Machine Learning Mastery With Weka

Analyze Data, Develop Models and Work Through Projects

Jason Brownlee

MACHINE LEARNING MASTERY

Jason Brownlee

Machine Learning Mastery With Weka

Analyze Data, Develop Models and Work Through Projects

Machine Learning Mastery With Weka

Edition: v1.2

Contents

I Introduction 1

1 Welcome 2
- 1.1 Applied Machine Learning the Wrong Way . . . 2
- 1.2 Applied Machine Learning with Weka . . . 2
- 1.3 Book Overview . . . 3
- 1.4 Your Outcomes From This Process . . . 5
- 1.5 What This Book is Not . . . 5
- 1.6 Summary . . . 6

2 Rapidly Accelerate Your Progress in Applied Machine Learning With Weka 7
- 2.1 Starting in Applied Machine Learning is Hard . . . 7
- 2.2 Focus on Learning Just One Thing . . . 8
- 2.3 Learn the Process of Applied Machine Learning . . . 8
- 2.4 How to Best Use Weka . . . 8
- 2.5 Summary . . . 9

3 A Gentle Introduction to the Weka Machine Learning Workbench 10
- 3.1 What is Weka . . . 10
- 3.2 Introduction to the Weka Graphical Interface . . . 11
- 3.3 Why You Should Use Weka . . . 12
- 3.4 Summary . . . 12

4 How to Make Best Use of Weka For Applied Machine Learning 13
- 4.1 Harness The Number One Benefit of Weka . . . 13
- 4.2 Build a Machine Learning Portfolio . . . 14
- 4.3 Practice On Small In-Memory Datasets . . . 15
- 4.4 Benefits of the Repository . . . 16
- 4.5 Summary . . . 16

II Lessons 18

5 How to Download and Install the Weka Machine Learning Workbench 19
- 5.1 Download Weka . . . 19
- 5.2 Install The All-In-One Version of Weka . . . 19
- 5.3 Install Java and Weka Separately . . . 21
- 5.4 Install Weka On Linux And Other Platforms . . . 22

5.5 Summary . . . 23

6 A Tour of the Weka Machine Learning Workbench 24
6.1 Weka GUI Chooser . . . 24
6.2 Weka Explorer . . . 27
6.3 Weka Experiment Environment . . . 33
6.4 Weka KnowledgeFlow Environment . . . 36
6.5 Weka Workbench . . . 37
6.6 Weka SimpleCLI . . . 38
6.7 Weka Java API . . . 39
6.8 Summary . . . 39

7 How To Load CSV Machine Learning Data 41
7.1 How to Talk About Data in Weka . . . 41
7.2 Data in Weka . . . 42
7.3 Load CSV Files in the *ARFF-Viewer* . . . 42
7.4 Load CSV Files in the *Weka Explorer* . . . 45
7.5 Use Excel for Other File Formats . . . 46
7.6 Summary . . . 46

8 How to Load Standard Machine Learning Datasets 47
8.1 Standard Weka Datasets . . . 47
8.2 Binary Classification Datasets . . . 49
8.3 Multiclass Classification Datasets . . . 50
8.4 Regression Datasets . . . 51
8.5 Summary . . . 52

9 How to Better Understand Your Machine Learning Data 53
9.1 Descriptive Statistics . . . 53
9.2 Univariate Attribute Distributions . . . 56
9.3 Visualize Attribute Interactions . . . 58
9.4 Summary . . . 61

10 How to Normalize and Standardize Your Machine Learning Data 63
10.1 About Data Filters in Weka . . . 63
10.2 Normalize Your Numeric Attributes . . . 65
10.3 Standardize Your Numeric Attributes . . . 68
10.4 Summary . . . 69

11 How to Transform Your Machine Learning Data 70
11.1 Discretize Numerical Attributes . . . 70
11.2 Convert Nominal Attributes to Dummy Variables . . . 73
11.3 Summary . . . 77

12 How To Handle Missing Values In Machine Learning Data 78
12.1 Mark Missing Values . . . 78
12.2 Remove Missing Data . . . 80
12.3 Impute Missing Values . . . 82

12.4 Summary . . . 84

13 How to Perform Feature Selection With Machine Learning Data 85
13.1 Feature Selection in Weka . . . 85
13.2 Correlation Based Feature Selection . . . 88
13.3 Information Gain Based Feature Selection . . . 89
13.4 Learner Based Feature Selection . . . 90
13.5 Select Attributes in Weka . . . 92
13.6 What Feature Selection Techniques To Use . . . 94
13.7 Summary . . . 94

14 How to Use Machine Learning Algorithms 95
14.1 Weka Machine Learning Algorithms . . . 95
14.2 Which Algorithm To Use . . . 98
14.3 Linear Machine Learning Algorithms . . . 99
14.4 Nonlinear Machine Learning Algorithms . . . 99
14.5 Ensemble Machine Learning Algorithms . . . 99
14.6 Machine Learning Algorithm Configuration . . . 99
14.7 Get More Information on Algorithms . . . 100
14.8 Summary . . . 102

15 How To Estimate The Performance of Machine Learning Algorithms 104
15.1 Model Evaluation Techniques . . . 104
15.2 Which Test Option to Use . . . 105
15.3 What About The Final Model . . . 106
15.4 Performance Summary . . . 106
15.5 Summary . . . 109

16 How To Estimate A Baseline Performance For Your Models 110
16.1 Importance of Baseline Results . . . 110
16.2 Zero Rule For Baseline Performance . . . 111
16.3 Baseline Performance for Classification Problems . . . 112
16.4 Summary . . . 113

17 How To Use Top Classification Machine Learning Algorithms 114
17.1 Classification Algorithm Tour Overview . . . 114
17.2 Logistic Regression . . . 115
17.3 Naive Bayes . . . 117
17.4 Decision Tree . . . 119
17.5 k-Nearest Neighbors . . . 122
17.6 Support Vector Machines . . . 125
17.7 Summary . . . 127

18 How To Use Top Regression Machine Learning Algorithms 128
18.1 Regression Algorithms Overview . . . 128
18.2 Linear Regression . . . 129
18.3 k-Nearest Neighbors . . . 130

18.4 Decision Tree . . . 132
18.5 Support Vector Regression . . . 134
18.6 Multilayer Perceptron . . . 136
18.7 Summary . . . 139

19 How to Use Top Ensemble Machine Learning Algorithms 140
19.1 Ensemble Algorithms Overview . . . 140
19.2 Bootstrap Aggregation . . . 141
19.3 Random Forest . . . 143
19.4 AdaBoost . . . 145
19.5 Voting . . . 147
19.6 Stacked Generalization . . . 149
19.7 Summary . . . 151

20 How To Compare the Performance of Machine Learning Algorithms 152
20.1 Best Machine Algorithm For A Problem . . . 152
20.2 Compare Algorithm Performance in Weka . . . 153
20.3 Design The Experiment . . . 154
20.4 Run The Experiment . . . 156
20.5 Review Experiment Results . . . 157
20.6 Debugging Errors With Experiments . . . 161
20.7 Summary . . . 163

21 How to Tune the Parameters of Machine Learning Algorithms 164
21.1 Improve Performance By Tuning . . . 164
21.2 Algorithm Tuning Experiment Overview . . . 165
21.3 Design The Experiment . . . 165
21.4 Run The Experiment . . . 169
21.5 Review Experiment Results . . . 170
21.6 Summary . . . 173

22 How to Save Your Machine Learning Model and Make Predictions 174
22.1 Tutorial Overview . . . 174
22.2 Finalize a Machine Learning Model . . . 175
22.3 Save Finalized Model To File . . . 176
22.4 Load a Finalized Model . . . 177
22.5 Make Predictions on New Data . . . 179
22.6 Summary . . . 183

III Projects 184

23 How To Work Through a Multiclass Classification Project 185
23.1 Tutorial Overview . . . 185
23.2 Load Dataset . . . 186
23.3 Analyze the Dataset . . . 187
23.4 Evaluate Algorithms . . . 190

23.5 Finalize Model and Present Results . . . 196
23.6 Summary . . . 197

24 How To Work Through a Binary Classification Project 198
24.1 Tutorial Overview . . . 198
24.2 Load the Dataset . . . 198
24.3 Analyze the Dataset . . . 199
24.4 Prepare Views of the Dataset . . . 202
24.5 Evaluate Algorithms . . . 207
24.6 Finalize Model and Present Results . . . 211
24.7 Summary . . . 211

25 How to Work Through a Regression Machine Learning Project 213
25.1 Tutorial Overview . . . 213
25.2 Load the Dataset . . . 214
25.3 Analyze the Dataset . . . 214
25.4 Prepare Views of the Dataset . . . 217
25.5 Evaluate Algorithms . . . 222
25.6 Tune Algorithm Performance . . . 225
25.7 Evaluate Ensemble Algorithms . . . 229
25.8 Finalize Model and Present Results . . . 232
25.9 Summary . . . 232

IV Conclusions 234

26 How Far You Have Come 235

27 Getting More Help 236
27.1 Weka Offline Documentation . . . 236
27.2 Weka Online Documentation . . . 239
27.3 Stack Overflow . . . 240
27.4 Summary . . . 240

Part I

Introduction

Chapter 1

Welcome

Welcome to Machine Learning Mastery With Weka. This book is your guide to applied machine learning. You will discover the step-by-step process that you can use to get started and become good at machine learning for predictive modeling using the Weka platform.

1.1 Applied Machine Learning the Wrong Way

Here is what you should not do when you start in applied machine learning:

- Get really good at the math that underlies machine learning theory.
- Deeply study the underlying theory and parameters for machine learning algorithms.
- Avoid or lightly touch on all of the other tasks needed to complete a real project.

This approach can work for some people, but it is a really slow and a roundabout way of getting to your goal. It teaches you that you need to spend all your time learning how to use individual machine learning algorithms. It also does not teach you the process of building predictive machine learning models that you can actually use to make predictions. Sadly, this is the approach used to teach machine learning that I see in almost all books and online courses on the topic.

1.2 Applied Machine Learning with Weka

This book focuses on a specific sub-field of machine learning called predictive modeling. This is the field of machine learning that is the most useful in industry and the type of machine learning that the Weka platform excels at facilitating.

Unlike statistics, where models are used to understand data, predictive modeling is laser focused on developing models that make the most accurate predictions at the expense of explaining why predictions are made. Unlike the broader field of machine learning that could feasibly be used with data in any format, predictive modeling is primarily focused on tabular data (e.g. tables of numbers like a spreadsheet). This book was written around three themes designed to get you started and practicing applied machine learning effectively and quickly. These three parts are as follows:

- **Weka**: Weka is the very best platform for beginners getting started and practicing applied machine learning.
- **Lessons**: Learn how the subtasks of a machine learning project map onto Weka and the best practice way of working through each task.
- **Projects**: Tie together all of the knowledge from the lessons by working through case study predictive modeling problems.

These are the three pillars of this book that will quickly and effectively take you from where you are now to your goal of confidently working through and delivering results on your own applied machine learning projects.

1.3 Book Overview

This book was carefully designed to quickly and effectively take you from beginner to confident machine learning practitioner capable of working through your own projects end-to-end. As such, this book is divided into 4 parts:

- Part 1: Introduction
- Part 2: Lessons
- Part 3: Projects
- Part 4: Conclusions

1.3.1 Part 1: Introduction

The introduction makes the case that Weka is the best platform for beginners getting started in applied machine learning. It covers:

- Why applied machine learning is so hard and how Weka makes it easy.
- What the Weka machine learning workbench provides.
- How to make best use of Weka by developing a portfolio of completed projects.

After completing this part you will be ready to actually get started learning applied machine learning using the Weka workbench.

1.3.2 Part 2: Lessons

This part provides the meat of the book, providing you with specific instruction on how to use Weka for applied machine learning. Each tutorial is standalone. The benefit of this is that you can dip in to specific lessons if and when you need them, or work through them sequentially one-by-one until you have all the knowledge you need to work through a problem. Each lesson will teach you one key skill in using Weka for applied machine learning. The full list of the 18 lessons provided are as follows:

- **Lesson 01**: How to Download and Install the Weka Machine Learning Workbench.
- **Lesson 02**: A Tour of the Weka Machine Learning Workbench.
- **Lesson 03**: How To Load CSV Machine Learning Data.
- **Lesson 04**: How to Load Standard Machine Learning Datasets.
- **Lesson 05**: How to Better Understand Your Machine Learning Data.
- **Lesson 06**: How to Normalize and Standardize Your Machine Learning Data.
- **Lesson 07**: How to Transform Your Machine Learning Data.
- **Lesson 08**: How To Handle Missing Values In Machine Learning Data.
- **Lesson 09**: How to Perform Feature Selection With Machine Learning Data.
- **Lesson 10**: How to Use Machine Learning Algorithms.
- **Lesson 11**: How To Estimate The Performance of Machine Learning Algorithms.
- **Lesson 12**: How To Estimate A Baseline Performance For Your Models.
- **Lesson 13**: How To Use Top Classification Machine Learning Algorithms.
- **Lesson 14**: How To Use Top Regression Machine Learning Algorithms.
- **Lesson 15**: How to Use Top Ensemble Machine Learning Algorithms.
- **Lesson 16**: How To Compare the Performance of Machine Learning Algorithms.
- **Lesson 17**: How to Tune the Parameters of Machine Learning Algorithms.
- **Lesson 18**: How to Save Your Machine Learning Model and Make Predictions.

After completing all of the lessons, you will be ready to work through standalone projects, end-to-end.

1.3.3 Part 3: Projects

This part contains three end-to-end projects that tie together the lessons from the previous part. Each project focuses on a different type of problem. The projects increase in complexity, starting off easy and straightforward and finish by using many advanced techniques you have learned. The projects you will work through in this part include:

- **Project 01**: Multiclass classification project to predict iris flower species from flower measurements.
- **Project 02**: Binary class classification project to predict the onset of diabetes from patient medical details.
- **Project 03**: Regression project to predict suburban house price from suburb details.

After completing this part, you will have solidified you knowledge of working through applied machine learning projects end-to-end and be ready to take on your own projects.

1.3.4 Part 4: Conclusions

Now that you are ready to take on your own projects, this part takes a moment to look back at how far you have come. The skills of applied machine learning are in great demand and it is important to appreciate exactly what you have learned and how you can bring those skills to your own projects. This part also lists valuable resources that you can consult to get more information and get answers to the inevitable technical questions that will come up.

1.4 Your Outcomes From This Process

This book will lead you from being a developer who is interested in applied machine learning to a developer who has the resources and capability to work through a new dataset end-to-end using Weka and develop accurate predictive models. Specifically, you will know:

- How to work through a small to medium sized dataset end-to-end.
- How to deliver a model that can make accurate predictions on new unseen data.
- How to complete all subtasks of a predictive modeling problem with Weka.
- How to learn new and different techniques in Weka.
- How to get help with Weka.

From here you can start to dive into the specifics of the techniques and algorithms used with the goal of learning how to use them better in order to deliver more accurate predictive models, more reliably in less time.

1.5 What This Book is Not

This book was written for professional developers who want to know how to build reliable and accurate machine learning models.

- **This is not a machine learning textbook**. We will not be getting into the basic theory of machine learning (e.g. induction, bias-variance trade-off, etc.). You are expected to have some familiarity with machine learning basics, or be able to pick them up yourself.
- **This is not an algorithm book**. We will not be working through the details of how specific machine learning algorithms work (e.g. random forest). You are expected to have some basic knowledge of machine learning algorithms or how to pick up this knowledge yourself.
- **This is not a programming book**. We will not be writing any code at all. Weka provides a Java API, but this API will not be covered in this book. We will focus exclusively on developing models using the Weka graphical user interface.

The beauty of Weka is that you can learn the process of applied machine learning and get good at delivering results without a strong background in algorithms or machine learning theory. The details and theory can come later, as you work to get better at the process of applied machine learning and delivering robust predictions and predictive models.

1.6 Summary

I hope you are as excited as me to get started. In this introduction chapter you learned that this book is unconventional. Unlike other books and courses that focus heavily on machine learning algorithms and theory and focus on little else, this book will walk you through each step of a predictive modeling machine learning project.

1.6.1 Next

Let's dive in. The next section will make the case as to why Weka is the best platform for beginners in applied machine learning.

Chapter 2

Rapidly Accelerate Your Progress in Applied Machine Learning With Weka

Why start with Weka over another tool like the R environment or Python for applied machine learning? In this chapter you will discover why Weka is the perfect platform for beginners interested in rapidly getting good at applied machine learning. After reading this chapter you will know:

- Why getting started in applied machine learning is hard.
- The one most important thing to focus on when getting started in applied machine learning.
- How to make best use of Weka when getting started in applied machine learning.

Let's get started.

2.1 Starting in Applied Machine Learning is Hard

When you start out in applied machine learning, there is so much to learn. For example:

- There are the algorithms.
- There is the data.
- There is the specific problem you are working on.
- There is the mathematics behind it all.
- There is the tool that you plan to use.

Often you are convinced that you need to learn a new programming language before you can get started in applied machine learning, like Python or more esoteric languages like Matlab or R. This does not have to be the case. It is so much easier to learn one thing well rather than try, and possibly fail to learn a host of new things.

2.2 Focus on Learning Just One Thing

The one thing to learn when you are starting in machine learning is how to deliver a result. That is, given a problem, how to work through it and deliver a set of predictions or how to deliver a model that can generate predictions. Not just predictions, but accurate predictions that can be delivered robustly and reliably, that you can put your name or your company's name against and in which you can feel confident. This is the most important skill to learn. It often involves steps like:

1. Defining your problem.
2. Preparing your data.
3. Evaluating a suite of algorithms.
4. Improving your results with tuning and ensembles.
5. Finalizing your model and present results.

This is the process of applied machine learning.

2.3 Learn the Process of Applied Machine Learning

The best tool to learn this process is the Weka machine learning workbench. There are 3 main reasons why this is the case:

- **Speed**: you can work your problem fast, giving you more time to try lots of ideas.
- **Focus**: it is just you and your problem, the tool gets out of your way.
- **Coverage**: it provides lots of state-of-the-art algorithms to choose from.

It saves you from the cruft that you can encounter with other platforms. You do not need to spend weeks learning a new language or API, and can focus on learning how to work through problems efficiently and effectively. You can focus on the one valuable thing you need to learn: the process of applied machine learning and delivering a result. Later, you can learn how to use more and different tools.

2.4 How to Best Use Weka

There is a specific way that you can use Weka to best aid you on your machine learning journey.

- **Practice on small in-memory datasets**. These are datasets with hundreds or thousands of instances so they are fast to work with and are standard datasets in the field, so that they are well understood.
- **Practice on different problem types**. Select standard datasets from a range of problem domains, such as biology, physics and advertising, and a range of problem types, such as binary and multiclass classification, regression, unbalanced datasets, and more.

- **Practice by exercising different parts of the tool**. Use a range of different techniques on different problems, including filtering methods, machine learning algorithms and even unsupervised methods like clustering and association rules.

These three simple principles will help you greatly accelerate your progress in developing skills in applied machine learning. Your learning will be focused on working through a problem and delivering a result in the form a set of accurate and reliable predictions or a model that can make ongoing predictions. We will go into more detail on how to make the best use of Weka in Chapter 4. The benefits of this approach will mean that you can greatly outpace others starting out in the field that are:

- Still figuring out how to implement an algorithm from scratch in code.
- Still figuring out how to use an esoteric programming language or API.
- Still figuring out how to setup their environment.

In applied machine learning, fast, reliable and systematic turnaround of results is more important than most other things. For this and more, Weka is your way forward.

2.5 Summary

In this chapter you discovered the importance of the Weka machine learning workbench for beginners in applied machine learning. You learned:

- That getting started in applied machine learning is hard because there is so much to learn.
- That the one most important thing to focus on in applied machine learning is delivering a reliable and robust result.
- That Weka can best be used by practicing on a suite of standard machine learning datasets.

2.5.1 Next

In the next section we will take a closer look at the Weka workbench and the features and benefits it provides to beginners in applied machine learning.

Chapter 3

A Gentle Introduction to the Weka Machine Learning Workbench

Some statistical and machine learning work benches like R provide very advanced tools but require a lot of manual configuration in the form of scripts and programming. The tools can also be fragile, written by and for academics rather than written to be robust and used in production environments. In this chapter you will get a gentle introduction to the Weka machine learning workbench. After reading this chapter, you will know:

- What the Weka machine learning workbench is and what it supports.
- The Weka graphical interface and the best parts to focus on.
- Why Weka is the best tool for beginners who are learning how to work through their own machine learning problems.

Let's get started.

3.1 What is Weka

The Weka machine learning workbench is a modern platform for applied machine learning. Weka is an acronym which stands for Waikato Environment for Knowledge Analysis. It is also the name of a New Zealand bird the Weka. Five of the top features for using Weka are:

- **Open Source**: Weka is released as open source software under the GNU GPL. It is dual licensed and Pentaho Corporation owns the exclusive license to use the platform for business intelligence in their own product.
- **Graphical Interface**: It has a Graphical User Interface (GUI). This allows you to complete your machine learning projects without programming.
- **Command Line Interface**: All features of the software can be used from the command line. This can be very useful if you are looking to take your usage of the platform to the next level and start automating common tasks.

