Masaryk University
Faculty of Informatics

Research of Fedora Status for Machine Learning

Bachelor's Thesis

Dominik Tuchyňa

Brno, Spring 2019

Masaryk University
Faculty of Informatics

Research of Fedora Status for Machine Learning

Bachelor's Thesis

Dominik Tuchyňa

Brno, Spring 2019

This is where a copy of the official signed thesis assignment and a copy of the Statement of an Author is located in the printed version of the document.

Declaration

Hereby I declare that this paper is my original authorial work, which I have worked out on my own. All sources, references, and literature used or excerpted during elaboration of this work are properly cited and listed in complete reference to the due source.

Dominik Tuchyňa

Advisor: Adam Rambousek

Acknowledgements

I would like to thank to my leader Adam Rambousek and consultant Jan Horák from Red Hat for their tremendous support and help in writing of this thesis. I would like to thank to doc. RNDr. Lubomír Popelínský, Ph.D. for giving me a chance to explain to him and his students at his seminar the dependency issue I was trying to solve.

Abstract

The topic of this thesis was to find out whether the Fedora distribution attracts machine learning developers and also to create a guideline content for newcomers in this new, progressively developed field. In addition, part of the thesis research was also focused on solving the SRPM packaging dependency issue by using recommendation system based on machine learning algorithm. The model has two different variations, one is using the association rule learning algorithms Apriori and FP-Growth, and the other is based on neural network embedding layers approach, which is actually a Word2Vec model implementation from library Gensim. Targeted programming language was Python.

Keywords

python, machine learning, association rule learning, data mining, word2vec, neural network embeddings, embedding layers, gensim, rpm, srpm, jupyter, apriori, fp-growth, gpu acceleration, fedora, ubuntu, cuda, opencl

Contents

• CHAPTER 1
INTRODUCTION

In recent years the Machine Learning (ML) has caught a lot of attention and curiosity - whether it is news informing us about new artificial intelligence breakthroughs, students interested in computer science, programmers trying to adapt on the latest machine learning software or companies interested in profiting from using the machine learning within various tasks.

Analogically, company Red Hat has also interest in the area of latest artificial intelligence advancements like machine learning or more specifically deep learning. The question of the thesis was at first focused on the likeability of the Fedora distribution by developers when it comes to the machine learning development - what aspects do they take into account when choosing the right distribution and whether are these decisions correlated to the support of machine learning libraries, environment and deployment or not.

This question was then (after discussions with my leader and consultant) shifted to more or less task to help create guideline content for newcomers in this progressively developed field, because it allows a more computer science approach to be used within thesis than just comparing distributions side by side, which I think would consist of rather subjective opinions of questioned developers about their distribution preferences than some objective conclusions. Nevertheless, the task to identify and highlight the main pain points in Fedora distribution is still a part of the thesis research.

I decided that guideline content would be based on implementing the solution for yet an unsolved issue that Red Hat developers often meet when deploying an RPM package, therefore the guideline content creation has two main benefits - in the first place it would help new

and inexperienced developers to start with their first machine learning project and in the second place it should help RPM packaging programmers in resolving the unsuccessful builds.

At the beginning of the thesis, I provide the state of the art machine learning approaches and some of the algorithms used within them, followed by a quick glance on a few of the platform differences compared to another Linux distribution, Ubuntu.

During the implementation of the recommendation system I had come across several issues which are often part of any machine learning oriented development - all of the problems I have encountered are described later in the thesis and at the end, I present my solution and its implementation.

The finished guideline content is available in the GitHub repository written in Jupyter Notebook. The reasons I have chosen the Jupyter Notebook are described further on in thesis. The recommendation models can also be found in the same GitHub repository.

The basic tests and results of the implemented recommendation systems can be found in attached copy of the GitHub repository along with guideline content written in Jupyter Notebook.

Finally, in the end, I lay down some of the most important conclusions of the thesis where I also suggest several other improvements to the existing guideline content and implemented recommendation system.

Bibliography chapter can be found at the very end of the thesis.

• CHAPTER 2
BACKGROUND INFORMATION

Python, thanks to its recent boom [19] and understandable, easy-to-use and readable code can offer a rich variety of different APIs specifically for AI development - many machine learning libraries and algorithms are implemented in Python [10] and since a lot of them are open source, the Linux platform is preferred - algorithm libraries are often at the beginning of their deployment suited primary for the UNIX based platforms, by contrast to Microsoft Windows whose support can be delayed for a later stage. UNIX based platforms are getting a lot of popularity lately [21] , leaving Windows less attractive for developers each upcoming year.

As a result of the enormous diversity of the GNU/Linux distributions, developers have a large set of options from which they can choose. The major interest from the Red Hat perspective is to find out the pain points in Fedora distribution and prepare content on the Fedora Developer Portal to help new-comers to begin in the field of artificial intelligence.

