Evolving Text Classifier Using Genetic Programming

Ema Eminagic and Hoang Ngo

2020

Master's Thesis in
Biomedical Engineering
Faculty of Engineering LTH
Department of Biomedical Engineering
Supervisor: Anders Brodin
In Colaboration with Oxide AB

Evolving Text Classifier Using Genetic Programming

Published by
Department of Biomedical Engineering
Faculty of Engineering LTH, Lund University
P.O. Box 118, SE-221 00 Lund, Sweden

Subject: Biomedical Engineering (BMEM01)
Supervisor: Anders Brodin
Co-supervisor: Axel Stålhand at Oxide AB
Examiner: Ingrid Svensson

Abstract

Text classification is one of the main tasks within the field of natural language processing, which has been growing significantly during the last decade with applications in different industries. Despite different approaches to text classification showing good results, such as Machine Learning and Deep Learning, their shortcomings give substance to the need for further research on other approaches. In this thesis, we propose a genetic programming algorithm - a technique inspired by biological evolution, which is capable of producing text classification models by means of string matchers and character n-grams. This approach does not require domain-specific knowledge and manual feature engineering and can provide interpretability in the model. The performance of the classification models produced by the proposed algorithm gives promising results, especially on topic detection.

Keywords: genetic programming, evolutionary algorithm, evolutionary computation, text classification, n-grams, natural language processing.

Acknowledgements

First of all we would like to thank Oxide for giving us the opportunity to write this thesis. It has been a great experience to explore this exciting area within evolutionary computation. We are grateful for the support, guidance and help with interesting ideas and concepts from everyone at Oxide; Arthur, Axel, Jimmy and Lars. We also want to express our appreciation for the interest which they have shown towards us by including us in their Evodays and team building activities where Overcooked was a particularly excellent team building exercise. Special thanks to Axel Stålhand, our supervisor at Oxide, who guided us on a regular basis throughout the thesis and provided us with useful material. Also thanks to Lars Hård, who introduced us to genetic programming with such contagious and inspiring enthusiasm. Furthermore we would like to thank Anders Brodin, our supervisor at the Department of Biology, who has been there to answer our questions, discuss our thoughts and provide us with helpful feedback.

Lund, August 2020

Ema Eminagic and Hoang Ngo

Contents

List of Acronyms

DL	Deep Learning
EA	Evolutionary Algorithm
GA	Genetic Algorithm
GP	Genetic Programming
ML	Machine Learning

Chapter 1

Introduction

This chapter covers the background, the aim and the delimitations of this master thesis and also a brief introduction to the company Oxide where this master thesis was carried out.

1.1 Background

Text classification is the task of assigning the proper tags, labels or categories to unstructured textual data within the field of natural language processing. Ever since the digitization of text, text classification has become an increasingly important field providing governments, companies and agencies at different levels of society with valuable information. Human language text is constantly produced in different forms and numbers, from the billions of social media posts created every second - to published research papers and news articles - carrying various kinds of highly useful and important information. Some common applications of text classification are e-mail spam detection [1], business intelligence [2], hate speech detection [3] and detection of panic conversations on social media which helps speeding up authorities' emergency response systems [4]. Even though these examples of text classification applications might seem advanced the current techniques are far from perfected. A good example is the conversations with chatbots, whose purpose is to emulate human speech. Since the chatbots still struggle to understand human language they often fail to respond in a human-like fashion [5], [6].

The techniques of today for text classification commonly involve classical machine learning (ML) or deep learning (DL) [7]. Most ML approaches require a fair amount of manually performed feature engineering [8], which is the process of extracting features from input data [9]. The features are required for the ML models to be able to operate on the text. The work of producing good features is usually tedious and requires good domain knowledge [8]. DL has the advantage of operating without handcrafted features and may still produce rather accurate classifications. However, a downside of deep learning is the low interpretability caused by the complexity of the models, which is commonly referred to as the black box problem [10]. The lack of interpretability is considered a great disadvantage and even a risk to society when used for high stakes decision making for instance in healthcare and criminal justice, which has become increasingly common [11]. Another downside is the large size of the DL models, which require large amounts of processing power and model compression [12], [13], [14]. Further disadvantages are the expenses that come with training a deep learning model. Since the models demand large amounts of training data, and high-quality training data is expensive to obtain, preparing the model for work becomes costly [15]. This may be a problem, especially for startup companies. To deal with this problem the models which are used are frequently pre-trained. This means that it is trained, frequently by other parties, on data which might not be optimal for the particular task. Some suggest that this lack in high-quality training data has seriously constrained major breakthroughs in the field of AI [16]. This gives good reason to explore other approaches to text classification. One such approach is genetic programming; a metaheuristic search algorithm with the potential of finding classification models that require no manual feature engineering yet include some level of interpretability.

1.2 Oxide

Oxide is an R&D company focused on developing AI technology with strong inspiration from evolutionary principles and mechanisms found in biology. Currently the company is mainly focused on computational finance in the biotech sector, and the next step is to launch the Polychaos AI engine - a new kind of search engine with abilities to automate research. The search engine will be launched within the health and life science sector.

The Polychaos AI engine is built up of several layers of different evolutionary processes working together as a large intelligent self-improving system. Each layer in this system is assigned a certain task. One of these tasks is to extract information from textual data by performing binary text classification. The classifications are performed according to certain categories which are provided by another layer of the system in response to an incoming query. It is this task of text classification that has set the foundation for this thesis.

1.3 Aim

The aim of this thesis is to develop a genetic programming algorithm as an approach to text classification which fulfills the following criteria:

- requires no domain-specific knowledge
- requires no manual feature engineering
- provides interpretability

1.4 Delimitations

The scope of this work is limited to binary text classification. The development and evaluation of the performance of the genetic programming algorithm is based on datasets of medical papers and social media posts. All text data is limited to the English language.

Chapter 2

Theoretical Background

This chapter covers the theory that is necessary to understand the proposed genetic programming algorithm in this thesis and also gives an overview of different existing text classification algorithms.

2.1 Genetic programming

2.1.1 Evolutionary algorithms - brief history and applications

Genetic programming is a subfield within evolutionary algorithms (EA), in particular genetic algorithms (GA), which are a group of metaheuristic search algorithms based on the principles of evolution. EAs are in turn a subset of Evolutionary computation, which is a field within AI. The history of evolutionary algorithms and evolutionary computation in general dates back to the 1950s. This is when the first digital computer models imitating natural evolution appeared in public documentations. Some of the proposed areas of application at that time were automatic programming and numerical optimization problems. By the mid 60s the basics of EA and its main variants were established. Two of the most basic forms of EA are evolution strategies (ES) and genetic algorithms (GA). These branches were developed in parallel, ES by Ingo Rechenberg and Hans-Paul Schwefel and their students in Germany and GA by John Henry Holland and his students in the US. GAs became popular in the 70s as a result of Holland's work "Adaptation in Natural and Artificial Systems" published in 1975. From the work with GA another branch gradually evolved, which was named genetic programming (GP). This category was thoroughly established by the publications of John Koza in the 1990s, a PhD student of Holland's, who often is referred to as the father of Genetic Programming. Today many algorithms freely combine elements from all the different categories of EA and the lines between the categories, in particular between ES and GA, have become more blurred. [17], [18]

Evolutionary algorithms can be applied to various kinds of problems. Today they are popular tools for search, optimization, machine learning and for solving design problems. [19] One of the more extraordinary and recent applications is the design of so-called xenobots. Xenobots are small synthetic organisms constructed from stem cells with the ability to move over a surface in non-random directions [20]. In the future these "robots" might be used for targeted drug delivery or tissue repair

[20] combating difficult diseases, especially cancer. They might also help deal with environmental problems such as pollution by cleaning the oceans from microplastics or by replacing technologies made from ecologically harmful materials [21].

Another interesting application, with great potential, is the design of quantum circuits using evolutionary algorithms [22], [23]. Quantum circuit models are the most used models for quantum computing, which began developing in the early 1980s [24]. Not until recently have such models been possible to physically realise. Still, quantum computers are not quite ready for robust execution of quantum models [25]. However, they will have a profound impact on technological capacity when they are fully developed.

2.1.2 Basic principles of evolution

Evolution is the process of change in the biological traits of a population over successive generations, which is key to biological diversity and the origin of new species. The process is thoroughly described in Charles Darwin's On the Origin of Species by Means of Natural Selection [26], where the basic principle is that all species arise and evolve due to natural selection. The meaning of natural selection is simply that small differences between individuals affect their ability to compete with other individuals for survival and reproduction, also known as "survival of the fittest". The conditions for evolution can be summarized into four criteria described by Koza:

- "An entity must have the ability to reproduce itself.
- There must be a population of such self-reproducing entities.
- There must be some variety among the population.
- There must be some difference in ability to survive in the environment associated with the variety." [27]

When the criteria are fulfilled evolution occurs through a couple of processes that increase or decrease the diversity in a gene pool. Mechanisms that increase the diversity are mutation, recombination and interspecific genetic exchange, while mechanisms such as natural selection and genetic drift decrease it. These mechanisms operate genetically on the DNA, more specifically on the combinations of 4 nucleic acids. This means that it only takes 4 different components to represent and construct something as immensely complex as the human brain. Even more incredible is the evolution of something relatively simple, as a single-celled organism, into something much more complex as a human being. How is it possible? From a mathematical perspective the answer can be seen as parallel computation. As each individual organism carries a unique genome, nature can be considered to "compute" billions of variants of genomes simultaneously. This speeds up the process of finding a combination of base-pairs that results in meaningful traits. To find meaningful combinations is an extremely complex task, considering the combinatorics of a DNA string. Human DNA consists approximately in 3 billion base-pairs [28], meaning there are 4^3 billions of possible combinations. This shows the power of evolution and its mechanisms, as well as why it is of interest to mimic in the context of computational problem solving and AI.

2.1.3 Genetic algorithm

Mimicking the evolutionary mechanisms in nature, genetic algorithms differ from most other search techniques by simultaneously involving a parallel search with hundreds or thousands of points in the search space.

In nature the evolutionary mechanisms operate on the DNA strings contained in the chromosomes of the individuals within a population. In analogy to nature the solutions generated by the GA are referred to as individuals, which are represented by a "chromosome" - a fixed-length character string. In a GA the evolutionary mechanisms of nature are translated into code and operate on these character strings. The mechanisms can be applied in various different orders but the basic genetic algorithm can be summed up into the following 6 steps [29]:

1. **Start**: Initialize the first generation of individuals by randomly generating n numbers of chromosomes.

2. **Fitness evaluation**: Evaluate the fitness of each individual/chromosome and rank the individuals according to fitness.

3. **New population**: Create a new population by iterating through the following steps a-c until the offspring match the current population in number.

   (a) **Selection**: Select parents, that is the individuals used for reproduction, according to their fitness rank. The better the fitness the higher the chances of becoming a parent.

   (b) **Crossover**: Mate the parents by crossing over the parents' chromosomes. The parents mate according to a predefined probability rate. This means that all parents will not mate. If no crossover occurs the offspring is identical to the parents.

   (c) **Mutation**: Mutate the offspring with a probability rate at each character in the chromosome.

4. **Replace**: Replace the current population with the offspring.

5. **Stop Criterion**: If the stop criterion is not met, then use the newly produced population to repeat the previous steps starting at step 2.

6. **Stop**: When the stop criterion is met the individual with the highest fitness represents the best solution.

2.1.3.1 The traveling salesman

A classic and challenging problem is the traveling salesman problem. The problem consists of a salesman and a set of cities which the salesman has to visit. All cities must be visited and the salesman must return to the original city. The cities can only be visited once. The problem to be solved is to find the shortest route. This is very easy to describe, yet very difficult to solve. The traveling salesman problem is known to be NP-hard, which means the problem cannot be solved in polynomial time. However, it can be solved by a GA in a reasonable time. Below are different aspects and mechanisms described that are required in order to go through with the basic GA steps listed above. [30]

Representation
To solve the problem with a GA finding a good representation is the first step. Consider a set of cities denoted A-H (figure 1). The route can now be described in terms of a sequence of letters A-H. The sequence order of the letter corresponds to the order in which to visit the city. The letter sequence is referred to as the chromosome and each letter as a gene, meaning each gene represents a city (A-H). The chromosome consists of the same number of genes as there are cities and each gene appears only once within the same chromosome.

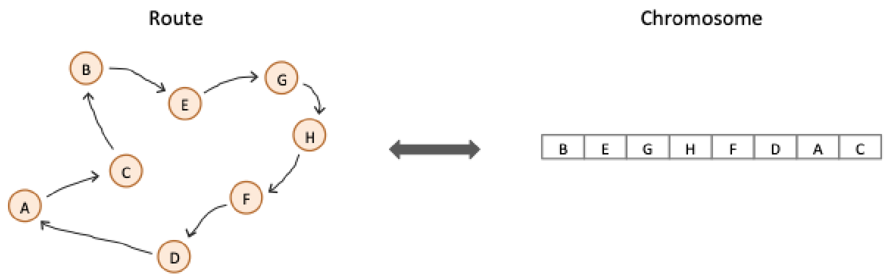


Figure 1: The representation of a route in the form of a chromosome. The route is a solution to the "traveling salesman" problem, where each letter represents a city and the sequence of letters (a chromosome) represents the city route.

Fitness
A fitness function, f(x), is designed to address the problem. In this case the fitness function calculates the length of the route, which in the simplest form is conducted either from a table of distances between each city or by calculating the euclidean distance between the city coordinates.

Selection
The selection of parents can be performed using one of the numerous already existing selection methods and strategies. Some of the classic methods are tournament selection, rank selection and fitness proportionate selection [27]. Each of these methods can be modified and customized, and thus the variations are many. For instance, the tournament selection can be carried out by randomly selecting a set number of individuals and choosing the individual with the best fitness as the first parent. The second parent is then chosen in the exact same course of action, with a new set of random individuals.

Crossover
In order to produce offspring as a result of combining two or more parent individuals, as is the case with sexual reproduction in nature, the parent's chromosomes are crossed over. In the simplest case a crossover point is chosen at random in the parents' chromosomes, at which the halves from each chromosome are spliced together. However, this approach does not consider the rule of no doublet cities. To follow this rule a so-called ordered crossover [31] can be used (figure 2). The strategy is to choose a random subset of the first parent's chromosome and replace the rest of the chromosome with the genes from the second parent's chromosome in the order they appear, while excluding the genes that already occur in the chosen subset.

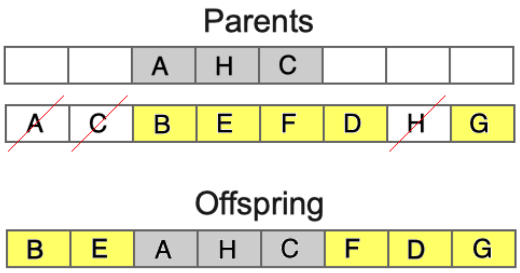


Figure 2: Crossover - an evolutionary mechanism for genetic exchange where two or more parent individuals combine subsets of their chromosomes to form an offspring. This particular example illustrates the crossover between two parents representing city routes for cities A-H. The method is an ordered variant of crossover preventing duplicate genes in the offspring chromosome.