- **Java API**: It is written in Java and provides a API that is well documented and promotes integration into your own applications. Note that the GNU GPL means that in turn your software would also have to be released as GPL.
- **Community**: The interface and algorithms are well documented and the large community using the platform means that there is plenty of support if you need it.

3.2 Introduction to the Weka Graphical Interface

The Weka workbench provides two main ways to work on your problem: The Explorer for playing around and trying things out and the Experiment Environment for controlled experiments.

3.2.1 Weka Explorer

The explorer is where you play around with your data and think about what transforms to apply to your data, what algorithms you want to run as experiments. The Explorer interface is divided into 5 different tabs:

- **Preprocess**: Load a dataset and manipulate the data into a form that you want to work with.
- **Classify**: Select and run classification and regression algorithms to operate on your data.
- **Cluster**: Select and run clustering algorithms on your dataset.
- **Associate**: Run association algorithms to extract insights from your data.
- **Select Attributes**: Run attribute selection algorithms on your data to select those attributes that are relevant to the feature you want to predict.
- **Visualize**: Visualize the relationship between attributes.

3.2.2 Weka Experiment Environment

This interface is for designing experiments with your selection of algorithms and datasets, running experiments and analyzing the results. The tools for analyzing results are powerful, allowing you to consider and compare results that are statistically significant over multiple runs. The Experiment interface is divided into 3 different tabs:

- **Setup** for designing your controlled experiments.
- **Run** for executing your experiments.
- **Analyse** for analyzing the results from your experiments.

We will take a tour of the Weka interface in Chapter 6 where you will get to use each tool for the first time, before we start making heavy use of them in later lessons.

3.3 Why You Should Use Weka

The main reason Weka is so good for beginners is that you can work through the process of applied machine learning using the graphical interface without having to write a single line of code. This is a big deal because getting a handle on the process, handling data and experimenting with algorithms is what a beginner should be learning about, not learning yet another scripting language. Here are some additional reasons:

- It provides a simple graphical user interface that encapsulates the process of applied machine learning.
- It facilitates algorithm and dataset exploration as well as rigorous experiment design and analysis.
- It is free and open source, meaning you can download it and start using it now.
- It is cross-platform and runs on Windows, Mac OS X and Linux, meaning it will run on whatever environment you're using.
- It contains state-of-the-art algorithms with an impressive list of Decision Trees, Rule-Based Algorithms and Ensemble methods, as well as others.

3.4 Summary

In this chapter you received a gentle introduction to the Weka machine learning workbench. You learned:

- That Weka is a free open source platform for applied machine learning.
- That the key graphical interfaces you need to learn are the Explorer and the Experimenter.
- That the main reason Weka is great for beginners is because you can focus on applied machine learning without also learning yet another programming language or library.

3.4.1 Next

In the next section we will take a closer look at how you can incorporate Weka into a broader strategy for learning, demonstrating and mastering applied machine learning.

Chapter 4

How to Make Best Use of Weka For Applied Machine Learning

Weka is an easy to use and powerful platform for applied machine learning, but how can you make the most out of using it? In this chapter you will discover how you can best use the Weka platform to accelerate your journey into applied machine learning. After reading this chapter you will know:

- The number one benefit of using the Weka machine learning workbench and how to harness it.
- The importance of developing a portfolio to demonstrate your developing skills in applied machine learning.
- The value is practicing on a range of varied standard machine learning datasets in order to build out your portfolio.

Let's get started.

4.1 Harness The Number One Benefit of Weka

The Weka graphical user interface can make exploring and working through a machine learning problem fast and easy. So much so, that it can be a lot of fun. You can leverage this enormous benefit of Weka to develop a portfolio of completed projects that you can use to demonstrate your growing skills in applied machine learning to current and prospective employers.

The most productive way of building up a portfolio of completed projects is to work on a suite of standard machine learning datasets. The UCI Machine Learning repository hosts hundreds of standard machine learning datasets that you can use for practice that are small, fit into memory and are well understood. You can use this strategy to harnessing the ease of use of the Weka platform to develop a portfolio of completed projects and learn how to best use a variety of different techniques on a mix of different problem types. It can help to accelerate your progress in developing skills in applied machine learning and demonstrate those skills with tangible work product. Let's dive a little deeper.

4.2 Build a Machine Learning Portfolio

If you are just starting out as a beginner in machine learning or you are a hardened veteran, a machine learning portfolio can keep you on track and demonstrate your skills. Creating a machine learning portfolio is a valuable exercise. Building up a collection of completed machine learning projects can keep you focused, motivated and be leveraged on future projects.

- **Focus**: Each project has a well defined purpose and end point. Small projects constrained in effort and resources can keep your velocity high.
- **Knowledge Base**: The corpus of completed projects provide a knowledge base for you to reflect on and leverage as you push into projects further from your comfort zones.
- **Trajectory**: There are so many shiny things to investigate, reminding yourself that you are looking for a consistent collection of projects can be used as a lever to keep you on track.

A machine learning portfolio is a collection of completed independent projects, each of which uses machine learning in some way. The portfolio presents the collection of projects and allows review of individual projects. Five properties of an effective machine learning portfolio include:

- **Accessible**: I advocate making the portfolio public in the form of a publicly accessible webpage or collection of public code repositories. You want people to find, read, comment on, and use your work if possible.
- **Small**: Each project should be small in scope in terms of effort, resources, and most importantly, your time (10-to-20 hours). You're busy and it's hard to keep focus.
- **Completed**: Small projects help you have finished projects. Set a modest project objective and achieve it. Like mini-experiments, you present the findings of your successes and your failures, they are all useful learnings.
- **Independent**: Each project should be independent so that it can be understood in isolation. This does not mean you can't leverage prior work, it means that the project makes sense on it's own as a standalone piece of work.
- **Understandable**: Each project must clearly and effectively communicate it's purpose and findings (at the very least). Spend some time and make sure a fresh set of eyes understand what you did and why it matters.

The idea of a project portfolio is not new, it was baked into websites like GitHub. What is interesting is that in recent interviews with data scientists and managers, portfolios are being requested even desired along with participation in machine learning competitions and completion of online training. Like sample code in programming interviews, Machine Learning portfolios are getting to become a serious part of hiring.

4.3 Practice On Small In-Memory Datasets

It is hard to get good practice for applied machine learning. It is one thing to practice on the same dataset over and over, and you need to move on and challenge your developing skills The best way to practice is to choose standard machine learning datasets that have specific traits. Select traits that you will encounter and need to address when you start working on problems of your own such as:

- Different types of supervised learning such as classification and regression.
- Different sized datasets from tens, hundreds, thousands and millions of instances.
- Different numbers of attributes from less than ten, tens, hundreds and thousands of attributes.
- Different attribute types from real, integer, categorical, ordinal and mixtures.
- Different domains that force you to quickly understand and characterize a new problem in which you have no previous experience.

You can create a program of traits to study and learn about and the algorithms you need to address them, by designing a program of test problem datasets to work through. Such a program has a number of practical requirements, for example:

- **Real-World**: The datasets should be drawn from the real world (rather than being contrived). This will keep them interesting and introduce the challenges that come with real data.
- **Small**: The datasets need to be small so that you can inspect and understand them and that you can run many models quickly to accelerate your learning cycle.
- **Well-Understood**: There should be a clear idea of what the data contains, why it was collected, what the problem is that needs to be solved so that you can frame your investigation.
- **Baseline**: It is also important to have an idea of what algorithms are known to perform well and the scores they achieved so that you have a useful point of comparison. This is important when you are getting started and learning because you need quick feedback as to how well you are performing (close to state-of-the-art or something is broken).
- **Plentiful**: You need many datasets to choose from, both to satisfy the traits you would like to investigate and (if possible) your natural curiosity and interests.

For beginners, you can get everything you need and more in terms of datasets to practice on from the UCI Machine Learning Repository. The UCI Machine Learning Repository is a database of machine learning problems that you can access for free[1]. It is hosted and maintained by the Center for Machine Learning and Intelligent Systems at the University of California at Irvine[2]. It was originally created by David Aha as a graduate student at UC Irvine. For more

[1] https://archive.ics.uci.edu/ml/index.html
[2] http://cml.ics.uci.edu/

than 25 years it has been the go-to place for machine learning researchers and machine learning practitioners that need a dataset.

Each dataset gets its own webpage that lists all the details known about it including any relevant publications that investigate it. The datasets themselves can be downloaded as ASCII files, often in the useful CSV format. For example, here is the webpage for the Abalone Data Set that involves the prediction of the age of abalone from their physical measurements[3].

4.4 Benefits of the Repository

Some beneficial features of the library include:

- Almost all datasets are drawn from the domain (as opposed to being synthetic), meaning that they have real-world qualities.
- Datasets cover a wide range of subject matter from biology to particle physics.
- The details of datasets are summarized by aspects like attribute types, number of instances, number of attributes and year published that can be sorted and searched.
- Datasets are well studied which means that they are well known in terms of interesting properties and expected *good* results. This can provide a useful baseline for comparison.
- Most datasets are small (hundreds to thousands of instances) meaning that you can readily load them in a text editor or Excel and review them, you can also easily model them quickly on your workstation.

Browse the 300+ datasets using this handy table that supports sorting and searching[4]. We will cover how to load CSV files such as those from the UCI Machine Learning Repository in Chapter 7, and how to load the standard datasets that are distributed with Weka in Chapter 8.

4.5 Summary

In this chapter you discovered how you can harness the best quality of the Weka machine learning workbench. You learned:

- That the best quality of Weka is ease of use and how quickly you can work through a problem end-to-end.
- That you can harness the ease of use of Weka to develop a portfolio of completed projects that you can use to demonstrate your skills in applied machine learning.
- That you can leverage the large number of standard machine learning datasets on the UCI Machine Learning Repositories website in order to challenge your developing skills and contribute your growing portfolio.

[3] http://archive.ics.uci.edu/ml/datasets/Abalone
[4] http://archive.ics.uci.edu/ml/datasets.html

4.5.1 Next

This is the end of Part I. Next we will start the first lesson in Part II of this book by downloading and installing the Weka machine learning workbench.

Part II

Lessons

Chapter 5

How to Download and Install the Weka Machine Learning Workbench

The Weka machine learning workbench is a powerful and yet easy to use platform for predictive modeling. In this lesson you will discover how you can install Weka on your workstation fast, and get started with machine learning. After reading this lesson you will know:

- How to install the all-in-one version of Weka for Windows or Mac.
- How to install Java and Weka separately on Windows or Mac.
- How to install Weka on Linux and other platforms.

Let's get started.

5.1 Download Weka

All versions of Weka can be downloaded from the Weka download webpage[1]. Select the version of Weka that you would like to install then visit the Weka download page to locate and download your preferred version of Weka. Your options include:

- Install the all-in-one version of Weka for Windows or Mac OS X.
- Install Java and Weka separately for Windows or Mac OS X.
- Install the standalone version of Weka for Linux and other platforms.

5.2 Install The All-In-One Version of Weka

Weka provides an all-in-one installation version for Windows and Mac OS X. This installation includes both the Weka platform that you can use for predictive modeling, as well as the version of Java needed to run the Weka platform.

[1] http://www.cs.waikato.ac.nz/ml/weka/downloading.html

5.2.1 Windows

On windows the all-in-one version of Weka is provided as a self-extracting executable. You must choose whether you would like the 32-bit version of the package or the 64-bit version of the package. If you have a modern version of Windows, you should select the 64-bit version. On the Weka download webpage, these packages are called:

- Self-extracting executable for 64-bit Windows that includes Oracle's 64-bit Java.
- Self-extracting executable for 32-bit Windows that includes Oracle's 32-bit Java.

The download is about 100 megabytes. After you have downloaded the package, double click on the icon to start the installation process. Follow the prompts for the installation and Weka will be added to your Program Menu. Start Weka by clicking on the bird icon.

5.2.2 Mac OS X

On OS X the all-in-one version of Weka is provided as a disk image. On the Weka download webpage, this package is called:

- Disk image for OS X that contains a Mac application including Oracle's Java.

The download is about 120 megabytes. The disk image includes two versions of Weka, one with the Java version bundled and one standalone. I recommend installing both. Drag both the folder and the icon into your Applications folder.

Figure 5.1: Expanded Weka Disk image for OS X

- Start Weka by clicking on the bird icon.

5.3 Install Java and Weka Separately

You may already have the Java Runtime Environment or Java Development Kit installed on your workstation or you may like to install Java separately from Weka so that you can use Java with other applications. Weka provides a version that you can download that does not include the Java Runtime Environment. I recommend this installation of Weka if you would like to access the data files and documentation provided with the Weka installation. Weka requires at least Java 1.7 installed. If you do not have Java installed and would like to install Java separately from Weka, you can download Java from the Java Download webpage[2]. The webpage will automatically determine the version of Java you need for your workstation and download the latest version. The Java download is about 60 megabytes.

5.3.1 Windows

Weka provides a version for Windows that does not include Java. You must choose whether you would like the 32-bit version of the package or the 64-bit version of the package. If you have a modern version of Windows, you should select the 64-bit version. On the Weka downloads page, this version is named as follows:

- Self-extracting executable for 64-bit Windows without a Java VM.
- Self-extracting executable for 32-bit Windows without a Java VM.

The download is about 50 megabytes. After you have downloaded the package, double click to start the installation process. Follow the prompts for the installation and Weka will be added to your Program Menu. Start Weka by clicking on the bird icon.

5.3.2 Mac OS X

There is only a single download version of Weka for OS X. It is a disk image that includes both the version of Weka bundled with Java as well as the standalone version. On the Weka download webpage, this package is called:

- Disk image for OS X that contains a Mac application including Oracle's Java.

The download is about 120 megabytes. Open the disk image and drag the standalone version of Weka (the folder) into your Applications folder.

[2] https://java.com/en/download/

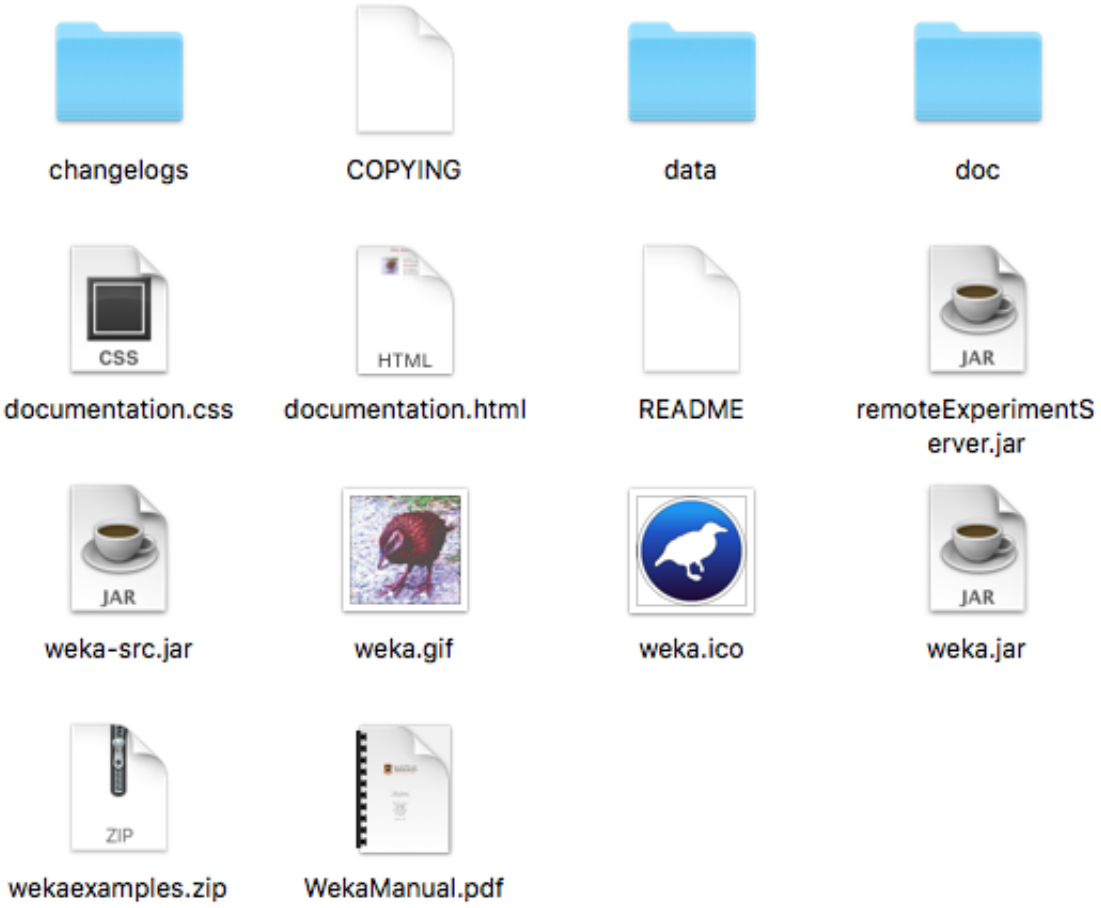


Figure 5.2: Weka Installation Directory

Start Weka by double clicking on the `weka.jar` file. You can also start Weka on the command line, assuming Java is in your path.

- 1. Change directory into your Weka installation directory. For example

```
cd /Applications/weka-3-8-0
```

Listing 5.1: Change directory to the installation location.

- 2. Start the Java virtual machine with the `weka.jar` file. For example:

```
java -jar weka.jar
```

Listing 5.2: Start Weka manually.

5.4 Install Weka On Linux And Other Platforms

Weka also provides a standalone version that you can install on Linux and other platforms. Weka runs on Java and can be used on all platforms that support Java. It is a zip file and has the following name of the Weka download webpage:

- Zip archive containing Weka.

Download the zip file and unzip it. You can also start Weka on the command line, assuming Java is in your path.

```
  35147 14 Apr 12:58 COPYING
  16171 14 Apr 12:58 README
6621937 14 Apr 12:58 WekaManual.pdf
   1938 14 Apr 12:58 changelogs
    918 14 Apr 12:58 data
    578 14 Apr 12:58 doc
    510 14 Apr 12:58 documentation.css
   1863 14 Apr 12:58 documentation.html
  42900 14 Apr 12:58 remoteExperimentServer.jar
10759024 14 Apr 12:58 weka-src.jar
  30414 14 Apr 12:58 weka.gif
 359270 14 Apr 12:58 weka.ico
10997325 14 Apr 12:58 weka.jar
14758799 14 Apr 12:58 wekaexamples.zip
```

Figure 5.3: Weka Installation Files

As above, you can change directory into the Weka installation location and start Weka on the command line.

5.5 Summary

In this lesson you discovered how you can download and install the Weka machine learning workbench. Specifically, you learned:

- How to install Weka using the all-in-one version on Windows and Mac OS X.
- How to install Weka using the standalone version on Windows and Mac OS X.
- How to install Weka using the standalone version on Linux and other platforms.

5.5.1 Next

Now that we have Weka installed, in the next lesson we will take a tour of the Weka graphical user interface and pay particular attention to the *Weka Explorer* and *Weka Experiment Environment*.

Chapter 6

A Tour of the Weka Machine Learning Workbench

Weka is an easy to use and powerful machine learning platform. It provides a large number of machine learning algorithms, feature selection methods and data preparation filters. In this lesson you will discover the Weka machine learning workbench and take a tour of the key interfaces that you can use on your machine learning projects. After reading this lesson you will know about:

- The interfaces supported by the Weka machine learning workbench.
- Those interfaces that are recommended for beginners to work through their problems, and those that are not.
- How to at least click through each key interface you will need in Weka and generate a result.

Let's get started.

6.1 Weka GUI Chooser

The entry point into the Weka interface is the *Weka GUI Chooser*. It is an interface that let's you choose and launch a specific Weka environment.

Figure 6.1: Screenshot of the *Weka GUI Chooser*.

In addition to providing access to the core Weka tools, it also has a number of additional utilities and tools provided in the menu. There two important utilities to note in the *Tools* menu:

- 1. The *Package Manager* which let's you browse and install third party add-ons to Weka such as new algorithms.

