Evolving kernel functions for Support Vector Machines classification:

A Genetic Programming Approach

Padraig Leavey

A thesis submitted to University College Dublin in part fulfillment of the requirements of the degree of Master of Science in Business Analytics

Michael Smurfit Graduate School of Business,

University College Dublin

October 2014

Supervisors

Dr Séan McGarraghy Dr James McDermott

Dean of School: Professor Tom Begley

Table of Contents

Acknowledgements

I would like to thank Dr Séan McGarraghy and Dr James McDermott for the help and support they have given me during the course and completion of this thesis.

Abstract

Support Vector Machines (SVM) are a state-of-the-art classification and regression technique that have shown promising results in a wide number of application areas. Genetic Programming (GP) is an evolutionary search heuristic that solves machine-representable problems using analogues of natural selection. This project is an attempt to minimise the knowledge overhead required to successfully employ SVM in classification problems by using GP to automate the performance-critical process of kernel selection. Two types of commonly-used GP search heuristics, *TinyGp* and *ParetoGp*, are compared for the kernel evolution process. The results are tested on a number of data-sets sourced from the machine learning repositories maintained by the University of California at Irvine.

1 Introduction

The goal of this project is to demonstrate the ability to evolve combination kernel functions for Support Vector Machines (SVM) classifiers from a set of base kernels using Genetic Programming (GP), and to demonstrate that superior binary classification accuracy can be achieved on test data using the combined kernels than can be achieved using the individual base kernels. Therefore, this project is an attempt to hybridise two state-of-the-art machine learning techniques, SVM and GP, to improve outcomes in binary classification tasks. Two GP search heuristics, Tiny GP and Pareto-front GP, are compared for this project.

1.1 Business Contribution

In recent decades, due to the growth of the processing power of computers and the ability to store more and more data in digital format, businesses have become acutely interested in what is commonly called ‘Big Data’. ‘Analytics’ has very much become a buzz-word in business, and rightly so. The strive to derive meaningful, actionable insights from the vast quantities of data that most large companies collect, often incidentally, has become more and more urgent. Business leaders have recognised that making decisions based on intelligence derived from this data, rather than hunches or gut feelings, can lead to better outcomes.

But analytics itself can often seem an impenetrable and arcane world to those not trained in its intricacies. Though this project does not aim to emulate something as easy to use as IBM claim their Watson Analytics tool to be, it aims to make the use of the SVM algorithm more user-friendly.

When employing SVM, one must take account of many factors to optimise its performance. Some of these, such as SVM parameters like the penalty function, *C*, or the choice of kernel parameters, are not considered in this project. But a key element that determines the level of performance of the SVM is the choice of kernel function. Without domain expert knowledge this can be a trial and error process and may lead to unsatisfactory and possibly even misleading outcomes.

This projects aims to automate this process by providing a method that finds a near-optimal choice of kernel for the user. It does this by taking a library of base kernels and combining them in ways that produce new, valid kernel functions. It tests all the combinations in SVM classification of the data-set under investigation and returns to the user the one that has performed the best. In this way, the amount of time and expert knowledge to be invested is reduced and the user can proceed with confidence that the kernel function they are using is the best out of a large number of possibilities.

1.2 Academic Contribution

Several previous studies [1], [2] have looked at using Genetic Programming to evolve multiple kernels for Support Vector Machine classification. This project also aims to evolve kernels for SVM classification. Its primary academic contribution is the comparison of two different GP search heuristics, Tiny GP and Pareto-front GP. Details of these techniques are given in Section 2.

References

[1] Sullivan, Keith M., and Sean Luke. "Evolving kernels for support vector machine classification." In *Proceedings of the 9th annual conference on Genetic and evolutionary computation*, pp. 1702-1707. ACM, 2007.

[2] Dioşan, Laura, Alexandrina Rogozan, and Jean-Pierre Pecuchet. "Learning SVM with Complex Multiple Kernels Evolved by Genetic Programming."*International Journal on Artificial Intelligence Tools* 19, no. 05 (2010): 647-677.

2 Background and Literature Review

2.1 Support Vector Machines

Support Vector Machines (SVM) are a state-of-the-art supervised learning model used for classification of data. They were introduced by Boser et al in 1992 [1].They have proved particularly useful in text and image processing applications and have been used for pattern recognition tasks in areas as diverse as bioinformatics and finance.

2.1.1 Kernel Methods

SVM belong to the general class of kernel methods, a class of algorithm that makes use of the computational efficiency of kernel functions. A kernel function is a positive definite, symmetric, real-valued function of two arguments and is defined as a function k that for all $x, z \in X$ satisifies

$$k(x,z) = \langle \Phi(x), \Phi(z) \rangle$$

where Φ is a mapping from X to a (inner product) feature space F

$$\Phi : \mathrm{x} \longmapsto \Phi(\mathrm{x}) \in F$$

A kernel function may be thought of as the measure of similarity between two objects, with the real-valued number it returns giving the degree of similarity.

Four key aspects of the kernel method approach are [2]:

1. Data items are embedded in a vector space called the feature space.
2. Linear relations are sought among the images of the data items in the feature space.
3. The kernel based learning algorithms are implemented in such a way that they do not require the coordinates of the data items in the feature space; they instead merely require pairwise inner products.
4. The kernel function efficiently computes the pairwise inner products from the original data items.

This computational shortcut is known as the kernel trick. The feature space may be of a very high dimension. Computing the feature space coordinates explicitly could become computationally very costly. A kernel function allows us to avoid this cost by implicitly mapping the data items and computing only the inner products.

2.1.2 Support Vector Machines

Support Vector Machines (SVM) are a kernel-based learning algorithm used for classification and regression. They based on statistical learning theory. For classification tasks, as in this project, SVM build a decision function based on training and use this decision function as the class boundary when classifying unseen data.

Mathematically the decision function is the optimal decision surface and can be shown as:

$$sgn\big(f(x)\big) = f(x) = \sum_{i,j=1}^{l} y_i \alpha_i K\big(x_i, x_j\big) + b$$

They are characterised by the number of support vectors. The SVM's task is to maximise the distance between support vectors that lie on either side of the decision boundary.

2.1.3 The Kernel Matrix

Given a kernel $k$and inputs $x_1, \dots, x_n \in X$, the n x n matrix

$$K := \left(k(x_i, x_j)\right)_{i,j}$$

is called the Gram matrix, or kernel matrix, of k with respect to $x_1, \dots, x_n$. [3]. It is the evaluation of the kernel function on all the pairs of data points, and contains all the information about the data that is required by the learning algorithm. The kernel matrix plays a key role in this project, as it is evaluated for all the base kernels for the data-sets under investigations and the resulting set of kernel matrices are used as inputs for the GP component of the algorithm.

2.1.4 Kernel Closure

Kernel functions allow us to compute inner products in higher dimensional feature spaces without having to explicitly map the data items to the feature space. New kernel functions can be obtained by manipulating known kernel functions with appropriate mathematical operators. From [4]:

Let k_1 and k_2 be kernels over $X \times X, X \subseteq \mathbb{R}, \; a \in \mathbb{R}^+$. Then the following functions are kernels:

$(a) k(x,z) = k_1(x,z) + k_2(x,z)$

$(b) k(x,z) = a * k_1(x,z)$

$(c) k(x,z) = k_1(x,z) * k_2(x,z)$

$(d) k(x,z) = e^{k_1(x,z)}$

These properties of kernels are used to guide the combinations that are allowed as part of the GP runs.

2.2 Genetic Programming

2.2.1 Introduction: Evolutionary Algorithms

Evolutionary Algorithms (EAs) are a family of computer-based problem-solving systems that use computational models of some of the known mechanisms of evolution as key elements in their design and implementation. Many computational problems require searching through a vast number of possible solutions. This is analogous to the process of evolution, where the possible solutions are the possible genetic sequences and the desired solutions are highly-fit individuals who are well-adapted to surviving and reproducing [5]. In this way, evolution can be seen as a method for designing innovative solutions to complex problems, and the mechanisms of evolutions seem well-suited to solving many computational problems

The features shared by all EAs are given as the following [6]:

(i) The use of a population of individuals as candidate solutions.