Mutation
After the crossover the offsprings are mutated at a certain probability rate to introduce variation into the population. The mutation is usually very simple and involves random flipping of each gene according to the mutation rate. If the mutation rate is set to 4% then 4% of the genes in a chromosome will most likely mutate. In the case of the traveling salesman the mutation involves random swapping of two genes in the chromosome since a city is only allowed to occur once in the route.

Termination
At some point the execution of the algorithm must stop, hence a termination criterion or function must be defined. The simplest approach is to set a certain generation as the last generation, meaning the number of generations is predefined. Another common approach is to base the termination function on some sort of convergence, usually fitness convergence. If this was to be applied on the traveling salesman

problem the execution would stop when the mean fitness did not change over a certain number of generations - in other words when the algorithm stopped producing shorter routes. If the problem is not of a too complex nature, this stagnation indicates that the best solution has been found, in this case the shortest route.

2.1.3.2 Representation problem

The biggest issue with GAs is the representation problem which lies in the fixed-length character strings. This representation scheme is suitable for many problems where the structure of the solutions are already known, such as the simple example mentioned above with the traveling salesman. Here, different rules and the formula to calculate the route are already known, the only unknown factor is the order of the cities. For many problems, the structure and size of the solutions are not known in advance. The fixed-length representation would therefore put a limitation that makes the problem unsolvable. Another disadvantage of this representation scheme is the difficulty to represent the solutions with a hierarchical structure. All of this demands a more flexible representation where the size, shape and structural complexity is also part of the solution. [27]

2.1.4 Genetic programming

2.1.4.1 Representation

Genetic programming (GP) is an extension of genetic algorithms. In GP, the population consists of computer programs that evolve with the aim to solve a user-defined problem. Depending on the problem, the term "computer program" can refer to a mathematical expression, a model, a transfer function, a control strategy, decision tree or an actual computer program.

The main difference between GA and GP is the representation of the solution candidates. GP deals with the representation problem in GA by increasing the complexity of the structures. Instead of using the fixed-length character strings, GP uses programs with variable shapes and sizes and ability to represent hierarchical structure. [27]

The most common representation of hierarchical programs in GP is the tree-based representation. An example is shown in figure 3, where the tree represents the simple mathematical expression "3x+1-y". The structure of the tree consists of nodes and links. A node specifies a program instruction to be executed and produces a certain output. A link specifies the input for the instruction. The tree's leaves, which are the nodes with no child nodes, are called terminals and the other nodes are called functions. Each function's child nodes must be executed and return a value before the parent node can be executed. [32]

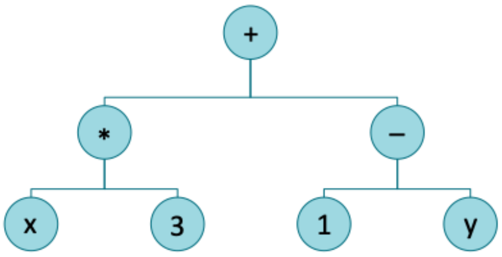


Figure 3: A tree-based representation of a hierarchical GP program which represents the arithmetical expression 3x+(1-y).

In this thesis, all the descriptions are referred to the tree-based representation. However, there are two other GP representations that are interesting and worth mentioning: the linear and graphical representations. The linear GP is very similar to GA with linear chromosomes; however, the size of the chromosome is not fixed. In graphical GP, the programs are represented as graphs with links that indicate the flow of the program. This allows parallelism in the flow and reuse of results. [33]

2.1.4.2 Problem-specific components

As a preparatory step, a set of functions and terminals has to be specified. This set of components defines the search space of possible programs since the programs are compositions of the available functions and terminals in the set. [32]

The set of functions and terminals is specific to the particular problem. One of the best known applications of GP is the symbolic regression problem. For this problem, the program is a mathematical expression, the function set consists of arithmetic functions and the terminal set consists of external inputs and numerical constants. [32]

This means that the choice of the set of functions and terminals is extremely important. If the function and terminal set is insufficient, the GP may not be able to create a program that can solve the problem. [32]

2.1.4.3 Genetic operations

The genetic operations in GP are similar to GA. There are crossovers and mutations. Because of the structural difference, these operations are performed differently.

The crossover operation is performed by swapping a random subtree of one of the parents with a random subtree of the other parent (figure 4). The crossover points in the parents are the same in GA to maintain the fixed size, whereas in GP, the crossover points can be different.

The mutation operation is performed by modifying a randomly chosen subtree in the tree. This is usually performed by replacing the subtree by a new randomly generated subtree (figure 5). Mutation can be seen as randomly sampling new points in the search space to prevent premature convergence. [33]

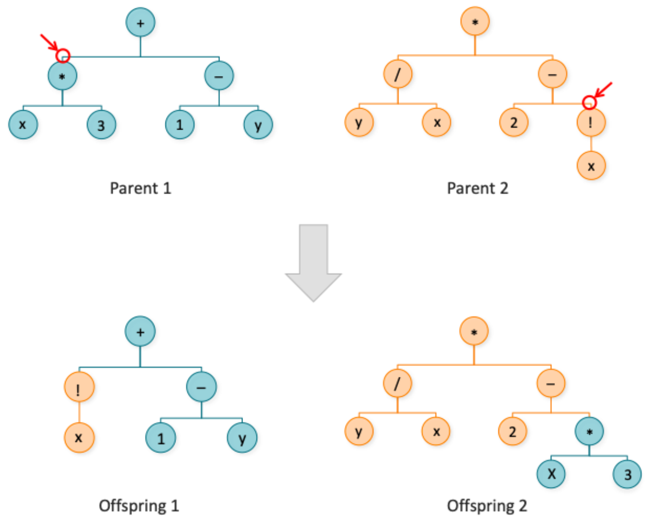


Figure 4: The process of crossover between tree-based representations of hierarchical GP programs. The crossover is performed at the marked crossover-points.

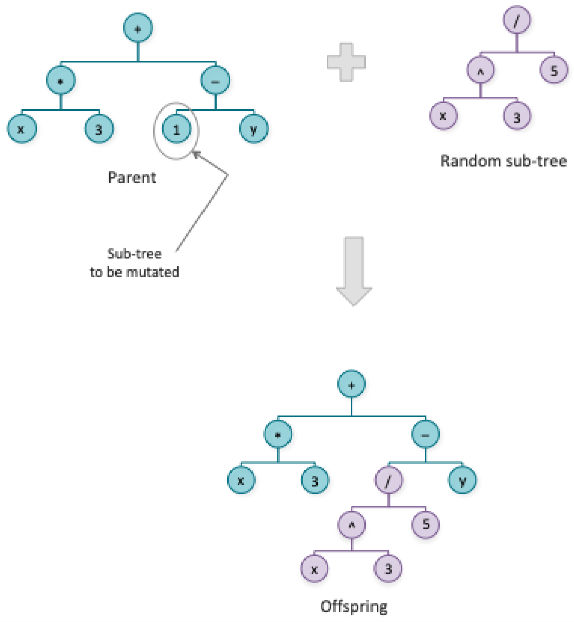


Figure 5: The process of mutation, where a parent tree (blue tree) replaces one of its subtrees with a randomly generated sub-tree (purple tree).

In case there is a need of correct syntax for the program to be able to run, the genetic operations have to preserve syntax. For example, when performing crossover, the two randomly chosen subtrees need to have the same type of output in order to preserve syntax.

2.1.4.4 Search space

As mentioned briefly above, the search space is the space of all possible programs that the GP searches. Because of the dynamically varying sizes and shapes of the programs and combinations of functions and terminals, the search space for GP can be of a very high dimensionality and complexity with the number of programs growing exponentially. For instance, with a function set of 3 functions and a terminal set of 2 inputs, there are $2^{(s+1)/2} * 3^{(s-1)/2} * \frac{(s-1)!}{(s+1)/2! * (s-1)/2!}$ programs of size s. For $s = 3$, there are 12 programs. For $s = 7$, there are 2160 programs. For $s = 15$, there are 240185088 programs. For $s = 31$, there are $9.12 * 10^{18}$ programs. The search space grows very quickly. It should be noted that this calculation is for a syntax-free program. Syntax preservation will naturally limit the combinatorics and reduce the number of programs of a certain size. Nevertheless, if there is no limit placed on the size of the programs, the search space is infinite. [34]

2.1.4.5 Genetic diversity

An important property of the GP search is the genetic diversity in the population. Genetic diversity is commonly seen as a key aspect that encourages efficient and effective search so that the population does not get stuck in a local optimum.

Genetic diversity can be categorised into genotypic and phenotypic diversity. Genotypic diversity refers to the variation in terms of structure and content of the program. Phenotypic diversity refers to the variation in terms of functionality of the program. It is usually difficult to know the right type of diversity to control and the ideal level of diversity to maintain. It has been shown that the behavior of different diversity measures during the evolutionary process is problem specific. [35]

Genetic diversity also correlates with the exploration-exploitation abilities of the algorithm. Exploration refers to a global search, where the algorithm searches for new solutions in new regions of the search space. Exploitation refers to a local search, where the algorithm improves the quality of the already existing solution by searching for better solutions in a limited neighboring region in the search space. A low level of genetic diversity suggests exploitation and a high level of genetic diversity suggests exploration. It is important to find a balance between exploration and exploitation at different stages of the evolutionary process. Too much exploration risks missing and failing to exploit good solutions, while too much exploitation risks getting stuck in a local optimum. It is possible to control and promote diversity to achieve this balance. [36]

2.1.4.6 Bloat

Bloat is a well-known issue in GP that is about the growth in size of the programs during the evolution without a corresponding growth in fitness. The bloat issue can cause many problems for the GP search. First and foremost, big programs are computationally costly to evaluate which causes the execution time of the search to become immense. Bloated programs are also more difficult to evolve in a meaningful way, which means to be modified by genetic operations to discover better programs. In other words, bloat can destruct the search mechanisms in GP. [37]

There are a number of techniques to counteract bloat. One common and effective technique, that is easy to implement, is to place maximum limits on the size of the programs. For the tree-based representation, the size of a program can be the number of tree-nodes or the height of the tree. Another common technique is to favor the smaller programs when selecting individuals for reproduction. This technique usually refers to the principle of parsimony (also known as Occam's Razor)[38], which argues that the simplest explanation is the most likely to be correct. Designing genetic operations that specifically control bloat is also an approach tried by many. [34]

2.2 Text classification

Text classification is the task of categorizing textual data into different predefined categories to extract information from its content. Some of the most common use cases of text classification are sentiment analysis, topic detection, spam detection, language detection, question answering and news categorization. Text classification is one of the main tasks within the field of natural language processing. Natural language is the language that humans use to communicate with each other, such as English and Swedish. It is different from computer languages that humans use to communicate with computers. Natural language processing is a field that is developed for computers to be able to understand human languages. [39], [8]

2.2.1 Different approaches to text classification

The approaches to text classification can be grouped into three categories:

- Rule-based
- Data driven
- Hybrid

A rule-based approach uses a set of predefined rules that determine the categories. In the simplest form the rules include different keywords and their appearances in the text. For instance if a category is "animals", then the appearance of words such as "dog", "hedgehog", "paw" or "claw" would be used as some of the rules that the category is classified by. Other simple examples are identifying phone numbers as any sequence of numbers with a length of 9, or identifying email addresses as any word containing the character @. The advantage of this approach is the simplicity of the algorithm. However, the rules are usually manually crafted which requires deep domain knowledge in order to produce high-quality rules and thereby high-quality predictions. Another downside is that the rules are highly fixed for a certain task, meaning that each new task requires a new set of manually produced rules. Such rule-based systems are difficult to maintain, hence it is common to combine rule-based methods with other natural language processing methods. A method commonly used with rule-based techniques is word embeddings. Word embeddings take advantage of neural networks to vectorize words, allowing for the words to be numerically compared which can be used to find better rules. This step is performed as a sort of preprocessing step and is a hybrid approach to text classification. Neural networks are a technique within the ML field, and are just one among many other ML models that can be combined with rule-based systems to automatically find rules for the rule-based model. [8]

Data driven approaches to text classification are based on ML or DL. The advantage of such approaches over rule-based methods is the ability to find patterns in the data a human would struggle to find or not find at all. However, this does not necessarily demand less manual work. ML approaches require a set of features extracted from the text as input, which usually involve tedious manual work and good domain knowledge in order to produce high-quality features. In respect to this, the ML models are not easily generalized to fit new problems or tasks. The ML models also require large amounts of training data which may be very costly. This applies to deep learning models as well. A great advantage over classic ML models however, is that the feature extraction in deep learning is performed by neural networks meaning no human work is demanded. The neural networks perform both feature engineering and classification in an end-to-end fashion. This makes it more transferable between different tasks and applications. However, deep learning models are large which lead to great computational expenses and memory requirements. Even though current model compression techniques help deploying the deep learning models to applications with low computational resources, there are still issues. Some of them are limitations in terms of configuration freedom of the compressed model and dramatic change in model's performance and stability [14]. As previously mentioned, the black box mechanism is also another problem in deep learning models. The interpretability on feature level and inside each layer in the neural network is more or less absent. [8]

2.2.2 Standard text classification pipeline using data driven approach

A standard text classification pipeline using the data driven approach consists of three main steps: text preprocessing, feature engineering and model training [40].

2.2.2.1 Text preprocessing

During text preprocessing, the raw text is transformed into a more digestible form that makes the text easier to analyze. This step consists of a number of different tasks, some of which are generally applicable to most cases while some are dependent on the dataset. For instance, converting the whole text to lowercase is useful in most cases while a process called stemming can degrade classification accuracy. Stemming is the process of reducing a word to its canonical form by cutting off its suffix or prefix. For example, the word "studies" can be reduced to "studi". There is another process called lemmatization that is similar to stemming. The difference is that lemmatization transforms a word based on its intended meaning. The same word "studies" would be reduced to "study" instead. Other widely used preprocessing methods are stop-word removal and noise removal. Stop-words are the commonly used words in a language, such as "the", "an", "a", "is", that have a low level of information about the content of the text. Noise removal is a more domain dependent task, as noise can be different in different datasets. For instance, noise can refer to HTML metadata, file headers, special characters and domain-specific keywords.

Text preprocessing will generally improve the classification accuracy. However, the level of meaningful preprocessing is different for different datasets. It is therefore necessary to verify each preprocessing task that is added to the pipeline. [41], [42]

2.2.2.2 Feature engineering

In this context, features can be defined as some kind of numerical representations of a text, that can be used as input to text classifiers. This transformation is required in order for the text to become comprehensible for the classifier. Feature engineering is the process of extracting and selecting suitable features for a certain task and classifier. There are mainly two kinds of features, frequency based and prediction based [43]. One of the most common and simplest frequency based features are count vectors, also commonly known as bag of words vectors. Each position at a count vector represents a term from the corpus, where corpus is the collection of different texts to be individually classified. The number at a position represents the frequency count for that term in the specific text which the vector represents.