To try to simulate the development of the machine learning project in the guideline content, I have chosen to work on an example task that could use the power of the artificial intelligence - analyze dependencies in Fedora packages and create a model that would suggest missing entries. This example task has not been implemented yet and its solution would help Red Hat programmers reduce their time spent on searching for the missing entries and also minimize resources used by unsuccessful SRPM builds.

The issue is that the deployment of SRPM package can often fail because of the missing dependencies which were forgotten to be included in the SRPM spec file (short for specification file, basically a ”recipe” for the SRPM package building). The main goal would be to solve this

problem by applying one of the machine learning methods - the association rule learning - on lists of correctly included dependencies from all of the available SRPM spec files in Fedora. By this approach we can build a recommendation system, that would either suggest programmer missing dependencies or would be part of an automation pipeline for unsuccessful builds resolving.

As one of the earliest definitions of machine learning by Samuel Arthur says:

> "Machine learning is the field of study that gives computers the ability to learn without being explicitly programmed." [20]

the usage of the machine learning algorithm is the key feature here - rather than implementing a specific static analysis tool for each programming language to find out the missing dependencies and fill them in the spec file - which would end up with a lot of code utilities that would need to be maintained and expanded - we only analyze the associations between dependencies, no matter what language did developers use in the source code of a package.

• CHAPTER 3
STATE OF THE ART IN MACHINE LEARNING

We can generally categorize different ML approaches by:

- Learning and training set - categories describe the training set that was used (supervised, semi-supervised, unsupervised, reinforcement learning)
- Adaption to new data - algorithm ability of how it learns the data, on-fly or else (online learning, batch learning)
- Generalization - how the test data is associated with the trained information (instance based vs model based)

These categories are not restricted and machine learning models are often a combination of them. In terms of state of the art, we can take a closer look on each class and mention some of the latest and most popular algorithms used within them.

3.1 Learning and training set

3.1.1 Supervised learning

Supervised method means that each one of the training examples is labeled with its solution. That is where the word supervision comes from, we implicitly supervise the algorithm learning with our right answers to the questions. The main tasks in supervised ML are classification and regression.

3.1.1.1 Classification

Classification is desired when the problem of labeling something has to be solved. The algorithm classifies the input data to any of the learned classes from the training. This classification is based on the similarities between the entity on input and the class it will be associated with. Classification is then also categorized by the amount of classes the algorithm classifies to (or simply said by the amount of classes the algorithm has learned) - unary, binary or a multi-class classification. Often a binary problem is the most common for beginners to solve (for example spam filtering - label a mail as a spam or no-spam). Some of the algorithms used:

- Neural Networks
- Hidden Markov Models (Speech recognition)
- Support Vector Machines

3.1.1.2 Regression

In regression we rather predict a target - for example, we try to find the best mathematical function that would describe the trends or patterns the closest to the actual representation of the data (for example 2D or 3D graph). Based on the learned function, the model predicts where the test data from input would appear in the graph. Some of the popular algorithms used:

- Decision Trees (Customer behaviour prediction)
- Random Forests (Stock prediction)
- Support Vector Machines (Image detection)

3.1.2 Unsupervised learning

The opposite to the supervised approach is unsupervised learning. In contrast to supervision, here we feed data as unlabeled and the algorithm is expected to learn by itself without any help by giving the right answer. These algorithms are in general supposed to preserve the structure of the data for later research. This way we can better understand

the input data and discover any possible hidden patterns. Therefore, we can say that we rely on some sort of complexity of the given data.

Unsupervised ML algorithms are generally categorized by an issue or a problem they are supposed to solve:

- Clustering problem (we try to group some entities, for example image search of a chosen object) - k-Means Clustering
- Association rule problem (we want to discover relations between attributes or entities) - Apriori, FP-Growth, Eclat
- Visualization or dimensionality reduction (we want to represent or simplify the data without loosing too much information) - Principal Component Analysis

3.1.3 Semi-supervised leaning

Sometimes the process of labeling the training data is rather impossible or unwanted due to its required time-consuming or expensive amount of work. We also might want to purposely avoid excessive labeling of the training data - this reduces the possibility of a human error. The solution for solving such a problem is semi-supervised training.

Semi-supervised learning is a combination of previous two categories, it learns with the help of usually a little bit of labeled data on a big amount of unlabeled data. "For example, deep belief networks (DBN) are based on supervised components called restricted Boltzmann machines (RBM) stacked on top of one another. RBMs are trained sequentially in an unsupervised manner and in the end the whole system is fine-tuned using supervision" [12, page 34].

3.1.4 Reinforcement Learning

This type of learning is probably the closest to animal or human behaviour in the nature. We have an agent that decides which action to perform in the environment. These actions have consequences and return to the agent either a reward or penalty, depending on how useful was the action for achieving the goal of the agent. Based on these penalties and rewards the agent builds a policy - a strategy that tells the agent how to behave and which actions to choose when given certain circumstances.