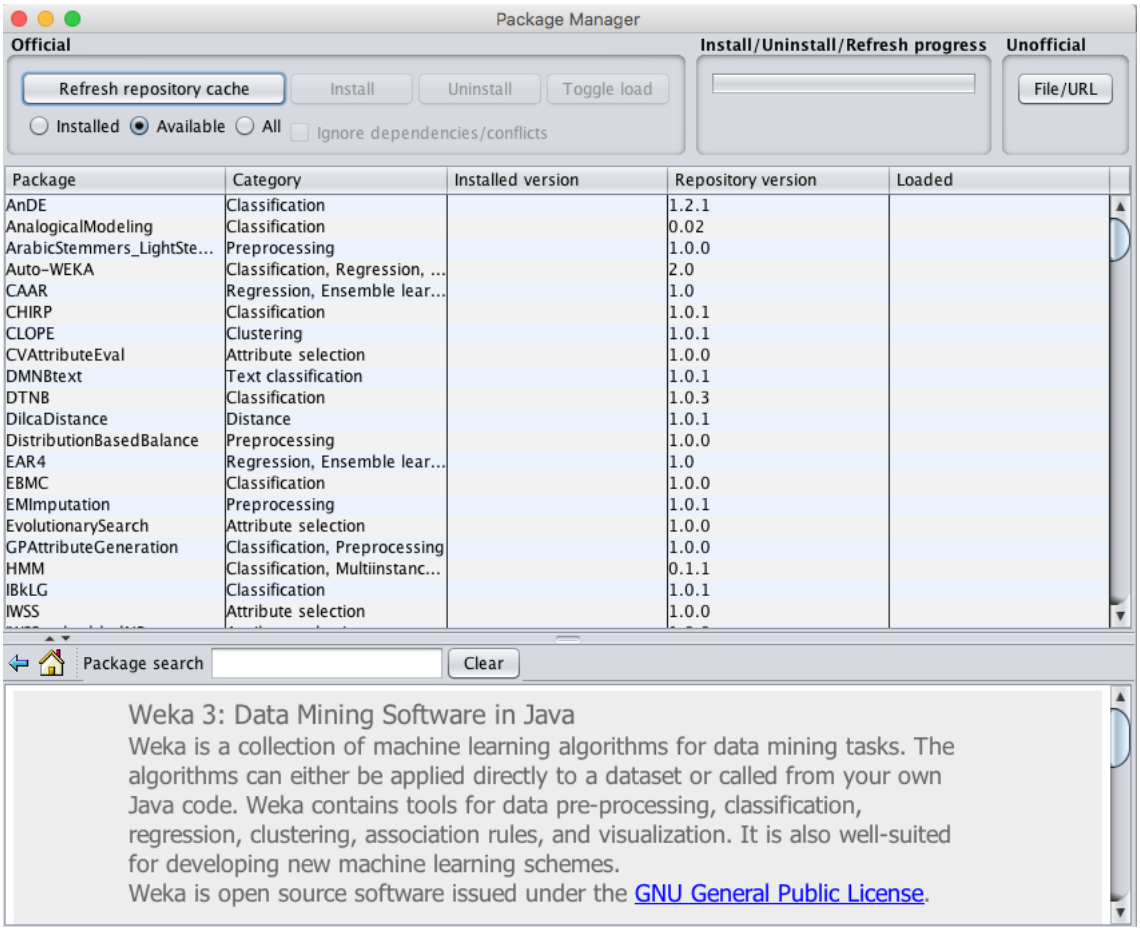
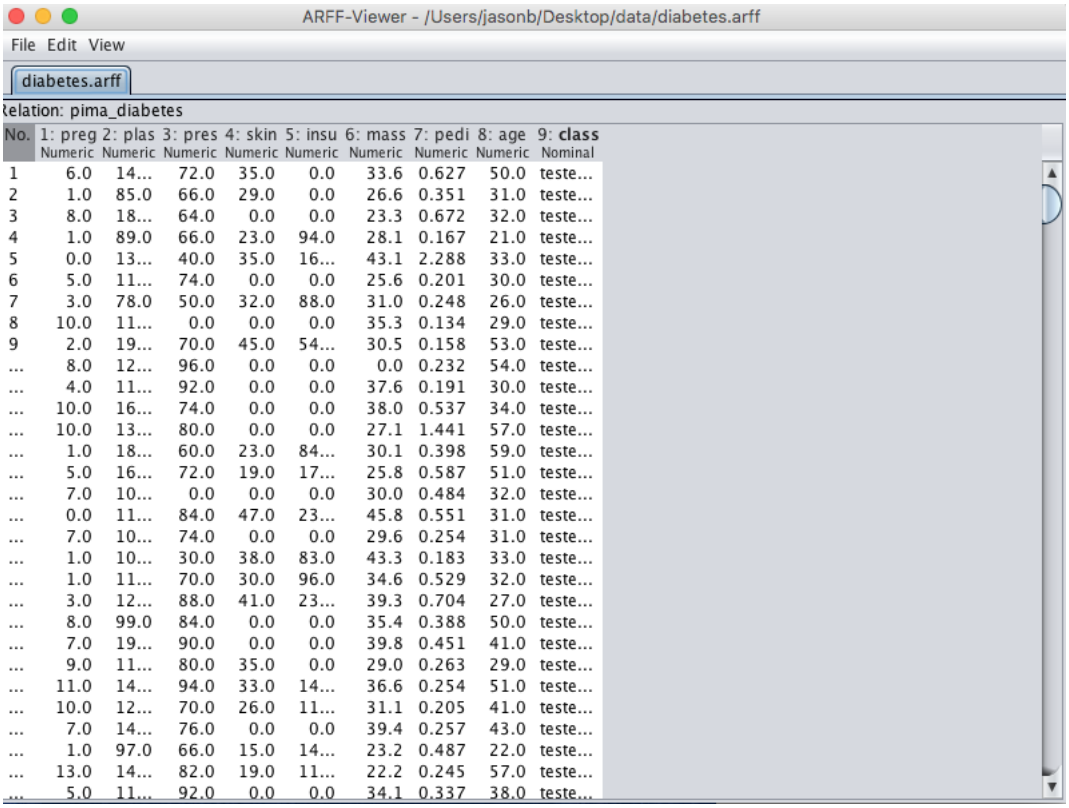


Figure 6.2: Screenshot of the Weka *Package Manager*.

- 2. The *ARFF-Viewer* that allows you to load and transform datasets and save them in ARFF format.



No.	1: preg Numeric	2: plas Numeric	3: pres Numeric	4: skin Numeric	5: insu Numeric	6: mass Numeric	7: pedi Numeric	8: age Numeric	9: class Nominal
1	6.0	14...	72.0	35.0	0.0	33.6	0.627	50.0	teste...
2	1.0	85.0	66.0	29.0	0.0	26.6	0.351	31.0	teste...
3	8.0	18...	64.0	0.0	0.0	23.3	0.672	32.0	teste...
4	1.0	89.0	66.0	23.0	94.0	28.1	0.167	21.0	teste...
5	0.0	13...	40.0	35.0	16...	43.1	2.288	33.0	teste...
6	5.0	11...	74.0	0.0	0.0	25.6	0.201	30.0	teste...
7	3.0	78.0	50.0	32.0	88.0	31.0	0.248	26.0	teste...
8	10.0	11...	0.0	0.0	0.0	35.3	0.134	29.0	teste...
9	2.0	19...	70.0	45.0	54...	30.5	0.158	53.0	teste...
...	8.0	12...	96.0	0.0	0.0	0.0	0.232	54.0	teste...
...	4.0	11...	92.0	0.0	0.0	37.6	0.191	30.0	teste...
...	10.0	16...	74.0	0.0	0.0	38.0	0.537	34.0	teste...
...	10.0	13...	80.0	0.0	0.0	27.1	1.441	57.0	teste...
...	1.0	18...	60.0	23.0	84...	30.1	0.398	59.0	teste...
...	5.0	16...	72.0	19.0	17...	25.8	0.587	51.0	teste...
...	7.0	10...	0.0	0.0	0.0	30.0	0.484	32.0	teste...
...	0.0	11...	84.0	47.0	23...	45.8	0.551	31.0	teste...
...	7.0	10...	74.0	0.0	0.0	29.6	0.254	31.0	teste...
...	1.0	10...	30.0	38.0	83.0	43.3	0.183	33.0	teste...
...	1.0	11...	70.0	30.0	96.0	34.6	0.529	32.0	teste...
...	3.0	12...	88.0	41.0	23...	39.3	0.704	27.0	teste...
...	8.0	99.0	84.0	0.0	0.0	35.4	0.388	50.0	teste...
...	7.0	19...	90.0	0.0	0.0	39.8	0.451	41.0	teste...
...	9.0	11...	80.0	35.0	0.0	29.0	0.263	29.0	teste...
...	11.0	14...	94.0	33.0	14...	36.6	0.254	51.0	teste...
...	10.0	12...	70.0	26.0	11...	31.1	0.205	41.0	teste...
...	7.0	14...	76.0	0.0	0.0	39.4	0.257	43.0	teste...
...	1.0	97.0	66.0	15.0	14...	23.2	0.487	22.0	teste...
...	13.0	14...	82.0	19.0	11...	22.2	0.245	57.0	teste...
...	5.0	11...	92.0	0.0	0.0	34.1	0.337	38.0	teste...

Figure 6.3: Screenshot of the Weka *ARFF-Viewer*.

6.2 Weka Explorer

The *Weka Explorer* is designed to investigate your machine learning dataset. It is useful when you are thinking about different data transforms and modeling algorithms that you could investigate with a controlled experiment later. It is excellent for getting ideas and playing what-if scenarios. The interface is divided into 6 tabs, each with a specific function:

The *Preprocess* tab is for loading your dataset and applying filters to transform the data into a form that better exposes the structure of the problem to the modeling processes. Also provides some summary statistics about loaded data. Load a standard dataset in the `data/` directory of your Weka installation, specifically `data/breast-cancer.arff`. This is a binary classification problem that we will use on this tour. You will learn more about this dataset in Section 8.2.2.

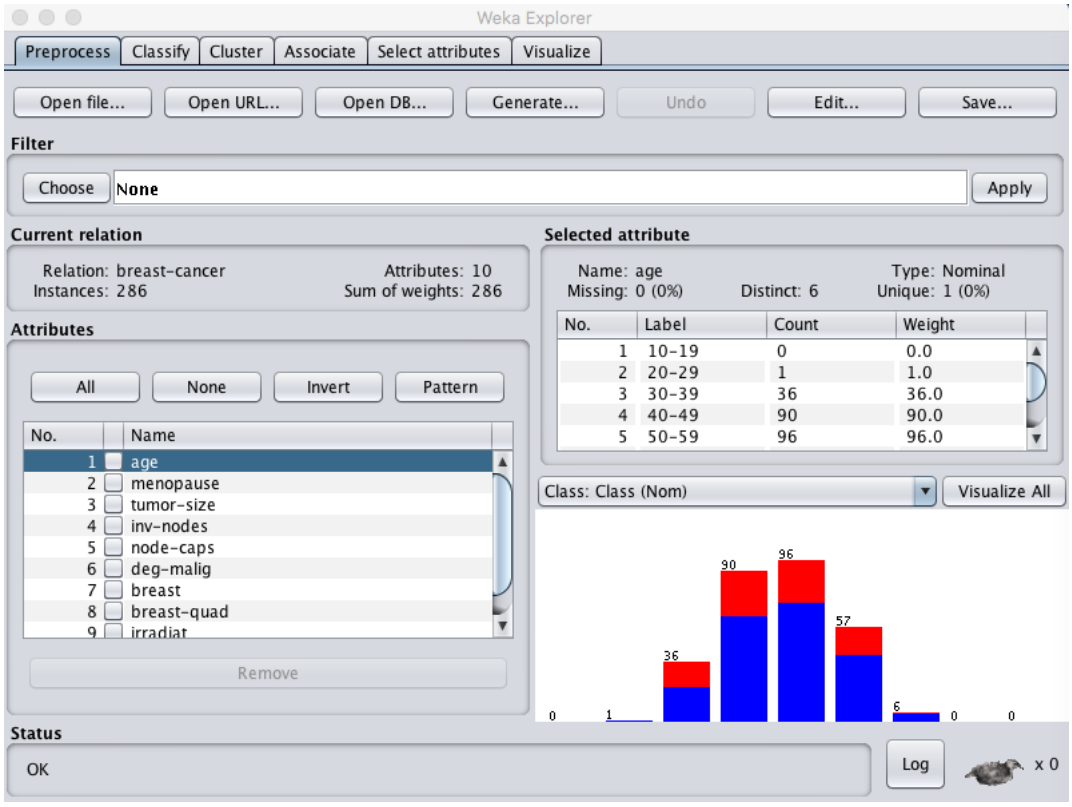


Figure 6.4: Screenshot of the *Weka Explorer Preprocess* Tab.

The *Classify* tab is for training and evaluating the performance of different machine learning algorithms on your classification or regression problem. Algorithms are divided up into groups, results are kept in a result list and summarized in the main *Classifier output* pane.

- Click the *Start* button to run the *ZeroR* classifier on the dataset and summarize the results.

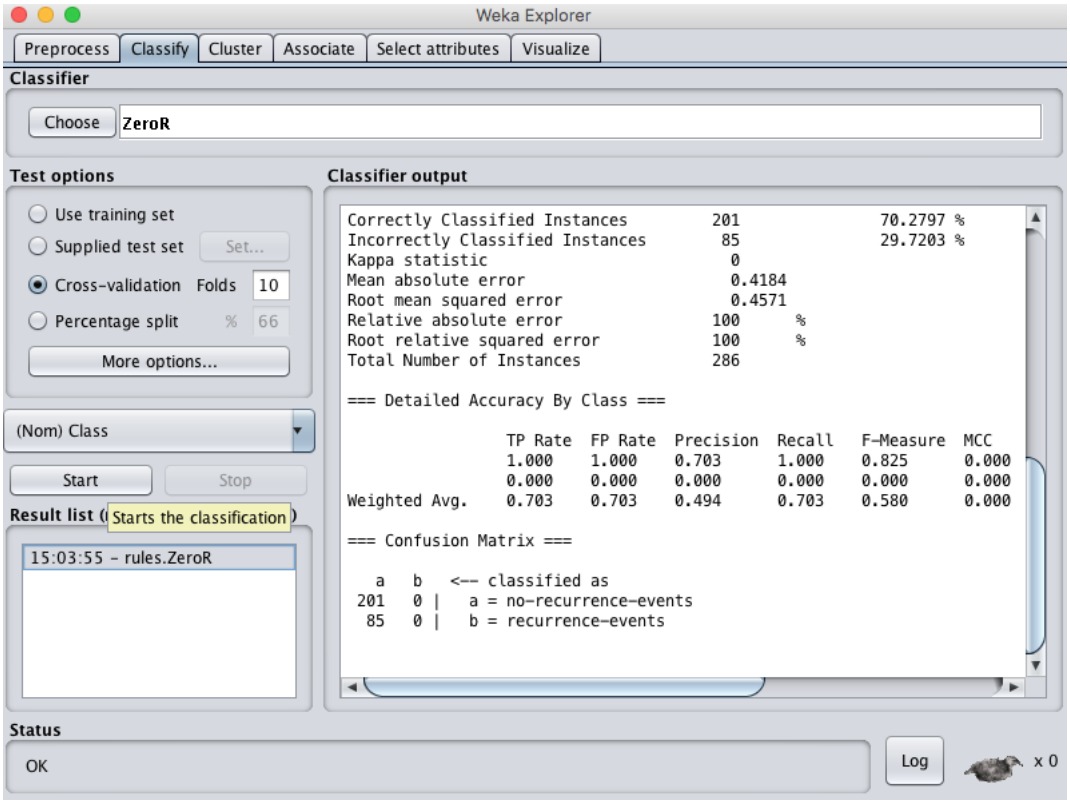


Figure 6.5: Screenshot of the *Weka Explorer Classify* Tab.

The *Cluster* tab is for training and evaluating the performance of different unsupervised clustering algorithms on your unlabeled dataset. Like the *Classify* tab, algorithms are divided into groups, results are kept in a result list and summarized in the main *Clusterer output* pane.

- Click the *Start* button to run the *EM* clustering algorithm on the dataset and summarize the results.

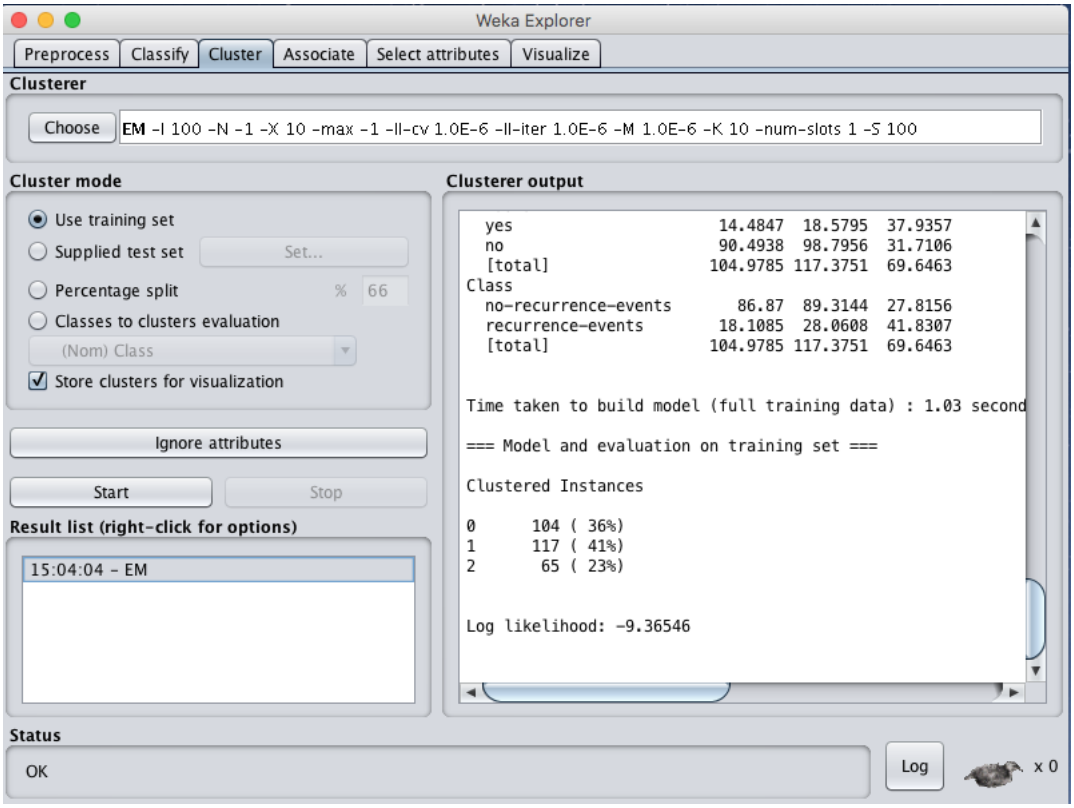


Figure 6.6: Screenshot of the *Weka Explorer Cluster* Tab.

The *Associate* tab is for automatically finding associations in a dataset. The techniques are often used for market basket analysis type data mining problems and require data where all attributes are categorical.

- Click the *Start* button to run the *Apriori* association algorithm on the dataset and summarize the results.

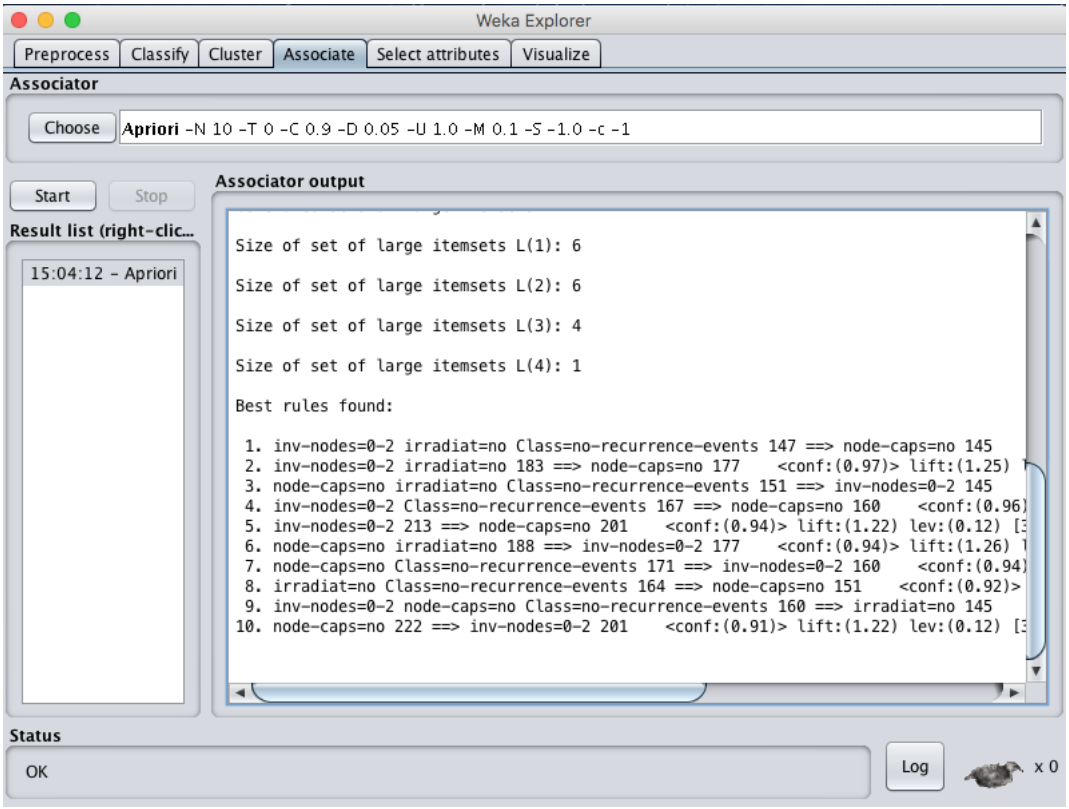


Figure 6.7: Screenshot of the *Weka Explorer Associate* Tab.

The *Select attributes* tab is for performing feature selection on the loaded dataset and identifying those features that are most likely to be relevant in developing a predictive model.

- Click the *Start* button to run the *CfsSubsetEval* algorithm with a *BestFirst* search on the dataset and summarize the results.

