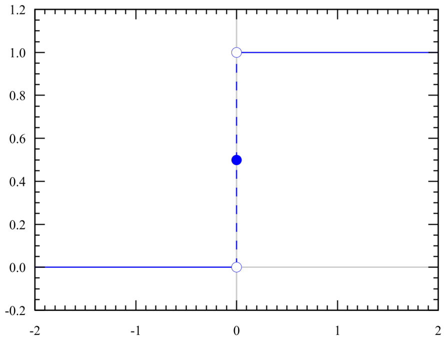


Figure 4.4. Step activation function

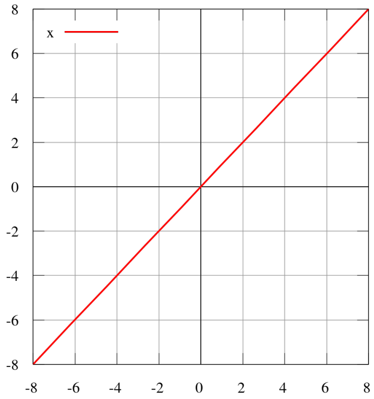


Figure 4.5. Linear activation function

$$f(x) = max(0, x) \tag{4.5}$$

Although it looks like a linear function, ReLU has a derivative function and allows for backpropagation another advantage is that is computationally efficient. The main problem of this function is its behaviour with value near zero or even negative, the gradient of the function become zero and the network can't perform backpropagation and can't learn.

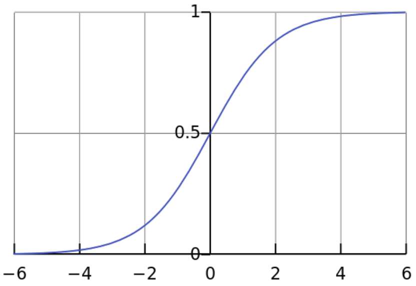


Figure 4.6. Sigmoid curve

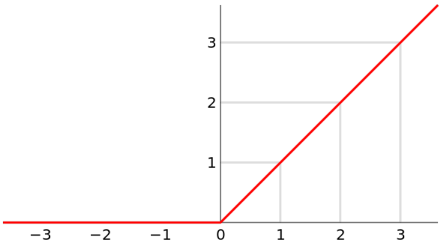


Figure 4.7. ReLU activation function

Delta learning rule uses the difference between target output and the obtained activation to drive the learning. According to this rules, each time an output is computed, the weight of the neuron is adjusted according to an error function trying to reduce the difference between output and target.

Backpropagation short for *"backward propagation of errors"*, is an algorithm for supervised learning of artificial neural networks using gradient descent. Given an artificial neural network and an error function, the method calculates the gradient of the error function with respect to the neural network's weights. It is a generalization of the delta rule for perceptrons to multilayer feedforward neural networks. [6].
The main characteristics of this technique is that, the gradient, proceeds backward through the network, with the gradient of the final layer calculated before the first. This solution allow an efficient calculation of the gradient for each layer. The algorithm is structured in this form:

1. Compute the error term for the output units
2. From the output layer, repeat until last hidden reached:
    (a) Propagating the error term back to the previous layer
    (b) Updating the weights between the two layers

The BP suffer the Vanishing Gradient problem, more layers are embedded in the network and more difficult become to train it. Due to the nature of the backpropagation, when an output is generated, the weight of the neuros is fixed according to the rule, each layer back the correction value is smaller, following the derivate of the activation function, in case of slallow network this problem is solved easily, for deep network this can become a real issue. The ReLU activation function manage better this problem. Another solution is the batch normalization, that rescale the input value between $[-1,1]$ in order to keep the input value far from the activation function edges. Exist a kind of network, Long Short Term Memory, developed to mitigate the vanishing gradient problem, will be discussed lately.

Recurrent Neural Networks (RNN) is a class of NN that keep connections between nodes and a temporal sequence. The main difference, respect NN, is that has feedback connections, this memory allow to keep track of temporal dynamic behaviour, they can process single data point or entire sequence of data, like video or speech. They become useful as their intermediate values can store information about past inputs for a time that is not fixed a priori.
This kind of structures is used in a lot of different fields: image classification, captioning, sentiment analysis, machine traslation, video classification, etc...

Long Short Term Memory (LSTM) normal recurrent neural networks can connect short term event with the present, this feature can be useful in some case, other context could require more long term connection, this special RNN can handle this sort of correlations. Sometimes the recent information is enough to perform the present task, for example, trying to predict the last word of a sentence like: "Sun bright in the *sky*" the previous words are enough to corretly predict the last one,

no more context is required. In case of more complex sentences some additional information are needed, for the phrase: "I was born in Italy and I speak *italian*" the context become fundamental to solve the problem. In this kind of problem LSTM can be really helpful.
LSTM are a special kind of RNN, they were introduced by Hochreiter & Schmidhuber (1997) [7] and lately studied and improved. They works really well on a large variety of problems. The special kind of network is explicitly design for long term connections. The structure of a RNN is normally a single layer, in case of LSTM the structure is more complicated, is made upon four different layers, as shown in figure 4.8.