Another common frequency feature is TF-IDF vectors, which are also based on term frequency but in addition carry information on the relative importance of a word in a text or a corpus. The prediction based features are more complex than the frequency based, and usually work by combining several neural network models. The problem with numerical text transformation is the degradation of information that the transformation automatically gives rise to. Such degradation is for instance loss of sequential, contextual and semantic information.

Modern complex prediction features do succeed in capturing some of that information though. A well known and commonly used technique is Word2Vec which maps words to each other in a great vector space, allowing for similarity measures to be created between word vectors. An example that shows that the technique manages to capture some semantic information is the following computation: "King - Man + Woman = Queen". This illustrates the simple vector calculations between the word vectors for "King", "Man" and "Woman" which results in a vector representing the word "Queen" [44]. This example is extraordinary and might seem very impressive, but when applied to real world problems there are obvious limitations which quickly show. For instance, the words "good" and "bad" are located very closely in the vector space, something which is a problem when performing sentiment analysis. [45]

2.2.2.3 Model training

After the features have been created, they are used to train a classification model. There are many machine learning models to choose from. Some well-known models are the support vector machine, the Naïve Bayes algorithm, the random forest algorithm and artificial neural networks. [45]

Chapter 3

Experimental Work

This chapter covers the structure and motivations behind the proposed GP algorithm in this thesis, as well as the methodology used to analyse the performance of the algorithm. Firstly, the algorithm is presented together with the motivations behind each section of the algorithm. The output of the proposed algorithm is demonstrated through one example run. This is followed by a description of the parameter screening, a strategy with the intention of finding influential parameters. There will be a discussion where the influence of different parameters on the GP algorithm is analysed. Finally, procedure and results of the testing of the algorithm are presented.

3.1 Proposed algorithm

The proposed algorithm is a GP algorithm producing text classifiers by means of string matchers. String matching is a technique of finding occurrences of a given text string pattern within another text string. The algorithm and the produced string matchers perform automatic text classification in an end-to-end fashion where no domain-specific knowledge is required. The string matchers also provide interpretability. This is due to the straightforward representation scheme, which is described in section 3.1.1.

The algorithm is inspired by a GP algorithm called "neat-GP" [46]. The "neat-GP" was chosen with respect to its successes in data classification in combination with its bloat controlling properties and use of speciation [46]. The specific concepts from "neat-GP" that we got inspiration from are speciation and reproduction, which will be described in section 3.1.3 3.1.6. It should be noted that we did some modifications to these concepts to fit our representation scheme.

We realized the importance of bloat control early on, as the string matchers grew uncontrollably. This caused excessive execution time for the evaluation of the string matchers. If no measures had been taken, the development of the algorithm would seriously have been limited.

We also realized it would be a good idea to make use of several string matchers and not only the "fittest". This can be done by incorporating speciation, which means to divide the population into different species. The "fittest" string matcher from each species can then be used in a voting classifier. This speciation step will be further explained in sections 3.1.3 and 3.1.8.

To construct the algorithm we used the Deap python library [47], a framework for prototyping and testing evolutionary algorithms. The proposed algorithm is illustrated in figure 6. Each part in the flowchart will be described in sections 3.1.2-3.1.8.

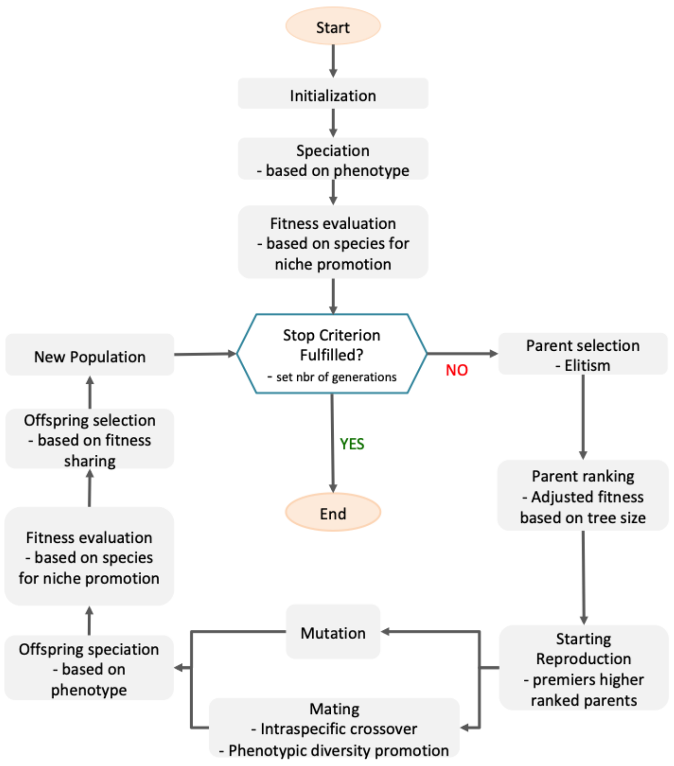


Figure 6: A flow chart illustrating the proposed algorithm.

3.1.1 Representation

As previously mentioned, the algorithm produces string matchers, which in this application refers to classifiers that take a text as input and outputs boolean values, true or false, in response to a set of criterions expressed by the string matcher. These criterions are built up of logical operators and short character n-grams.

Logical operators are symbols or words that are used as connections to create compound expressions. Some common logical operators are AND, OR, XOR and NOT (tables 1-4).

Table 1: Truth table showing logical relationship of the logical operator AND.

X	**Y**	**X AND Y**
False	False	False
True	False	False
False	True	False
True	True	True

Table 2: Truth table showing logical relationship of the logical operator OR.

X	**Y**	**X OR Y**
False	False	False
True	False	True
False	True	True
True	True	True

Table 3: Truth table showing logical relationship of the logical operator XOR.

X	**Y**	**X XOR Y**
False	False	False
True	False	True
False	True	True
True	True	False

Table 4: Truth table showing logical relationship of the logical operator NOT.

X	**NOT X**
False	True
True	False

Character n-grams are sequences of n number of letters which can appear as part of a word. For example, the word "lucky" can be sliced into multiple 3-grams, "luc", "uck", "cky". A simple string matcher might combine some logical operators and character n-grams into criteria that translate into a query. For example, the criteria "AND('tre','ees','swe',NOT('fin')) can translate into the query: "Does this text contain the n-grams 'tre' AND 'ees' AND 'swe' but NOT 'fin'?" . Perhaps the motive behind such a query is to check whether the text is about forests or trees in Sweden as part of a system behind a tourist guide listing beautiful places to visit in Sweden.

There are a number of motives behind choosing this approach. The most important one is that no numerical text transformations are required in order to classify the text data, the text itself is the input to the classification model. This is a great advantage since the transformation of text to numerical vectors degrades information in the data. In other words, the GP algorithm is given access to the entire text which creates opportunities to extract any possible information. Another advantage with this approach is that no text preprocessing is required, simplifying the task even more. This is due to the nature of n-grams, that they are very small components. This makes them resistant to variations in the text, such as spelling and inflection, meaning that the need for preprocessing tasks like stemming can be avoided. [48]

Producing n-grams based string matchers with a GP for text classification is a rather unexplored approach. This makes it an interesting area of research, which is another motive for choosing this approach.

The individuals generated by the GP, that is the string matchers, are represented by binary trees, that is a tree structure with each node having two children at most (figure 7). The function set consists of the following logical operators: AND, OR, XOR and NOT, and the terminal set consists of character n-grams.

The n-grams are randomly generated or randomly selected from a set of n-grams which is extracted from the training data. The alphabet used to generate the n-grams is the Latin alphabet in lowercase form. The n-grams are therefore case-insensitive. Numbers and non-alphanumeric characters are also represented by special characters (€ for numbers and @ for non-alphanumeric characters).

When processing a text, the tree outputs true or false, which represents which class the text has been assigned to. In other words, a true output is synonymous with positive class, and false with negative class. Hence, we view each tree as a classifier. However, each tree can also be viewed as a feature with binary values that can be used to train another classification model. We will later use both definitions to extensively evaluate the quality of the GP algorithm.

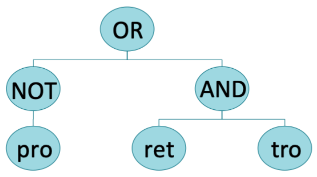


Figure 7: An example of a tree-representation representing a string matcher, which can be translated into the following query: "Does this text contain both 'ret' AND 'tro' OR does it exclude/NOT contain 'pro'?"

3.1.2 Population initialization

At initialization, the population is generated randomly from the function set and the list of n-grams extracted from the training data. The random generation of individuals is a random sampling from the search space. Because of the large number of possible n-grams, the search space becomes enormous. The random sampling might easily miss huge parts of the search space which can cause the evolution to concentrate around one or several smaller parts of the search space where the optimal solution might not be found. In other words, the search might get stuck in the wrong areas of the search space. To combat this risk, a common method is to initialize the population with a large number of individuals. This is derived from a statistical technique, called bootstrap [49]. In this report we will refer to the increased initial random sampling as bootstrap.

By generating a large number of individuals in the first generation, the exploration of the search space is accelerated and spread across bigger parts of the space. However, the evaluation of such a large number of individuals comes at a very high computational cost in terms of execution time. Thus, the size of the population can be decreased in the following generations. In this algorithm the population size is decreased at the third generation to the size given by the population size parameter. This parameter defines the fixed size of the population throughout the rest of the evolutionary process.

3.1.3 Speciation

Speciation is, just as in nature, the division of a population into different species. The purpose of this mechanism in evolutionary computation is to promote diversity and different lineages [50]. The proposed GP algorithm is inspired by the "neat-GP" algorithm as mentioned. One of the inspirations is the speciation mechanism. This mechanism is assumed particularly appropriate for text classification considering the complexity of finding one string matcher that successfully covers all variation in the text data. By speciating the population, different niches within the population could evolve which, hopefully will give rise to individuals covering different variations of the data set.

The speciation is based on phenotypic similarity, which refers to the functionality of the individuals. In this case the functionality is how the individuals label the training data. Speciation can also be based on genotype, meaning the similarity in structure and content would be measured instead [35]. However, the genotype of two individuals might differ a lot without a greater impact on the difference in functionality between the two. Simultaneously a small genotypic difference might greatly impact the functionality. Since the functionality of the individual can be assumed to be of greater importance when dealing with text classification, the phenotypic similarity is a better measure to use for speciation.

To conduct the phenotypic similarity measurement, the classification results of a subset of the training data is vectorized for each individual. The subset used for speciation is balanced, meaning it consists of equal parts of text documents belonging to the negative and positive class. The same subset is used for all speciation throughout the run. To speciate an individual the euclidean distance [51] between the individual's and each of the species representatives' classification vector is calculated. The individual is assigned whichever species it is closest to. The radius which defines a species is controlled by a speciation threshold parameter. This threshold decides whether an individual is similar enough to belong to a certain species. If the individual does not meet this criterion with any of the existing species it gives rise to a new species. When a new species arises the individual giving rise to it becomes that species's representative. At the beginning of the first generation no species have yet been defined, meaning there are no species to compare the first individual to. For this reason, the first individual in the first generation will always represent and define the first species.

To further promote niche-formation the different species are trained on different subsets of the training data. The idea is to simulate different environments in analogy to nature, where different environments can cause a species to develop in different directions, which promotes genetic variation [52].

3.1.4 Fitness evaluation

To evaluate the fitness of the individuals, we first considered three metrics based on classification performance; accuracy, balanced accuracy and F1-score. However, downsides of these metrics were quickly revealed as they create loopholes in the system, which allow for individuals of bad quality to gain high fitness scores. An example is an individual that always outputs the same class - regardless of the input. If the training data are highly unbalanced, meaning one class constitutes for instance 90% of the data, then the individuals that always output the majority class will perform at an accuracy of 90%. If the majority class is the positive class then these individuals would get an F1-score of 95%. This is due to the F1-score only focusing on one class and not taking true negatives into account [53]. Even with the metric balanced accuracy that deals with unbalanced datasets, such individuals would still perform at a balanced accuracy of 50%. We considered this score to still be too high for such meaningless functions.

This is problematic since such individuals appear rather commonly and may outperform better individuals during training. Hence these individuals might hinder the evolution of more promising individuals. Such behaviour can be referred to as specification gaming, a well known challenge within the field of AI, and is defined as " ...a behaviour that satisfies the literal specification of an objective without achieving the intended outcome..." [54].

To combat this behaviour we developed a "rule based" fitness function, which rewards an individual for each correct classification and punishes it even more for each inaccurate classification. Three versions of the "rule based" fitness are used; a neutral, a positive and a negative version. The neutral treats all correct and incorrect classifications the same, while the negative and positive versions focus only on the negative or the positive class. To explain further, the negative version only rewards correct classifications of the negative class and punishes incorrect classifications of the negative class while the classifications of the positive class is not considered. During the development of the algorithm we discovered that the positive version is more suitable for classification scenarios where it is more important to get low false positives, and the negative version is more suitable for classification scenarios where it is more important to get low false negatives. The neutral version is most suitable when it is important to get as high accuracy as possible.

3.1.5 Parent selection

The selection of parents, that is the individuals used for reproduction, is based on elitism [27]. This means that the individuals with the highest fitness are selected. In more detail, for every species, individuals from that species are ranked according to their fitness and the individuals with the highest fitness are selected. The number of selected individuals is dictated by the survival rate.

Before entering the reproduction phase, all the parents are ranked according to an adjusted fitness, irrespective of the species each parent belongs to. The adjusted fitness takes into account the difference between the size of the individual and the expected size. The size of an individual is the number of nodes in the tree, and the expected size is a parameter that guides the individuals towards a certain size. This is done to benefit the reproduction of individuals that are not too small and not too big. Too small individuals would not have enough complexity to produce meaningful classification, while too big individuals might contain bloat.

3.1.6 Reproduction

The parents are reproduced in an iterative fashion, starting with the highest ranked parent, according to the adjusted fitness. A few mechanisms in this step are inspired by the "neat"-algorithm.

A mechanism inspired by "neat" is that every parent gets a number of expected offspring. This limits the number of offspring that one parent can produce and is calculated based on fitness of the parent and the survival rate of the population. This mechanism prevents a single parent from producing too many offspring which would lower the genetic diversity in the population.

Another mechanism inspired by "neat" is that a parent will either mutate or mate with another parent to produce offspring. The probability of mutation is decided by a mutation rate. A parent will only mate with another parent if it does not mutate. Mating is restricted to the same species. This is also inspired by "neat". The other parent is the highest ranked parent of the same species, provided that it has not already reached its expected offspring limit. If the first parent happens to be the highest ranked parent, then the other parent is the second highest ranked parent. The same procedure is repeated, for the highest to the lowest ranked parent, until a large enough number of offspring has been produced.

If a parent is the only representative of its species, it will be allowed to mate with a random parent from other species. Such interspecific mating can counter the niche-formation promoting mechanism of the GP algorithm. It is therefore limited so that there is a 50% chance the parent will mutate instead of partaking in interspecific mating.