One of the most popular algorithms used within reinforcement learning is perhaps the Monte Carlo Tree Search algorithm, which was used in Google's *AlphaGo* application.

3.2 Adaption to new data

3.2.1 Batch learning, non-incremental adaption

There are systems that learn only with all of the available data at once. In order to adapt to a new information with preserving the old one that was learned before, the model has to be trained all over again from dataset (also called a batch) containing both the old and the new data. This process is often resource-demanding and time consuming, which can lead to offline learning - the system is trained offline first and after that it is launched to production. The launched model cannot train anymore and in order to adapt to a new information it has to be trained all over again from a batch at once. This process can be also described as a non-incremental adaption (in opposite to incremental adaption, which is described next).

3.2.2 Online learning, incremental adaption

In contrast to batch learning, the system using this approach can incrementally adapt to a new data. "By feeding the training data sequentially either individually or by small groups called mini-batches" [12, page 36]. This process is far less resource-costly and can be done on-fly (online). One of the benefits of using the online learning is that once system is trained on new data, it can discard them if they are no longer needed.

Online learning can be also used as an out-of-core learning - the situation when the dataset cannot be fed into the computer memory. In out-of-core approach only a part of the data is loaded (a small batch), which is then used in the training process. This is repeated until the whole dataset is processed.

3.2.3 Note

One thing to discuss is whether the offline and online term is used correctly - it is an often practice that both the batch (non-incremental) and the online (incremental) learning is done offline, so sometimes the used term misleads to wrong understanding which type was actually implemented.

3.3 Generalization

"Having a good performance measure on the training data is indeed good, but insufficient - the true goal is to perform well on new instances - on the test data." [12]

3.3.1 Model based ML

We can often meet with ML applications that use the model based learning. The model can be represented as a mathematical function that models any pattern or trend found in the training data. Based on this function we can predict where the test data would appear in the model and give back the required answer. For this function that models the training data we also need another function which would describe how well and precise the modeling function defines the data. We can either use an utility function (fitness function) that tells us how good the model is or a cost function that describes how bad the model is performing. In general, we would first study the data, select a promising model and then train the model during which the learning algorithm searched for the best parameters to either maximize the utility function or minimize the cost function. After the model is trained we can make predictions out of the input - this final step is called inference. Some of the models used are:

- Linear Regression model
- Polynomial Regression model

3.3.2 Instance based ML

Here we try to measure the similarities between trained examples and a test data. Based on the similarity measurement we generalize the

test data to required solution. This method is sometimes referred to as learning by heart. Some of the instance based algorithms used:

- k-Nearest Neighbours
- Kernel methods

• CHAPTER 4
MACHINE LEARNING LIBRARIES IN PYTHON

Here I tried to summarize some of the most popular ML libraries used in Python. I cannot objectively measure the popularity of a library, but to support the statement I am attaching GitHub statistics for each of the mentioned frameworks. The last update of the attached statistics was: May 22, 2019.

4.1 Tensorflow

Currently one of the most popular deep learning libraries in the world. It is named Tensorflow because it takes a multi-dimensional array (also known as tensors) as an input. Then, a sort of flowchart of operations (called a Graph) is applied on the input throughout the model. The input (tensor) goes in, and then it flows through this system of multiple operations and comes out the other end as output, hence the name Tensorflow.

This library also offers a GPU acceleration supported variant called `tensorflow-gpu`.

Commits: 56,355; Contributors: 1,999; Stars: 128,033; Forks: 74,887

4.2 Scikit-learn

Scikit-learn contains a range variety of common used machine learning algorithms. It contains tools for statistical modeling including classification, regression, clustering and dimensionality reduction. Scikit-learn

has also useful tools for preprocessing the data (for example a LabelEncoder which encodes the elements of a dataset to integers).

The library is built upon the SciPy (Scientific Python) library.

Commits: 24,019; Contributors: 1,309; Stars: 35,228; Forks: 17,197

4.3 Pandas

Pandas excels in data analysis and is one of the most used in data wrangling. Through pandas, you get acquainted with your data by cleaning, transforming, and analyzing it.

Pandas is built on top of the NumPy package, meaning a lot of the structure of NumPy is used or replicated in Pandas. Data in pandas is often used to feed statistical analysis in SciPy, plotting functions from Matplotlib and machine learning algorithms in Scikit-learn.

Commits: 19,385; Contributors: 1,488; Stars: 19,611; Forks: 7,754

4.4 Keras

Keras is a high-level neural networks API, written in Python and capable of running on top of Theano or Tensorflow. It was developed with a focus on enabling fast experimentation. Being able to go from idea to result with the least possible delay is the key to doing a good research.