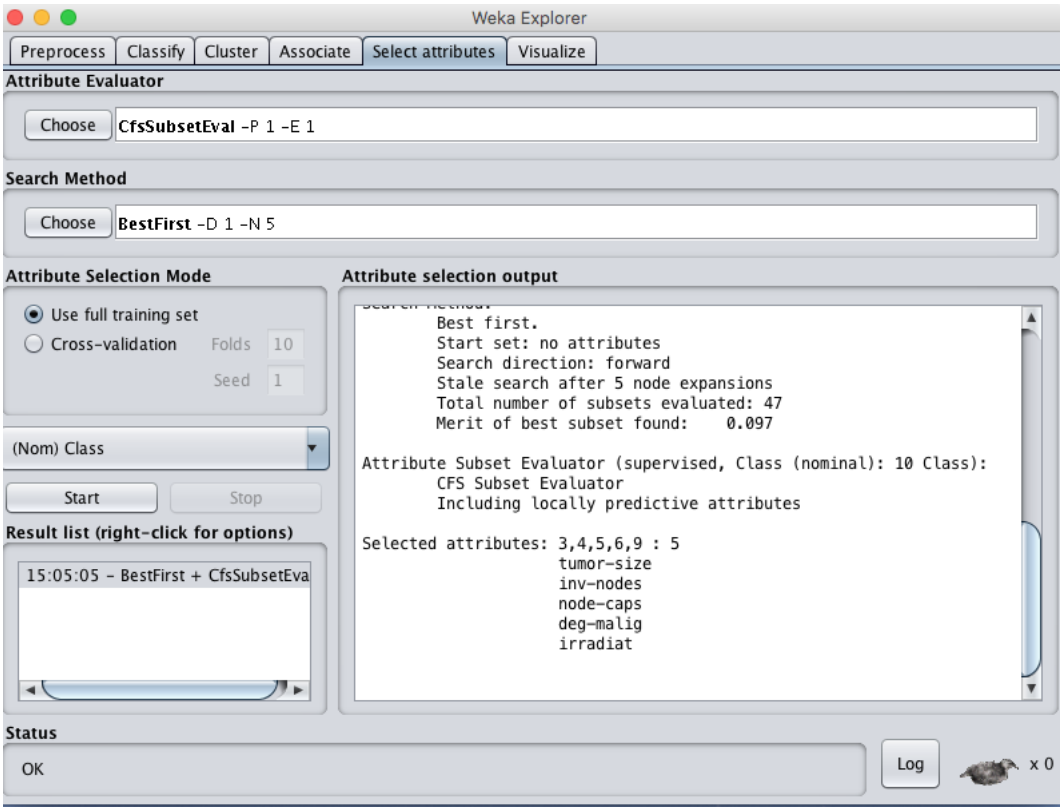


Figure 6.8: Screenshot of the *Weka Explorer Select Attributes* Tab.

The *Visualize* tab is for reviewing pairwise scatter plot matrix of each attribute plotted against every other attribute in the loaded dataset. It is useful to get an idea of the shape and relationship of attributes that may aid in data filtering, transformation and modeling.

- Increase the *PointSize* and the *Jitter* and click the *Update* button to set an improved plot of the categorical attributes of the loaded dataset.

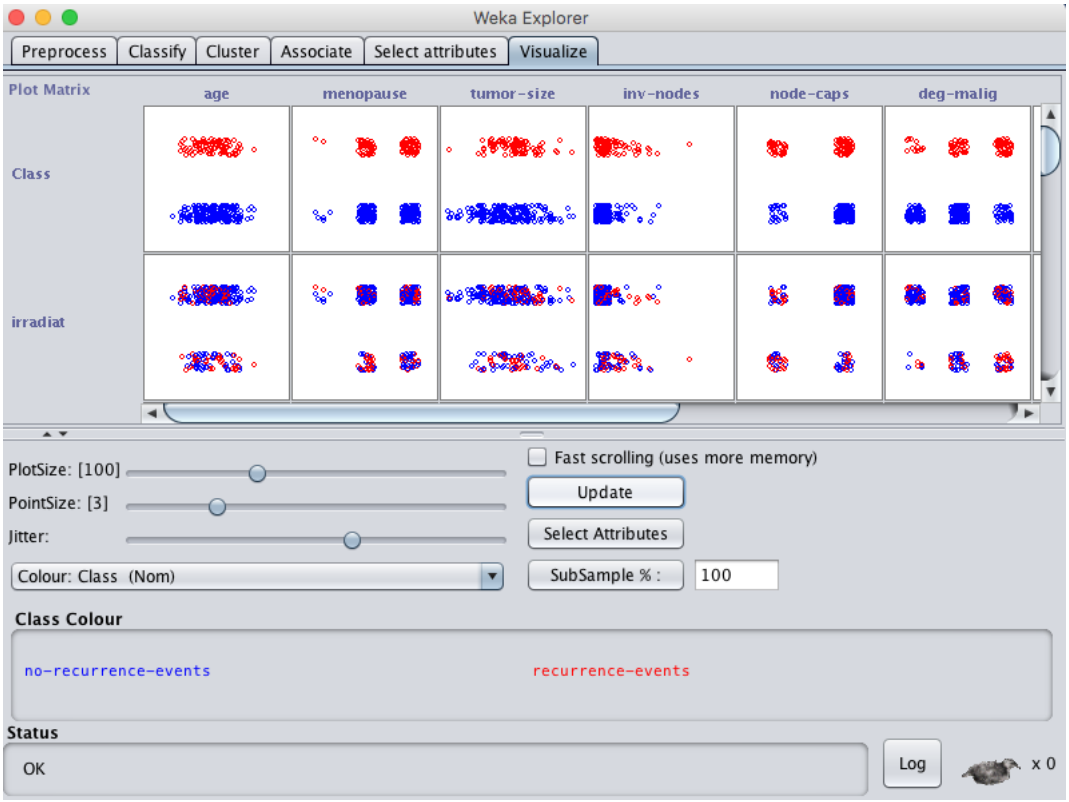


Figure 6.9: Screenshot of the *Weka Explorer* Visualize Tab.

6.3 Weka Experiment Environment

The *Weka Experiment Environment* is for designing controlled experiments, running them, then analyzing the results collected. It is the next step after using the *Weka Explorer*, where you can load up one or more views of your dataset and a suite of algorithms and design an experiment to find the combination that results in the best performance. The interface is split into 3 tabs.

The *Setup* tab is for designing an experiment. This includes the file where results are written, the test setup in terms of how algorithms are evaluated, the datasets to model and the algorithms to model them. The specifics of an experiment can be saved for later use and modification. In this section we will use the onset of diabetes binary classification problem. You will learn more about this dataset in Section 8.2.1.

- Click the *New* button to create a new experiment.
- Click the *Add New...* button in the *Datasets* pane and select the `data/diabetes.arff` dataset.
- Click the *Add New...* button in the *Algorithms* pane and click *OK* to add the *ZeroR* algorithm.

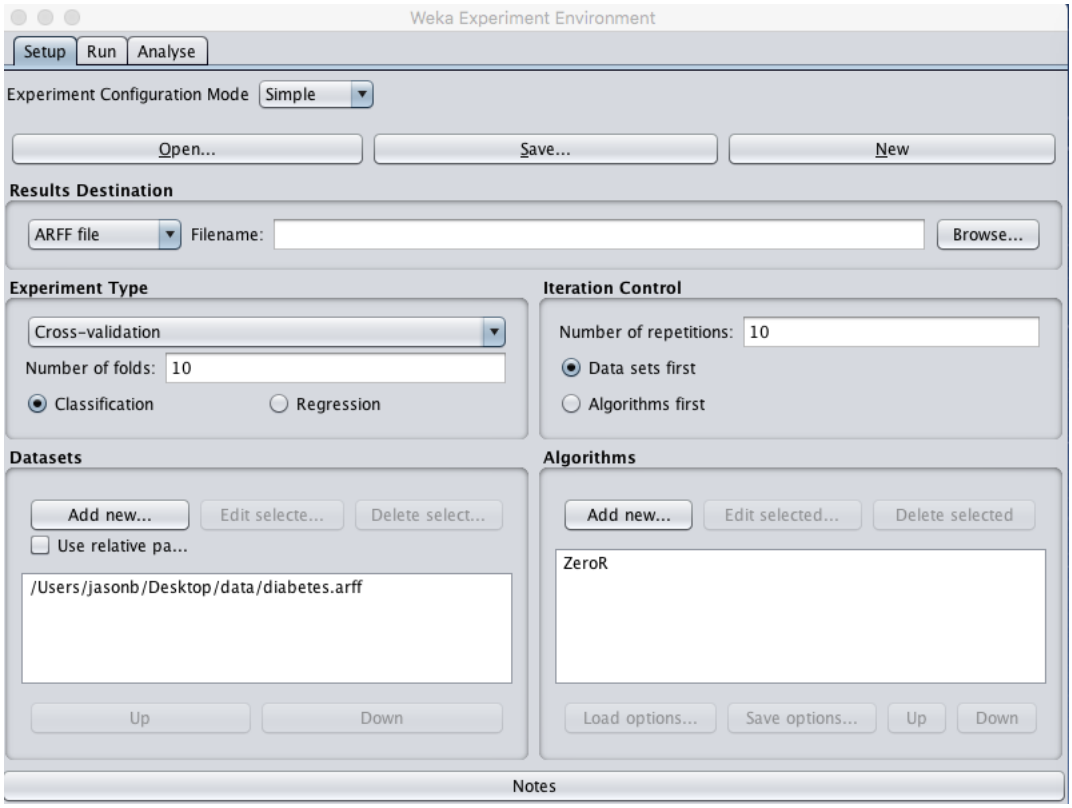


Figure 6.10: Screenshot of the *Weka Experiment Environment Setup* Tab.

The *Run* tab is for running your designed experiments. Experiments can be started and stopped. There is not a lot to it.

- Click the *Start* button to run the small experiment you designed.

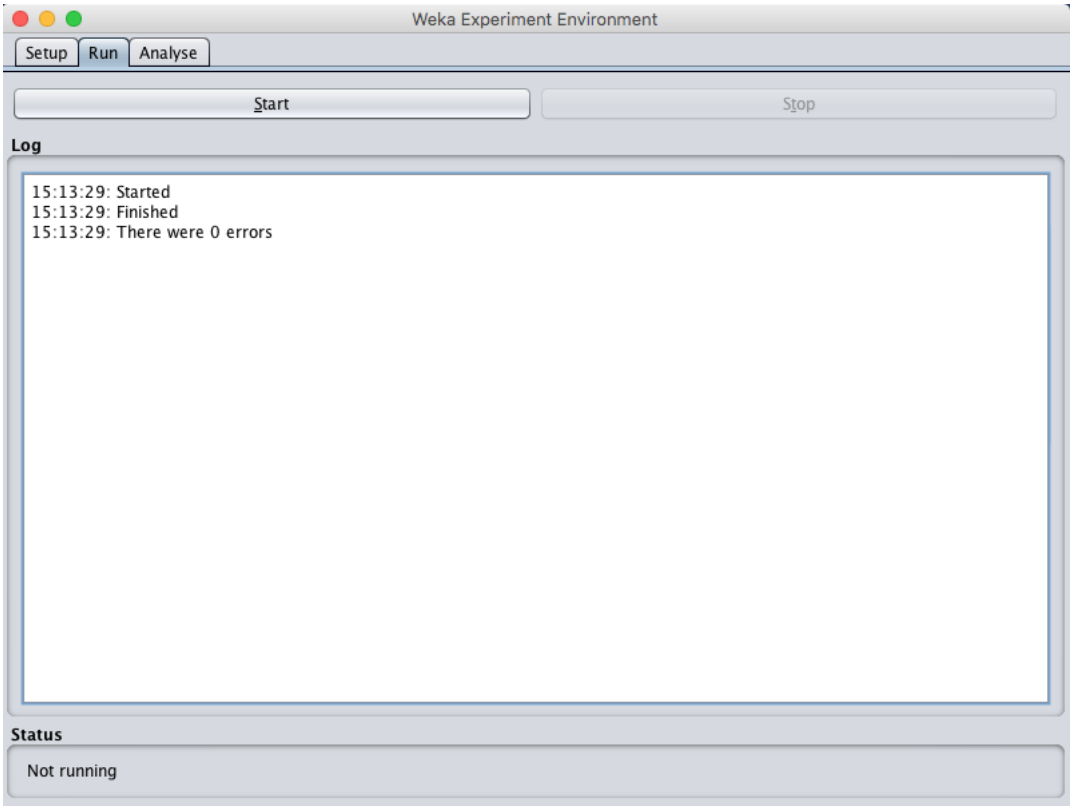


Figure 6.11: Screenshot of the *Weka Experiment Environment Run* Tab.

The *Analyse* tab is for analyzing the results collected from an experiment. Results can be loaded from a file, from the database or from an experiment just completed in the tool. A number of performance measures are collected from a given experiment which can be compared between algorithms using tools like statistical significance.

- Click the *Experiment* button the *Source* pane to load the results from the experiment you just ran.
- Click the *Perform Test* button to summary the classification accuracy results for the single algorithm in the experiment.

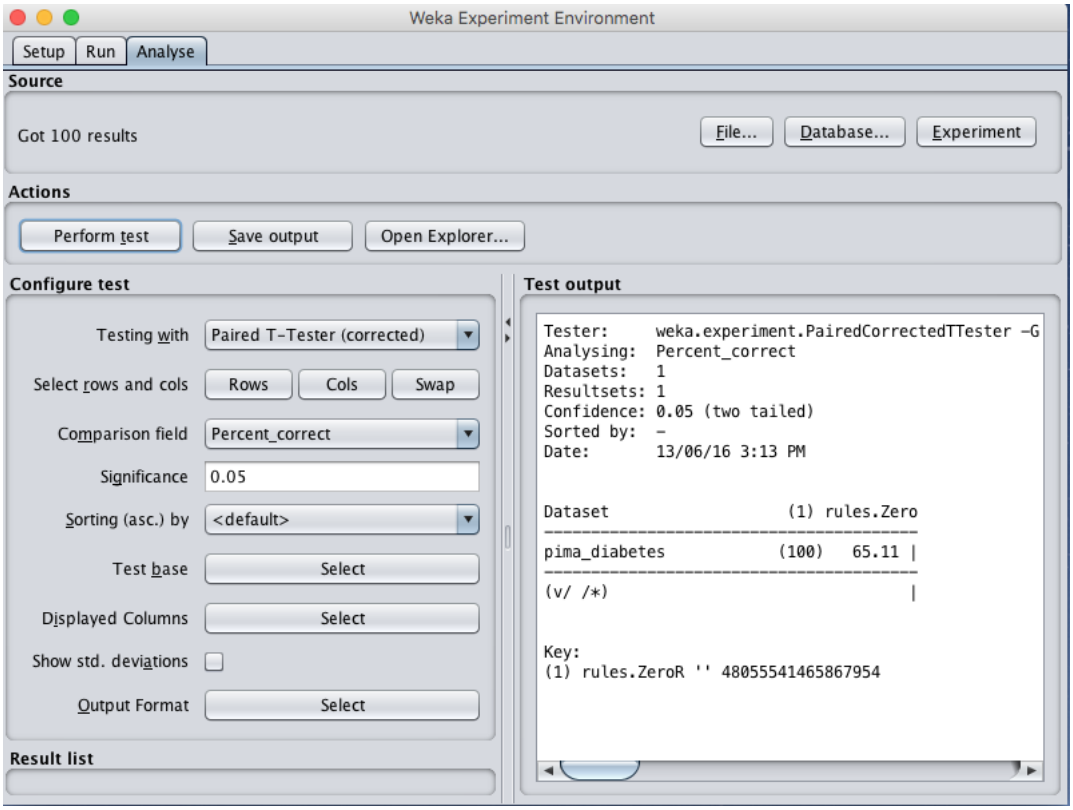


Figure 6.12: Screenshot of the *Weka Experiment Environment Analyse* Tab.

6.4 Weka KnowledgeFlow Environment

The *Weka KnowledgeFlow Environment* is a graphical workflow tool for designing a machine learning pipeline from data source to results summary, and much more. Once designed, the pipeline can be executed and evaluated within the tool.

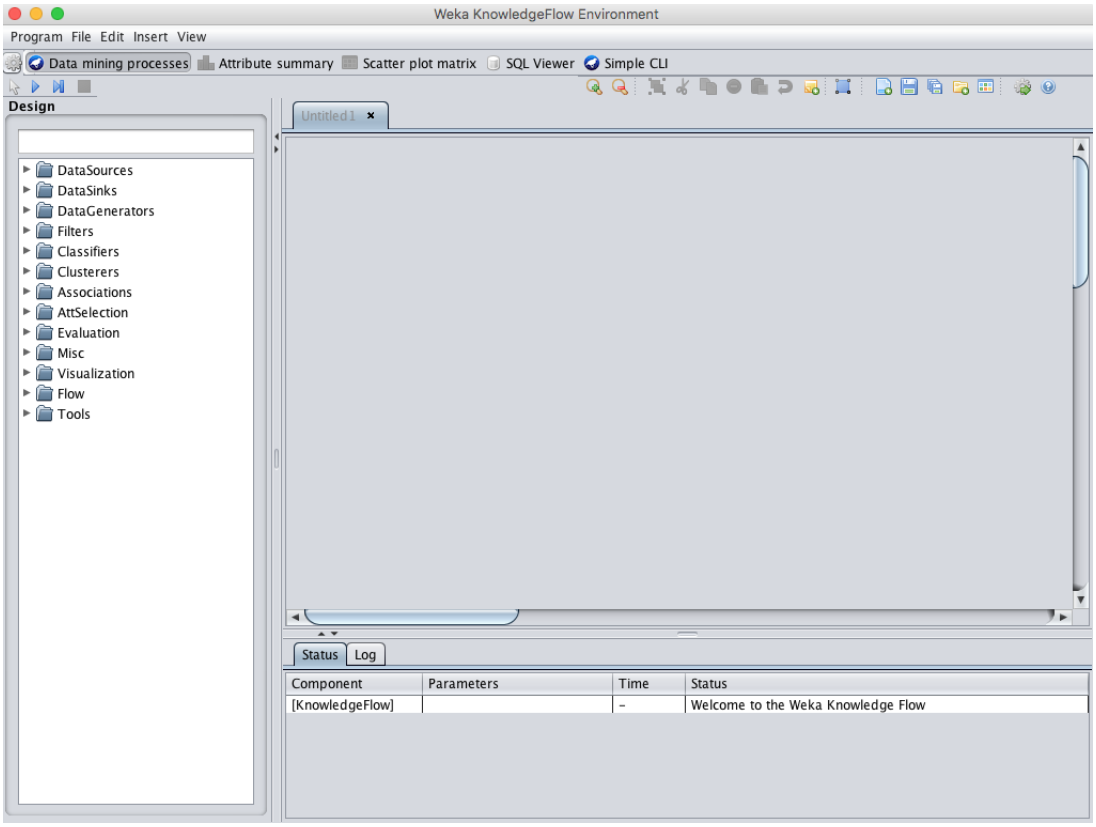


Figure 6.13: Screenshot of the Weka KnowledgeFlow Environment.

The *Weka KnowledgeFlow Environment* is a powerful tool that I do not recommend for beginners until after they have mastered use of the *Weka Explorer* and *Weka Experiment Environment*.

6.5 Weka Workbench

The *Weka Workbench* is an environment that combines all of the GUI interfaces into a single interface. It is useful if you find yourself jumping a lot between two or more different interfaces, such as between the *Weka Explorer* and the *Weka Experiment Environment*. This can happen if you try out a lot of what-if's in the Explorer and quickly take what you learn and put it into controlled experiments.

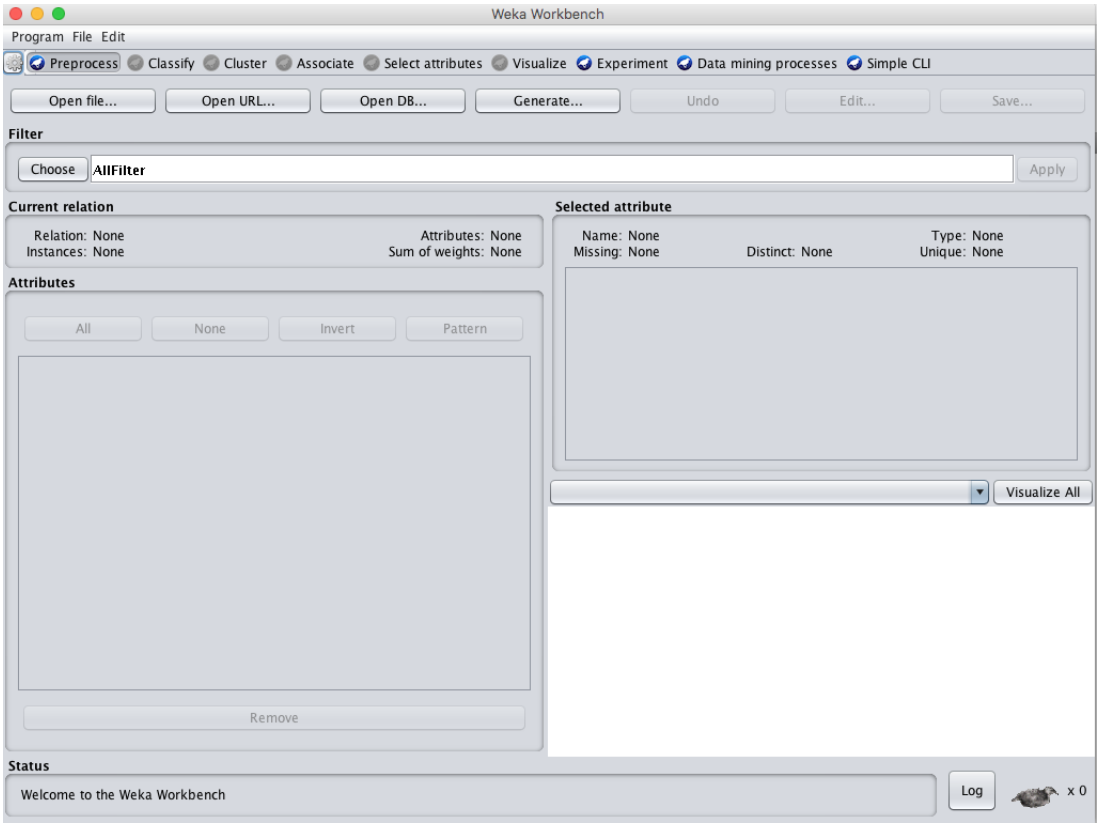


Figure 6.14: Screenshot of the Weka Workbench.