Mutation

When mutation occurs, the genome of the parent is copied and mutated to produce the offspring. The mutation method is a standard subtree mutation, meaning that the subtree with the randomly chosen node as the root is replaced by a randomly generated subtree. The randomly generated subtree is limited by a maximum depth that determines how big the mutation can become.

Crossover

Mating is performed by means of crossover between the parents' genomes, that is the tree structures and contents. If the randomly chosen crossover nodes on both parents are terminal nodes, the crossover is performed on the n-gram level instead of the subtree level.

Two crossover methods are explored. The first one is referred to as the short branch crossover, and the second as leaf-biased crossover. Both of these methods are focused on bloat control by limiting crossover of big subtrees. Short branch crossover allows crossover of small subtrees only, while leaf-biased crossover is not as strict. The latter method is performed by either choosing a random terminal node (leaf node) or a random tree node regardless of node type. To what extent terminal nodes are chosen as crossover points is dictated by a terminal probability parameter. A high probability means that most crossovers will be performed on terminal nodes.

Phenotypic diversity promotion
To promote genetic diversity, we introduced a phenotypic diversity promotion mechanism. When crossover occurs, the offspring is compared to its parents in terms of functionality in the same fashion that phenotypic similarity measurement is performed during the speciation procedure. The similarity distance is evaluated against a diversity threshold. This threshold determines how different the offspring's functionality has to be from its parents' to be accepted. The crossover is repeated until the offspring passes the diversity threshold. After a predefined number of times, if the crossover does not produce a different enough offspring, the last produced offspring is chosen. [55]

3.1.7 New population

After producing a large number of offspring, they are ranked based on fitness sharing [50]. The idea is that if two individuals of two different species have the same fitness, the individual coming from a species with smaller population will be ranked higher. In other words, the fitness is shared within a species which will have a more negative effect on bigger populations. The highest ranked offspring will survive and be added to the new population. The number of surviving offspring is the same as the number of individuals that are killed in the old population.

The motivation behind this mechanism is to promote balance between different species populations. During development of the algorithm, it was observed that many new species could not grow because the species only consisted of one individual and its offspring never survived. At the same time, there are species that keep getting larger and eventually take over the whole population.

3.1.8 Convergence

The evolution is terminated after a predefined number of generations. At termination the best individual from each species is chosen as representative. All representatives are combined into a voting classifier [56] which is used to perform the classification. This means that the majority of the representatives decide which class to assign to the text.

3.1.9 Result

Following is the result from one of the runs on the retrospective/prospective study type dataset that we used during the development of the proposed algorithm and for screening. In this particular run, the GP algorithm produced a population with 5 species. The trees representing the top-ranked individual from each of the five species are shown in figures 8-12. While the sizes, structures and contents (in terms of n-grams) of these individuals differ greatly, there are some similarities between them. The first observation is that the root of all the trees is "OR". "OR" is also the most common node in most trees, while “AND” is the least common one. This indicates that the GP algorithm finds “OR” to be more rewarding than “AND” . It should be noted, however, that all operators are used.

Another observation is that most trees tend to grow more on the right side. Additionally, there are certain n-grams that appear in multiple species, such as “etr” and “ros”. These n-grams are part of the word “retrospective”, which corresponds to the positive class in the dataset and this is probably the reason why they are common. At the same time, some n-grams such as “hxs” and “qdk” do not seem to be meaningful as they do not belong to the English language. These can either have been randomly generated or are a product of crossover between n-grams. However, n-grams that seem to bear no meaning are not necessarily useless because they can still contribute to the ability of the algorithm to explore.

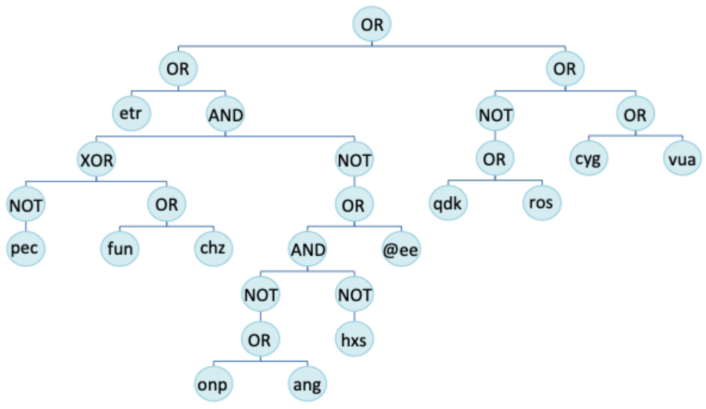


Figure 8: The top-ranked individual from species 1 at the end of evolution

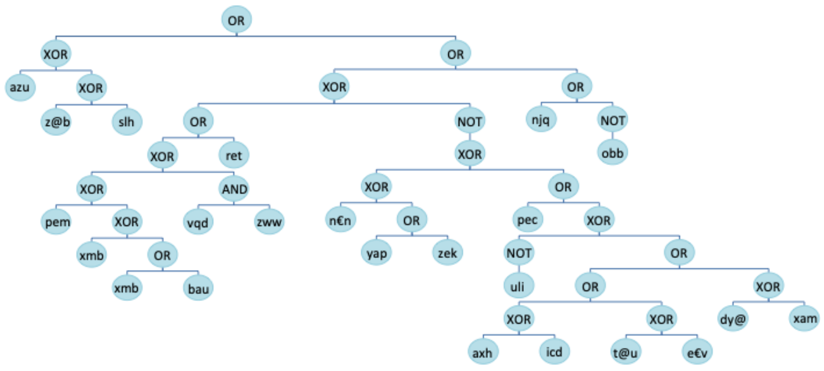


Figure 9: The top-ranked individual from species 2 at the end of evolution

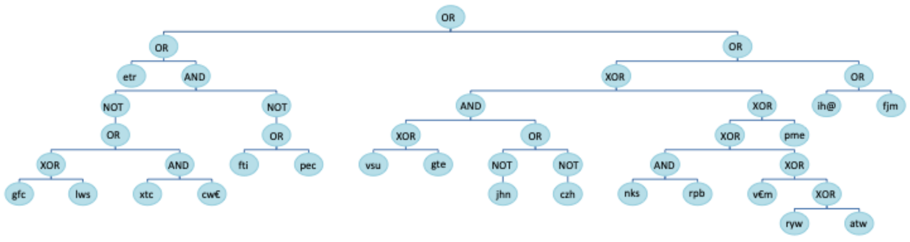


Figure 10: The top-ranked individual from species 3 at the end of evolution

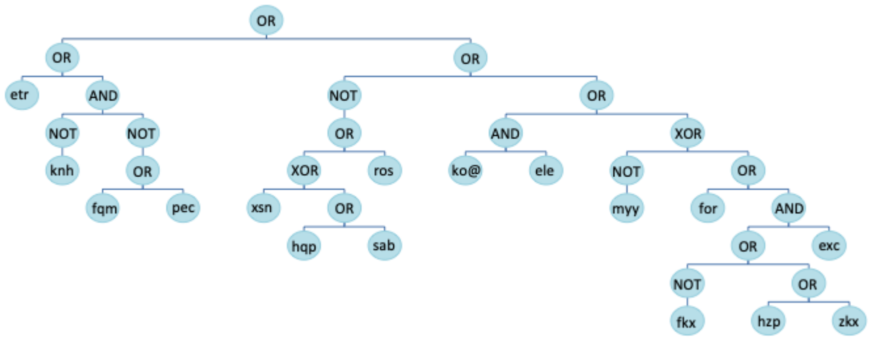


Figure 11: The top-ranked individual from species 4 at the end of evolution

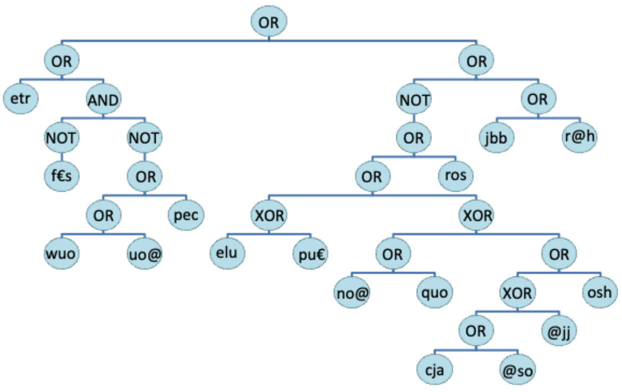
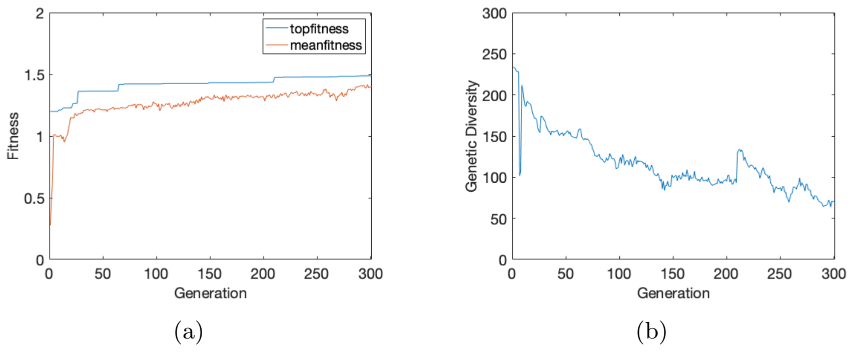


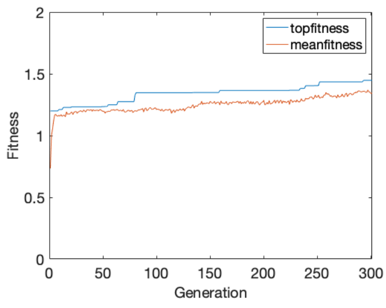
Figure 12: The top-ranked individual from species 5 at the end of evolution

The fitness of the whole population and the population of different species show similar behaviours throughout the run (figures 13-18). The top fitness shows the highest-ranked individual in the whole population. The mean fitness is the average of all individuals' in the population. The mean fitness increases very fast in the beginning but slower afterwards, while the top fitness frequently is at a high level already at the start and only increases sporadically throughout the run. Exceptions are the mean fitness of the species 2 and 3 as they fluctuate greatly. This is probably due to the small size of these populations (figure 20). An individual with low fitness affects the mean fitness of a small population much more than it would in a big population. Therefore, these fluctuations are not visible in the mean fitness of the whole population.

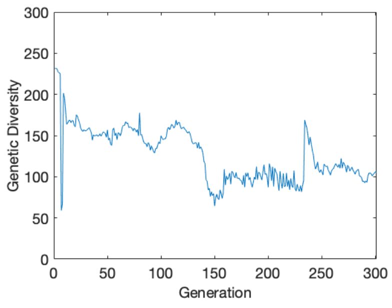


(a) (b)

Figure 13: (a) Fitness and (b) genetic diversity of the whole population

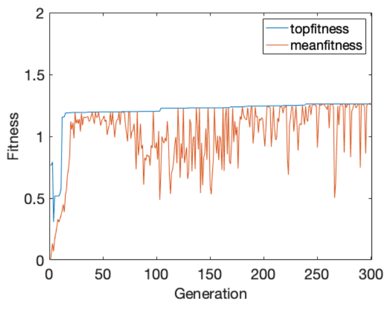


(a)

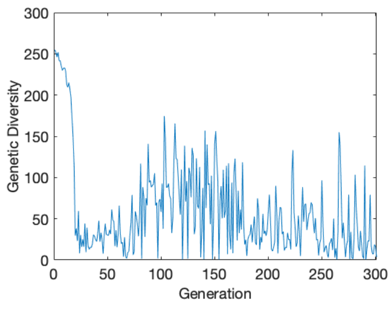


(b)

Figure 14: (a) Fitness and (b) genetic diversity of species 1's population

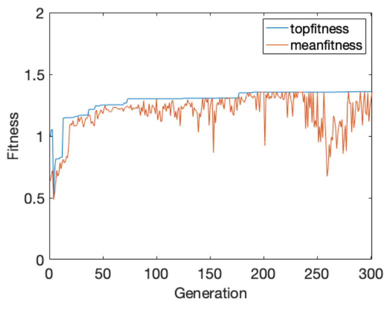


(a)

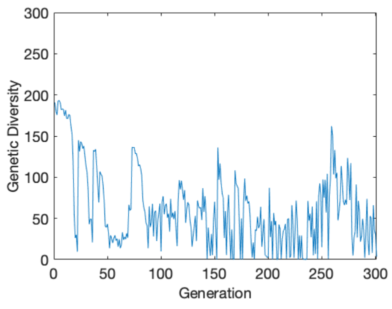


(b)

Figure 15: (a) Fitness and (b) genetic diversity of species 2's population

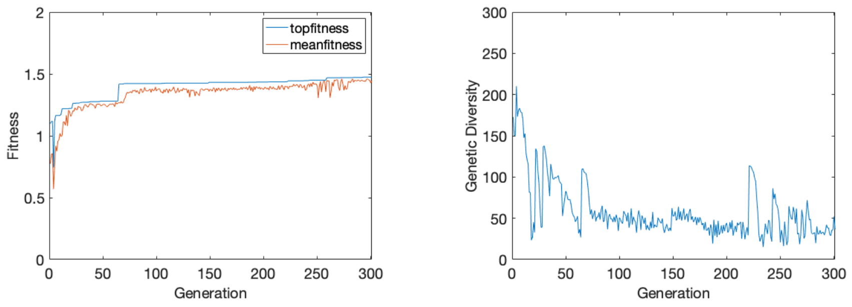


(a)



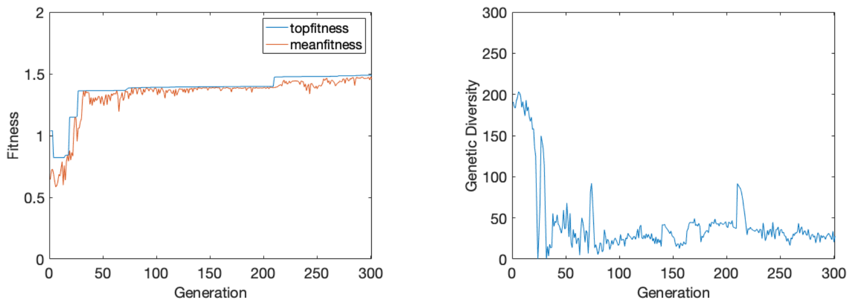
(b)

Figure 16: (a) Fitness and (b) genetic diversity of species 3's population



(a) (b)

Figure 17: (a) Fitness and (b) genetic diversity of species 4's population



(a) (b)

Figure 18: (a) Fitness and (b) genetic diversity of species 5's population

The genetic diversity of the population is based on the same phenotypic similarity measurement that we used in the speciation and phenotypic diversity promotion. The difference is that we have used the whole training dataset instead of just a subset. The diversity is calculated by averaging the euclidean distance between the top-ranked individual in the population with the rest of the individuals in the population. The purpose was to check to which degree the top-ranked individual would spread its genes to the whole population. It should be noted that the largest possible euclidean distance between two vectors used here is approximately 300, which explains the scale of the genetic diversity (figures 13-18).