Commits: 5,112; Contributors: 794; Stars: 41,343; Forks: 15,726

4.5 PyTorch

Sometimes a young and inexperienced developer who uses TensorFlow and Keras encounters a problem with debugging these deep learning libraries. Also, Keras and Tensorflow do not seem to play well with Python libraries such as NumPy, SciPy or Scikit-learn. PyTorch should have answers to these issues – its benefit is in being a deep learning library that is more at home in Python than the mentioned Tensorflow or Keras. It has features such as dynamic computational graph construction as opposed to the static computational graphs present in TensorFlow or Keras.

Commits: 18,086; Contributors: 1,042; Stars: 28,282; Forks: 6,762

4.6 Gensim

Gensim is a library designed to efficiently extract semantic topics from documents with human language. Gensim is designed to process unstructured and raw digital texts (often referred to as plain texts).

Algorithms in Gensim such as Word2Vec, Semantic Analysis, Latent Dirichlet Allocation or FastText discover the semantic structure of documents by examining statistical co-occurrence patterns within a corpus of training documents. No supervision is needed - these algorithms are unsupervised and are based on a big amount of complex text data.

Commits: 3,810; Contributors: 313; Stars: 9,220; Forks: 3,380

4.7 Theano

Theano is a Python library where the programmer can define, optimize and evaluate mathematical expressions, especially those that use multi-dimensional arrays. By using Theano it is possible to attain simiar computational speed compared to C implementations for problems involving large amounts of data.

Theano combines aspects of a computer algebra system (CAS) with aspects of an optimizing compiler. It can also generate customized C code for mathematical operations. Commits: 28,080; Contributors: 332; Stars: 8,795; Forks: 2,483

4.8 Jupyter

A Jupyter Notebook is a browser application where you can write, run and publish your code. It is a free software primary used by scholars, scientists or corporations to share and present their research.

The advantage of Jupyter Notebook is its form of programming called literate programming. This approach emphasizes a structure with readable text inserted between the individual blocks of code, which servers as a sort of documentation and description of the source code. It excels at demonstration and teaching objectives especially for science.

Commits: 11,433; Contributors: 384; Stars: 5,821; Forks: 2,477

4.9 Pypi and Anaconda Cloud repository

There are several other Python libraries targeted exclusively either for machine or deep learning, but an honorable mention goes to PyPi (an official third-party software repository for Python) and for package manager called Conda (a part of a free and open-source distribution of the Python named Anaconda that aims to simplify package management).

Both PyPi and Anaconda Cloud repository contain all kinds of libraries for Python. Libraries listed in PyPi can be often installed on the system by using the `pip` tool, but the PyPi also includes libraries that can be installed only from downloaded sources (from GitHub for example). On the other hand, all of the libraries from Anaconda Cloud can be installed by using the `conda` tool.

Besides popular APIs like those described in this chapter, you can also find all sorts of custom implementations for a big variety of machine learning algorithms that are not present in any of the mentioned libraries.

• CHAPTER 5
FEDORA COMPARED TO THE UBUNTU DISTRIBUTION

Ubuntu is nowadays a widely used and popular distribution. Some can suggest that Ubuntu is indeed more popular than Fedora, more precisely in the community of a new Linux users, but this does not say anything about the preference amongst developers in machine learning field.

Since there are no specific statistics on the topic of distribution selection when it comes to machine learning field, the only reasonable way to compare Fedora and Ubuntu seems to compare what each platform can offer for machine learning development.

5.1 GPU acceleration support

One of the main advantages of a distribution would be the support of GPU acceleration for complex machine learning tasks, because except for a good performing model we should also consider the amount of time at which the model is training or performing.

GPU acceleration generally offers faster execution time when it comes to the amount of calculations to be computed. Since a machine-learning task is often based on preprocessing large datasets and then performing various mathematical computations on selected entries of the dataset, a parallel processing approach is preferred for handling multiple tasks at once.

According to the Jon Peddie Research [18] the current leading GPU seller is Intel, after that comes the Nvidia and within the third place with a small difference to Nvidia, the AMD. Therefore, when considering GPU acceleration as one of the major factors, the support of all these three platforms must be inspected (although the GPU accelera-

tion is most common for dedicated graphic cards only, which involves mostly Nvidia and AMD products, I also explored what Intel offers for its integrated graphics cards for the purpose of complete inspection).

Nvidia has its own technology for GPU acceleration called CUDA® available for all Nvidia GPUs from the G8x series onwards, including GeForce, Quadro and the Tesla line. This technology is closed-source and is not available for any other GPU brand.

Open-source alternative to CUDA® is OpenCL™, which can be used on any platform including Intel, AMD and also Nvidia graphic cards.

5.1.1 CUDA® support

At the time of writing this thesis, the latest release is CUDA® Toolkit v10.1.105 [17]. By the date release notes were written, it supported both newest version of Fedora and Ubuntu (Ubuntu 18.04.1 LTS released in July 26, 2018 [7]; Fedora 29 released in October 30, 2018 [3]).

Release notes also state that Ubuntu had known issue with installation of CUDA® 10.1 and NVIDIA 418 drivers resulting in error pointing at unmet (uninstalled) dependency `xserver-xorg-core` [17, section 3.6.1]. There are not described any other known issues with Fedora distribution.