(ii) An inheritance method in the form of genetic operators that are applied to the individuals of a population in order to create the next generation of individuals. The two main operators are crossover and mutation.

(iii) A fitness function to measure the quality of an individual as a candidate solution. The better the fitness of an individual, the higher the probability of that individual being used in the breeding of the next generation.

2.2.2 Genetic Programming

Genetic Programming (GP) was introduced by Koza in his 1992 book *Genetic Programming: On the Programming of Computers by Means of Natural Selection* [7]. In it he describes the problem of program induction: the inductive discovery, from the space of possible computer programs, of a computer program that produces some desired output when presented with some particular input.

Typically, GP solutions are represented as parse trees. Prior to beginning a GP programming run, the inputs are selected in the form of a function set and a terminal set. A function set may include mathematical operators such as {+, *, sin, cos, exp}. A terminal set may include constants and variables. In the parse tree representation all nodes have a parent node, except the root node. The number of children each parent has is called the arity of the parent node. Nodes that have one or more children are functions, and nodes that have no children (arity of zero) are the terminal nodes.

An example of an GP parse tree is shown in Figure 1 below. This tree represents the mathematical expression 3+2*x. '+' and '*' are functions. Each takes two arguments and thus have arity 2. '3', '2' and 'x' are terminals and have arity 0.

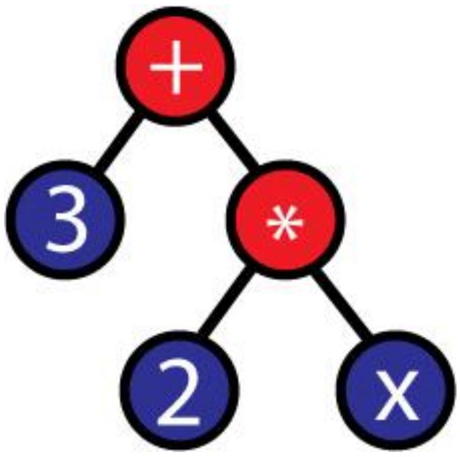

Figure 1: A GP parse tree

(source: http://geneticprogramming.us/Representation.html; accessed 27/09/14)

When implementing a GP run, the function and terminal set must be selected so that they satisfy the conditions of closure and sufficiency.

The closure property states that each function in the function set should be able to accept as its argument any value and data type that it may encounter. For example the function set *F* = {+, *, sqrt} and the terminal set *T* = {x, 5, 2} satisfy the closure property. The function set *F* = {+, *, sqrt} and the terminal set *T* = {x, 5, -2} do not satisfy the closure property, as the 'sqrt' function will be unable to handle the negative number, '-2'.

The sufficiency property requires that the function and terminal sets should have sufficient expressive power to represent the solution to the problem.

Recombination of GP individuals is carried out by two primary operators: crossover and mutation.

The most important is the crossover operation. Crossover involves two individuals and combines genetic material from both parents to form a new individual. The most common form of crossover operation is sub-tree crossover.

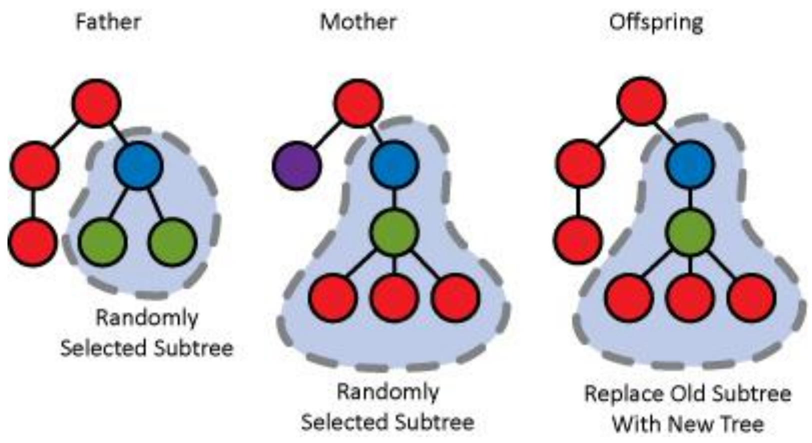
Father
Mother
Offspring
Randomly Selected Subtree
Randomly Selected Subtree
Replace Old Subtree With New Tree

Figure 2: Sub-tree crossover

(source: http://geneticprogramming.us/Genetic_Operations.html; accessed 27/09/14)

Mutation is the less important of the two genetic operators. It is used less as mutation events often decrease the fitness of an individual. However, it is still an important part of a GP run as it tends to increase diversity in the gene pool. Mutation operates on a single individual from the population, randomly replacing a node with another node of the same arity and return type, or randomly generating a new sub-tree and swapping it into the individual at a chosen mutation point.

2.2.3 *TinyGp*

TinyGp is a highly optimised GP system that was developed by Riccardo Poli in 2004 to demonstrate that GP software need not be big, complicated and difficult to use [8]. Its main characteristic is that it is steady-state. A generation is deemed completed when a number of crossover and mutation events equivalent to the population size have occurred. It uses tournament selection, sub-tree crossover and point mutation.

2.2.4 *ParetoGp*

ParetoGp is a GP variant put forward by Smits and Kotanchek in 2005 [9]. It seeks to exploit the Pareto front, an idea from multi-objective optimisation. In this variant the Pareto front represents the best individuals in the population and these are focussed on for genetic operations.

2.3 SVM kernel evolution by means of GP

A number of previous studies have experimented with kernel evolution by GP.

Howley and Madden [10] used GP to evolve kernel functions using an input variable of three kernel functions: polynomial, radial basis function (RBF) and sigmoid. The used a fitness function based on two factors: the classification accuracy on the training set and the sum of the support vector values. They did not use the returned kernel combinations to classify a test set of unseen data due to the computational cost involved.

Sullivan and Luke [4] used strongly-typed genetic programming and kernel closure properties. They used *k*-fold cross-validation in their fitness function. They also reported high computational costs with their technique.

Phienthrakul and Kijsirikul [11] used crossover and mutation operations to evolve kernel functions in the form of trees with adjustable kernel parameters. In their fitness function they used the bound of the generalisation error to evaluate the hybrid kernels.

Girdea and Ciortuz [12] used GP to evolve kernel combinations by addition and multiplication, and used training data management techniques inspired by the InfoBoost algorithm to focus on the most difficult data instances. They compared the boosted hybrid with kernel evolved by GP and concluded that the boosted kernels were superior.

Diosan et al [13] used a steady-state GP approach to evolve kernel combinations. Their fitness function was the classification accuracy of the combined kernels on test sets of unseen data.

References

[1] Boser, Bernhard E., Isabelle M. Guyon, and Vladimir N. Vapnik. "A training algorithm for optimal margin classifiers." In *Proceedings of the fifth annual workshop on Computational learning theory*, pp. 144-152. ACM, 1992.

[2] Shawe-Taylor, John, and Nello Cristianini. *Kernel methods for pattern analysis*. Cambridge university press, 2004.

[3] Hofmann, Thomas, Bernhard Schölkopf, and Alexander J. Smola. "Kernel methods in machine learning." *The annals of statistics* (2008): 1171-1220.

[4] Sullivan, Keith M., and Sean Luke. "Evolving kernels for support vector machine classification." In *Proceedings of the 9th annual conference on Genetic and evolutionary computation*, pp. 1702-1707. ACM, 2007

[5] Mitchell, Melanie. *An introduction to genetic algorithms*. MIT press, 1998.

[6] Espejo, Pedro G., Sebastián Ventura, and Francisco Herrera. "A survey on the application of genetic programming to classification." *Systems, Man, and Cybernetics, Part C: Applications and Reviews, IEEE Transactions on* 40, no. 2 (2010): 121-144.

[7] Koza, John R. *Genetic programming: on the programming of computers by means of natural selection*. Vol. 1. MIT press, 1992.

[8]Poli, Riccardo, William B. Langdon, Nicholas F. McPhee, and John R. Koza. *A field guide to genetic programming*. Lulu. com, 2008.