6.6 Weka SimpleCLI

Weka can be used from a simple Command Line Interface (CLI). This is powerful because you can write shell scripts to use the full API from command line calls with parameters, allowing you to build models, run experiments and make predictions without a graphical user interface. The *Weka SimpleCLI* provides an environment where you can quickly and easily experiment with the Weka command line interface commands.

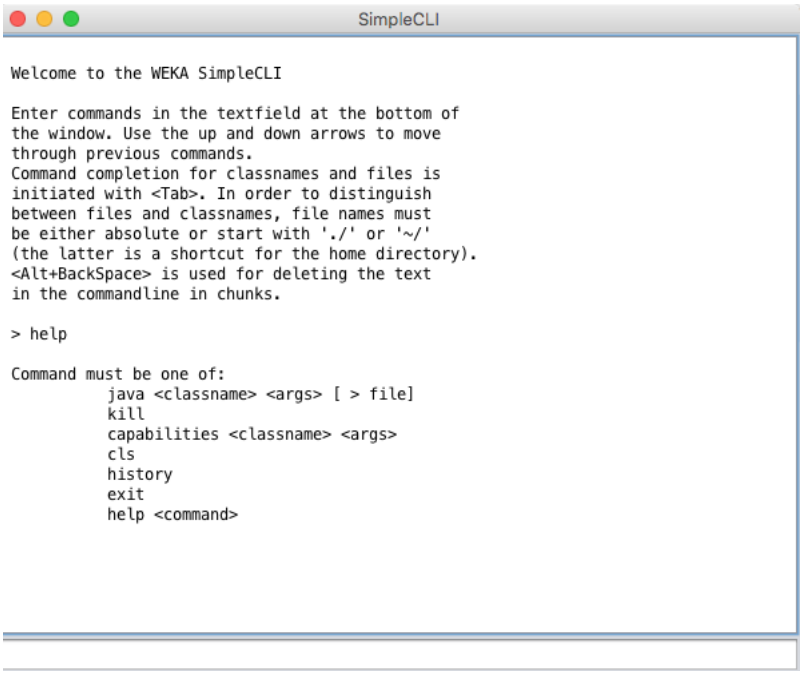


Figure 6.15: Screenshot of the Weka SimpleCLI.

Like the *Weka KnowledgeFlow Environment*, this is a powerful tool that I do not recommend for beginners until they have mastered use of the *Weka Explorer* and *Weka Experiment Environment*.

6.7 Weka Java API

Weka can also be used from the Java API. This is for Java programmers and can be useful when you want to incorporate learning or prediction into your own applications. This is an advanced feature that I do not recommend for beginners until they have mastered use of the *Weka Explorer* and *Weka Experiment Environment*.

6.8 Summary

In this lesson you discovered the Weka Machine Learning Workbench. You went on a tour of the key interfaces that you can use to explore and develop predictive machine learning models on your own problems. Specifically, you learned about:

- The *Weka Explorer* for data preparation, feature selection and evaluating algorithms.
- The *Weka Experiment Environment* for designing, running and analyzing the results from controlled experiments.
- The *Weka KnowledgeFlow Environment* for graphically designing and executing machine learning pipelines.
- The *Weka Workbench* that incorporates all of the Weka tools into a single convenient interface.

- The *Weka SimpleCLI* for using the Weka API from the command line.
- The Weka Java API that can be used to incorporate learning and prediction into your own applications.

6.8.1 Next

We are now familiar with the Weka interface. In the next lesson we will learn how to load machine learning data in CSV format into Weka.

Chapter 7

How To Load CSV Machine Learning Data

You must be able to load your data before you can start modeling it. In this lesson you will discover how you can load your CSV dataset in Weka. After reading this lesson, you will know:

- About the ARFF file format and how it is the default way to represent data in Weka.
- How to load a CSV file in the *Weka Explorer* and save it in ARFF format.
- How to load a CSV file in the *Weka ArffViewer* tool and save it in ARFF format.

Let's get started.

7.1 How to Talk About Data in Weka

Machine learning algorithms are primarily designed to work with arrays of numbers. This is called tabular or structured data because it is how data looks in a spreadsheet, comprised of rows and columns. Weka has a specific computer science centric vocabulary when describing data:

- **Instance**: A row of data is called an instance, as in an instance or observation from the problem domain.
- **Attribute**: A column of data is called a feature or attribute, as in feature of the observation.

Each attribute can have a different type, for example:

- **Real** for numeric values like 1.2.
- **Integer** for numeric values without a fractional part like 5.
- **Nominal** for categorical data like *dog* and *cat*.
- **String** for lists of words, like this sentence.

On classification problems, the output variable must be nominal. For regression problems, the output variable must be real.

7.2 Data in Weka

Weka prefers to load data in the ARFF format. ARFF is an acronym that stands for Attribute-Relation File Format. It is an extension of the CSV file format where a header is used that provides metadata about the data types in the columns. For example, the first few lines of the classic iris flowers dataset in CSV format looks as follows:

```
5.1,3.5,1.4,0.2,Iris-setosa
4.9,3.0,1.4,0.2,Iris-setosa
4.7,3.2,1.3,0.2,Iris-setosa
4.6,3.1,1.5,0.2,Iris-setosa
5.0,3.6,1.4,0.2,Iris-setosa
```

Listing 7.1: CSV Data from the iris dataset.

The same file in ARFF format looks as follows:

```
@RELATION iris

@ATTRIBUTE sepallength REAL
@ATTRIBUTE sepalwidth REAL
@ATTRIBUTE petallength REAL
@ATTRIBUTE petalwidth REAL
@ATTRIBUTE class {Iris-setosa,Iris-versicolor,Iris-virginica}

@DATA
5.1,3.5,1.4,0.2,Iris-setosa
4.9,3.0,1.4,0.2,Iris-setosa
4.7,3.2,1.3,0.2,Iris-setosa
4.6,3.1,1.5,0.2,Iris-setosa
5.0,3.6,1.4,0.2,Iris-setosa
```

Listing 7.2: ARFF version of the iris dataset.

You can see that directives start with the at symbol (`@`) and that there is one for the name of the dataset (e.g. `@RELATION iris`), there is a directive to define the name and datatype of each attribute (e.g. `@ATTRIBUTE sepallength REAL`) and there is a directive to indicate the start of the raw data (e.g. `@DATA`). Lines in an ARFF file that start with a percentage symbol (`%`) indicate a comment. Values in the raw data section that have a question mark symbol (`?`) indicate an unknown or missing value. The format supports numeric and categorical values as in the iris example above, but also supports dates and string values. Depending on your installation of Weka, you may or may not have some default datasets in your Weka installation directory under the `data/` subdirectory. These default datasets distributed with Weka are in the ARFF format and have the `.arff` file extension. You will learn more about these datasets in the next Lesson.

7.3 Load CSV Files in the *ARFF-Viewer*

Your data is not likely to be in ARFF format. In fact, it is much more likely to be in Comma Separated Value (CSV) format. This is a simple format where data is laid out in a table of rows and columns and a comma is used to separate the values on a row. Quotes may also be used to surround values, especially if the data contains strings of text with spaces. The CSV format is

easily exported from Microsoft Excel, so once you can get your data into Excel, you can easily convert it to CSV format.

Weka provides a handy tool to load CSV files and save them in ARFF. You only need to do this once with your dataset. Using the steps below you can convert your dataset from CSV format to ARFF format and use it with the Weka workbench. If you do not have a CSV file handy, you can use the iris flowers dataset. Download the file from the UCI Machine Learning repository[1] and save it to your current working directory as `iris.csv`. You will learn more about the iris dataset in Section 8.3.1.

- 1. Start the *Weka GUI Chooser*.

Figure 7.1: Screenshot of the *Weka GUI Chooser*.

- 2. Open the ARFF-Viewer by clicking *Tools* in the menu and select *ArffViewer*.
- 3. You will be presented with an empty *ARFF-Viewer* window.

[1] https://archive.ics.uci.edu/ml/machine-learning-databases/iris/iris.data

Figure 7.2: Screenshot of the Weka ARFF Viewer.

- 4. Open your CSV file in the *ARFF-Viewer* by clicking the *File* menu and select *Open*. Navigate to your current working directory. Change the *Files of Type:* filter to *CSV data files (*.csv)*. Select your file and click the *Open* button.

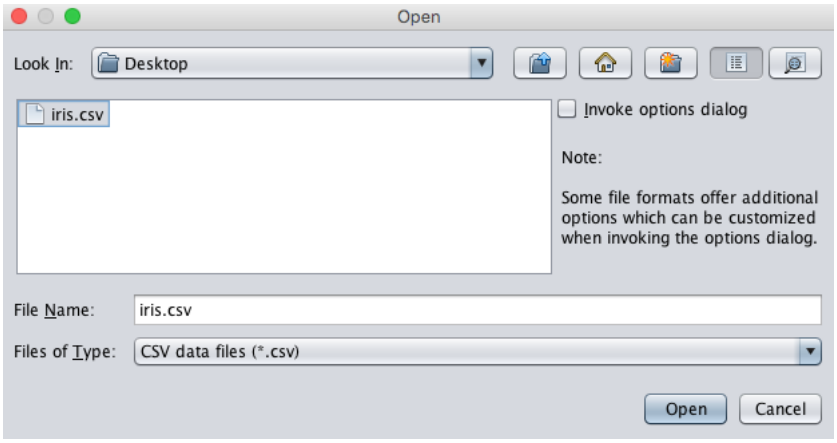


Figure 7.3: Load CSV In ARFF Viewer.

- 5. You should see a sample of your CSV file loaded into the *ARFF-Viewer*.
- 6. Save your dataset in ARFF format by clicking the *File* menu and selecting *Save as...*. Enter a filename with a `.arff` extension and click the *Save* button.

You can now load your saved `.arff` file directly into Weka. Note, the *ARFF-Viewer* provides options for modifying your dataset before saving. For example you can change values, change the name of attributes and change their data types. It is highly recommended that you specify the names of each attribute as this will help with analysis of your data later. Also, make sure that the data types of each attribute are correct.

7.4 Load CSV Files in the *Weka Explorer*

You can also load your CSV files directly in the *Weka Explorer* interface. This is handy if you are in a hurry and want to quickly test out an idea. This section shows you how you can load your CSV file in the *Weka Explorer* interface. You can use the iris dataset again, to practice if you do not have a CSV dataset to load.

- 1. Start the *Weka GUI Chooser*.
- 2. Launch the *Weka Explorer* by clicking the *Explorer* button.

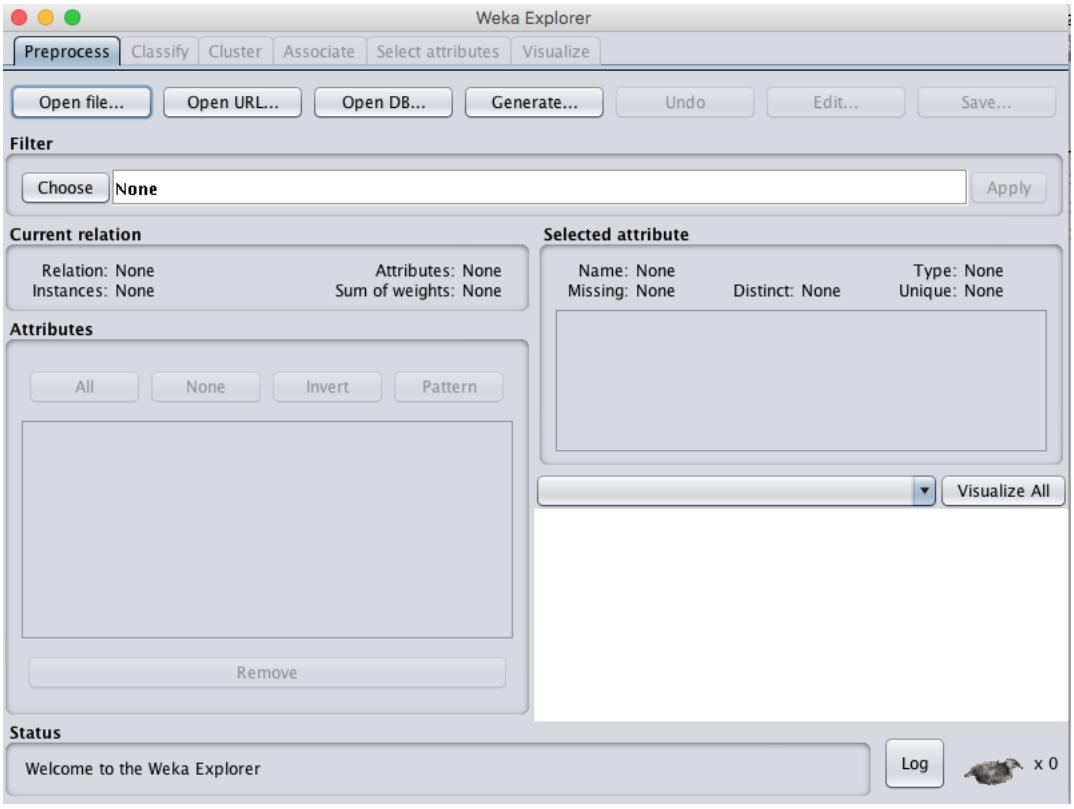


Figure 7.4: Screenshot of the *Weka Explorer*.

- 3. Click the *Open file...* button.
- 4. Navigate to your current working directory. Change the *Files of Type* to *CSV data files (*.csv)*. Select your file and click the *Open* button.

You can work with the data directly. You can also save your dataset in ARFF format by clicking the *Save* button and typing a filename.

7.5 Use Excel for Other File Formats

If you have data in another format, load it in Microsoft Excel first. It is common to get data in another format such as CSV using a different delimiter or fixed width fields. Excel has powerful tools for loading tabular data in a variety of formats. Use these tools and first load your data into Excel. Once you have loaded your data into Excel, you can export it into CSV format. You can then work with it in Weka, either directly or by first converting it to ARFF format.

7.6 Summary

In this lesson you discovered how to load your CSV data into Weka for machine learning. Specifically, you learned:

- About the ARFF file format and how Weka uses it to represent datasets for machine learning.
- How to load your CSV data using *ARFF-Viewer* and save it into ARFF format.
- How to load your CSV data directly in the *Weka Explorer* and use it for modeling.

7.6.1 Next

Weka is distributed with standard machine learning datasets that we can use as practice. In the next lesson you will learn about these standard datasets and specific examples that we can and should use as practice.

Chapter 8

How to Load Standard Machine Learning Datasets

It is a good idea to have small well understood datasets when getting started in machine learning and learning a new tool. The Weka machine learning workbench provides a directory of small well understood datasets in your Weka installation. In this lesson you will discover some of these small well understood datasets distributed with Weka, their details and where to learn more about them. We will focus on a handful of datasets of differing types. After reading this lesson you will know:

- Where the sample datasets are located or where to download them afresh if you need them.
- Specific standard datasets you can use to explore different aspects of classification and regression predictive models.
- Where to go for more information about specific datasets and state-of-the-art results.

Let's get started.

8.1 Standard Weka Datasets

An installation of the open source Weka machine learning workbench includes a `data/` directory full of standard machine learning problems.

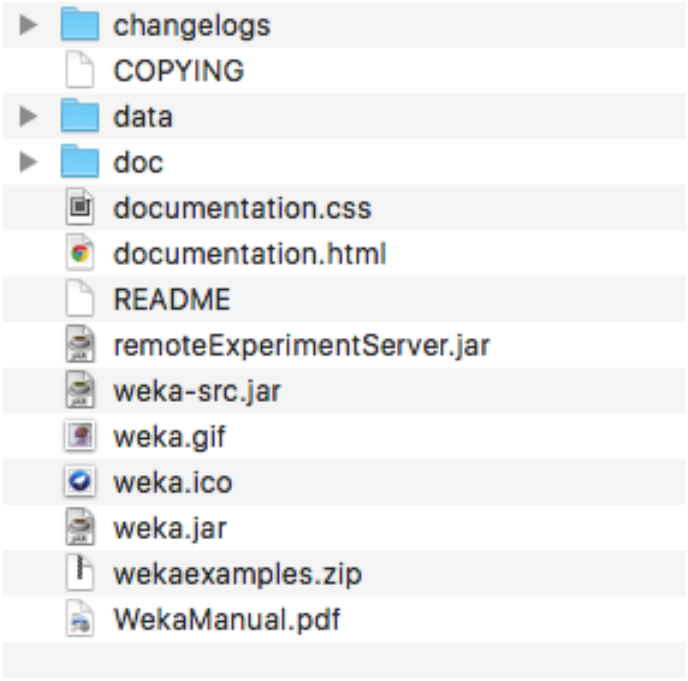


Figure 8.1: Weka Installation Directory.

This is very useful when you are getting started in machine learning or learning how to get started with the Weka platform. It provides standard machine learning datasets for common classification and regression problems, for example, below is a snapshot from this directory:

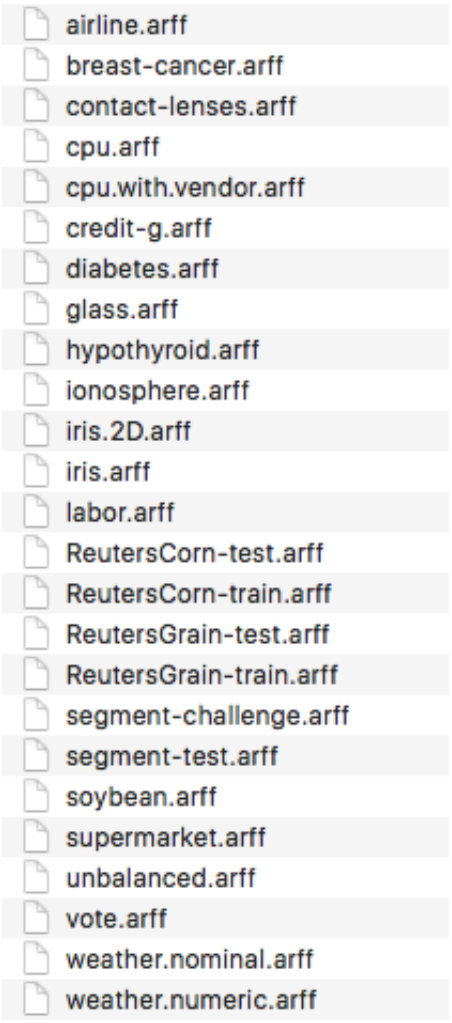


Figure 8.2: Provided Datasets in Weka Installation Directory.

All datasets are in the Weka native ARFF file format and can be loaded directly into Weka, meaning you can start developing practice models immediately. There are some special distributions of Weka that may not include the `data/` directory. If you have chosen to install one of these distributions, you can download the `.zip` distribution of Weka, unzip it and copy the `data/` directory to somewhere that you can access it easily from Weka. There are many datasets to play with in the `data/` directory, in the following sections I will point out a few that you can focus on for practicing and investigating predictive modeling problems.

8.2 Binary Classification Datasets

Binary classification is where the output variable to be predicted is nominal comprised of two classes. This is perhaps the most well studied type of predictive modeling problem and the type of problem that is good to start with. There are three standard binary classification problems in the `data/` directory that you can focus on:

8.2.1 Pima Indians Onset of Diabetes

Each instance represents medical details for one patient and the task is to predict whether the patient will have an onset of diabetes within the next five years. There are 8 numerical input variables all of which have varying scales.

- Dataset File: `data/diabetes.arff`
- More Info: https://archive.ics.uci.edu/ml/datasets/Pima+Indians+Diabetes
- Top results are in the order of 77% accuracy: http://www.is.umk.pl/projects/datasets.html#Diabetes

8.2.2 Breast Cancer

Each instance represents medical details of patients and samples of their tumor tissue and the task is to predict whether or not the patient has breast cancer. There are 9 input variables all of which a nominal.

- Dataset File: `data/breast-cancer.arff`
- More Info: https://archive.ics.uci.edu/ml/datasets/Breast+Cancer
- Top results are in the order of 75% accuracy: http://www.is.umk.pl/projects/datasets.html#Ljubljana

8.2.3 Ionosphere

Each instance describes the properties of radar returns from the atmosphere and the task is to predict whether or not there is structure in the ionosphere. There are 34 numerical input variables of generally the same scale.