In the diversity plots, the genetic diversity decreases throughout the run, which is expected. However, the decrease is not consistent as it may remain stable for a number of generations or even increase occasionally. This suggests that the GP algorithm promotes and maintains genetic diversity in the population. For species 2 and 3 the diversity fluctuates greatly, which also is the case with the fitness. As mentioned earlier this is probably due to the small size of these populations.

To evaluate the effect of the genetic diversity promotion mechanism, we tested the algorithm with and without the mechanism. Both settings were run 100 times and the stop criteria were either to reach 75% accuracy and 75% balanced accuracy or to reach 1000 generations (figure 19). It could be seen that the genetic diversity promotion mechanism improved the results. There were less runs that did not reach the accuracy criterion when the diversity promotion mechanism was activated. Only 11% of the runs with diversity promotion failed while 25% of the runs without diversity promotion failed. Also the number of generations needed to reach the accuracy criterion was lower and more consistent when the diversity promotion mechanism was activated.

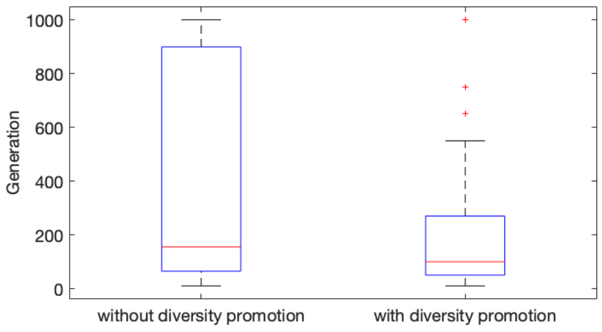


Figure 19: Number of generations to reach stop criterion of 100 runs for different diversity promotion settings.

The population sizes of different species throughout the run suggest that the mechanism that keeps the balance between the different species populations works (figure 20). There is no dominant species that grows and takes over the whole population and there is no species that only consists of one individual. For instance, species 1 has the biggest population after the initial sampling. This indicates that the individuals from this species originally have higher fitness than individuals from other species. It could be expected that this species would keep on growing. However, it instead decreases and remains at a lower level. It should be noted that the initial sampling with a large number of individuals is not included in the plot as it only lasted for the first 3 generations.

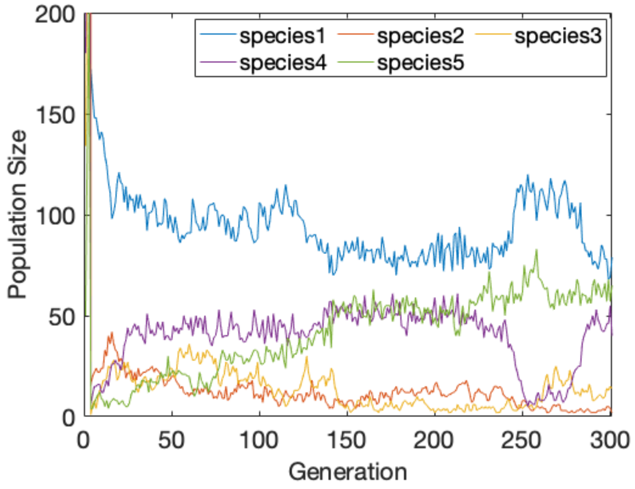


Figure 20: Population size of different species.

The average size of the individuals in different species' population is measured as the number of tree-nodes. Even though an upward trend can be observed throughout the run, the individuals' size does not keep growing exponentially, which would be expected if there was a bloating problem (figure 21). It is expected that individuals should grow somewhat in size as their fitness increases. This shows that the bloat control mechanism works. Another remark is that the species that grow remarkably have a small population. This growth might be due to a weaker size-based selection pressure on smaller populations.

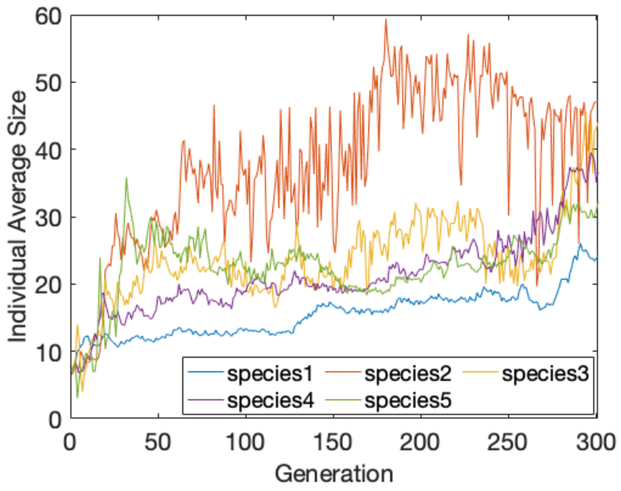


Figure 21: Average size of individuals in different species.

The classification performance on the validation data is well reflected in the training data as the curves follow each other exactly (figure 22). The classifier even gives a slightly better performance on the validation data. This shows that the classifier produced by the GP algorithm is capable of generalizing, which means it performs well on both training data and new, unseen data.

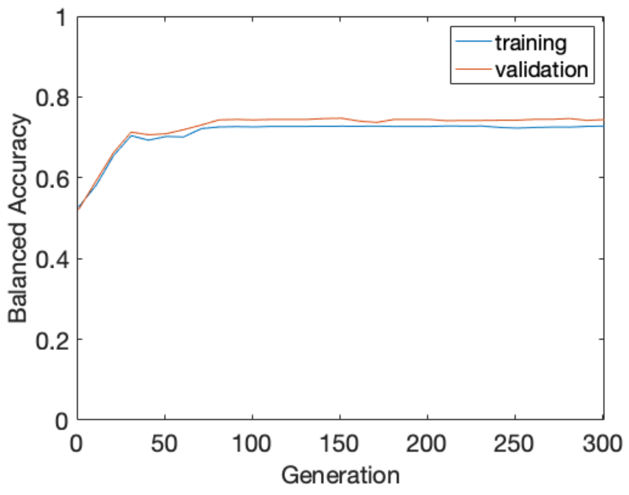


Figure 22: Classifier evaluation on both training data and validation data.

3.2 Parameter screening

3.2.1 Procedure

There are many parameters in the proposed algorithm that can affect the performance of the algorithm. Parameter optimization is the task of detecting an optimal set of parameters for a particular model on a particular dataset. It is a time-consuming task, especially in the case where there are many to choose from. [57]

Since the focus of this thesis is not to find the optimal set of parameters to produce the best possible results using the proposed algorithm on different datasets, the parameter screening is rather to identify the influential parameters and to get a rough estimation of a good value for each of them.

Table 5 shows all parameters in the proposed algorithm and the different values of each of these that we tested. For all the parameters, there are low, medium and high values that are determined. The choices of these different values are empirical and based on the experiences gained throughout the development of the proposed algorithm. It might look like a simple optimization problem but there are approximately 1 million combinations just from these few parameter values.

Table 5: List of parameters and corresponding values used for screening

Parameter	Values
n-gram length	2, 3, 4, 5, 6, varying between 2 and 5
speciation threshold	2.5, 2.75, 3, 3.25, 3.5, 3.75, 4, 4.5
mutation rate	0.3, 0.5, 0.7
survival rate	0.3, 0.5, 0.7
crossover method	short-branch, leaf-biased (terminal probability = 0.6, 0.8)
diversity threshold	1.5, 2, 2.5
expected size	0, 5, 10, 15, 20, 25, 30, 35, 40, 45, 50
bootstrap size	no bootstrap, 2000, 10000
population size	100, 200, 300
mutation max depth	2, 4, 6

To assess the performance of the algorithm with different parameters, the k-fold cross-validation method with k = 10 is used. This means that we split the data into 10 parts and trained 10 different models; each model was trained with 9 parts and evaluated on the tenth part. The final evaluation is the average of all 10 evaluations. This method helps using all the data for both training and validation and gives a more reliable result by evaluating multiple slightly different scenarios. For each GP run, the evolution is terminated after 300 generations. This seems to be long enough for the GP algorithm to produce reasonable results that are fair to compare between different parameter settings. [58]

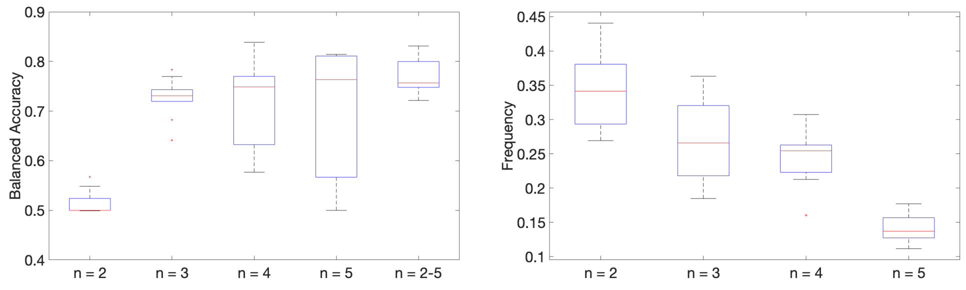
The dataset that we used for this part consists of medical papers that were categorized into two classes: retrospective study (positive class) and prospective study (negative class). This dataset is imbalanced with the positive class being the majority class. The suitable evaluation metric is therefore balanced accuracy. [59]

3.2.2 Result

3.2.2.1 N-gram length

Bigrams (n=2) perform remarkably worse than n-grams of other lengths in the range 3-5 (figure 23a). This is not surprising considering the generality of such short letter sequences which probably makes bigrams too general to capture any meaningful patterns in the text. The other n-grams appear to have the ability to reach approximately the same level of performance in terms of balanced accuracy, but 3-grams perform most consistently. A possible explanation for the somewhat inconsistent performance of 4-grams and 5-grams is that the longer the sequence the more combinations exist, making it more difficult to find suitable n-grams. There is also an aspect of the nature of the dataset as N-grams of a particular length might perform better on some datasets than on others.

The highest balanced accuracy was reached using n-grams of variable length, ranging from 2-5. This indicates that the best approach is to let the GP explore on its own which lengths to use. Studying appearance frequency of different n-gram lengths, it shows that the GP algorithm prefers bigrams, followed by 3- and 4-grams (figure 23b). 5-grams make up only around 14% of the terminals. This shows that it is valuable to combine terminals with different levels of generality. For instance, the bigrams appear frequently despite the low performance when operating on their own. Perhaps it is the high level of generality that makes it easier to find suitable bigrams faster that do not have a negative effect in combination with other n-grams. With the same logic, the reason for the relatively low frequency with which 5-grams appear might depend on that they are too specific and hence difficult for the GP to make use of. Furthermore, as 5-grams are the n-grams that are least common in the training data, the chances to randomly choose a 5-gram may be smaller. Because of the high performance when we used varying length n-grams, we chose this setting for the testing. The setting with 3-grams is also used for the testing because of its consistency in performance.



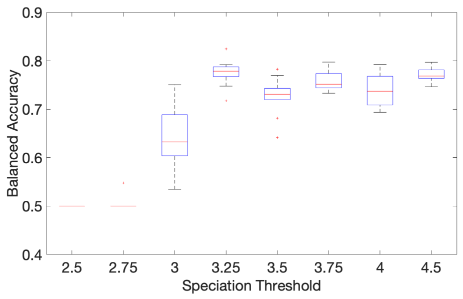
(a) (b)

Figure 23: (a) Classifier evaluation on validation data for different n-gram length settings. The plot indicates that the GP performs best when combining n-grams of varying length. (b) The appearance frequency of different n-gram lengths in the best individuals from all species when running the GP on varying n-gram length. The GP seems to prefer shorter n-grams.

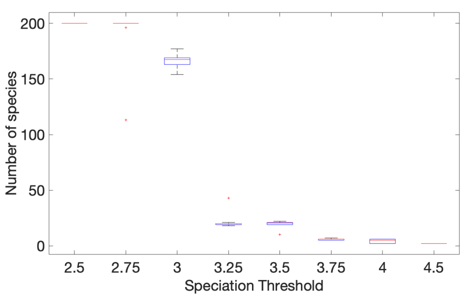
3.2.2.2 Speciation threshold

As anticipated, it can be seen that the smaller the speciation threshold is the more species will arise (figure 24b). A drastic drop in the number of species is also notable when increasing the speciation threshold from 3 to 3.25. The change in number of species is well reflected, showing the GP algorithm performance for the different speciation thresholds (figure 24a). Studying these two figures together shows a relation between performance and number of species. The GP clearly performs worse when the number of species is too high, especially as high as up to 200 as is the case with the two lowest speciation thresholds. This is the same as the population size, meaning that each individual is its own species. When this is the case, there is no room for evolution. This is reflected in the diversity plot, where the diversity for the two lowest thresholds are relatively constant throughout the run (figure 24c). This means that the population probably has not changed much during the run. The performance is still quite unsatisfactory when the number of species is a bit lower than 200, but still quite high. A reasonable explanation is that few offspring can survive to the next generation. Hence the algorithm gets stuck in a small part of the search space, which decreases the performance.

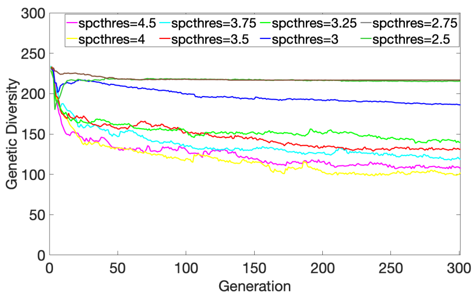
For speciation thresholds above 3.25, no notable change in performance can be observed (figure 24a) and no relation between number of species and performance is suggested (figues 24a & 24b). The best results are achieved using threshold 3.25 or 4.5. Even though the differences between the results for speciation thresholds above 3.25 are not remarkable, we decided to use both thresholds 3.25 and 4.5 since they yielded different numbers of species, which can be interesting to compare.



(a)



(b)

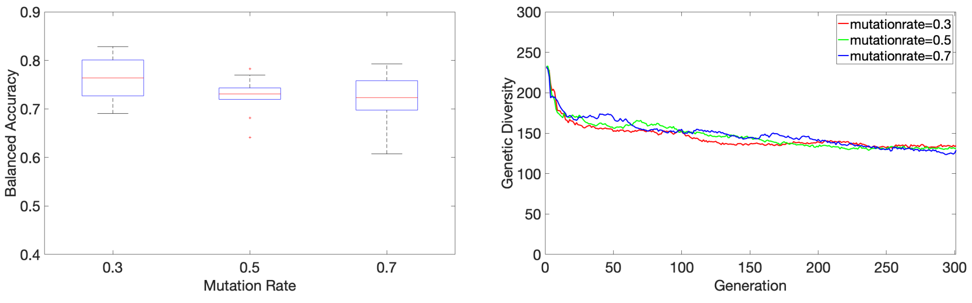


(c)

Figure 24: (a) Classifier evaluation on validation data generated with different speciation thresholds. (b) The number of species arising as a result of the different speciation thresholds. The results are anticipated, the smaller the threshold, the more species arise. This is natural since a higher similarity is required within a species meaning that less individuals will fall into the same category. (c) The mean genetic diversity in the population at each generation using different speciation thresholds.