[9] Smits, Guido F., and Mark Kotanchek. "Pareto-front exploitation in symbolic regression." In *Genetic programming theory and practice II*, pp. 283-299. Springer US, 2005

[10] Howley, Tom, and Michael G. Madden. "The genetic kernel support vector machine: Description and evaluation." *Artificial Intelligence Review* 24, no. 3-4 (2005): 379-395.

[11] Phienthrakul, Tanasanee, and Boonserm Kijsirikul. "GPES: An algorithm for evolving hybrid kernel functions of support vector machines." In *Evolutionary Computation, 2007. CEC 2007. IEEE Congress on*, pp. 2636-2643. IEEE, 2007

[12] Gîrdea, Marta, and Liviu Ciortuz. "A Hybrid Genetic Programming and Boosting Technique for Learning Kernel Functions from Training Data." In *SYNASC*, pp. 395-402. 2007.

[13] Dioşan, Laura, Alexandrina Rogozan, and Jean-Pierre Pecuchet. "Learning SVM with Complex Multiple Kernels Evolved by Genetic Programming."*International Journal on Artificial Intelligence Tools* 19, no. 05 (2010): 647-677.

3 Method

The core of the method is the calculation of the kernel matrix for each of the eight base kernels on the data-set under investigation. This matrix, also called the Gram matrix, contains the evaluation of the kernel function on all of the input data points.

A high level overview of the program is as follows:

1.) Read in the data-set and save it in matrix format.
2.) Compute the value of each of the eight base kernels on the data matrix. These are saved as the kernel matrices.
3.) The kernel matrices are passed to the Genetic Programming part of the code as the input variable set.
4.) The kernel matrices are combined by addition, multiplication and exponentiation operators. Random constants are drawn from a normal distribution and used as scalars in the combining process.
5.) The combined kernels are checked that they are valid kernel matrices i.e. symmetric and positive semi-definite matrices.
6.) Valid kernel matrices are used for classification in the GP fitness function.
7.) The kernel combination with the best classification accuracy on the test data is returned as the output of that GP run.

3.1 The Kernlab Package

Kernlab is an R package that includes implementation of kernel-based machine learning methods, including Support Vector Machines (SVM).

When using its SVM functionality, the R Kernlab package allows the user to either specify the kernel function from a list of eight base kernels or to provide the pre-computed kernel matrix. For this project, the decision was made to use the pre-computed kernel matrices.

The eight base kernel functions provided by Kernlab are:

- Linear (vanilladot): $k(x,x') = \langle x,x' \rangle$
- Radial Basis Function (rbfdot): $k(x,x') = \exp(-\|x-x'\|^2)$
- Polynomial (polydot): $k(x,x') = (scale.\langle x,x' \rangle + offset)^{degree}$
- Sigmoid (tanhdot): $k(x,x') = tanh(scale.\langle x,x' \rangle + offset)$
- Bessel (besseldot): $k(x,x') = \frac{Bessel_{(v+1)}^{n}(\sigma\|x-x'\|)}{(\|x-x\prime\|)^{-n(v+1)}}$
- ANOVA (anovadot): $k(x,x') = \left(\sum_{k=1}^{n} \exp(-\sigma(x^k - x'^k)^2)\right)^d$
- Laplace (laplacedot): $k(x,x') = \exp(-\sigma\|x-x'\|)$
- Spline (splinedot): $k(x,x') = 1 + xx' + xx' \min(x,x') - \frac{x+x'}{2} min(x,x')^2 + \frac{\min(x,x')^3}{3}$

In this project, for seven of the eight kernel functions the kernel parameters were left at their default values. One implication of this, which did not become clear to the author until some time into the testing phase of the project, is that the Linear and Polynomial kernel functions are equivalent. The Polynomial kernel function takes three arguments: degree, scale and offset. Leaving these at their default of 1, 1 and 1 means that it is functionally identical to the Linear kernel.

The only base kernel that did not have its parameters left at their default value was the Bessel function. During the initial testing phase it was found to frequently error by failing to find any support vectors. Therefore its degree parameter was changed to 2 from its default value of 1. This eliminated the errors. This change to the Bessel function's degree parameter was used for all experiments in the project.

Once the kernel matrices have been computed for each of the base kernels on the input data-set, they are passed to the genetic programming part of the code.

3.2 RGP package

RGP is a genetic programming package integrated into the R environment. The system implements classical tree-based genetic programming, as well as variants including Pareto genetic programming.

Its core components are a function set, a variable set, a constant set and a fitness function. In this project, these are defined as follows:

Input function set: addition (+), multiplication (*) and exponentiation (exp).

Input variable set: The eight base kernel matrices pre-computed on the data-set under investigation, Kvanilla (linear), Krbf (radial basis function), Kpoly (polynomial), Ktanh (sigmoid), Kbessel (Bessel), Kanova (ANOVA), Klaplace (Laplace) and Kspline (spline).

Input constant set: Random numbers drawn from a normal distribution with mean 0 and standard deviation 1.

Fitness function: The classification accuracy on the test data of the individual (kernel combination) produced by the GP run.

RGP allows the user to control parameters such as initial population size, number of evolution steps (crossover and mutation events) and the maximum tree depth of new individuals. It also allows users to choose the search heuristic used in the GP runs. Two heuristics were used in this project: *makeTinyGpSearchHeuristic* and *makeAgeFitnessComplexityParetoGpSearchHeuristic*. The results are compared in Section 5.

3.3 Program

The program executes the following steps:

1.) The data-set under investigation is imported as a .csv file using the *read.csv* function. This function saves the data in a data frame.
2.) The class labels are extracted and saved in a data frame.
3.) Both the data and class labels data frames are converted to matrices.
4.) The data is scaled to be in the range 0 to 1. This step is optional and is dependent on the characteristics of the input data-set.
5.) The eight base kernel matrices are computed.
6.) The GP input function set is defined.

7.) The GP input variable set is defined.

8.) The GP input constant set is defined.

9.) The GP population size is defined.

10.) The maximum tree depth for individuals is defined.

11.) The function, variable and constant sets, the population size and the maximum tree depth are passed to the *geneticProgramming* function. In this function the search heuristic to be used is also defined, as is the stopping condition. In this project the stopping condition is a defined number of evolution steps. Each search heuristic has a default crossover probability. These defaults were not changed for this project. See Section 4 for the details for each experiment.

12.) The fitness function:
Once an individual has been produced it is passed to the fitness function.

- i) The fitness function checks that the individual is a matrix. This is to rule out the possibility that the individual is a scalar. If it passes this check it is assigned as the combined kernel matrix.
- ii) The combined kernel matrix is checked for 'NaN' values.
- iii) It is checked for positive and negative infinity values.
- iv) A check that the combined kernel matrix is symmetric is performed.
- v) A check that the combined kernel is positive semi-definite is performed.
- vi) If the combined kernel fails any of these checks a fitness value of 1 is returned. The fitness function seeks to minimise the value of the function, so 1 is the least fit value it can return.
- vii) If it passes all these checks, it is passed to *ksvm,* Kernlab's SVM function, trained on the training data and used to classify the test data.
- viii) The fitness function returns a value of 1-classification accuracy.

13.) The best individual, i.e. the kernel combination with the best classification accuracy, produced by the GP run is returned.

4 Results and Analysis

Three data-sets from the machine learning repositories maintained by the University of California at Irvine (http://archive.ics.edu/ml/) were used to test the algorithm developed for this project: Wisconsin breast cancer data-set, Pima Indians diabetes data-set and the Ionosphere data-set.

4.1 Wisconsin breast cancer data

The Wisconsin breast cancer data-set contains information on the characteristics of cell nuclei extracted from breast masses. The data-set consists of 569 instances and 32 attributes. The first attribute is the ID number. The second is the diagnosis, either benign or malignant. These are the binary class labels. The remaining 30 attributes contain ten real-valued features computed for each cell nucleus, the standard error calculated for each feature and the maximum value recorded for each feature. The data-set and its information can be found at https://archive.ics.uci.edu/ml/datasets/Breast+Cancer+Wisconsin+(Diagnostic).

The data was saved as a .csv file and imported into R using the *read.csv* function. The class labels were extracted and the remaining 31 attributes were scaled in the range 0 to 1. The decision was made to scale the data to avoid very large and very small values in the kernel matrices. Such values introduce a risk of rounding errors and the possibility of infinite values following certain matrix operations. The data was then partitioned into train and test sets. The train set contained 399 instances and the test set contained 170 instances, a split of approximately 70:30.