- Dataset File: `data/ionosphere.arff`
- More Info: https://archive.ics.uci.edu/ml/datasets/Ionosphere
- Top results are in the order of 98% accuracy: http://www.is.umk.pl/projects/datasets.html#Ionosphere

8.3 Multiclass Classification Datasets

There are many classification type problems, where the output variable has more than two classes. These are called multiclass classification problems. This is a good type of problem to look at after you have some confidence with binary classification. Three standard multiclass classification problems in the `data/` directory that you can focus on are:

8.3.1 Iris Flowers Classification

Each instance describes measurements of iris flowers and the task is to predict to which species of 3 iris flower the observation belongs. There are 4 numerical input variables with the same units and generally the same scale.

- Dataset File: `data/iris.arff`
- More Info: https://archive.ics.uci.edu/ml/datasets/Iris

8.3.2 Large Soybean Database

Each instance describes properties of a crop of soybeans and the task is to predict which of the 19 diseases the crop suffers. There are 35 nominal input variables.

- Dataset File: `data/soybean.arff`
- More Info: https://archive.ics.uci.edu/ml/datasets/Soybean+(Large)

8.3.3 Glass Identification

Each instance describes the chemical composition of samples of glass and the task is to predict the type or use of the class from one of 7 classes. There are 10 numeric attributes that describe the chemical properties of the glass and its refractive index.

- Dataset File: `data/glass.arff`
- More Info: https://archive.ics.uci.edu/ml/datasets/Glass+Identification

8.4 Regression Datasets

Regression problems are those where you must predict a real valued output. The selection of regression problems in the `data/` directory is small. Regression is an important class of predictive modeling problem. As such I recommend downloading the free add-on pack of regression problems collected from the UCI Machine Learning Repository. It is available from the datasets page on the Weka webpage[1] and is the first in the list called:

- A jar file containing 37 regression problems, obtained from various sources (`datasets-numeric.jar`)

It is a `.jar` file which is a type of compressed Java archive. You should be able to unzip it with most modern unzip programs. If you have Java installed (which you very likely do to use Weka), you can also unzip the `.jar` file manually on the command line using the following command in the directory where the jar was downloaded:

```
jar -xvf datasets-numeric.jar
```

Listing 8.1: Uncompress numerical datasets for Weka.

Unzipping the file will create a new directory called `numeric` that contains 37 regression datasets in ARFF native Weka format. Three regression datasets in the `numeric/` directory that you can focus on are:

[1] http://www.cs.waikato.ac.nz/ml/weka/datasets.html

8.4.1 Longley Economic Dataset

Each instance describes the gross economic properties of a nation for a given year and the task is to predict the number of people employed as an integer. There are 6 numeric input variables of varying scales.

- Dataset File: `numeric/longley.arff`

8.4.2 Boston House Price Dataset

Each instance describes the properties of a Boston suburb and the task is to predict the house prices in thousands of dollars. There are 13 numerical input variables with varying scales describing the properties of suburbs.

- Dataset File: `numeric/housing.arff`
- More Info: https://archive.ics.uci.edu/ml/datasets/Housing

8.4.3 Sleep in Mammals Dataset

Each instance describes the properties of different mammals and the task is to predict the number of hours of total sleep they require on average. There are 7 numeric input variables of different scales and measures.

- Dataset File: `numeric/sleep.arff`

8.5 Summary

In this lesson you discovered the standard machine learning datasets distributed with the Weka machine learning platform. Specifically, you learned:

- Three popular binary classification problems you can use for practice: diabetes, breast-cancer and ionosphere.
- Three popular multiclass classification problems you can use for practice: iris, soybean and glass.
- Three popular regression problems you can use for practice: longley, housing and sleep.

8.5.1 Next

Now that we know how to load data, we can start working through problems. In the next lesson you will discover how you can learn more about a load dataset by reviewing descriptive statistics and data visualizations.

Chapter 9

How to Better Understand Your Machine Learning Data

It is important to take your time to learn about your data when starting on a new machine learning problem. There are key things that you can look at to very quickly learn more about your dataset, such as descriptive statistics and data visualizations. In this lesson you will discover how you can learn more about your data in the Weka machine learning workbench by reviewing descriptive statistics and visualizations of your data. After reading this lesson you will know about:

- The distribution of attributes from reviewing statistical summaries.
- The distribution of attributes from reviewing univariate plots.
- The relationship between attributes from reviewing multivariate plots.

Let's get started

9.1 Descriptive Statistics

The *Weka Explorer* will automatically calculate descriptives statistics for numerical attributes.

1. Open the *Weka GUI Chooser*.
2. Click *Explorer* to open the *Weka Explorer*.
3. Load the Pima Indians datasets from `data/diabetes.arff`.

The Pima Indians dataset contains numeric input variables that we can use to demonstrate the calculation of descriptive statistics. You can learn more about this dataset in Section 8.2.1. Firstly, note that the dataset summary in the *Current Relation* section. This pane summarizes the following details about the loaded datasets:

- Dataset name (relation).
- The number of rows (instances).

- The number of columns (attributes).

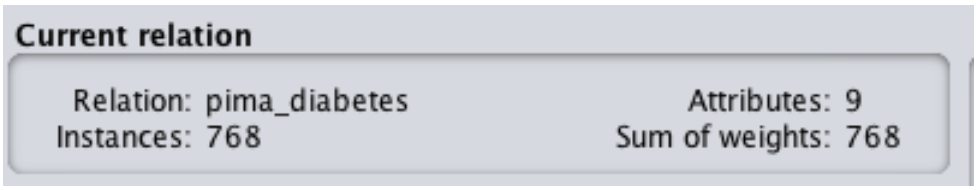


Figure 9.1: Weka Summary of Dataset.

- Click on the first attribute in the dataset in the *Attributes* pane.

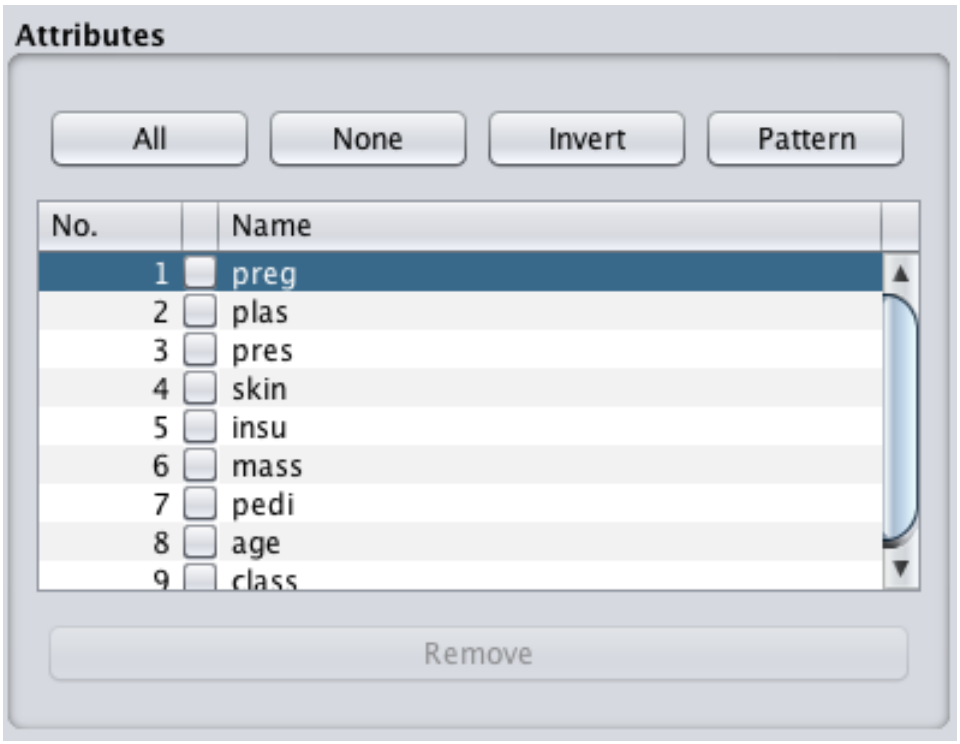


Figure 9.2: Weka List of Attributes.

Take note of the details in the *Selected attribute* pane. It lists a lot of information about the selected attribute, such as:

- The name of the attribute.
- The number of missing values and the ratio of missing values across the whole dataset.
- The number of distinct values.
- The data type.

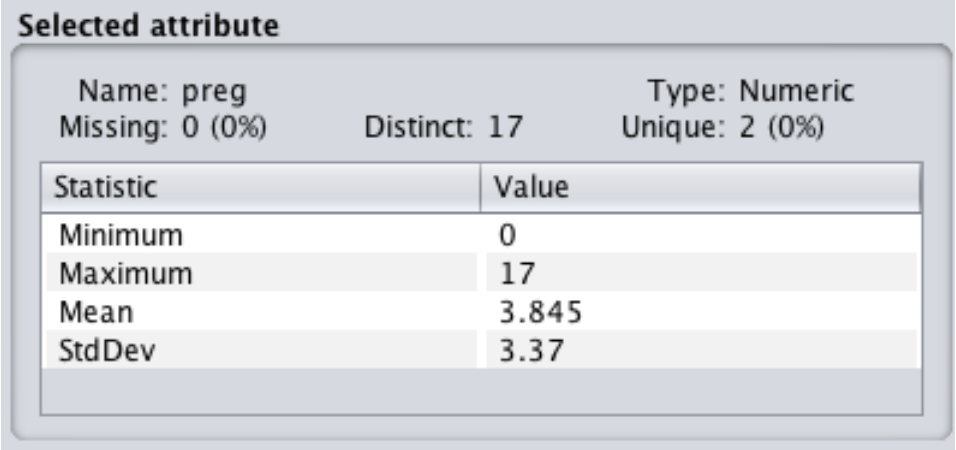


Figure 9.3: Weka Summary of Attribute.

The table below lists a number of descriptive statistics and their values. A useful four number summary is provided for numeric attributes including:

- Minimum value.
- Maximum value.
- Mean value.
- Standard deviation.

You can learn a lot from this information. For example:

- The presence and ratio of missing data can give you an indication of whether or not you need to remove or impute values.
- The mean and standard deviation give you a quantified idea of the spread of data for each attribute.
- The number of distinct values can give you an idea of the granularity of the attribute distribution.
- Click the class attribute. This attribute has a nominal type. Review the *Selected attribute* pane.

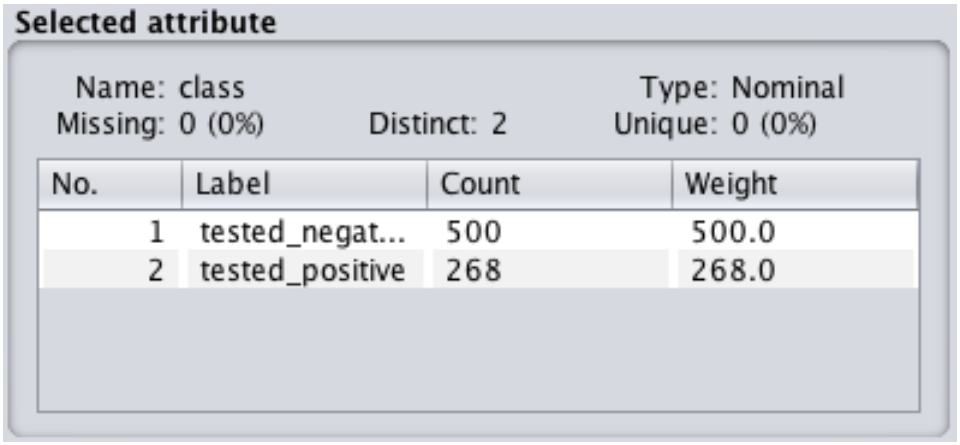


Figure 9.4: Weka Summary of Class Attribute.

We can now see that for nominal attributes that we are provided with a list of each category and the count of instances that belong to each category. There is also mention of weightings, which we can ignore for now. This is used if we want to assign more or less weight to specific attribute values or instances in the dataset.

9.2 Univariate Attribute Distributions

The distribution of each attribute can be plotted to give a visual qualitative understanding of the distribution. Weka provides these plots automatically when you select an attribute in the *Preprocess* tab. We can follow on from the previous section where we already have the Pima Indians dataset loaded.

- Click on the `preg` attribute in the *Attributes* pane and note the plot below the *Selected attribute* pane.

You will see the distribution of `preg` values between 0 and 17 along the x-axis. The y-axis shows the count or frequency of values with each `preg` value.

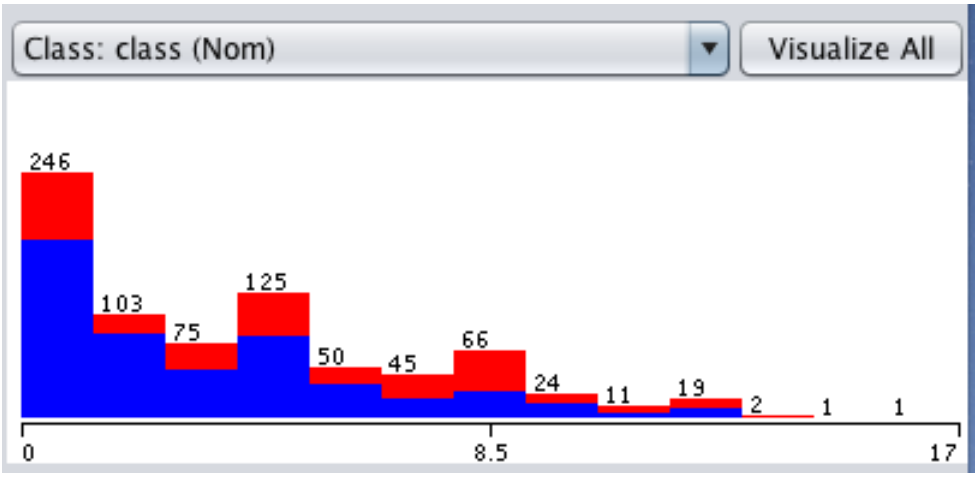


Figure 9.5: Weka Univariate Attribute Distribution.

Note the red and blue colors referring to the positive and negative classes respectively. The colors are assigned automatically to each categorical value. If there were three categories for the `class` value, we would see the breakdown of the `preg` distribution by three colors rather than two. This is useful to get a quick idea of whether the problem is easily separable for a given attribute, e.g. all the red and blue are cleanly separated for a single attribute. Clicking through each attribute in the list of *Attributes* and reviewing the plots, we can see that there is no such easy separation of the classes. We can quickly get an overview of the distribution of all attributes in the dataset and the breakdown of distributions by class by clicking the *Visualize All* button above the univariate plot.

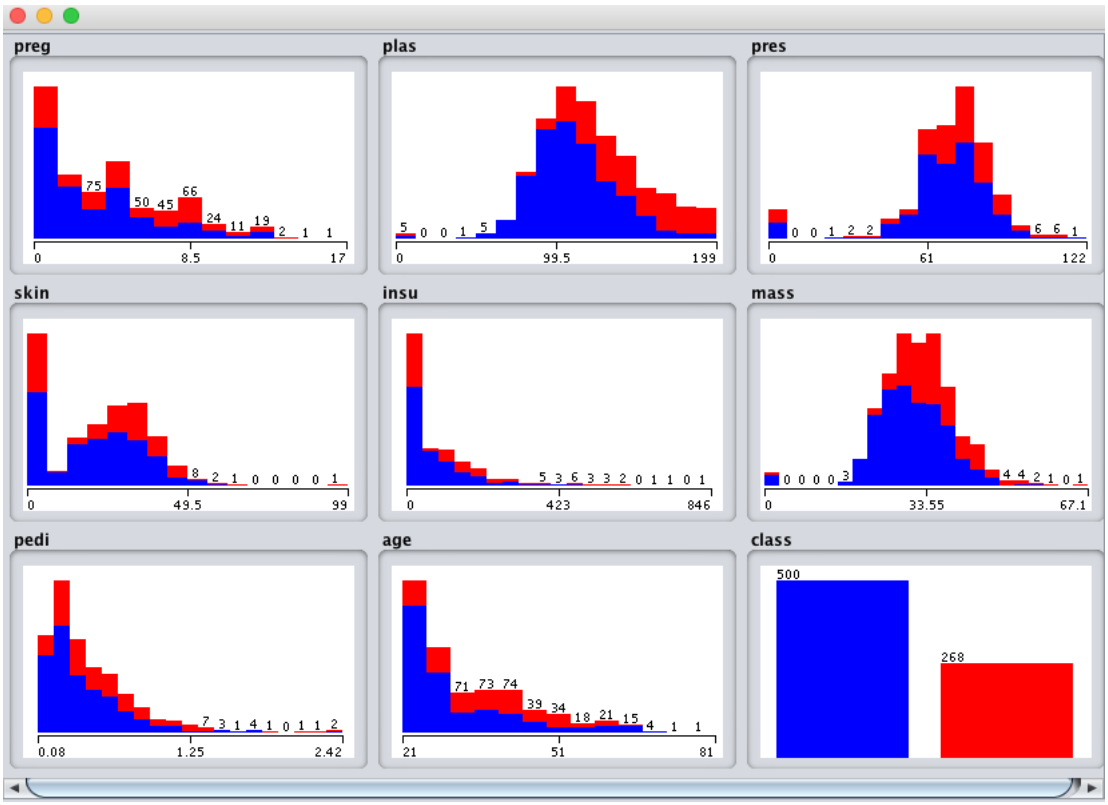


Figure 9.6: Weka All Univariate Attribute Distributions.

Looking at these plots we can see a few interesting things about this dataset.

- It looks like the `plas`, `pres` and `mass` attributes have a nearly Gaussian distribution.
- It looks likes `pres`, `skin`, `insu` and mass have values at 0 that look out of place.

Looking at plots like this and jotting down things that come to mind can give you an idea of further data preparation operations that could be applied (like marking zero values as corrupt) and even techniques that might be useful (like linear discriminant analysis and logistic regression that assume a Gaussian distribution in input variables).

9.3 Visualize Attribute Interactions

So far we have only been looking at the properties of individual features, next we will look at patterns in combinations of attributes. When attributes are numeric we can create a scatter plot of one attribute against another. This is useful as it can highlight any patterns in the relationship between the attributes, such as positive or negative correlations. We can create scatter plots for all pairs of input attributes. This is called a scatter plot matrix and reviewing it before modeling your data can shed more light on further pre-processing techniques that you could investigate. Weka provides a scatter plot matrix for review by default in the *Visualise* tab.

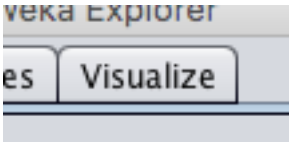


Figure 9.7: Weka Visualize Tab.

Continuing on from the previous section with the Pima Indians dataset loaded, click the *Visualize* tab, and make the window large enough to review all of the individual scatter plots.

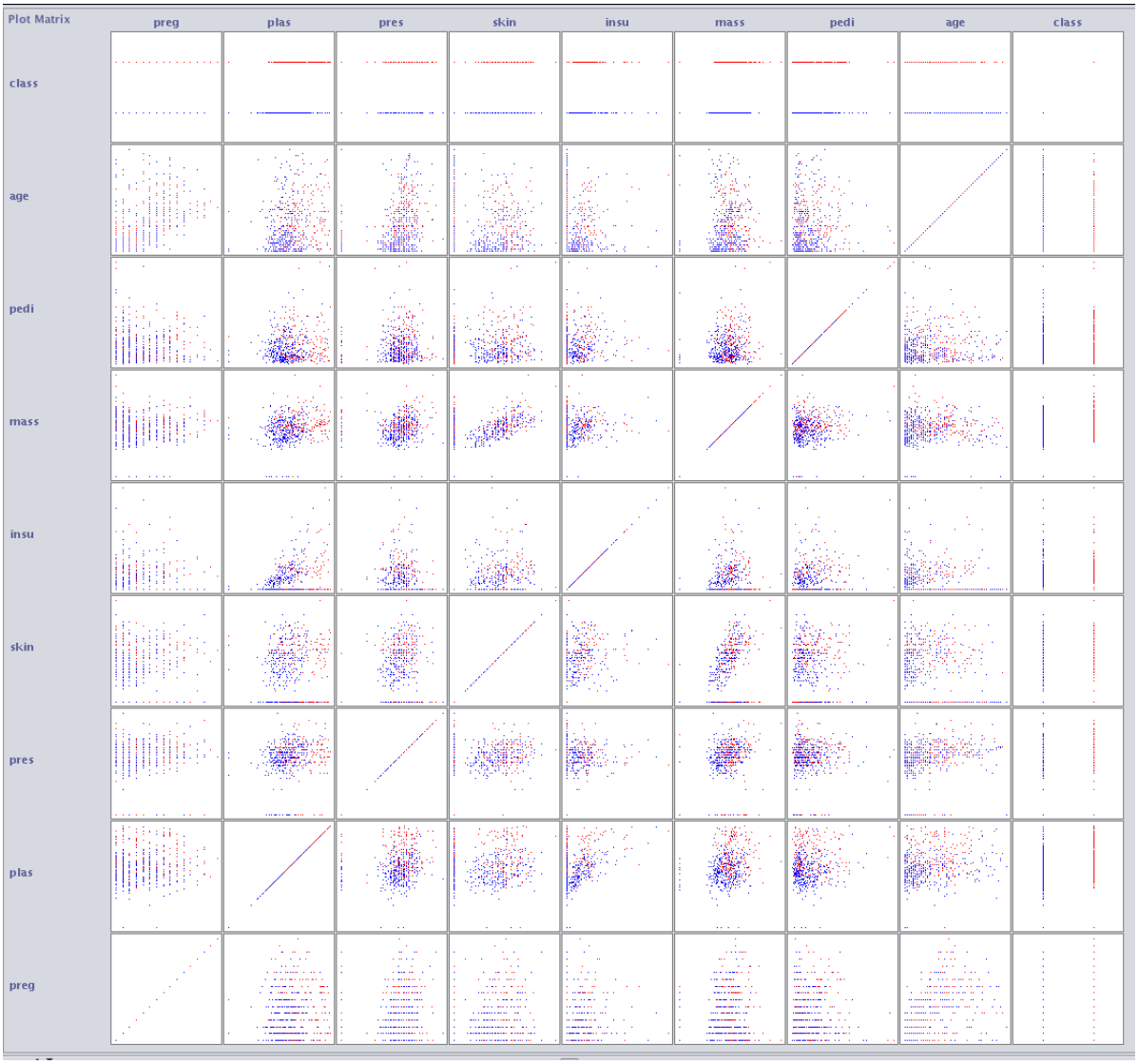


Figure 9.8: Weka Scatter Plot Matrix.