However, when I have looked at the install instructions (which is a completely separate page apart from the documentation and release notes) that specify install instructions for Linux platform [16], I have found out that Fedora seems to have known issues within installation. It states that the CUDA® driver may not work if a custom-built `xorg.conf` file (at `/etc/X11/xorg.conf`) is present. It also states that a user must add the RPMFusion free repository to the package manager repository database before installing the Nvidia driver RPM packages, or missing dependencies will prevent the installation from proceeding. The guide then also states that installation might fail if a RPMFusion non-free repository is enabled - user has to temporarily shut it down. Furthermore, there are other issues that can occur: when a system has installed both packages with the same instance of `dnf` some driver components may be missing; the i686 libvdpau package dependency may fail to install; when a non-default partition scheme is used it may be necessary to rebuild the grub configuration files [16, section 3.3]. However, no installation issues are described in instructions for Ubuntu.

5.1.2 OpenCL™ support

OpenCL is an open-source framework and there are variety of implementations developer can choose from, depending on which brand of GPU he or she wants to use in accelerating calculations.

5.1.2.1 Intel OpenCL™

As the largest graphics card manufacturer [18], platform support of OpenCL™ for the Intel would be a major advantage for a distribution. The truth is, however, that Intel GPU acceleration would be in my opinion the least demanded, as Intel does not currently make dedicated graphics cards which are used for GPU acceleration for their better performance as opposed to the integrated ones. However it should be also mentioned that Intel is planning to enter the discrete GPU market in 2020 [8] [5]. Unfortunately, in the developer guide for Intel® SDK for OpenCL™ applications [13], there are no specifications whether the Ubuntu or Fedora platform is supported by default.

5.1.2.2 Nvidia OpenCL™

If Nvidia GPU supports CUDA®, it should also have a support for OpenCL™. Nevertheless, Nvidia states that "OpenCL support is included in the latest NVIDIA GPU drivers" [15]. However, this does not give us any description which GPUs are supported and which are not.

5.1.2.3 AMD ROCm

Although labeled only as ROCm, this implementation was initiated and is being supported by AMD [9] and therefore is naturally targeted for AMD GPUs. Currenlty the ROCm 2.4.x platform supports the Ubuntu 16.04.3 up to the 18.04.2 version [6, section Supported Operating Systems], whereas the Fedora is not supported by default, stating that "Fedora may work but may not be compatible with the `rock-dkms` kernel driver" [6, section CentOS/RHEL 7 (7.4, 7.5, 7.6) Support].

5.2 Distribution performance

Distribution performance in machine learning oriented tasks would be also a major factor when selecting the right platform, specifically a run-time complexity telling how the performance drops with increasing data or model size.

Unfortunately, I did not find any research on this topic and I currently do not possess required equipment and hardware to do a similar research, so this still remains a question yet to be answered.

5.3 Developer user experience

Operating system user experience as a developer is a very subjective matter. Both distributions have currently GNOME 3 as a default desktop environment, although each one has a slightly different appearance.

The only comparable trait I consider important is how these distribution handle several functionalities in means of working with console and packages, and one of these issues were pointed earlier in subsection 5.1.1 - Fedora does not by default support libraries that contain closed-source licences.

5.4 Support and stability

In terms of choosing the right distribution for reliable stability, it seems like the Ubuntu is a better option, whereas Fedora focuses rather on the newer software with constant updates. As stated by the Cannonical [1], LTS releases are published every two years in April. An estimated 95 percent of all Ubuntu installations are LTS releases, with more than 60 percent of large-scale production clouds running on the most popular OS images - Ubuntu 18.04, 16.04 and 14.04 LTS [2]. It was 3 years before, now it is enhanced and it is 5 years.

On the other hand, Fedora does not have an LTS and its advantage is probably more in innovation than support. As stated on the official Fedora Project Wiki website, Fedora has new releases every 6 months [4], which induces that it should support latest technologies far more than Ubuntu.

• CHAPTER 6
GUIDELINE CONTENT

Guideline content consists of two parts, the environment part (python setup and installation), and the development part (a quick look on the issue and its solution). The former is focused primarily on the newest python3 and its compilers, the latter offers a preview on basic workflow of a machine learning project development.

The guideline is written in Jupyter Notebook which I have described earlier in chapter 4.

The guideline content can be found in attached GitHub repository copy to the online version of this thesis.

• CHAPTER 7
SRPM DEPENDENCIES INCLUSION

To further understand the conclusions that were made in selecting the right technique and algorithm for the project, the issue that is being solved must be presented and explained.