4.1.1 Base kernels

The classification accuracies of each of the kernlab's base kernels on the test set are shown in the Table 4.1 below. Also shown is whether or not the kernel matrix is symmetric and PSD. As discussed in Section 3, these properties have implications for how the matrices are used by the algorithm.

Base Kernel	**Classification Accuracy**	**Symmetric**	**PSD**
Linear (Kvanilla)	0.9647059	Yes	Yes
RBF (Krbf)	0.9764706	No	N/A
Polynomial (Kpoly)	0.9647059	Yes	Yes
Sigmoid (Ktanh)	0.1764706	Yes	No
Bessel (Kbessel)	0.9882353	Yes	No
ANOVA (Kanova)	0.9823529	Yes	Yes
Laplace (Klaplace)	0.9882353	Yes	Yes
Spline (Kspline)	0.9352941	Yes	Yes

Table 4.1: Base kernel test set classification accuracy and matrix properties

4.1.2 *TinyGp*

The first GP experiment used the *makeTinyGpSearchHeuristic*. Thirty GP runs were performed, taking a total time of 4.30 hours. The following GP parameters were used:

- Population Size: 80
- Maximum Tree Depth: 3
- Step Limit: 801
- Crossover Probability: 0.9 (*TinyGp* default value)

Table 4.2 below shows the results of the 30 *TinyGp* runs, showing the kernel combination that resulted in the best classification accuracy on the test set, and its classification accuracy.

Run Number	**Kernel Combination**	**Classification Accuracy**
1	Klaplace	0.9882353
2	Kanova * Kbessel + (Kvanilla + Klaplace)	0.9882353
3	Kvanilla + Kvanilla * Kbessel	0.9941176
4	Kpoly * Ktanh * Kbessel	0.9941176
5	Klaplace + Kpoly	0.9882353
6	Kbessel * Kpoly + Kbessel	0.9941176
7	Klaplace	0.9882353
8	Kpoly + Klaplace	0.9882353
9	Klaplace + Ktanh * 0.273859406859867	0.9882353
10	Kvanilla * Kbessel + Kanova	0.9941176
11	Klaplace	0.9882353
12	Klaplace	0.9882353
13	Klaplace	0.9882353
14	Kpoly * Kbessel + Kbessel + Kbessel	0.9941176
15	Klaplace	0.9882353
16	Klaplace	0.9882353
17	Klaplace + Kanova	0.9882353
18	Klaplace + 0.0589412450068675	0.9882353
19	Klaplace	0.9882353
20	Klaplace * 0.532829703426835 + Kbessel * Kanova	0.9941176
21	Klaplace	0.9882353
22	Kpoly * Kbessel	0.9941176
23	Klaplace	0.9882353
24	Klaplace	0.9882353
25	Klaplace	0.9882353
26	Klaplace	0.9882353
27	Kbessel * Kpoly	0.9941176
28	Kpoly + (-0.537499308378501 + Klaplace + Kanova + Kpoly)	0.9882353
29	Klaplace	0.9882353
30	Klaplace	0.9882353

Table 4.2: *TinyGp* best kernel combinations and classification accuracies

The average classification accuracy of the kernel combinations returned by *TinyGp* runs is 0.989803913. As can be seen from comparison with Table 4.1, none of the base kernels match this level of classification accuracy, with the closest being the Bessel and Laplace kernels that both have an accuracy of 0.9882353. The maximum classification accuracy returned by the GP runs was 0.9941176, which was returned by a number of different kernel combinations.

The *TinyGp* runs found combinations that improved on the base kernels in 7 of the 30 runs. On the remaining 23 occasions it either returned the best performing base kernel (Laplace; the Bessel kernel is not returned as its kernel matrix is not PSD and, as discussed in Section 3, non-PSD kernel matrices are not used for classification in the fitness function) or a kernel combination that matched the classification accuracy of the best performing base kernel.

4.1.3 *ParetoGp*

The second experiment used the *makeAgeFitnessComplexityParetoGpSearchHeuristic*. Thirty GP runs were performed with the following parameters.

- Population Size: 80
- Maximum Tree Depth: 3
- Step Limit: 21 (equivalent to 20 generations)
- Crossover Probability: 0.5 (*ParetoGp* default value)

Table 4.3 below shows results of the 30 *ParetoGp* runs, showing the kernel combination that resulted in the best classification accuracy, and the classification accuracy.

Run Number	Kernel Combination	Classification Accuracy
1	Kbessel * Kpoly	0.9941176
2	-0.326383641949083 * Kbessel + Kpoly * Kbessel	0.9941176
3	Kpoly * Kbessel	0.9941176
4	Klaplace	0.9882353
5	Kpoly * Kbessel	0.9941176
6	Klaplace	0.9882353
7	Klaplace	0.9882353
8	Klaplace	0.9882353
9	Klaplace	0.9882353
10	Klaplace	0.9882353
11	Kbessel * Kpoly + exp(1.11795959156545)	0.9941176
12	Kpoly * Kbessel	0.9941176
13	Kbessel + -0.035887941893063 + Kvanilla * Kbessel	0.9941176
14	Ktanh * Kbessel * (0.238193182989636 + Kpoly)	0.9941176
15	Kpoly * Kbessel + 0.567201535634003 * Kvanilla	0.9941176
16	Klaplace	0.9882353
17	Kbessel * Kpoly	0.9941176
18	Klaplace * Kspline + Kpoly * Kanova	0.9941176
19	Klaplace	0.9882353
20	Klaplace	0.9882353
21	Kvanilla * Kbessel + 1.33074709459145 * Kpoly	0.9941176
22	Klaplace	0.9882353
23	Klaplace	0.9882353
24	Klaplace	0.9882353
25	Klaplace	0.9882353
26	Klaplace	0.9882353
27	Klaplace	0.9882353
28	Kpoly * Kbessel + exp(-1.23615299497577)	0.9941176
29	Klaplace	0.9882353
30	Kbessel * Kpoly	0.9941176

Table 4.3: *ParetoGp* best kernel combinations and classification accuracies

The average classification accuracy of the thirty *ParetoGp* runs was 0.990980373. As in the *TinyGp* runs, the average accuracy of the *ParetoGp* runs outperforms any of the base kernels. And, again, the maximum accuracy was 0.9941176, which was returned by a number of different kernels.

The *ParetoGp* runs found combinations that improved on the base kernels in 14 of the 30 runs. In the other 16 runs it returned the Laplace kernel, the best performing base kernel that gets used for classification.

4.1.4 Statistical Analysis

To determine the statistical significance of the results, Student's t-tests were performed. One-sample t-tests were performed to determine the significance of the average accuracy of the GP runs against each of the base kernels. A two-sample t-test was performed to compare the *TinyGp* runs against the *ParetoGp* runs.

***TinyGp* vs base kernels:**

The null hypothesis for these tests is that the difference between the classification accuracy of the base kernel under test and the average classification accuracy of the sample of 30 *TinyGp* runs is zero. The alternative hypothesis is that it the difference is non-zero. Two-tailed t-tests were used as it was not clear if the GP runs would be different from the classification accuracy of the base kernels. The calculations were performed using Excel's Data Analysis tool.

Table 4.4 shows the p-values for the base kernels tested against the *TinyGp* runs.

Base Kernel	**p-value**	**Null hypothesis**
Linear	3.56966×10^{-30}	Rejected
RBF	2.27207×10^{-22}	Rejected
Polynomial	3.56966×10^{-30}	Rejected
Sigmoid	6.44811×10^{-74}	Rejected
Bessel	0.002939291	Rejected
ANOVA	1.62965×10^{-15}	Rejected
Laplace	0.002939291	Rejected
Spline	6.84943×10^{-40}	Rejected

Table 4.4: t-test outcomes base kernels vs *TinyGp*

For all the base kernels the null hypothesis is rejected at the 0.05 significance level i.e. there is significant evidence that the *TinyGp* runs improve on the classification accuracies of the base kernels.