You can see that all combinations of attributes are plotted in a systematic way. You can also see that each plot appears twice, first in the top left triangle and again in the bottom right triangle with the axes flipped. You can also see a series of plots starting in the bottom left and continuing to the top right where each attribute is plotted against itself. These can be ignored. Finally, notice that the dots in the scatter plots are colored by their class value. It is good to look for trends or patterns in the dots, such as clear separation of the colors.

- Clicking on a plot will give you a new window with the plot that you can further play with.

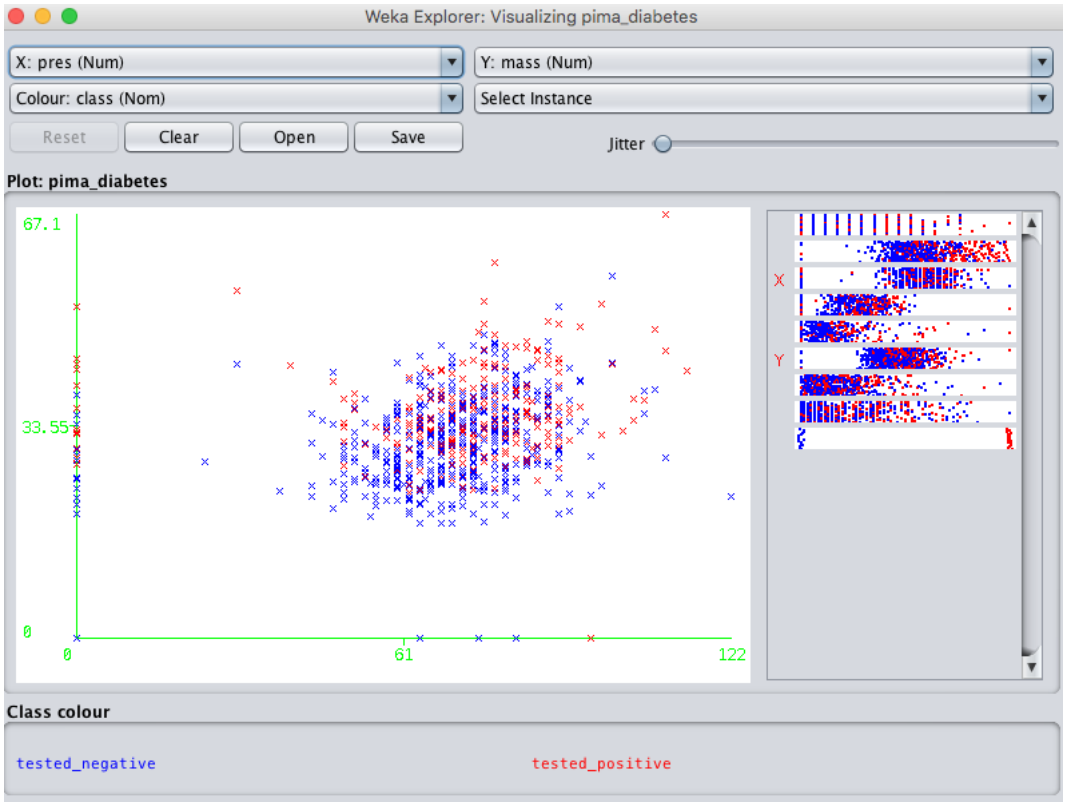


Figure 9.9: Weka Individual Scatter Plot.

Note the controls at the bottom of the screen. They let you increase the size of the plots, increase the size of the dots and add jitter. This last point about jitter is useful when you have a lot of dots overlaying each other and it is hard to see what is going on. Jitter will add some random noise to the data in the plots, spread out the points a bit and help you see what is going on. When you make a change to these controls, click the *Update* button to apply the changes.

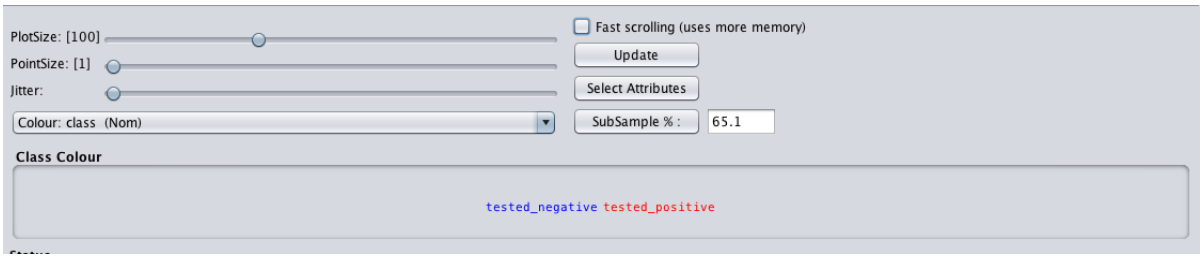


Figure 9.10: Weka Controls for Scatter Plot Matrix.

For example, below are the same plots with a larger dot size that makes it easier to see any trends in the data.

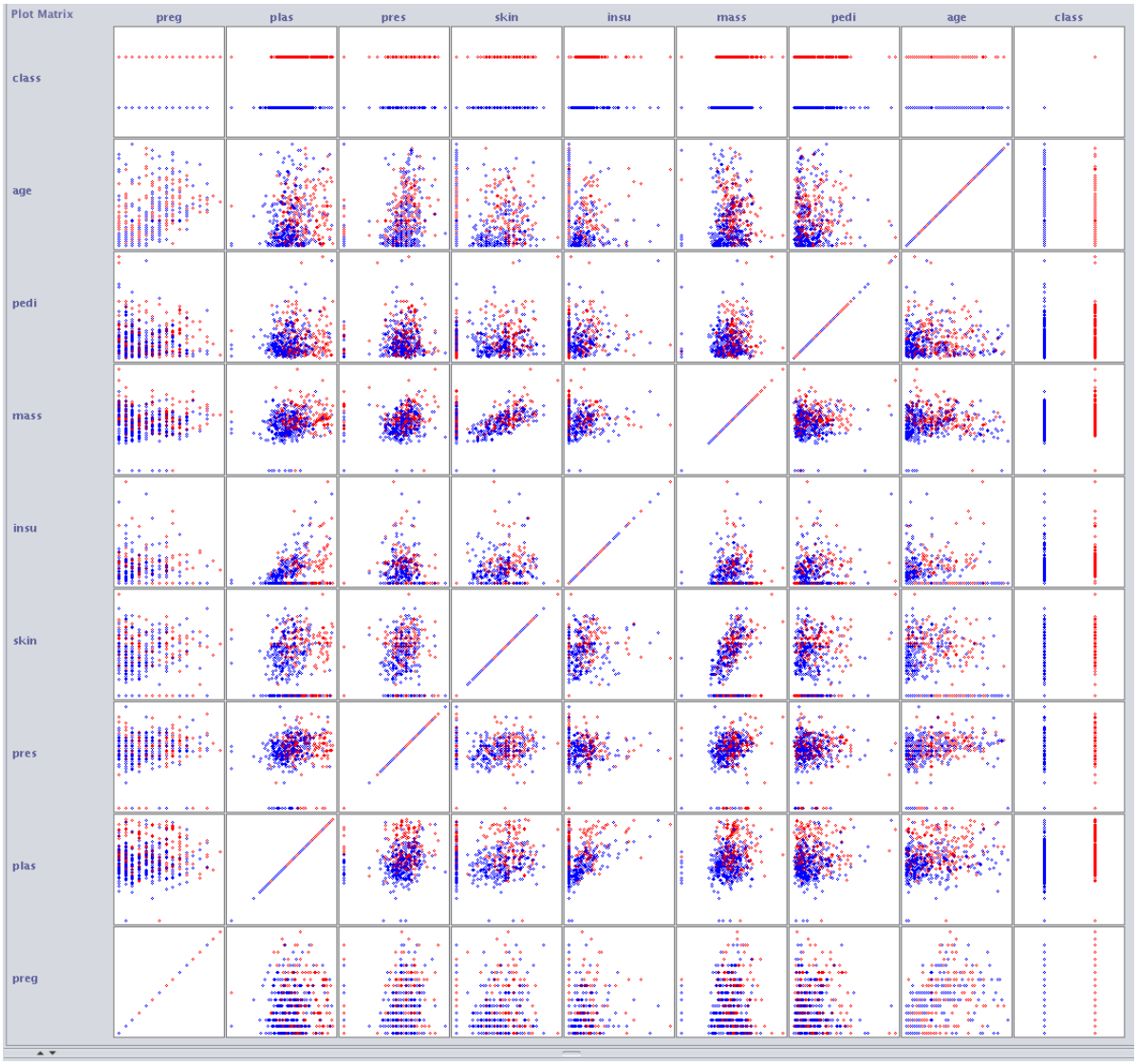


Figure 9.11: Weka Improved Scatter Plot Matrix.

9.4 Summary

In this lesson you discovered how you can learn more about your machine learning data by reviewing descriptive statistics and data visualizations. Specifically, you learned:

- That Weka automatically calculates descriptive statistics for each attribute.
- That Weka allows you to review the distribution of each attribute easily.
- That Weka provides a scatter plot visualization to review the pairwise relationships between attributes.

9.4.1 Next

We now know how to load and analyze data in Weka. In the next lesson we will learn about data filters and how they can be used to rescale numerical data.

Chapter 10

How to Normalize and Standardize Your Machine Learning Data

Machine learning algorithms make assumptions about the dataset you are modeling. Often, raw data is comprised of attributes with varying scales. For example, one attribute may be in kilograms and another may be a count. Although not required, you can often get a boost in performance by carefully choosing methods to rescale your data. In this lesson you will discover how you can rescale your data so that all of the data has the same scale. After reading this lesson you will know:

- How to normalize your numeric attributes between the range of 0 and 1.
- How to standardize your numeric attributes to have a zero mean and unit variance.
- When to choose normalization or standardization.

Let's get started.

10.1 About Data Filters in Weka

Weka provides filters for transforming your dataset. The best way to see what filters are supported and to play with them on your dataset is to use the *Weka Explorer*. The *Filter* pane allows you to choose a filter.

Figure 10.1: Weka Filter Pane for Choosing Data Filters.

Filters are divided into two types:

- **Supervised Filters**: That can be applied but require user control in some way. Such as rebalancing instances for a class.
- **Unsupervised Filters**: That can be applied in an undirected manner. For example, rescale all values to the range 0-to-1.

Personally, I think the distinction between these two types of filters is a little arbitrary and confusing. Nevertheless, that is how they are laid out. Within these two groups, filters are further divided into filters for Attributes and Instances:

- **Attribute Filters**: Apply an operation on attributes or one attribute at a time.
- **Instance Filters**: Apply an operation on instance or one instance at a time.

This distinction makes a lot more sense. After you have selected a filter, its name will appear in the box next to the *Choose* button. You can configure a filter by clicking its name which will open the configuration window. You can change the parameters of the filter and even save or load the configuration of the filter itself. This is great for reproducibility.

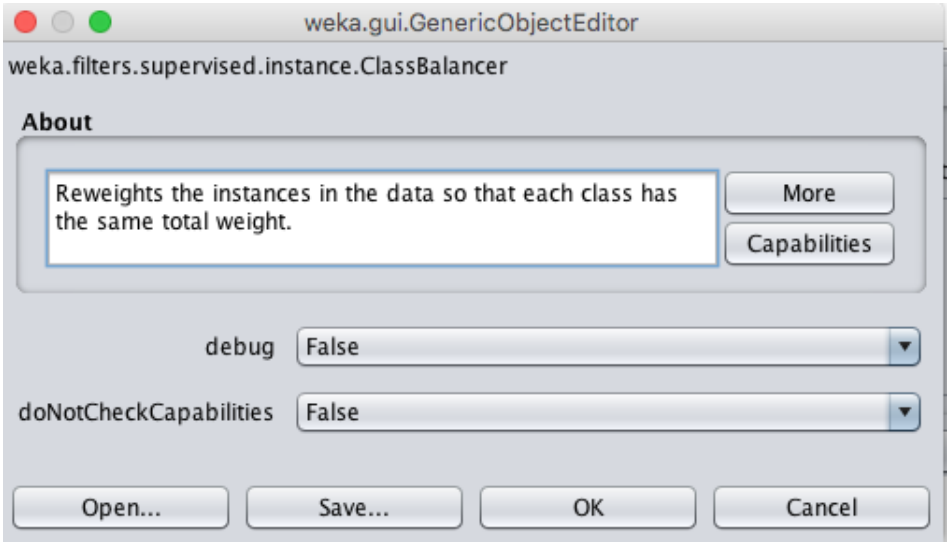


Figure 10.2: Weka Data Filter Configuration.

You can learn more about each configuration option by hovering over it and reading the tooltip. You can also read all of the details about the filter including the configuration, papers and books for further reading and more information about the filter works by clicking the *More* button.

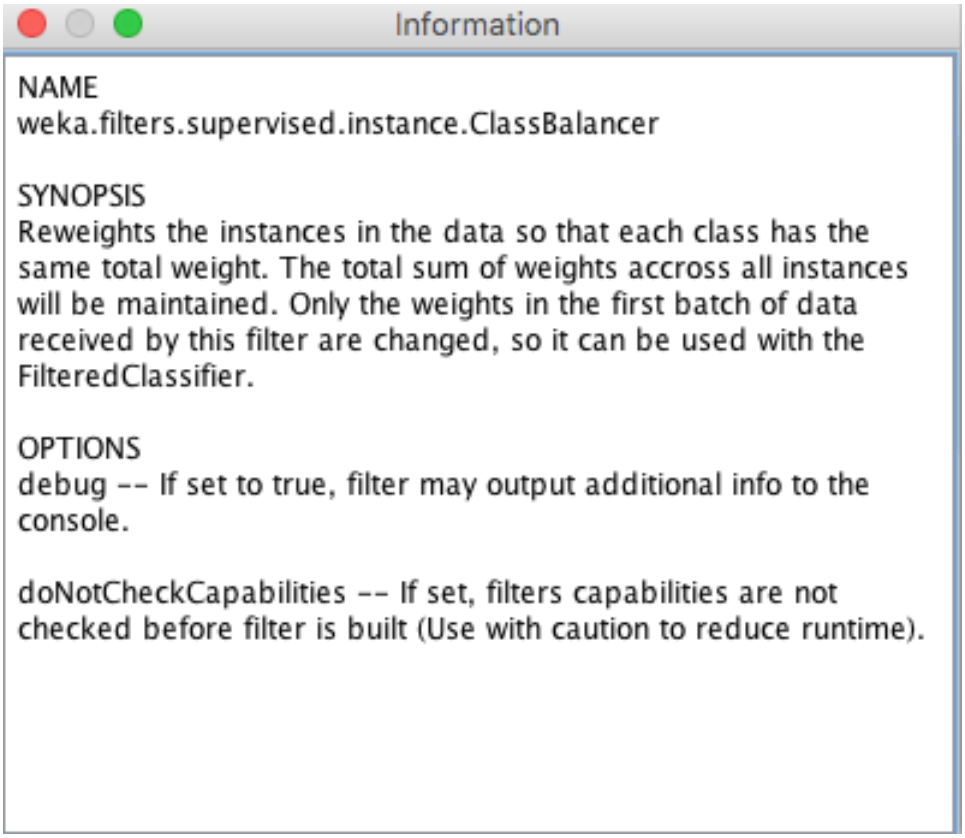


Figure 10.3: Weka Data Filter More Information.

You can close the help and apply the configuration by clicking the *OK* button. You can apply a filter to your loaded dataset by clicking the *Apply* button next to the filter name.

10.2 Normalize Your Numeric Attributes

Data normalization is the process of rescaling one or more attributes to the range of 0 to 1. This means that the largest value for each attribute is 1 and the smallest value is 0. Normalization is a good technique to use when you do not know the distribution of your data or when you know the distribution is not Gaussian (a bell curve).

The dataset used for this example is the Pima Indians onset of diabetes dataset. You can learn more about this dataset in Section 8.2.1. You can normalize all of the attributes in your dataset with Weka by choosing the *Normalize* filter and applying it to your dataset. You can use the following recipe to normalize your dataset:

- 1. Open the *Weka Explorer*.
- 2. Load the `data/diabetes.arff` dataset.

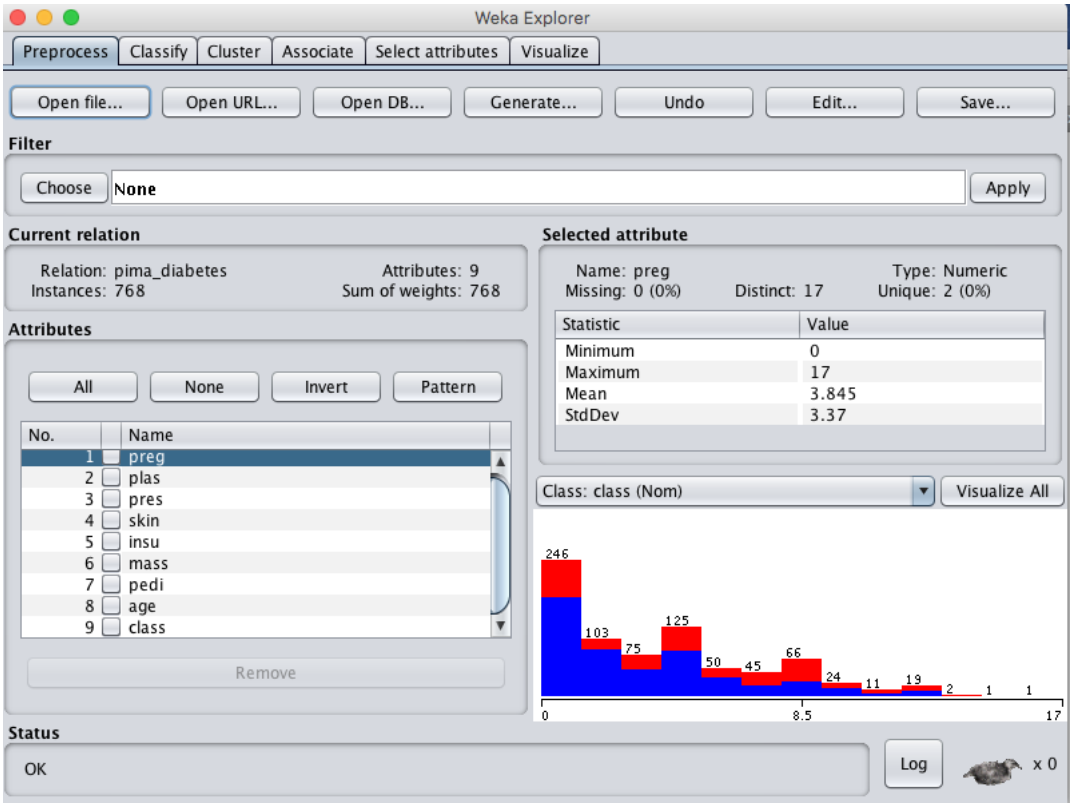


Figure 10.4: Weka Explorer Loaded Diabetes Dataset.

- 3. Click the *Choose* button and select the *unsupervised.attribute.Normalize* filter.