The RPM Package Manager (RPM) is a package management system, that makes it easier for developers to distribute, manage, and update software created for Red Hat Enterprise Linux, CentOS, and Fedora. An RPM package is simply an archive containing other files and information about them needed by the system. There are two types of RPM packages:

- binary RPM
- source RPM (SRPM)

Binary RPM contains the binaries built from the sources that can be optionally changed by patches. SRPM on the other hand contains source code, a spec file and optionally patches. This spec file is basically a recipe that determines how the source code will be build into a binary RPM. So, in order to have a package built and deployed, all the information about version, patches and dependencies required for built must be specified in the SRPM.

In order to properly build a SRPM package, the spec file must describe which dependencies to include that our project requires. After discussion with my leader and consultant, this was identified as a common reason behind the build failure, that is - developers often forget to include dependencies required by the project, thus the build fails during the process of building the SRPM spec file. Now, the project can be coded in any programming language. The make of a static analysis tool that would scan the code and then output missing dependencies

not included in the spec file seems like the best solution, but it has one main downside - implementing tool for every possible programming language present in SRPM package would be too complex and time consuming for development. This static analysis tool would be also hard to maintain, programming languages are changing from time to time (sometimes without backwards compatibility, see Python) and for any new language that would occur in the SRPM source codes the tool would had to be expanded.

However, we can abstract this problem and go one layer above the static analysis tool, metaphorically speaking. Rather than analyzing the source code, we can analyze the relations and associations between dependencies themselves. It is probable that certain dependencies are often used within each other and that these similar dependencies can form various groups based on a common occurrence in the spec files. This is when the use of machine learning has two main advantages - complex analysis of these relations between dependencies without human supervision and adoption to the new data (new programming languages).

• CHAPTER 8
RECOMMENDATION SYSTEM MODEL

So, as written earlier in previous chapter, instead of identifying problem as a source code inspection, we can abstract it as a concept of analyzing the relations and associations between dependencies alone.

Suppose we have supermarket customers who buy products and we want to somehow find out which items are often being bought together. So, we take a list of transactions as an input, where every transaction is the list of items purchased by customer (or a content of a customer basket that was bought). As an output we want to produce rules or associations between items that are frequently bought together. This way we can recommend, that for example if a basket contains item A and item B, it has an eighty percent chance of containing item C, and therefore we implicitly (by placing item A, B and C together next to each other) recommend customer to buy item C.

This idea can be also used to help solve the dependency inclusion issue. To acquire data we iterate over the whole Fedora package repository (where all of the correctly archived SRPM packages are located) and for every package the list of required dependencies is extracted from spec file. By this way we obtain the list of correctly included dependencies for every package in Fedora repository, which will serve as a train dataset for our machine learning model.

• CHAPTER 9
SOLUTION USING DATA MINING APPROACH

Association rule learning algorithms are often used in data mining to provide an insight to possibly hidden relationships between entities that humans alone do not necessarily see at first sight.

These algorithms produce association rules in shape of $\{X\} \Rightarrow \{Y\}$, where $\{X\}$ is the list of entities upon which the algorithm makes predictions, and $\{Y\}$ is the actual prediction (the use of the word prediction is here just for simplicity, actually there are no predictions, just mined frequently occurring patterns present in dataset). Size of the set $\{X\}$ is at least one, same applies for set $\{Y\}$. Each rule has following attributes: confidence, support and optionally lift. Support is an indication of how frequently the items appear in the data, confidence indicates the number of times the if-then statements are found true, and lift can be used to compare confidence with expected confidence.

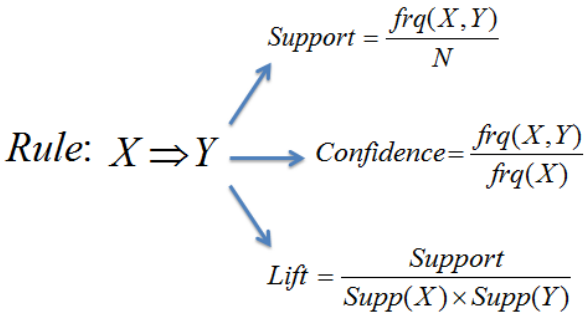

Figure 9.1: Association rules explained in graphic, picture from the website of Dr. Saed Sayad [11]

9.1 Issues and limitations

This would come as an excellent solution for the problem, because it does precisely what we want - feed the incomplete set of entities in, and produce the complete set of entities out. Unfortunately, severe issues happen to be present in this solution.

The first is the actual implementation in Python. One of the most popular association rule algorithms are Apriori, Frequent Pattern (FP) growth and Eclat [12]. Despite of association rule algorithms being categorized as machine learning, there are no available implementations in libraries listed in chapter chapter 4 nor in any other library with similar popularity. The only implementations I had found were custom libraries from PyPi repository, but all of them either lacked a better API, design or a better performance.

The second one is the overall computational complexity. The major problem when having a large list of rules is iterating through them. Of course, this can be solved by using a hashing data structure, but unfortunately only one of the algorithms from PyPi uses it and even that implementation is partially ineffective (the preview on this subject can be found in the code attached to the online version of this thesis).