***ParetoGp* vs base kernels**

Table 4.5 below shows the p-values for the base kernels tested against the *ParetoGp* runs. Again, the null hypothesis is that the difference between the classification accuracy of the base kernel under test and the average classification accuracy of the sample of 30 *ParetoGp* runs is zero.

Base Kernel	**p-value**	**Null hypothesis**
Linear	3.04723×10^{-29}	Rejected
RBF	6.21526×10^{-22}	Rejected
Polynomial	3.04723×10^{-29}	Rejected
Sigmoid	2.04115×10^{-72}	Rejected
Bessel	2.28641×10^{-5}	Rejected
ANOVA	8.28038×10^{-18}	Rejected
Laplace	2.28641×10^{-5}	Rejected
Spline	1.20878×10^{-38}	Rejected

Table 4.5 t-test outcomes base kernels vs *ParetoGp*

Again, the null hypothesis is rejected for all the base kernels, indicating significant evidence that the classification accuracies of kernel combinations returned by the *ParetoGp* runs improve on the classification accuracies of the base kernels.

TinyGp* vs *ParetoGp

For the 30 samples recorded the average classification accuracy of the *ParetoGp* runs was slightly higher than the *TinyGp* runs, at 0.990980373 vs 0.989803913. A two-sample t-test was performed to determine if this difference was statistically significant. The null hypothesis was that the difference between the means is zero. The p-value calculated was 0.111714027, and so the null hypothesis is not rejected. Therefore, it can be said that there is no statistically significant difference between the two types of GP run.

4.1.5 Summary

There is significant statistical evidence that the algorithm produces combination kernels that improve on the accuracy of the base kernels. There is no statistical evidence that there is any difference in the performance of the *TinyGp* and *ParetoGp* search heuristics at finding kernel combinations.

4.2 Pima Indians diabetes data-set

The Pima Indians diabetes data set contains 768 instances and 9 attributes, including the class label. The data relates to females of at least 21 years of age of Pima Indian heritage and whether or not they tested positive for diabetes. The class labels are the diagnosis, with class value 1 signifying a positive test for diabetes. The data-set contains 500 negative diagnoses and 268 positive diagnoses. The remaining 8 attributes are numeric-valued medical features such as body mass index, age and number of times pregnant. This data-set and its information can be found at https://archive.ics.uci.edu/ml/datasets/Pima+Indians+Diabetes.

The data-set was saved as .csv file and imported into R. The class labels were extracted and saved as a matrix. The remaining attributes, in matrix format, were scaled to be in the range 0 to 1. The matrix was then partitioned into train and test sets. The train set contained 599 instances and the test set contained 169 instances, an approximate 78:22 split.

4.2.1 Base kernels

The classification accuracies of kernlab's 8 base kernels on the test set are shown in Table 4.6. Also shown are the matrix properties symmetric and PSD.

Base Kernel	**Classification Accuracy**	**Symmetric**	**PSD**
Linear (Kvanilla)	0.7751479	Yes	Yes
RBF (Krbf)	0.7928994	No	N/A
Polynomial (Kpoly)	0.7751479	Yes	Yes
Sigmoid (Ktanh)	0.3076923	Yes	No
Bessel (Kbessel)	0.7810651	Yes	Yes
ANOVA (Kanova)	0.7692308	Yes	Yes
Laplace (Klaplace)	0.8047337	Yes	Yes
Spline (Kspline)	0.7928994	Yes	Yes

Table 4.6: Base kernel test set classification accuracy and matrix properties

The best performing base kernel is the Laplace kernel, with a classification accuracy of 0.8047337 on the test set.

4.2.2 *TinyGp*

The first GP experiment used the *makeTinyGpSearchHeuristic*. Thirty GP runs were performed using the following parameters:

- Population Size: 80
- Maximum Tree Depth: 3
- Step Limit: 801
- Crossover Probability: 0.9 (*TinyGp* default value)

Table 4.7 below shows the results of the 30 *TinyGp* runs, showing the kernel combination that resulted in the best classification accuracy on the test set, and its classification accuracy.

Run Number	**Kernel Combination**	**Classification Accuracy**
1	Klaplace * Ktanh * (Kvanilla + Klaplace)	0.8284024
2	Klaplace + exp(Klaplace)	0.8224852
3	exp(Klaplace)	0.8106509
4	exp(Klaplace) + Klaplace	0.8224852
5	Klaplace + Ktanh + exp(Klaplace)	0.8224852
6	Kspline * Klaplace + Kspline	0.816568
7	exp(Klaplace) + Klaplace	0.8224852
8	Kbessel + Ktanh + Klaplace * Kspline	0.816568
9	exp(Klaplace) + Klaplace	0.8224852
10	(Kbessel + Klaplace) * exp(Kbessel)	0.8284024
11	Kspline * Klaplace * Ktanh	0.816568
12	Kspline * Klaplace + Kvanilla	0.816568
13	Klaplace + exp(Klaplace)	0.8224852
14	Klaplace * (0.562339203777333 + Kpoly)	0.8284024
15	exp(Klaplace) + (Klaplace + Kvanilla)	0.8224852
16	exp(Klaplace) + Klaplace	0.8224852
17	(Kbessel + Kpoly) * Klaplace	0.8284024
18	exp(Kbessel) * (Kbessel + Klaplace)	0.8284024
19	Ktanh + exp(Klaplace) + Klaplace	0.8224852
20	exp(Kbessel) * Klaplace	0.8224852
21	Klaplace * 0.400848640248593 * (Klaplace * Kspline)	0.8224852
22	Kpoly * (Klaplace * Kbessel)	0.816568
23	Klaplace + (1.33806425435492 + Klaplace) + (Kspline + Klaplace)	0.8224852
24	exp(Kbessel * Klaplace * Kbessel * Kbessel)	0.8224852
25	Klaplace + exp(Klaplace)	0.8224852
26	(0.263857008080993 + Kbessel) * exp(Klaplace)	0.8224852
27	Klaplace * (Klaplace + Kvanilla)	0.8284024
28	Klaplace * (Kbessel + 1.93663826985605) + 1.77898957755768	0.8224852
29	Klaplace * Kpoly + Klaplace	0.8284024
30	exp(Kbessel) * (Klaplace + Kbessel)	0.8284024

Table 4.7: *TinyGp* best kernel combinations and classification accuracies

The average classification of the combination kernels generated by the 30 GP runs is 0.822682443. This improves on the classification accuracy of the best performing base kernel, the Laplace kernel with a 0.8047337. The maximum classification accuracy of the combination kernels is 0.8284024, which was returned by 8 of the GP runs using a number of different kernel combinations. This represents classifying 140 of the 169 test instances correctly. All 30 of the combined kernels outperformed the best-performing base kernel.

4.2.3 *ParetoGp*

The second experiment used the *makeAgeFitnessComplexityParetoGpSearchHeuristic*. Thirty GP runs were performed with the following parameters:

- Population Size: 80
- Maximum Tree Depth: 3
- Step Limit: 21 (equivalent to 20 generations)
- Crossover Probability: 0.5 (*ParetoGp* default value)

Table 4.8 below shows results of the 30 *ParetoGp* runs, showing the kernel combination that resulted in the best classification accuracy on the test set for each run, and the classification accuracy.