3.2.2.3 Mutation rate

No difference in performance or diversity can be observed for different mutation rates (Figures 25a & 25b). A mutation rate of 0.3 performs somewhat better and is thereby chosen for the testing.



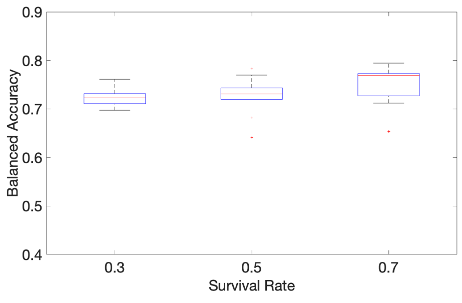
(a) (b)

Figure 25: (a) Classifier evaluation on validation data generated with different mutation rates. (b) The mean genetic diversity in the population at each generation using three different mutation rates.

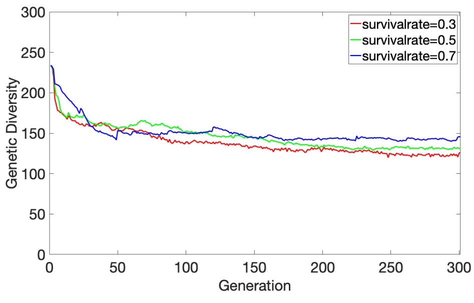
3.2.2.4 Survival rate

The diversity tends to be lower when the survival rate is also lower (figure 26b). This makes sense since a lower survival rate means there are fewer individuals left for reproduction, which in turn means that more individuals will share the same parents. This should produce less diverse offspring than reproduction involving a higher number of parents. A weak trend can be observed that the higher the survival rate is, the better the performance will be (figure 26a). The highest survival rate, 0.7, also yields the highest diversity. Therefore we chose this setting for the testing. An additional advantage of using a higher survival rate is also that it requires less execution time. The reason is that fewer offspring need to be produced and thereby evaluated. The fitness evaluation of new individuals is the most time costly part of the GP.

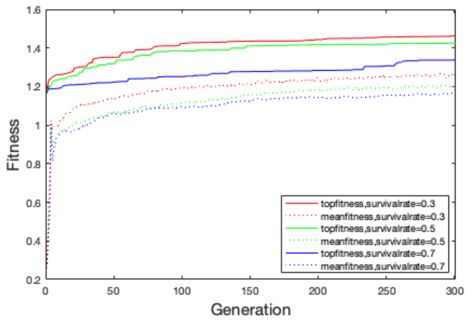
The fitness plot shows how the top fitness is evolving throughout the run for different survival rates (figure 26c). The highest survival rate seems to cause a more even fitness evolution throughout the run, while the other two rates cause a faster fitness climb in the beginning which then stops more or less. This might indicate that the algorithm gets more easily stuck in a subarea of the search space when using lower survival rates. As mentioned above, lower survival rates yield lower diversity, something which further indicates that the algorithm gets stuck more easily.



(a)



(b)

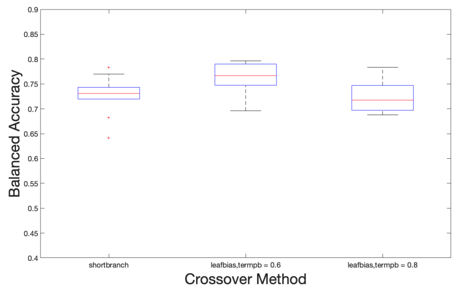


(c)

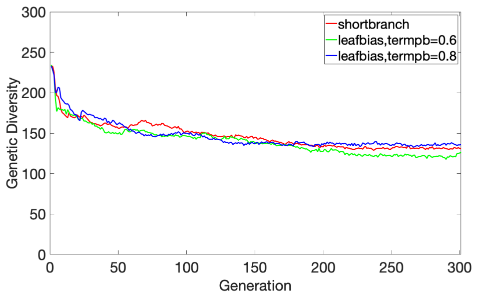
Figure 26: (a) Classifier evaluation on validation data for classifiers generated with different survival rates. (b) The mean genetic diversity in the population at each generation using three different survival rates. (c) The average of the top-fitness and mean-fitness of the population at each generation using three different survival rates.

3.2.2.5 Crossover method

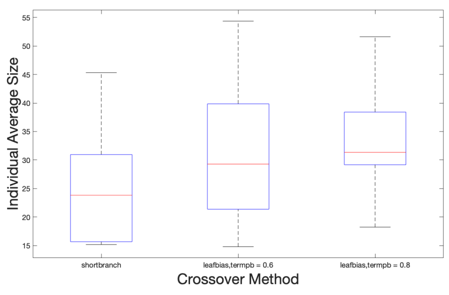
No difference in performance, diversity or bloat control can be detected for different crossover settings (Figure 27). On average, the leaf biased crossover with a terminal rate of 0.6, yields the highest balanced accuracy and therefore we chose this setting for the testing.



(a)



(b)

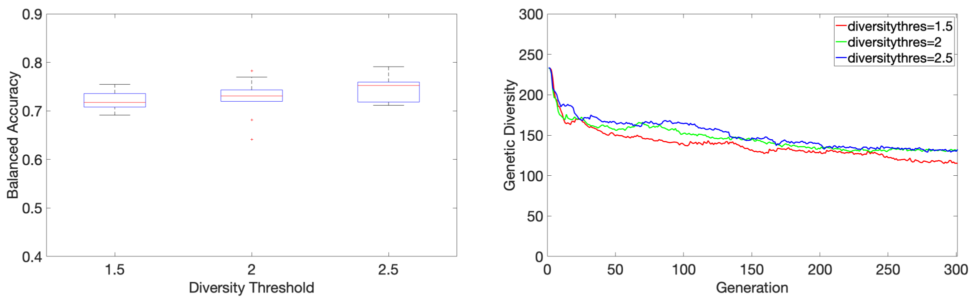


(c)

Figure 27: (a) Classifier evaluation on validation data generated using different crossover methods. (b) The mean genetic diversity in the population at each generation using three different crossover settings. (c) The average size of the individuals (in terms of number of tree nodes) in the population generated by using different crossover methods.

3.2.2.6 Diversity threshold

An increase in diversity threshold seems to improve both the performance and the diversity (figure 28). This might be due to the diversity threshold affecting the diversity, which in turn has an impact on the performance. Therefore, we chose the setting with the highest diversity threshold for the testing.

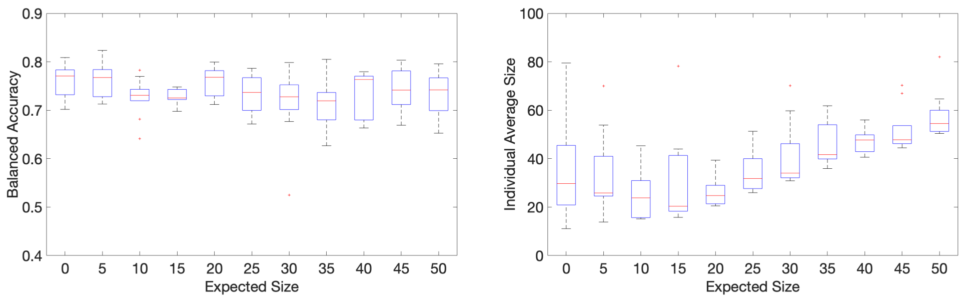


(a) (b)

Figure 28: (a) Classifier evaluation on validation data generated using different diversity thresholds. (b) The mean genetic diversity in the population at each generation using three different diversity thresholds.

3.2.2.7 Expected size

The size of the individuals in the population does not seem to have any influence on the GP algorithm's performance (figure 29). Hence, the parameter controlling the expected size is probably not an influential parameter in terms of performance. A small expected size is therefore chosen for the testing since this will decrease execution time. The expected node-size yielding the highest balanced accuracy is 5, which is small and suitable for testing.

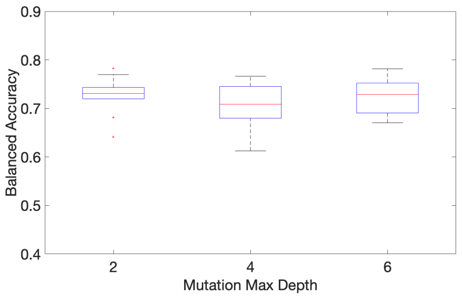


(a) (b)

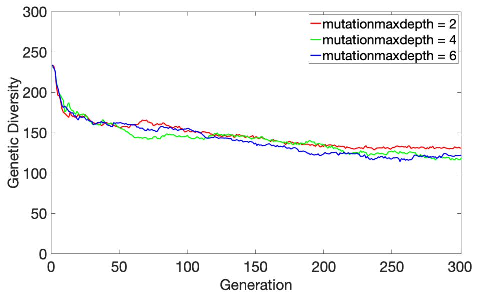
Figure 29: (a) Classifier evaluation on validation data for classifiers generated with different expected size. (b) The average size of the individuals (in terms of number of tree nodes) in the population generated by using different expected sizes.

3.2.2.8 Mutation max depth

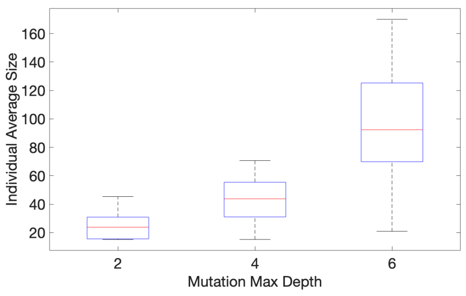
The mutation maximum depth, which controls the maximum size of the subtrees that are produced for mutation, does not have a strong influence on the performance or the diversity (figures 30a & 30b). However, it is clear that this parameter has an exponential effect on the size of individuals within the population (figure 30c). Since the performance was not affected by the size of the individuals, we chose the lowest mutation maximum depth for the testing.



(a)



(b)

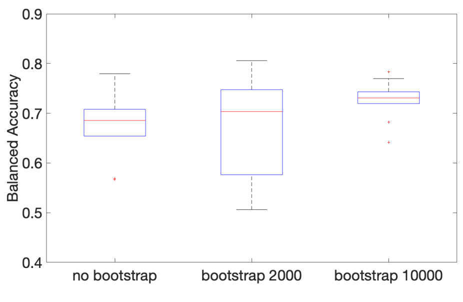


(c)

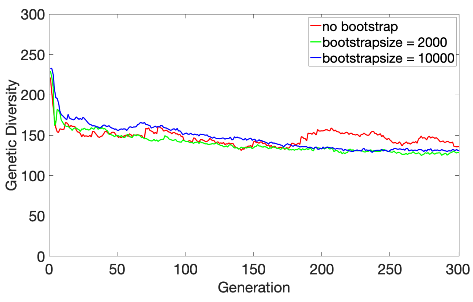
Figure 30: (a) Classifier evaluation on validation data for classifiers generated with different mutation max depth. (b) The mean genetic diversity in the population at each generation using three different mutation max depths. (c) The average size of the individuals (in terms of number of tree nodes) in the population generated by using different mutation max depth. It can be seen that the higher the mutation max depth is, the bigger the individuals become.

3.2.2.9 Bootstrap size

When comparing the median balanced accuracies a weak trend can be observed, that when greatly increasing the sampling size, the balanced performance increases somewhat (figure 31a). The sampling size does not seem to have a bigger effect on the diversity (figure 31b). Even though it can be seen that the diversity is somewhat bigger at the end of the run without bootstrap, the increase is not significant. Also, a higher sampling size should theoretically increase the exploration and therefore bootstrap was used for the testing. Since a sampling size of 10000 yielded the most consistent results this setting was chosen for the testing.



(a)

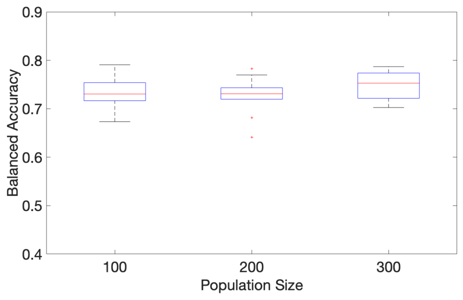


(b)

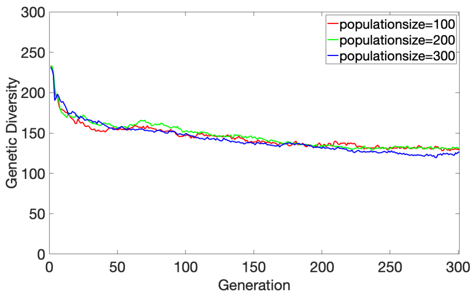
Figure 31: (a) Classifier evaluation on validation data for classifiers generated with different bootstrap settings. (b) The mean genetic diversity in the population at each generation using three different bootstrap settings.

3.2.2.10 Population size

The population size does not seem to have an effect on the performance and diversity (figure 32). The highest balanced accuracy is however achieved by the largest population consisting of 300 individuals. Thereby this setting is chosen for the testing.



(a)



(b)

Figure 32: (a) Classifier evaluation on validation data for classifiers generated with different population sizes. (b) The mean genetic diversity in the population at each generation using three different population sizes.

3.3 Testing

3.3.1 Procedure

Based on the results from parameter screening, we decided to test 2 different values for both n-gram length and speciation threshold (Table 6). The 4 different settings of the GP algorithm will be evaluated on two different tasks: topic detection and sentiment analysis.

Table 6: The chosen settings for the hyperparameters based on the results from the screening

Parameter	**Setting 1**	**Setting 2**	**Setting 3**	**Setting 4**
n-gram length	3	varying	3	varying
speciation threshold	3.25	3.25	4.5	4.5
mutation rate	0.3	0.3	0.3	0.3
survival rate	0.7	0.7	0.7	0.7
crossover method	leaf-biased, terminal probability = 0.6	leaf-biased, terminal probability = 0.6	leaf-biased, terminal probability = 0.6	leaf-biased, terminal probability = 0.6
diversity threshold	2.5	2.5	2.5	2.5
expected size	5	5	5	5
bootstrap size	10000	10000	10000	10000
population size	300	300	300	300
mutation max depth	2	2	2	2

For topic detection, medical papers with different topics are chosen as test sets. For each topic, there are balanced and imbalanced datasets with the negative class being the majority class. The topics are as follows:

- Retrospective study (Positive class) and Prospective (Negative class)
- Animal (Positive class) and Human (Negative class)
- DNA (Positive class) and RNA (Negative class)
- Virus (Positive class) and Bacteria (Negative class)

For sentiment analysis, we used social media posts as test sets. Just as with the medical papers, there are balanced and imbalanced datasets with the negative class being the majority class.

Since the datasets are both balanced and imbalanced, the suitable evaluation metric is balanced accuracy. [59]

The evolution run is terminated after 1000 generations.