The third issue paradoxically comes from the previously mentioned simplicity of the solution. It is space and computational complexity. For example, the recommendation system takes an incomplete set of entries `{foo, bar, qux, corge}` as an input and the task is to suggest missing entries. It should search for the rules where `{foo, bar, qux, corge}` is the left side of the rule. The problem is, when the left side `{foo, bar, qux, corge}` does not exist in any mined rule. This can happen, when at least one item from `{foo, bar, qux, corge}` is either an unknown dependency (which is used for its first time in SRPM packaging) or has no association between the others. For example, there can be rules with the left side `{foo, bar, qux}`, `{foo, bar, corge}` or `{bar, qux, corge}`, but none of them is a complete set of `{foo, bar, qux, corge}`. We can iterate through these rules and implement comparing the set `{foo, bar, qux, corge}` with the rules, but doing this for every single rule in a large dataset would only result in a higher computational complexity.

9.2 Correction

All of the mentioned issues can be corrected by choosing the right data structure and applying limitations to the model.

9.2.1 Design

We can limit the size of the rules so that one rule has only one entry on the left side and one entry on the right side. By doing so we avoid looking for the entire set of incomplete entries in the rules list and the only thing that is searched are the individual, specific entries.

We can notice that this changed the strategy of how the problem should be solved. Instead of making suggestion of a module based on all the modules as a whole in the incomplete set, the system now makes suggestions individually on every entry present in the incomplete list and then decides which potential suggestions will be recommended. This decision can be done in various ways, depending on how the union and intersection of the found right sides would be implemented.

9.2.2 Space and time complexity

By choosing the hash table as a data structure to store the rules, the necessity of iteration over the mined associations is not present anymore.

Furthermore, the data structure can be optimized more if a one to one rule limitation from previous correction is used. Every rule now is in form $\{X\} \Rightarrow \{Y\}$, where multiple right sides occur for one left side. For example $\{A\} \Rightarrow \{B|C|D|E\}$. The hashing table can be modified so that the key is the name of the dependency on the left side of the rule ($\{A\}$), and value for the key is a list of all modules present on the right side of the rule ($\{B, C, D, E\}$).

• CHAPTER 10
NEURAL NETWORK EMBEDDING LAYERS APPROACH

Issues described in chapter 9 indicate, that rather a different solution would be more efficient and less time-consuming for the development (assuming that a programmer would either had to implement his own algorithm from scratch or edit the available Python implementations).

Instead of making associations for every desired item in the dataset (and repeat that process for other desired items as well, making rules redundant - if we have a frequently occurring itemset $\{X, Y\}$ there had to be both the rules $\{X\} \Rightarrow \{Y\}$ and $\{Y\} \Rightarrow \{X\}$), we may want a model where rules do not duplicate themselves and every rule exists there only once. Moreover, a complete removal of the rules from environment is possible, where only unique entries are in some considerable way stored in (one or multi-dimensional) space. Neural network embeddings are more than adequate for such corrections.

10.1 Concept

One notably successful use of deep learning is embedding, a method used to represent discrete variables as continuous vectors. This technique has found practical applications for word embeddings in translation and entity embeddings for categorical variables.

An embedding is a mapping of a categorical variable to a vector of continuous numbers. In the context of neural networks, embeddings are low-dimensional, learned continuous vector representations of variables. Neural network embeddings are useful because they can meaningfully represent entities in the transformed space.

The task for the neural network embedding implementation would then be finding the nearest neighbors in embedding space. These nearest neighbours can be used to make recommendations based on a cluster categories.

So, each one of the unique modules used as required dependencies in SRPM packages (merely 18,000) is represented as a vector of fixed size with several numbers

A common way to represent categorical values in neural networks is a method known as one-hot vector encoding, but using embedding layers is more beneficial and practical. If a one-hot vector approach is used, we would have one row and one column for each of the unique dependencies. For example, when having itemsets `{foo, bar}`, `{corge, baz}` and `{qux, quux, foo}`, the one-hot vector encoding would look like this:

	foo	bar	baz	qux	quux	corge
foo	1	1	0	1	1	0
bar	1	1	0	0	0	0
baz	0	0	1	0	0	1
qux	1	0	0	1	1	0
quux	1	0	0	1	1	0
corge	0	0	1	0	0	1

where zero means that entities do not have any association between them and one that they have. This is inconvenient as we store a lot of unnecessary information (zeros) and if not secured, the trained model may come up with a lot of unrelated conclusions (for example that the first column `foo` has a pattern 1 1 0 that repeats itself throughout the table).

10.2 Word2Vec model

Upon from its release [14], Word2Vec became quite a popular model in word embedding - a way of using earlier mentioned vectors as representations for words. I have decided to use Word2Vec (at least try and see the results) in solving the dependency issue, because its implemen-

tation was available in the Gensim library (which I have described in chapter 4), probably due to its current popularity.