Run Number	Kernel Combination	Classification Accuracy
1	Klaplace * exp(Kbessel)	0.8224852
2	Klaplace * exp(Kbessel)	0.8224852
3	(Kspline + Klaplace) * exp(Kanova) * exp(Kanova)	0.816568
4	exp(Klaplace) * Kvanilla	0.816568
5	exp(Kbessel) * Klaplace	0.8224852
6	exp(Klaplace)	0.8106509
7	Kspline * Klaplace + 0.0155806591534536	0.816568
8	exp(Klaplace) + (Klaplace + Kspline)	0.8224852
9	3.11813402004905 * Klaplace	0.8224852
10	Kspline * Klaplace + Kbessel	0.816568
11	Klaplace	0.8047337
12	exp(Klaplace) + Kpoly	0.8106509
13	Klaplace + exp(Klaplace)	0.8224852
14	Klaplace * exp(Kbessel)	0.8224852
15	Klaplace + exp(Klaplace)	0.8224852
16	Klaplace * Kpoly + Klaplace	0.8284024
17	Klaplace + exp(Klaplace)	0.8224852
18	Klaplace * exp(Kbessel)	0.8224852
19	Klaplace * Kspline	0.816568
20	Kspline * (Kbessel + Klaplace)	0.816568
21	exp(0.14745632954195 + (Klaplace + 0.0877830073405247))	0.8224852
22	Kbessel + Klaplace * Kspline	0.816568
23	0.928567372092567 + Klaplace + exp(Klaplace)	0.8224852
24	exp(Klaplace) + Klaplace	0.8224852
25	exp(Klaplace)	0.8106509
26	exp(Klaplace) * Kvanilla	0.816568
27	exp(Klaplace) + Kspline	0.816568
28	Kspline + Kspline * Klaplace	0.816568
29	exp(Kbessel) * Klaplace	0.8224852
30	Klaplace + exp(Klaplace)	0.8224852

Table 4.8: *ParetoGp* best kernel combinations and classification accuracies

The average classification accuracy of the 30 GP runs was 0.818934893, which again outperforms any of the base kernels. The maximum accuracy found was, as with *TinyGp,* 0.8284024. However only one of the GP runs returned a combination with this accuracy. Of the remaining 29 runs, 28 of them found kernel combinations that outperformed the Laplace base kernel. One run returned the Laplace kernel as the best performing combination.

4.2.4 Statistical Analysis

To determine the statistical significance of the results of the GP runs, a number of t-tests were performed.

***TinyGp* vs base kernels:**

The null hypothesis for these tests is that the difference between the classification accuracy of the base kernel under test and the average classification accuracy of the sample of 30 *TinyGp* runs is zero. The alternative hypothesis is that the difference is non-zero.

Table 4.9 shows the p-values for the base kernels tested against the *TinyGp* runs.

Base Kernel	**p-value**	**Null hypothesis**
Linear	1.91774×10^{-31}	Rejected
RBF	1.22908×10^{-25}	Rejected
Polynomial	1.91774×10^{-31}	Rejected
Sigmoid	2.12135×10^{-61}	Rejected
Bessel	8.7297×10^{-30}	Rejected
ANOVA	6.55047×10^{-33}	Rejected
Laplace	1.73262×10^{-19}	Rejected
Spline	1.22908×10^{-25}	Rejected

Table 4.9 t-test outcomes base kernels vs *TinyGp*

For all the base kernels the null hypothesis is rejected at the 0.05 significance level i.e. there is significant evidence that the *TinyGp* runs improve on the classification accuracies of the base kernels.

***ParetoGp* vs base kernels**

Table 4.10 below shows the p-values for the base kernels tested against the *ParetoGp* runs. Again, the null hypothesis is that the difference between the classification accuracy of the base kernel under test and the average classification accuracy of the sample of 30 *ParetoGp* runs is zero.

Base Kernel	**p-value**	**Null hypothesis**
Linear	4.91581×10^{-29}	Rejected
RBF	1.26479×10^{-22}	Rejected
Polynomial	4.91581×10^{-29}	Rejected
Sigmoid	6.62451×10^{-60}	Rejected
Bessel	3.14177×10^{-27}	Rejected
ANOVA	1.3056×10^{-30}	Rejected
Laplace	1.77492×10^{-15}	Rejected
Spline	1.24679×10^{-22}	Rejected

Table 4.10 t-test outcomes base kernels vs *ParetoGp*

As the null hypothesis is rejected for all of the base kernels, it is concluded that there is significant statistical evidence that the *ParetoGp* runs improve on the classification accuracy of the base kernels.

TinyGp* vs *ParetoGp

For the 30 samples recorded the average classification accuracy of the *TinyGp* runs was slightly higher than the *ParetoGp* runs, at 0.822682443 vs 0.818934893. A two-sample t-test was performed to determine if this difference was statistically significant. The null hypothesis was that the difference between the means is zero and the alternative hypothesis was that the difference between the means is non-zero. The p-value calculated was 0.00373814, and so the null hypothesis is rejected. Therefore there is a statistically significant difference between the two types of GP run. However, as the difference between the classification accuracies of the two search heuristics is 0.00374755 it is reasonable to consider whether or not the difference is practically significant when it comes time to decide between the *TinyGp* and *ParetoGp* search heuristics.

4.2.5 Summary

As for the Wisconsin breast cancer data-set, there is strong evidence that applying GP to find combination kernels leads to an improvement in classification accuracy over the base kernels for the Pima Indians diabetes data-set.

There is also statistical evidence that *TinyGp* is superior to *ParetoGp* on this dataset. However, considering the similarity in classification accuracies of the two heuristics, a user would be entitled to consider whether there is any practically significant difference between the two.

4.3 Ionosphere data-set

The ionosphere data-set consists of 351 instances of 35 attributes. 34 of the attributes consist of continuous real-valued observations from a radar array in Canada. The 35th attribute is the class label, either 'good' or 'bad', indicating whether or not the data show evidence of structure in the ionosphere. These are the binary class labels. The data-set and more information about it can be found at https://archive.ics.uci.edu/ml/datasets/Ionosphere.

The data-set was saved as a .csv and imported into R. The class labels were extracted. In this case the data was not scaled. Testing revealed that the scaled data frequently returned the error "No support vectors found; You might want to change your parameters". As discussed in Section 4, the SVM and kernel parameters were not changed as part of this project, so the data was left unscaled. No errors were reported running the base kernels on the unscaled data.

The data was partitioned into train and test sets, with 249 instances in the train set and 102 in the test set, a split of approximately 71:29.

4.3.1 Base kernels

The classification accuracies of kernlab's 8 base kernels on the test set are shown in Table 4.11. Also shown are the matrix properties symmetric and PSD.

Base Kernel	**Classification Accuracy**	**Symmetric**	**PSD**
Linear (Kvanilla)	0.9607843	Yes	Yes
RBF (Krbf)	0.8235294	No	N/A
Polynomial (Kpoly)	0.9607843	Yes	Yes
Sigmoid (Ktanh)	0.009803922	Yes	No
Bessel (Kbessel)	0.9705882	No	N/A
ANOVA (Kanova)	0.9411765	Yes	Yes
Laplace (Klaplace)	0.9411765	No	N/A
Spline (Kspline)	0.5490196	Yes	Yes

Table 4.11: Base kernel test set classification accuracy and matrix properties

The best performing base kernel is the Bessel kernel, with a classification accuracy of 0.9705882 on the test set. However, the Bessel kernel matrix is not symmetric. Therefore it will not be used for classification in the fitness function (see Section 3 for details).

4.3.2 *TinyGp*

The first GP experiment used the *makeTinyGpSearchHeuristic*. Thirty GP runs were performed using the following parameters:

- Population Size: 80
- Maximum Tree Depth: 3
- Step Limit: 801
- Crossover Probability: 0.9 (*TinyGp* default value)

Table 4.12 below shows the results of the 30 *TinyGp* runs, showing the kernel combination that resulted in the best classification accuracy on the test set, and its classification accuracy.