We compared the performance of the classifiers produced by the GP algorithm with a baseline classification. The baseline classification consisted of no preprocessing and we used count vectors as features and random forest as classification model.

As previously mentioned, each individual can be viewed as both a classifier and a feature. In order to further evaluate the quality of the GP algorithm, the outputs from each species' representative in the classification model are used as features to train a random forest classification model.

3.3.2 Results

The classification models produced by different parameter-settings overall give better classification results than the baseline classification (tables 7 & 8). 72.5% of the results are better than the baseline (figure 33).

On the balanced datasets, the results from all 4 GP settings of the GP algorithm are better than the baseline on 3 of the 4 topic detection datasets and on the sentiment analysis dataset. The only dataset on which the GP algorithm did not perform better than the baseline was the human/animal dataset.

For the unbalanced datasets, more results of the GP algorithm were no better than the baseline. On the topic detection datasets, the models produced by setting 4 failed to perform better than the baseline on the human/animal and rna/dna datasets and the models produced by setting 2 failed on the rna/dna dataset. On the sentiment analysis dataset, the models from all 4 settings all failed to perform better than the baseline. However, on the study type and virus/bacteria datasets, the models from all 4 settings all perform much better than the baseline.

When using the outputs from all species' top individuals as features to train the random forest model (which is the same ML-model that is used in the baseline), the number of results not better than the baseline decreases greatly to 5% (figure 33). The improvement in performance for some of the models is remarkable. For example, the performance of the model from setting 2 increases from 0.5 to 0.78 on the unbalanced rna/dna dataset. There are only three out of 40 results that are worse after using together with the random forest ML-model. These are all results on the sentiment analysis dataset, which is the most difficult dataset to analyze. This is due to the texts being much shorter and not as well-written as the medical papers, which means that there are much more variations, such as unusual abbreviations and misspellings.

Table 7: The test results using the chosen set of hyperparameters on balanced datasets. (with voting classifier / with ML)

Balanced Dataset	Baseline	Setting 1	Setting 2	Setting 3	Setting 4
Virus/ Bacteria	0.85	0.92 / 0.95	0.94 / 0.95	0.93 / 0.93	0.96 / 0.96
Study Type	0.57	0.79 / 0.85	0.86 / 0.88	0.84 / 0.84	0.85 / 0.86
RNA/DNA	0.75	0.83 / 0.83	0.78 / 0.84	0.81 / 0.83	0.78 / 0.83
Human/ Animal	0.70	0.68 / 0.71	0.68 / 0.74	0.65 / 0.65	0.62 / 0.71
Sentiment	0.53	0.61 / 0.60	0.64 / 0.60	0.57 / 0.59	0.58 / 0.58

Table 8: The test results using the chosen set of hyperparameters on unbalanced datasets. (with voting classifier / with ML)

Unbal-anced Dataset	Baseline	Setting 1	Setting 2	Setting 3	Setting 4
Virus/ Bacteria	0.51	0.88 / 0.93	0.84 / 0.96	0.88 / 0.92	0.75 / 0.78
Study Type	0.51	0.55 / 0.84	0.91 / 0.91	0.90 / 0.90	0.91 / 0.91
RNA/DNA	0.54	0.55 / 0.73	0.50 / 0.78	0.66 / 0.75	0.51 / 0.76
Human/ Animal	0.51	0.59 / 0.67	0.52 / 0.63	0.58 / 0.63	0.50 / 0.60
Sentiment	0.51	0.51 / 0.52	0.50 / 0.53	0.51 / 0.52	0.51 / 0.50

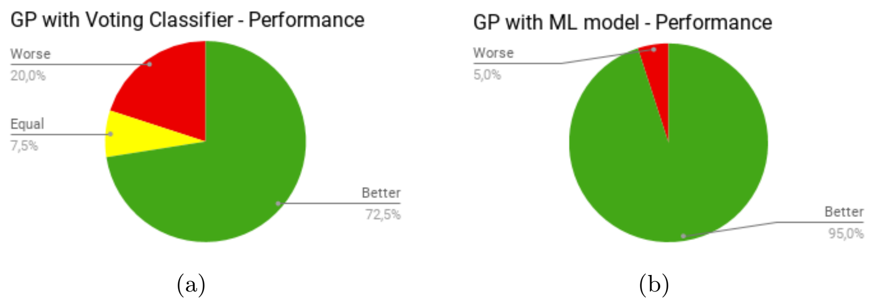


Figure 33: The proportion of classification results compared to the baseline classification, with the voting classifier (a) and with ML model (b).

Chapter 4

Discussion

In this chapter, we discussed some of the most important aspects of the algorithm, challenges and possible solutions for future work.

4.1 Results

Classification performance

Overall the proposed algorithm performs better than the baseline in most cases, but the performance varies greatly with different parameter settings. The performance also varies in terms of its consistency. This can be observed in several plots from the parameter screening, for instance in figure 23a which shows a great difference between the performances achieved using bigrams and 3-grams. It also shows an inconsistent performance when using 5-grams compared to 3-grams or bigrams. This indicates the necessity of parameter tuning before each new classification task.

The performance of the GP algorithm also varies greatly between different types of datasets. With the parameter settings chosen from the screening, the algorithm performs very well on some of the datasets. In most cases, the baseline performance is also good on those datasets which might indicate that these datasets are easier to perform text classification on. There are two datasets for which this is not the case however. In particular for the unbalanced study type dataset, the GP algorithm drastically outperforms the baseline model. This dataset is of the same type as the dataset used for the parameter screening, which further supports that the parameter settings have an important impact on the performance. This observation is promising, implying that satisfactory results can be achieved with accurate parameter tuning.

Another promising result is the performance achieved when using an ML model to weight the contributions from each species. This method improves the performance for almost all tests, especially for tests carried out with setting 2. The test results for setting 2 are on average worse than for the other settings, with 3 out of 10 results being worse than the baseline results. The improvement after using the ML model, however, is greater compared to the other settings. The main difference between setting 2 and the other settings is that it uses n-grams of varying length and produces more species due to a low speciation threshold. This yields a higher level of flexibility and produces a larger ensemble of models with different levels of generality and complexity. To give every species an equal vote through the voting classifier might not be optimal in this case. Thus, by letting the ML model weigh the contributions from each species, the results become notably better.

Representation

The proposed representation, involving character n-grams and logic operators, appears to work fairly well given the results from the testing. However, some information is missed using the current function sets that the GP algorithm is equipped with. For instance, the functions have no ability to capture sequential information that could capture contextual information. Also, the proposed GP algorithm is case-insensitive which gives some additional information loss as the letter case has a potential of providing contextual information. Another step in which some information loss may occur is the normalization of numbers and special characters. In this procedure numbers and non-alphanumeric characters are replaced with special characters. This can be addressed by excluding this step. It may, however, be necessary to keep it in order to keep the search space clean from unnecessary dimensions. Each time the terminal set or the function set is introduced with more alternatives, the search space becomes greater and more complex to search. Hence, there is always a trade-off between giving the GP all information and ability to use it freely, and preventing the search space from growing to unmanageable dimensions.

Diversity

Normally, a decrease in genetic diversity throughout the run is expected as the population consists of random individuals in the beginning but then gradually more individuals that share genes. However, the drastic drop in genetic diversity at the very beginning of almost every run does not seem normal. The only exception to this is when the survival rate is high and when the speciation threshold is so low that there are extremely many species in the population. The drop might be due to a combination of the effects of low survival rate, elitism selection and the mating mechanism.

A low survival rate leads to a low number of parents or in other words, a small gene pool to produce offspring for the next generation. It can therefore be assumed that the lower the survival rate, the less diverse the offspring will become. This can be connected to the exploration-exploitation balance. Too low survival rate might make the exploitation effect too strong.

Selection is often seen as being exploitative and genetic operations (mutation and crossover) are seen as explorative. The elitism selection method is highly exploitative and can have a negative effect on the genetic diversity. The reason is that the individuals with high fitness will always survive and spread their genes. When there is an individual in the population that is at a local optimum and has higher fitness than the rest of the population, this individual can influence the gene pool so much so that the whole population eventually converges to this local optimum. At the same time, the mating mechanism inspired by the "neat"-algorithm can make the crossover less explorative. The reason is that the high-ranked parents will have the chance to mate more often and therefore produce more offspring than the low-ranked parents. An individual that is at a local optimum will probably also be ranked high on the parent list and will hence be able to produce the maximum number of offspring.

The phenotypic diversity promotion mechanism might be the key process that increases genetic diversity. Even when offspring come from the same parents, they can be very different from each other and from their parents. The effect of this mechanism is clear in section 3.1.9 where the runs with diversity promotion perform much better than the runs without diversity promotion. This is also a clear example of the relationship between the genetic diversity and the quality of the search. Another interesting finding is that mutation normally is the key mechanism to exploration at new regions in the search space, while crossover only explores nearby subspace. In our algorithm, however, a high mutation rate did not increase the genetic diversity more than a low mutation rate. This might indicate that the diversity promotion mechanism makes crossover as explorative as mutation. Additionally, crossover is allowed to occur on n-gram level, which also makes crossover more explorative.

Speciation and niche-formation
Speciation is an important property to the performance of the classification model. The initial idea with speciation is to create an ensemble model which combines decisions from different models that could cover different parts of the dataset. The results from screening where different speciation thresholds are tested do not show whether different numbers of species affect the classification performance. However, the results from testing show that lower speciation thresholds are better than the results from the settings with higher speciation threshold. Out of 10 datasets, the settings with lower speciation threshold (settings 1 & 2) give better classification results on 5 datasets and the same classification results on 3 datasets, compared with the settings with higher speciation threshold (setting 3 & 4).

Moreover, by training different species with different subsets of the training dataset, the execution time can be reduced as the fitness evaluation is faster. This is a great advantage as the execution time can be immense with evolutionary algorithms.

One issue with speciation is the subset of samples used for speciation. This data might not be representative of the whole dataset by being too small or too different from the average of the whole dataset. Another issue with speciation is the formation of "bad" species. "Bad" species are not species with low fitness. They are rather the species that do not cooperate with other species and thereby worsen the overall classification performance. Because of the nature of the evolutionary process, a poor cooperation between species in one generation does not mean a poor cooperation in the next generation and vice versa. It is therefore difficult to detect a species that should be eradicated.

4.2 Future work

As previously mentioned, the overall performance of the proposed algorithm is better than the baseline reaching balanced accuracy scores up to 96%. Even so, there is room for improvement before the algorithm can be successfully applied to real-world problems. A simple method to achieve instant improvement is to integrate an automatic parameter tuner. This way the optimal parameter settings are automatically chosen before running the GP algorithm on new problems or new types of datasets. Using parameter tuners assumes that the parameters are kept constant throughout the run, which is a form of static parameter control. There are other approaches to parameter control as well, such as adaptive or self-adaptive parameter control [60].

In these approaches the parameters adapt or evolve throughout the run. To improve the algorithm further it could be interesting to implement one of these approaches instead of the parameter tuner.

If the results upon such implementation still are not satisfactory a possible direction forward would be to explore further how to use weights for the voting classifier. The test results from modifying the voting classifier with a simple ML model already shows promising results and suggests that such alterations could be useful. Suggested approaches would be to fine-tune the current ML model (a random forest classifier) or explore more complex ML models for instance models that include neural networks. Another possibility is to approach the weighting in an evolutionary fashion. EA:s are rather effective for these kinds of tasks.

Another interesting direction would be to equip the GP algorithm with additional functions. As discussed above, functions with ability to capture the sequential information can improve performance. A function with such ability could for instance be one that checks if certain n-grams appear before some other certain n-grams within a sentence or the entire text. Another example is a function that checks if two or more n-grams appear within a certain distance from each other. Furthermore, the so-called skip-grams can be introduced to the GP algorithm. Skip-grams are essentially n-grams with components that do not have to be consecutive. These have been shown to have some ability to overcome the data sparsity problem, which especially is a large problem with textual data. [61]

Other suggestions are to modify the evolutionary mechanisms in the proposed algorithm, such as the selection of parent individuals. The current selection method is based on pure elitism which has a negative effect on diversity. Thus, fitness sharing between individuals with similar genotype or phenotype could be integrated into parent selection to eliminate this negative effect.

Since the species work together through the voting classifier, something to be considered for future works would be to introduce the concept of coevolution. In biology, coevolution refers to the evolution of two or more species, which occurs due to their effects on each other. In our algorithm, coevolution can be implemented by determining the fitness of each individual based on how well it cooperates with the other species instead, and hence allows for the species to adapt to each other.

On a higher level, an interesting concept to introduce is modularization. In this context, modularization can imply that trees or parts of trees from highly fitted individuals are stored from each run. These trees are added to the function or terminal set for the next run to take advantage of. This enables the system to learn from previous runs, hopefully improving the efficiency of the GP search by proposing some already good solutions.

4.3 Ethical aspects

AI technology raises a number of ethical issues, issues which have been debated ever since the first mention of such technology. One great concern is the job loss that comes with automating work, and the wealth inequality that comes with it. This is a big and complex question. But from a historical view the technological advances, which initially caused severe economic harm to workers who lost their livelihood, have in the long run paved the way to a better standard of living and a more equal world than ever before. The world has never been wealthier and socially more just than it is today. When old jobs disappear new opportunities eventually arise. Historically the shift in jobs has gone from manual labour to more intellectual work. When AI takes over some intellectual tasks and job opportunities, we believe there will be a shift into more creative opportunities. More people might engage in creative work. The issue of distributing the wealth equally is a political concern, and not a technological concern we believe. Technical advances cannot be stopped, it lies in human nature.

In this particular case the subject of automation is research, in particular medical research. Such automation might find life saving discoveries which perhaps could never be found by human researchers. The time it takes to develop a new medicine might also be decreased. Finding new medicine is a long process, usually stretching from 10-20 years. If that time could decrease - the costs of developing it will be drastically decreased and the medicine will be possible to distribute to a much lower cost. In other words, it will be more economically accessible.

A great concern that comes with automating research is incorrect predictions. As mentioned before, many of today's AI models are too complex to interpret. This raises the question of how much power AI should be given to make decisions. In medicine it is of utter importance to base medical decisions on research. For this reason the interpretability of AI models is crucial. There has to be a chance for humans to validate the results. This is something which we have considered in this work by incorporating interpretability into our model from the very beginning.

Chapter 5

Conclusion

In this thesis, we proposed a GP algorithm which is capable of producing text classification models. The classification models are composed of string matchers that are built up of short character n-grams and logical operators. The proposed algorithm fulfills all the intent criteria; it requires no domain-specific knowledge, no manual feature engineering and also provides interpretability in the classification models.

To construct the algorithm, we combined the standard GP-algorithm with inspiration from the ”neat”-algorithm. One important component from the ”neat”-algorithm is speciation. In our algorithm, we perform speciation in order to create different species where each species is expected to cover different variations of the dataset. The representatives from all species are then combined to form an ensemble model to perform classification.