Gensim also offers an implementation of the Word2Vec model with both variants - Skip-Gram and Continuous Bag Of Words. These apporaches represent how the model is trained and how the targeting of words work.

In CBOW method, the trained model predicts the target word based on a context. So for example if we have an incomplete sentence In Skip-Gram approach, the model is trying to predict the context based on word.

• CHAPTER 11
CONCLUSIONS

11.1 General conclusions

From the analysis of a distribution preference, I have found that they can be only objectively compared in a functionality they offer and support for a developer. One of the main supported functionality was the support of GPU acceleration, which ultimately showed that Ubuntu is more supported than Fedora - both in the CUDA® and OpenCL™ technology. Also another downside of Fedora was by default unsupported libraries that contained patent-encumbered open-source. If a user wants to use this libraries, he or she has to explicitly enable the RPM fusion repository.

11.2 Summary of contribution

I tried to answer the yet unsolved issue Red Hat developers encounter when deploying SRPM packages. I suggested using recommendation system and approached the problem by using an association rule learning and the Word2Vec model. These models were not as successful as I thought (the results can be found in attached copy of the GitHub repository) and a rather different solution would be more convenient.

11.3 Future research

Future research on the Ubuntu versus Fedora topic should primary based on how these distributions perform when benchmarked specifically in a machine learning oriented tasks. Also the analysis of opinions about the distribution machine learning developers choose and use

should be done, although that is rather a survey targeted work than a computer science research.

For a better recommendation system I suggest implementing a custom algorithm from scratch using neural network embeddings. This implementation would be specifically aimed just for the dependencies required in the spec files. More focus should be also aimed at the testing of a model - using the right and bad associations, so that it would be statistically clear of how the model is actually performing.

Part of the testing should be done in real use case scenario by the help of Red Hat developers, who would use the recommendation system when deploying a new packages, because this is the situation we cannot reproduce on an existing packages with which the model was trained. Developers or an automated system would rate the efficiency and success at which the model is performing when recommending a dependency for a new package.

The guideline content should be expanded for bigger variety of simple yet nontrivial example tasks, where the suggested better recommendation system would be also described and written in the Jupyter Notebook.

BIBLIOGRAPHY

[1] https://www.ubuntu.com/about/release-cycle.

[2] About lts. https://wiki.ubuntu.com/LTS.

[3] Fedora 29 release dates.
https://fedoraproject.org/wiki/Releases/29/Schedule.

[4] Fedora release life cycle.
https://fedoraproject.org/wiki/Fedora_Release_Life_Cycle.

[5] Intel officially confirms: Entering dedicated gpu market in 2020.
https:
//wccftech.com/intel-official-discrete-gpu-market-2020.

[6] Official github repository for rocm.
https://github.com/RadeonOpenCompute/ROCm/.

[7] Ubuntu release dates. https://wiki.ubuntu.com/Releases.

[8] https://newsroom.intel.com/news-releases/
raja-koduri-joins-intel/#gs.dj5cex, 2017.

[9] AMD. Rocm, disclaimer attribution.
https://rocm.github.io/legal.html/.

[10] Bobriakov, I. Top 15 python libraries for data science in 2017.
http://medium.com/activewizards-machine-learning-company/
top-15-python-libraries-for-data-science-in-in-2017-ab61b4f9b4a7,
2017.

[11] Dr. Sayad, S. Association rules.
https://www.saedsayad.com/association_rules.htm.

[12] Géron, A. Hands-on machine learning with Scikit-Learn and TensorFlow : concepts, tools, and techniques to build intelligent systems. O'Reilly Media, Sebastopol, CA, 2017.

[13] Intel. Developer guide for intel® sdk for opencl™ applications. https://software.intel.com/en-us/openclsdk-devguide.

[14] Mikolov, T., Chen, K., Corrado, G. S., and Dean, J. Efficient estimation of word representations in vector space, 2013.

[15] Nvidia. Nvidia opencl™ information. https://developer.nvidia.com/opencl.

[16] Nvidia. Nvidia cuda installation guide for linux. https://docs.nvidia.com/cuda/cuda-installation-guide-linux/index.html, 2019.

[17] Nvidia. Nvidia cuda toolkit release notes. https://docs.nvidia.com/cuda/cuda-toolkit-release-notes/index.html, 2019.

[18] Research, J. P. https://www.anandtech.com/show/12107/q3-2017-graphics-cards-shipments-hit-5-year-high?fbclid=IwAR3FR9NpVESZVCqmdsKWvvDpDyT5Es6MyqNGYz2md8zY37EH5LSpuxyJ5vA.

[19] Robinson, D. The incredible growth of python. https://stackoverflow.blog/2017/09/06/incredible-growth-python, 2017.

[20] Samuel, A. Some studies in machine learning using the game of checkers. IBM, 1959.

[21] StackOverflow. Developer survey results. https://insights.stackoverflow.com/survey/2016#technology-desktop-operating-system, 2016.