Run Number	Kernel Combination	Classification Accuracy
1	Kanova	0.9411765
2	Kpoly	0.9607843
3	Kpoly	0.9607843
4	Kpoly	0.9607843
5	Kvanilla * Kpoly * exp(-0.184423688307749)	0.9803922
6	Kvanilla	0.9607843
7	0.173444030390435 * Kanova * (0.843281268695121 * 0.173444030390435)	0.9803922
8	Kpoly	0.9607843
9	Kvanilla	0.9607843
10	Kpoly	0.9607843
11	Kvanilla * Kpoly + Kvanilla * -1.13821567875988	0.9705882
12	Kvanilla	0.9607843
13	Kpoly * (Kpoly + 1.27022806656364)	0.9705882
14	Kvanilla	0.9607843
15	0.830332556253847 * 0.043389041805774 * (Kvanilla + Kpoly)	0.9705882
16	Kpoly	0.9607843
17	Kpoly	0.9607843
18	Kvanilla	0.9607843
19	Kvanilla	0.9607843
20	Kanova * (-0.140065206153455 * -0.275087279105598)	0.9803922
21	Kpoly	0.9607843
22	Kvanilla	0.9607843
23	Kpoly	0.9607843
24	exp(0.230929703118384 * Kvanilla)	0.9705882
25	Kpoly	0.9607843
26	Kvanilla	0.9607843
27	exp(Kanova * 0.298257367986263)	0.9803922
28	Kpoly * exp(-2.98183042985318)	0.9705882
29	Kvanilla	0.9607843
30	Kpoly * (Kvanilla + 2.35678060564807)	0.9705882

Table 4.12: *TinyGp* best kernel combinations and classification accuracies

The average classification accuracy of the combination kernels generated by the 30 GP runs is 0.964705873. This is less than the accuracy of the Bessel kernel, the best performing base kernel. However, as noted in Section 4.3.1, the Bessel kernel matrix is not symmetric and is therefore not passed to the *ksvm* function as an argument. The maximum classification accuracy of the combination kernels is 0.9803922, which is an improvement on the Bessel kernel. Combined kernels with this accuracy were found 4 times in the 30 GP runs. In Run 1, the GP returned the ANOVA kernel as the best kernel combination. This kernel is not even the best performing of the symmetric base kernels. However, in 10 of the 30 runs the GP returned combinations that matched or surpassed the accuracy of the Bessel kernel. In the remaining 19 runs, the GP returned either the Linear or Polynomial kernel as the best kernel combination. As has been noted in Section 3, the Linear and Polynomial kernels are equivalent when the Polynomial kernel is set to its default values, as is the case in this project.

4.3.3 *ParetoGp*

The second experiment used the *makeAgeFitnessComplexityParetoGpSearchHeuristic*. Thirty GP runs were performed with the following parameters:

- Population Size: 80
- Maximum Tree Depth: 3
- Step Limit: 21 (equivalent to 20 generations)
- Crossover Probability: 0.5 (*ParetoGp* default value)

Table 4.13 below shows results of the 30 *ParetoGp* runs, showing the kernel combination that resulted in the best classification accuracy on the test set for each run, and the classification accuracy.

Run Number	**Kernel Combination**	**Classification Accuracy**
1	Kpoly	0.9607843
2	Kpoly	0.9607843
3	Kvanilla	0.9607843
4	Kvanilla	0.9607843
5	Kpoly	0.9607843
6	Kvanilla	0.9607843
7	Kpoly	0.9607843
8	(Kpoly + -1.24300189819521) * Kvanilla	0.9705882
9	Kpoly	0.9607843
10	Kpoly	0.9607843
11	Kpoly	0.9607843
12	Kpoly	0.9607843
13	(Kpoly + Kvanilla) * 0.010059561765422	0.9705882
14	Kvanilla	0.9607843
15	Kpoly	0.9607843
16	Kvanilla	0.9607843
17	0.0228448881705308 * Kanova	0.9803922
18	Kvanilla	0.9607843
19	Kpoly	0.9607843
20	Kpoly	0.9607843
21	Kpoly	0.9607843
22	0.0596626681131614 * Kpoly	0.9705882
23	Kvanilla	0.9607843
24	Kvanilla	0.9607843
25	exp(-2.96891231472326 * Kanova)	0.9803922
26	Kvanilla	0.9607843
27	Kvanilla	0.9607843
28	Kpoly	0.9607843
29	Kpoly	0.9607843
30	exp(0.035467581078367 * Kvanilla)	0.9803922

Table 4.13: *ParetoGp* best kernel combinations and classification accuracies

The average classification accuracy of the combination kernels generated by the 30 GP runs is 0.96372548. As with the *TinyGp* runs, this is less than the accuracy of the Bessel kernel. The maximum classification accuracy of the combination kernels is 0.9803922, which is an improvement on the Bessel kernel. Combined kernels with this accuracy were found 3 times in the 30 GP runs. Another 3 runs found combinations that matched the accuracy of the Bessel kernel. The remaining 24 runs returned either the Linear or Polynomial kernel as the best combination. As has been noted, these two kernels are equivalent and produce the same classification accuracies.

4.3.4 Statistical Analysis

To determine the statistical significance of the results of the GP runs, a number of t-tests were performed.

***TinyGp* vs base kernels:**

The null hypothesis for these tests is that the difference between the classification accuracy of the base kernel under test and the average classification accuracy of the sample of 30 *TinyGp* runs is zero. The alternative hypothesis is that the difference is non-zero.

Table 4.14 shows the p-values for the base kernels tested against the *TinyGp* runs.

Base Kernel	**p-value**	**Null hypothesis**
Linear	0.015850811	Rejected
RBF	$2.3301\text{x}10^{-37}$	Rejected
Polynomial	0.015850811	Rejected
Sigmoid	$2.05096\text{x}10^{-61}$	Rejected
Bessel	0.000610664	Rejected
ANOVA	$1.77499\text{x}10^{-15}$	Rejected
Laplace	$1.77499\text{x}10^{-15}$	Rejected
Spline	$6.08998\text{x}10^{-51}$	Rejected

Table 4.14 t-test outcomes base kernels vs *TinyGp*

For all the base kernels the null hypothesis is rejected at the 0.05 significance level i.e. there is significant evidence that the classification accuracy of the base kernels differs from the combination kernels returned by the GP runs. Unlike the Wisconsin breast cancer and the Pima Indians diabetes data-sets, the difference is not all one-way i.e. improvement. The Bessel kernel accuracy of 0.9705882 is a statistically significant improvement over the *TinyGp* average of 0.964705873.

***ParetoGp* vs base kernels**

Table 4.15 below shows the p-values for the base kernels tested against the *ParetoGp* runs. Again, the null hypothesis is that the difference between the classification accuracy of the base kernel under test and the average classification accuracy of the sample of 30 *ParetoGp* runs is zero.

Base Kernel	p-value	Null hypothesis
Linear	0.01737244	Rejected
RBF	$1.08502\text{x}10^{-40}$	Rejected
Polynomial	0.01737244	Rejected
Sigmoid	$7.88594\text{x}10^{-65}$	Rejected
Bessel	$2.17194\text{x}10^{-6}$	Rejected
ANOVA	$4.04905\text{x}10^{-18}$	Rejected
Laplace	$4.04905\text{x}10^{-18}$	Rejected
Spline	$2.4385\text{x}10^{-54}$	Rejected

Table 4.15 t-test outcomes base kernels vs *ParetoGp*

Similar to the *TinyGp* runs, the null hypotheses are rejected i.e. there is a statistically significant difference between the classification accuracies of the base kernels and the GP runs. Again though, the Bessel kernel represents a statistically significant improvement over the mean classification accuracy of the GP runs.

TinyGp* vs *ParetoGp

For the 30 samples recorded the average classification accuracy of the *TinyGp* runs was slightly higher than the *ParetoGp* runs, at 0.964705873 vs 0.96372548. A two-sample t-test was performed to determine if this difference was statistically significant. The null hypothesis was that the difference between the means is zero and the alternative hypothesis was that the difference between the means is non-zero. The p-value calculated was 0.612401797, and so the null hypothesis is not rejected. Therefore there is no statistically significant difference between the samples of classification accuracies returned by the two search heuristics.

4.3.5 Summary

Unlike the Wisconsin breast cancer and Pima Indians diabetes data-sets, there is a base kernel, the Bessel kernel, that outperforms the average classification accuracies of both search heuristics as calculated from 30 run samples. This is due to an idiosyncrasy of the code, where the non-symmetric Bessel kernel matrix is rejected as a valid kernel for classification. This issue will be discussed further in Section 6. For all kernel matrices that are deemed valid for classification by the code, the mean classification accuracies of both search heuristics are higher.

There is no evidence that there is any statistically significant difference between the two search heuristics for this data-set.

4.4 Results Summary

Table 4.16 below shows a summary of the results across the three data-sets.

	Wisconsin	Pima	Ionosphere
Best Base	0.9882353	0.8047337	0.9705882
TinyGp average	0.98980319	0.822682443	0.964705873
TinyGp max.	0.9941176	0.8284024	0.9803922
ParetoGp average	0.990980373	0.818934893	0.96372548
ParetoGp max.	0.9941176	0.8284024	0.9803922

Table 5.16: Summary of results

As can be seen from this data, the combined kernels outperformed the base kernels in both the Wisconsin breast cancer and the Pima Indians diabetes data-sets, in both average and maximum classification accuracies.