Furthermore, the proposed algorithm has a diversity promotion mechanism, which improves the performance of the algorithm greatly. There is also a bloat control mechanism in the algorithm, which successfully prevents the population from growing uncontrollably.

The performance of the classification models produced by the GP algorithm shows great potential despite inconsistent results between different datasets and different parameter settings. This variation was observed during the parameter screening where we found n-gram length and speciation threshold to be the most influential parameters.

There are numerous possible future directions that can be explored to improve the algorithm. One direction can be to integrate an automatic parameter tuner in the algorithm or to implement adaptive parameters. Further directions can be to implement coevolution and to add functions with ability to capture the sequential information.

References

[1] W. Hu, J. Du, and Y. Xing. (2016). Spam filtering by semantics-based text classification. Presented at Eighth International Conference on Advanced Computational Intelligence. [Online]. Available: https://ieeexplore.ieee.org/document/7449809?arnumber=7449809

[2] A. S. Halibas, A. S. Shaffi, and M. A. K. V. Mohamed. (2018). Application of text classification and clustering of Twitter data for business analytics. Presented at Majan International Conference. [Online]. Available: https://ieeexplore.ieee.org/document/8363162

[3] G. B. Herwanto, A. M. Ningtyas, K. E. Nugraha, and I. N. P. Trisna. (2019). Hate Speech and Abusive Language Classification using fastText. Presented at International Seminar on Research of Information Technology and Intelligent Systems. [Online]. Available: https://ieeexplore.ieee.org/document/9034560?arnumber=9034560

[4] S. Gupta. "Text Classification: Applications and Use Cases." Towards Data Science. https://towardsdatascience.com/text-classification-applications-and-use-cases-beab4bfe2e62 (retrieved Aug. 3, 2020).

[5] J. A. Kumar, and P. A. Silva. (2020). Work-in-Progress: A Preliminary Study on Students' Acceptance of Chatbots for Studio-Based Learning. Presented at IEEE Global Engineering Education Conference. [Online]. Available: https://ieeexplore.ieee.org/document/9125183?arnumber=9125183

[6] A. Følstad, C. B. Nordheim, and C. A. Bjørkli. (2018). What Makes Users Trust a Chatbot for Customer Service? An Exploratory Interview Study. Presented at International Conference on Internet Science. [Online]. Available: https://link.springer.com/chapter/10.1007/978-3-030-01437-7_16

[7] S. Lakhotia, and X. Bresson. (2018). An Experimental Comparison of Text Classification Techniques. Presented at International Conference on Cyberworlds. [Online]. Available: https://ieeexplore.ieee.org/document/8590017?arnumber=8590017

[8] S. Minaee, N. Kalchbrenner, E. Cambria, N. Nikzad, M. Chenaghlu, and J. Gao. "Deep Learning Based Text Classification: A Comprehensive Review." arXiv. https://arxiv.org/pdf/2004.03705.pdf (retrieved Aug. 3, 2020).

[9] A. Shekhar. "What Is Feature Engineering for Machine Learning?" Medium. https://medium.com/mindorks/what-is-feature-engineering-for-machine-learning-d8ba3158d97a (retrieved Aug. 3, 2020).

[10] J. Hussain, "Deep Learning Black Box Problem," M.S. thesis, Department of Informatics and Media, Uppsala University, Uppsala, Sweden, 2019. [Online]. Available: https://www.diva-portal.org/smash/get/diva2:1353609/FULLTEXT01.pdf

[11] C. Rudin. "Stop explaining black box machine learning models for high stakes decisions and use interpretable models instead." nature. https://www.nature.com/articles/s42256-019-0048-x (retrieved Aug. 3, 2020).

[12] mc.ai. "Eight Deep Learning Pros and Cons." https://mc.ai/eight-deep-learning-pros-and-cons/ (retrieved Aug. 3, 2020).

[13] T. Arya. "Drawbacks of Deep Learning." MS&E 238 Blog. https://mse238blog.stanford.edu/2018/07/tanuarya/drawbacks-of-deep-learning/ (retrieved Aug. 3, 2020).

[14] Y. Cheng, D. Wang, P. Zhou, and T. Zhang. "A Survey of Model Compression and Acceleration for Deep Neural Networks." arXiv. https://arxiv.org/pdf/1710.09282.pdf (retrieved Aug. 3, 2020).

[15] P. Shrivastava. "Challenges in Deep Learning." Hackernoon. https://hackernoon.com/challenges-in-deep-learning-57bbf6e73bb (retrieved Aug. 3, 2020).

[16] A. Wissner-Gross. "Datasets Over Algorithms." Edge. https://www.edge.org/response-detail/26587 (retrieved Aug. 3, 2020).

[17] K. D. Jong, D. B. Fogel, and H. Schwefel. "A history of evolutionary computation." Research Gate. https://www.researchgate.net/publication/216300863_A_history_of_evolutionary_computation (retrieved Aug. 3, 2020).

[18] K. L. Ng, "Genetic Programming in Control Theory: On Evolving Programs and Solutions to Control Problems," M.S. thesis, Department of Automatic Control, Lund Institute of Technology, Lund, Sweden, 2000. [Online]. Available: https://lup.lub.lu.se/student-papers/search/publication/8848339

[19] D. Whitley. "An overview of evolutionary algorithms: practical issues and common pitfalls." Science Direct. https://www.sciencedirect.com/science/article/pii/S0950584901001884 (retrieved Aug. 3, 2020).

[20] P. Ball. "Living robots." nature. https://www.nature.com/articles/s41563-020-0627-6 (retrieved Aug. 3, 2020).

[21] S. Kriegman, D. Blackiston, M. Levin, and J. Bongard. "A scalable pipeline for designing reconfigurable organisms." PNAS. https://www.pnas.org/content/117/4/1853 (retrieved Aug. 3, 2020).

[22] M. Lukac, and M. Perkowski. "Evolving quantum circuits using genetic algorithm." Research Gate. https://www.researchgate.net/publication/3965782_quantum_circuits_using_genetic_algorithm (retrieved Aug. 3, 2020).

[23] A. Băutu, and E. Bautu. "Quantum Circuit Design by Means of Genetic Programming." Research Gate. https://www.researchgate.net/publication/228530934_Quantum_Circuit_Design_by_Means_of_Genetic_Programming (retrieved Aug. 3, 2020).

[24] P. Benioff. "The computer as a physical system: A microscopic quantum mechanical Hamiltonian model of computers as represented by Turing machines." SpringerLink. https://link.springer.com/article/10.1007%2FBF01011339 (retrieved Aug. 3, 2020).

[25] A. Cho. "The biggest flipping challenge in quantum computing." Science. https://www.sciencemag.org/news/2020/07/biggest-flipping-challenge-quantum-computing (retrieved Aug. 3, 2020).

[26] C. Darwin, *On the Origin of species by means of natural selection.* London, England: John Murray, 1859.

[27] J.R. Koza, *Genetic Programming: On the Programming of Computers by Means of Natural Selection.* Cambridge, MA, USA: MIT Press, 1992.

[28] H. Chial. "DNA Sequencing Technologies Key to the Human Genome Project." nature. https://www.nature.com/scitable/topicpage/dna-sequencing-technologies-key-to-the-human-828 (retrieved Aug. 3, 2020).

[29] E. G. Shopova, and N. G. Vaklieva-Bancheva. "BASIC—A genetic algorithm for engineering problems solution." ScienceDirect. https://www.sciencedirect.com/science/article/pii/S009813540600055X (retrieved Aug. 3, 2020).

[30] J. F. Miller, *Cartesian Genetic Programming.* Germany: Springer-Verlag Berlin Heidelberg, 2011.

[31] E. Stoltz. "Evolution of a salesman: A complete genetic algorithm tutorial for Python." Towards Data Science. https://towardsdatascience.com/evolution-of-a-salesman-a-complete-genetic-algorithm-tutorial-for-python-6fe5d2b3ca35 (retrieved Aug. 3, 2020).

[32] R. Poli. "A Genetic Programming Tutorial." Research Gate. https://www.researchgate.net/publication/2415604_A_Genetic_Programming_Tutorial (retrieved Aug. 3, 2020).

[33] M. Affenzeller, S. Wagner, S. Winkler, and A. Beham. *Genetic Algorithms and Genetic Programming: Modern Concepts and Practical Applications.* CRC Press, 2009.

[34] W. B. Langdon, and R. Poli. *Foundations of Genetic Programming.* Germany: Springer-Verlag Berlin Heidelberg, 2002.

[35] S. Gustafson. "An Analysis of Diversity in Genetic Programming." Research Gate. https://www.researchgate.net/publication/37245549_An_Analysis_of_ Diversity_in_ Genetic_Programming (retrieved Aug. 3, 2020).

[36] M. Črepinšek, S. Liu, and M. Mernik. "Exploration and Exploitation in Evolutionary Algorithms: A Survey." Research Gate. www.researchgate.net/publication/243055445_Exploration_and_Exploitation_in_ Evolutionary_Algorithms_A_Survey (retrieved Aug. 3, 2020).

[37] A. Purohit, and N. S. Choudhari. "Code Bloat Problem in Genetic Programming." Semantic Scholar. https://www.semanticscholar.org/paper/Code-Bloat-Problem-in-Genetic-Programming-Purohit-Choudhari/a5005fcd425ff7123550df171c1cbadccfe14683 (retrieved Aug. 3, 2020).

[38] P. Gibbs. "What is Occam's Razor?" The Physics and Relativity FAQ. http://math.ucr.edu/home/baez/physics/General/occam.html (retrieved Aug. 3, 2020).

[39] MonkeyLearn. "What is Text Classification?" https://monkeylearn.com/what-is-text-classification/ (retrieved Aug. 3, 2020).

[40] M. Zaveri. "Automatic Text Classification: A Technical Review." Research Gate. https://www.researchgate.net/publication/266296879_Automatic_Text_ Classification _A_Technical_Review (retrieved Aug. 3, 2020).

[41] K. Ganesan. "All you need to know about text preprocessing for NLP and Machine Learning." Towards Data Science. https://towardsdatascience.com/all-you-need-to-know-about-text-preprocessing-for-nlp-and-machine-learning-bc1c5765ff67 (retrieved Aug. 3, 2020).

[42] Y. HaCohen-Kerner, D. Miller, and Y. Yigal. "The influence of preprocessing on text classification using a bag-of-words representation." Plos One. https://journals.plos.org/plosone/article?id=10.1371/journal.pone.0232525 (retrieved Aug. 3, 2020).

[43] NSS. "An Intuitive Understanding of Word Embeddings: From Count Vectors to Word2Vec." Analytics Vidha. https://www.analyticsvidhya.com/blog/2017/06/word-embeddings-count-word2veec (retrieved Aug. 4, 2020).

[44] H. E. Boukkouri. "Arithmetic Properties of Word Embeddings." Medium. https://medium.com/data-from-the-trenches/arithmetic-properties-of-word-embeddings-e918e3fda2ac (retrieved Aug. 4, 2020).

[45] S. Bansal. "A Comprehensive Guide to Understand and Implement Text Classification in Python." Analytics Vidha. https://www.analyticsvidhya.com/blog/2018/04/a-comprehensive-guide-to-understand-and-implement-text-classification-in-python (retrieved Aug. 4, 2020).

[46] L. Trujillo, L. Muñoza, E. Galván-López, and S. Silva. "neat Genetic Programming: Controlling bloat naturally." Science Direct. https://www.sciencedirect.com/science/article/pii/S0020025515008038 (retrieved Aug. 4, 2020).

[47] *DEAP*. (2020). [Online]. Available: https://deap.readthedocs.io/en/master/

[48] W. Cavnar, and J. M. Trenkle. "N-Gram-Based Text Categorization." Research Gate. https://www.researchgate.net/publication/2375544_N-Gram-Based_Text_Categorization (retrieved Aug. 4, 2020).

[49] C. Taylor. "What Is Bootstrapping in Statistics?" ThoughtCo. https://www.thoughtco.com/what-is-bootstrapping-in-statistics-3126172 (retrieved Aug. 4, 2020).

[50] Y. W. Foo, C. Goh, and Y. Li. 2016. Speciation and diversity balance for Genetic Algorithms and application to structural neural network learning. Presented at International Joint Conference on Neural Networks. [Online]. Available: https://ieeexplore.ieee.org/document/7727345

[51] K. Teknomo. "Similarity Measurement." Revoledu.com. https://people.revoledu.com/kardi/tutorial/Similarity/EuclideanDistance.html (retrieved Aug. 4, 2020).

[52] Khan Academy. "Species & speciation". https://www.khanacademy.org/science/ biology/her/tree-of-life/a/species-speciation (retrieved Aug. 4, 2020).

[53] D. M. V. Powers. "What the F-measure doesn't measure: Features, Flaws, Fallacies and Fixes." arXiv. https://arxiv.org/abs/1503.06410 (retrieved Aug. 4, 2020).

[54] V. Krakovna, J. Uesato, V. Mikulik, M. Rahtz, T. Everitt, R. Kumar, Z. Kenton, J. Leike, and S. Legg. "Specification gaming: the flip side of AI ingenuity." Deep Mind. https://deepmind.com/blog/article/Specification-gaming-the-flip-side-of-AI-ingenuity (retrieved Aug. 4, 2020).

[55] D. Jackson. 2010. Promoting Phenotypic Diversity in Genetic Programming. Presented at International Conference on Parallel Problem Solving from Nature. [Online]. Available: https://link.springer.com/chapter/10.1007/978-3-642-15871-1_48

[56] S. Mangale. "Voting Classifier." Medium. https://medium.com/ @sanchitamangale12/voting-classifier-1be10db6d7a5 (retrieved Aug. 4, 2020).

[57] R. Silipo, and M. Lisovyi. "Machine learning algorithms and the art of hyperparameter selection." Knime. https://www.knime.com/blog/machine-learning-algorithms-and-the-art-of-hyperparameter-selection (retrieved Aug. 4, 2020).

[58] D. Shulga. "5 Reasons why you should use Cross-Validation in your Data Science Projects." Towards Data Science. https://towardsdatascience.com/5-reasons-why-you-should-use-cross-validation-in-your-data-science-project-8163311a1e79 (retrieved Aug. 4, 2020).

[59] Statistical Odds & Ends. ”What is balanced accuracy?” https://statisticaloddsandends.wordpress.com/2020/01/23/what-is-balanced-accuracy/ (retrieved Aug. 4, 2020).

[60] M. Gregor and J. Spalek. 2011. Fitness-based Adaptive Control of Parameters in Genetic Programming: Adaptive Value Setting of Mutation Rate and Flood Mechanisms. Presented at International Conference on Intelligent Computing and Intelligent Systems. [Online]. Available: https://arxiv.org/pdf/1605.01514.pdf

[61] D. Guthrie, B. Allison, W. Liu, and L. Guthrie. 2006. A Closer Look at Skip-gram Modelling. Presented at Fifth International Conference on Language Resources and Evaluation. [Online]. Available: ht-tps://www.researchgate.net/publication/266863668_A_Closer_Look_at_Skip-gram_Modelling