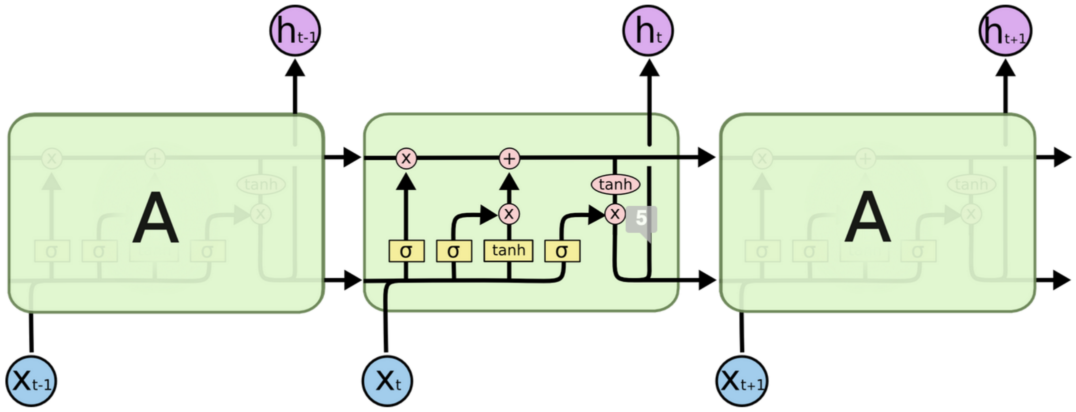


Figure 4.8. Long Short Term Memory layout [8]

The first layer filter the input and select what keep and not keep, the second layer will manage what information store in the cell state. The third state is in charge of update the state based on the previous step. The last one decide the output.

4.4 Evaluation metrics

Every model developed need to be evaluated, there are a lot of different methods that can be used in order to get a metrics about the quality of the system behaviour. The computation of this values is achieved via mathematical expression, for this reason, the results, are not always human readable, some error like mean absolute error, mean relative error and similar can be read and interpreted without any problem. For metrics like R2 or mean squared error the task become more complicated. The following is about the mathematical analysis and description of the different metrics used.

Most of metrics directly derive from the calculation of the simplest error, the *absolute error*, the difference between the target y and the prediction x:

$$\epsilon = |y - x| \tag{4.6}$$

Besides being the easiest to calculate is also the easiest to be understand because show directly the distance respect the desired value.
The first metric based on the previous metric is the *relative error*, that is computed dividing the absolute error by the target value:

$$\eta = \frac{\epsilon}{|x|} = \frac{|y - x|}{|x|} \tag{4.7}$$

This error rescale the output between [0,1] in order to better understand the error in the estimation. For example, in case of a value target value of 530 and a prediciton value of 570 the two error are:

$$\epsilon = |520 - 570| = 50 \tag{4.8}$$

$$\eta = \frac{50}{|520|} = 0.09 \tag{4.9}$$

the difference was 50, but respect the expected output di difference is really low, around 9%.
The previous errors are the basic ones, the following is the first based on aggregation of multiple errors calculation. The aggregation is fundamental to generate a single number to evaluate the quality of prediction of our model.
The *Mean Absolute Error* (MAE), is probably the easiest to be calculated and understand, is basically computed by calculating the average of all the absolute errors:

$$MAE = \frac{\sum_{i=1}^{n} |y_i - x_i|}{n} = \frac{\epsilon}{n} \tag{4.10}$$

Chapter 5

Forecasting

Chapter 6

Conclusion

Speaking about conclusion.

Bibliography

[1] M. Rath, P. Mäder, "The SEOSS 33 Dataset — Requirements, Bug Reports, Code History, and Trace Links for Entire Projects" in *Data in Brief*, v. 25, p. 104005, 05 2019. [Online]: https://doi.org/10.7910/DVN/PDDZ4Q

[2] M. Denil, de Freitas, in "Narrowing the Gap: Random Forests In theory and In Pratice"

[3] "Random Forest and its implementation." [Online]: https://towardsdatascience.com/random-forest-and-its-implementation-71824ced454f

[4] A. Natekin, A. Knoll, "Gradient boosting machines, a tutorial" in *Frontiers in Neurorobotics*, v. 7, p. 21, 2013. [Online]: https://www.frontiersin.org/article/10.3389/fnbot.2013.00021

[5] SciKit-Learn, "Ensable methods in Sci-Kit." [Online]: https://scikit-learn.org/stable/modules/ensemble.html#gradient-boosting

[6] C. W. John McGonagle, George Shaikouski, *et al.*, "Backpropagation." [Online]: https://brilliant.org/wiki/backpropagation/

[7] S. Hochreiter, J. Schmidhuber, "Long Short-term Memory" in *Neural computation*, v. 9, pp. 1735–80, 12 1997. [Online]: https://dl.acm.org/doi/10.1162/neco.1997.9.8.1735

[8] C. Olah, in "Understanding LSTM Networks" 8 2015. [Online]: https://colah.github.io/posts/2015-08-Understanding-LSTMs/