For the Ionosphere data-set, the average classification accuracy of both the *TinyGp* and *ParetoGp* runs was less than the classification accuracy of the best-performing base kernel, the Bessel kernel. However, this was because the Bessel kernel matrix was not symmetric and was thus not returned by the GP runs as a valid kernel matrix. The maximum accuracy of kernel combinations found by both the search heuristics was superior to the accuracy of the Bessel kernel.

For the Wisconsin breast-cancer and Ionosphere data-sets there was no statistically significant difference between the two search heuristics under investigation. For the Pima Indian diabetes data-set, the test statistic suggested that there was a statistically significant difference between the search heuristics. However, with average classification accuracies of 0.822682443 for *TinyGp* and 0.818934893 for *ParetoGp* there is little or no practically significant difference between the two.

5 Discussion

Generally the technique has shown good outcomes, with a total of 120 GP runs performed across 3 data-sets showing 119 runs returning classification accuracies that matched or outperformed the best-performing base kernels that get used for classification in the fitness function. However, it must be noted that the final part of that last sentence contains an important caveat.

5.1 Kernel matrices and the possibility of rounding errors

If we take as an example the Ionosphere and re-examine Table 4.11, reproduced below as Table 6.1, we note that the best-performing base kernel was the Bessel kernel.

Base Kernel	**Classification Accuracy**	**Symmetric**	**PSD**
Linear (Kvanilla)	0.9607843	Yes	Yes
RBF (Krbf)	0.8235294	No	N/A
Polynomial (Kpoly)	0.9607843	Yes	Yes
Sigmoid (Ktanh)	0.009803922	Yes	No
Bessel (Kbessel)	0.9705882	No	N/A
ANOVA (Kanova)	0.9411765	Yes	Yes
Laplace (Klaplace)	0.9411765	No	N/A
Spline (Kspline)	0.5490196	Yes	Yes

Table 5.1 Ionosphere base kernels

However, the Bessel kernel matrix is not symmetric and thus it will be disqualified from use in the fitness function. This, we believe, is a limitation of the kernel matrix approach used in this project.

As the kernel matrix is the matrix of pairwise inner products on the data items under investigation, it must be symmetric i.e $K(i,j) = K(i,j)$. That this is not so for the Bessel kernel matrix in this data-set is believed to be the result of rounding errors.

Some investigation of this problem was undertaken. The Bessel kernel matrix was subtracted element-wise from its transpose and assigned to the matrix object 'test' using the following R command:

```
> test <- t(Kbessel)-Kbessel
```

If the matrix was symmetric we would expect all the elements of the matrix 'test' to be zero. The maximum value of the resulting matrix 'test' was then determined.

```
> max(test)
[1] 2.397813e-10
```

As can be seen the maximum difference between any two elements in the Bessel kernel matrix and its transpose is of the order 10^{-10}. The number of non-zero entries in the 'test' matrix was then checked.

```
> nnzero(test)
[1] 878
```

That is, out of a total of 123,201 elements only 878 are non-zero, and the maximum value of the non-zero entries is 2.397813x10^{-10}. Though this is not conclusive in any way, it is suggestive of rounding errors in the calculation of the kernel matrices. This is a limitation of the kernel matrix approach.

The Bessel kernel matrix then never gets returned by the GP fitness function, as the fitness function includes a check for symmetry of all candidate matrix solutions. This check for symmetry was included as a consequence of the fact that the R function used to check for the PSD matrix property, *is.positive.semi.definite()*, only accepts a symmetric matrix as an argument. If the matrix is not symmetric the function returns an error and the program stops. The check for symmetry was included to prevent this error occurring.

It is believed that rounding errors may also play a part in the matrices being returned as non-PSD, though further investigation is also required to demonstrate this. Many kernel matrices that were calculated during testing had eigenvalues that were only very slightly negative. An R function, *nearPD()*, that converts matrices that are very close to being PD to the nearest PD matrix was investigated for this project, but ultimately not used.

The problems of non-symmetric and non-PSD matrices remains an outstanding limitation of this implementation.

5.2 SVM and kernel parameters

As mentioned in Section 1 of this document, the choice of kernel function is only one of the considerations the user must take into account when implementing SVM classification tasks. For this project the SVM parameter *C,* the soft margin parameter, was left at Kernlab's default value of 1. The kernel parameters were also left at their default parameter values, except for the Bessel kernel as discussed in Section 3.1.

The choice of SVM and kernel parameters are also critical to the classification performance of the algorithm. It is beyond the scope of this project to evolve these parameters using GP or genetic algorithms but there is much research done in this area and it would be a valuable extension to this project.

6 Conclusion

There is considerable evidence that this approach creates combination kernel functions that outperform the individual base kernels. Across the three data-sets, two (Wisconsin breast cancer and Pima Indian diabetes) showed superior average classification accuracies on the test data. The third data-set, Ionosphere, showed results where one of the base kernels outperformed the combined kernels. However, a clear reason for this is given in Section 4.

For the Wisconsin breast cancer and Ionosphere data-sets no statistically significant difference between the two search heuristics, *TinyGp* and *ParetoGp,* was found. For the Pima Indians diabetes data-set the test statistic showed evidence that *TinyGp* outperformed *ParetoGp.* However, the average classification accuracies of the two are so similar that there is little practical significance to this finding.

Overall it is concluded that this is a promising approach to automating kernel selection for SVM binary classification tasks. As discussed in Section 5, some work remains to be done on investigating the possibility of rounding errors causing some of the kernel matrices being returned as not symmetric.

Bibliography

Boser, Bernhard E., Isabelle M. Guyon, and Vladimir N. Vapnik. "A training algorithm for optimal margin classifiers." In *Proceedings of the fifth annual workshop on Computational learning theory*, pp. 144-152. ACM, 1992.

Dioşan, Laura, Alexandrina Rogozan, and Jean-Pierre Pecuchet. "Learning SVM with Complex Multiple Kernels Evolved by Genetic Programming."*International Journal on Artificial Intelligence Tools* 19, no. 05 (2010): 647-677.

Espejo, Pedro G., Sebastián Ventura, and Francisco Herrera. "A survey on the application of genetic programming to classification." *Systems, Man, and Cybernetics, Part C: Applications and Reviews, IEEE Transactions on* 40, no. 2 (2010): 121-144.

Gîrdea, Marta, and Liviu Ciortuz. "A Hybrid Genetic Programming and Boosting Technique for Learning Kernel Functions from Training Data." In *SYNASC*, pp. 395-402. 2007.

Hofmann, Thomas, Bernhard Schölkopf, and Alexander J. Smola. "Kernel methods in machine learning." *The annals of statistics* (2008): 1171-1220.

Howley, Tom, and Michael G. Madden. "The genetic kernel support vector machine: Description and evaluation." *Artificial Intelligence Review* 24, no. 3-4 (2005): 379-395.

Koza, John R. *Genetic programming: on the programming of computers by means of natural selection*. Vol. 1. MIT press, 1992.

Mitchell, Melanie. *An introduction to genetic algorithms*. MIT press, 1998.

Phienthrakul, Tanasanee, and Boonserm Kijsirikul. "GPES: An algorithm for evolving hybrid kernel functions of support vector machines." In *Evolutionary Computation, 2007. CEC 2007. IEEE Congress on*, pp. 2636-2643. IEEE, 2007

Poli, Riccardo, William B. Langdon, Nicholas F. McPhee, and John R. Koza. *A field guide to genetic programming*. Lulu. com, 2008.

Shawe-Taylor, John, and Nello Cristianini. *Kernel methods for pattern analysis*. Cambridge university press, 2004.

Smits, Guido F., and Mark Kotanchek. "Pareto-front exploitation in symbolic regression." In *Genetic programming theory and practice II*, pp. 283-299. Springer US, 2005.

Sullivan, Keith M., and Sean Luke. "Evolving kernels for support vector machine classification." In *Proceedings of the 9th annual conference on Genetic and evolutionary computation*, pp. 1702-1707. ACM, 2007.