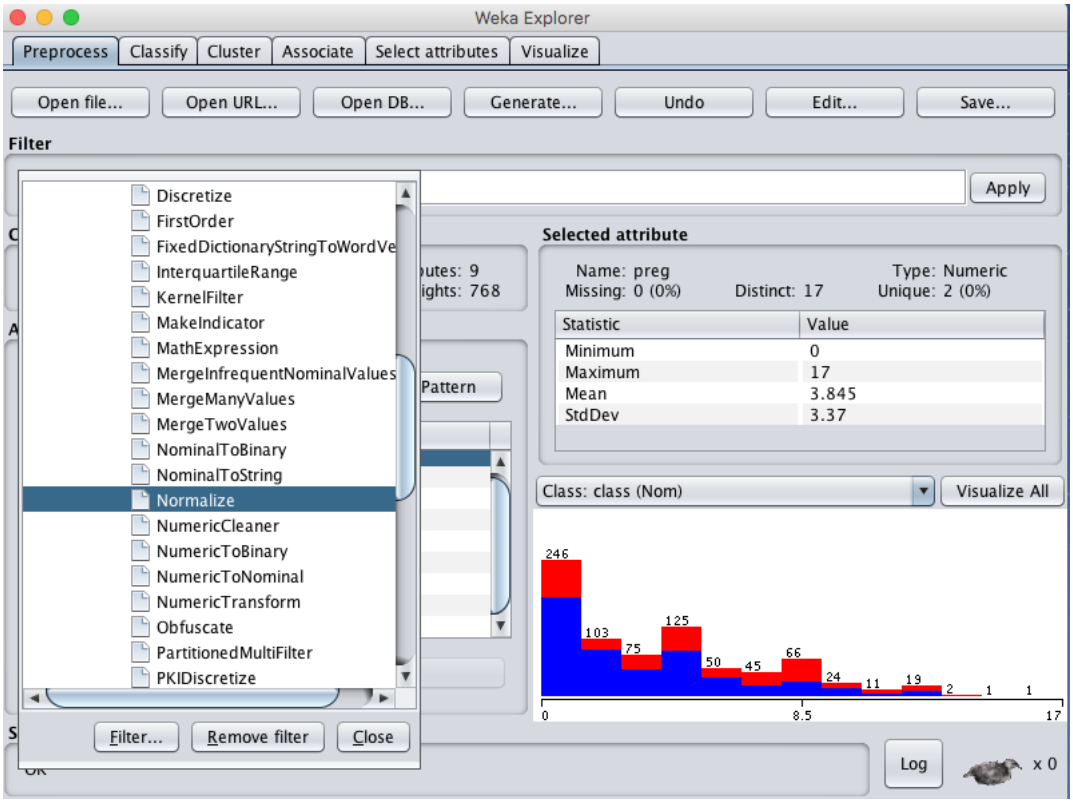


Figure 10.5: Weka Select Normalize Data Filter.

- 4. Click the *Apply* button to normalize your dataset.
- 5. Click the *Save* button and type a filename to save the normalized copy of your dataset.

Reviewing the details of each attribute in the *Selected attribute* window will give you confidence that the filter was successful and that each attribute was rescaled to the range of 0 to 1.

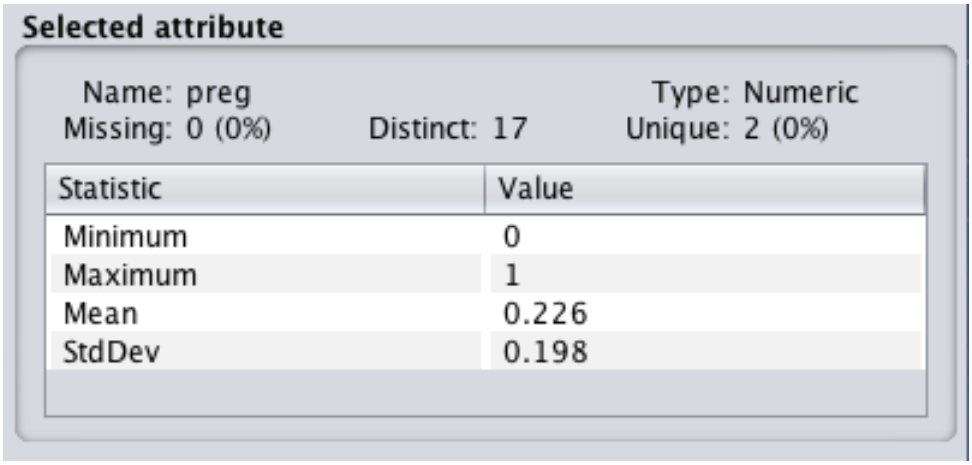


Figure 10.6: Weka Normalized Data Distribution.

You can use other scales such as -1 to 1, which is useful when using Support Vector Machines and AdaBoost. Normalization is useful when your data has varying scales and the algorithm

you are using does not make assumptions about the distribution of your data, such as k-Nearest Neighbors and Artificial Neural Networks.

10.3 Standardize Your Numeric Attributes

Data standardization is the process of rescaling one or more attributes so that they have a mean value of 0 and a standard deviation of 1. Standardization assumes that your data has a Gaussian (bell curve) distribution. This does not strictly have to be true, but the technique is more effective if your attribute distribution is Gaussian. You can standardize all of the attributes in your dataset with Weka by choosing the *Standardize* filter and applying it your dataset. You can use the following recipe to standardize your dataset:

- 1. Open the *Weka Explorer*.
- 2. Load the `data/diabetes.arff` dataset.
- 3. Click the *Choose* button to and select the *unsupervised.attribute.Standardize* filter.

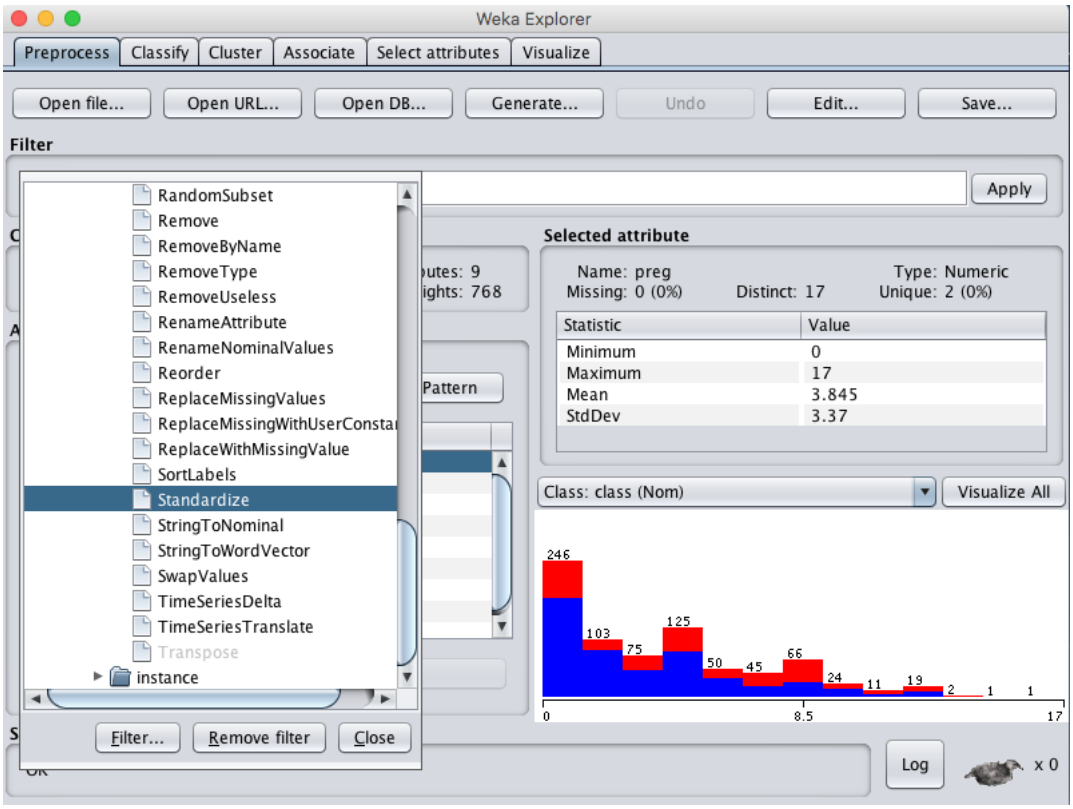


Figure 10.7: Weka Select Standardize Data Filter.

- 4. Click the *Apply* button to normalize your dataset.
- 5. Click the *Save* button and type a filename to save the standardized copy of your dataset.

Reviewing the details of each attribute in the *Selected attribute* window will give you confidence that the filter was successful and that each attribute has a mean of 0 and a standard deviation of 1.

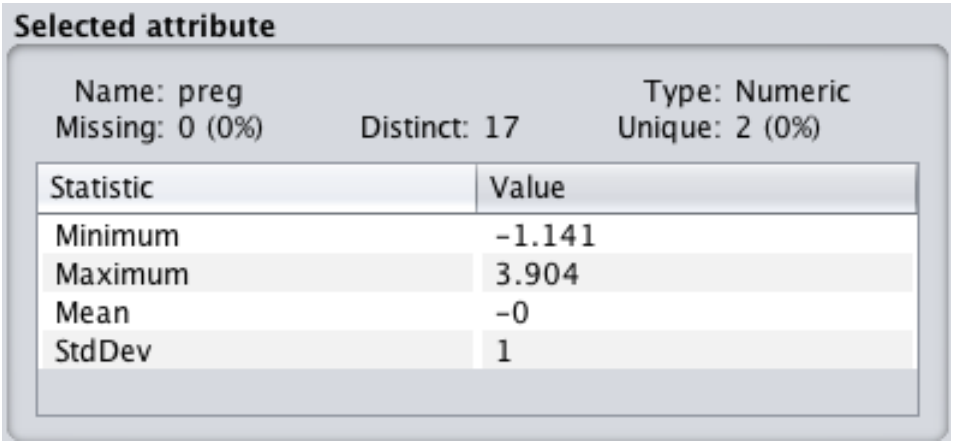


Figure 10.8: Weka Standardized Data Distribution.

Standardization is useful when your data has varying scales and the algorithm you are using does make assumptions about your data having a Gaussian distribution, such as linear regression, logistic regression and linear discriminant analysis.

10.4 Summary

In this lesson you discovered how to rescale your dataset in Weka. Specifically, you learned:

- How to normalize your dataset to the range 0 to 1.
- How to standardize your data to have a mean of 0 and a standard deviation of 1.
- When to use normalization and standardization.

10.4.1 Next

Weka provides a large assortment of data filters. In the next lesson you will learn how you can transform attributes using more advanced data filters.

Chapter 11

How to Transform Your Machine Learning Data

Often your raw data for machine learning is not in an ideal form for modeling. You need to prepare or reshape it to meet the expectations of different machine learning algorithms. In this lesson you will discover two techniques that you can use to transform your machine learning data ready for modeling. After reading this lesson you will know:

- How to convert a real valued attribute into a discrete distribution called discretization.
- How to convert a discrete attribute into multiple real values called dummy variables.
- When to discretize or create dummy variables from your data.

Let's get started.

11.1 Discretize Numerical Attributes

Some machine learning algorithms prefer or find it easier to work with discrete attributes. For example, decision tree algorithms can choose split points in real valued attributes, but are much cleaner when split points are chosen between bins or predefined groups in the real-valued attributes. Discrete attributes are those that describe a category, called nominal attributes. Those attributes that describe a category that where there is a meaning in the order for the categories are called ordinal attributes. The process of converting a real-valued attribute into an ordinal attribute or bins is called discretization.

You can discretize your real valued attributes in Weka using the *Discretize* filter. The tutorial below demonstrates how to use the *Discretize* filter. The Pima Indians onset of diabetes dataset is used to demonstrate this filter because of the input values are real-valued and grouping them into bins may make sense. You can learn more about this dataset in Section 8.2.1.

- 1. Open the *Weka Explorer*.
- 2. Load the `data/diabetes.arff` dataset.

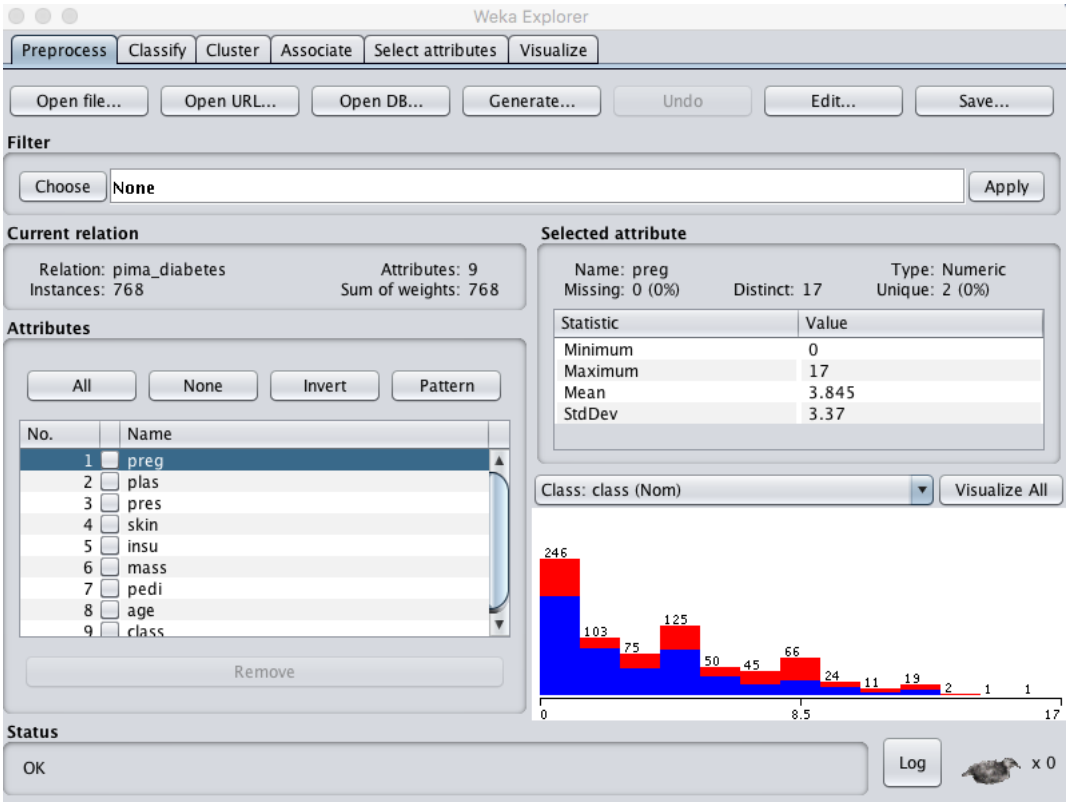


Figure 11.1: Weka Explorer Loaded Diabetes Dataset.

- 3. Click the *Choose* button for the Filter and select the *unsupervised.attribute.Discretize* filter.

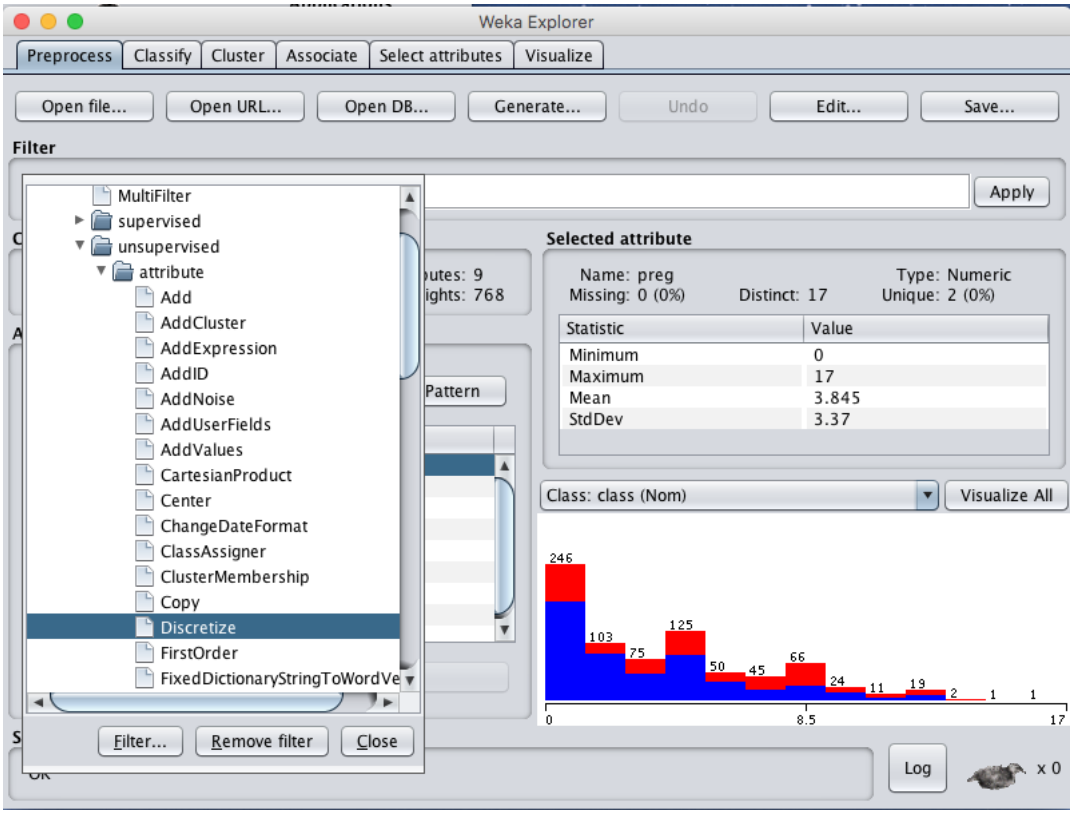


Figure 11.2: Weka Select Discretize Data Filter.

- 4. Click on the filter to configure it. You can select the indices of the attributes to discretize, the default is to discretize all attributes, which is what we will do in this case. Click the *OK* button.

- 5. Click the *Apply* button to apply the filter.

You can click on each attribute and review the details in the *Selected attribute* pane to confirm that the filter was applied successfully.

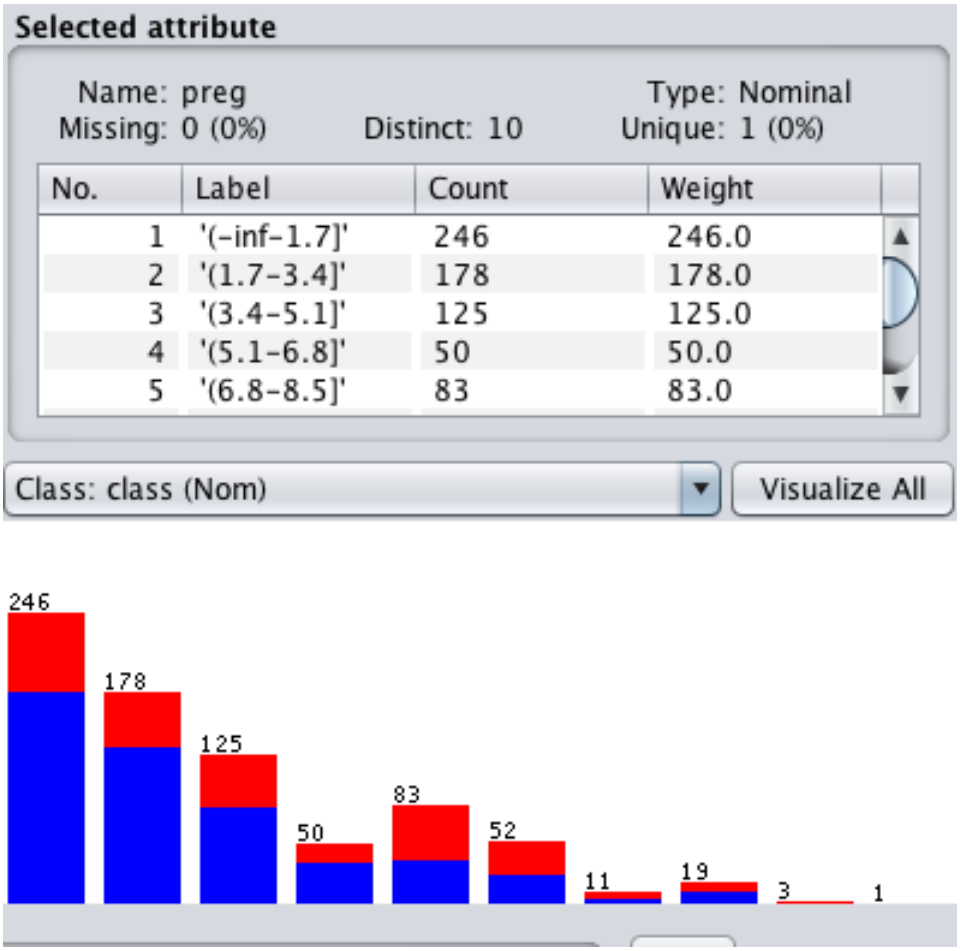


Figure 11.3: Weka Discretized Attribute.

Discretizing your real valued attributes is most useful when working with decision tree type algorithms. It is perhaps more useful when you believe that there are natural groupings within the values of given attributes.

11.2 Convert Nominal Attributes to Dummy Variables

Some machine learning algorithms prefer to use real valued inputs and do not support nominal or ordinal attributes. Nominal attributes can be converted to real values. This is done by creating one new binary attribute for each category. For a given instance that has a category for that value, the binary attribute is set to 1 and the binary attributes for the other categories is set to 0. This process is called creating dummy variables.

You can create dummy binary variables from nominal attributes in Weka using the *NominalToBinary* filter. The recipe below demonstrates how to use the *NominalToBinary* filter. The Contact Lenses dataset is used to demonstrate this filter because the attributes are all nominal and provide plenty of opportunity for creating dummy variables. You can download the Contact Lenses dataset from the UCI Machine learning repository. You can also access the dataset directory in your installation of Weka under the `data/` directory by loading the file `contact-lenses.arff`.

- 1. Open the *Weka Explorer*.
- 2. Load the `data/contact-lenses.arff` dataset.

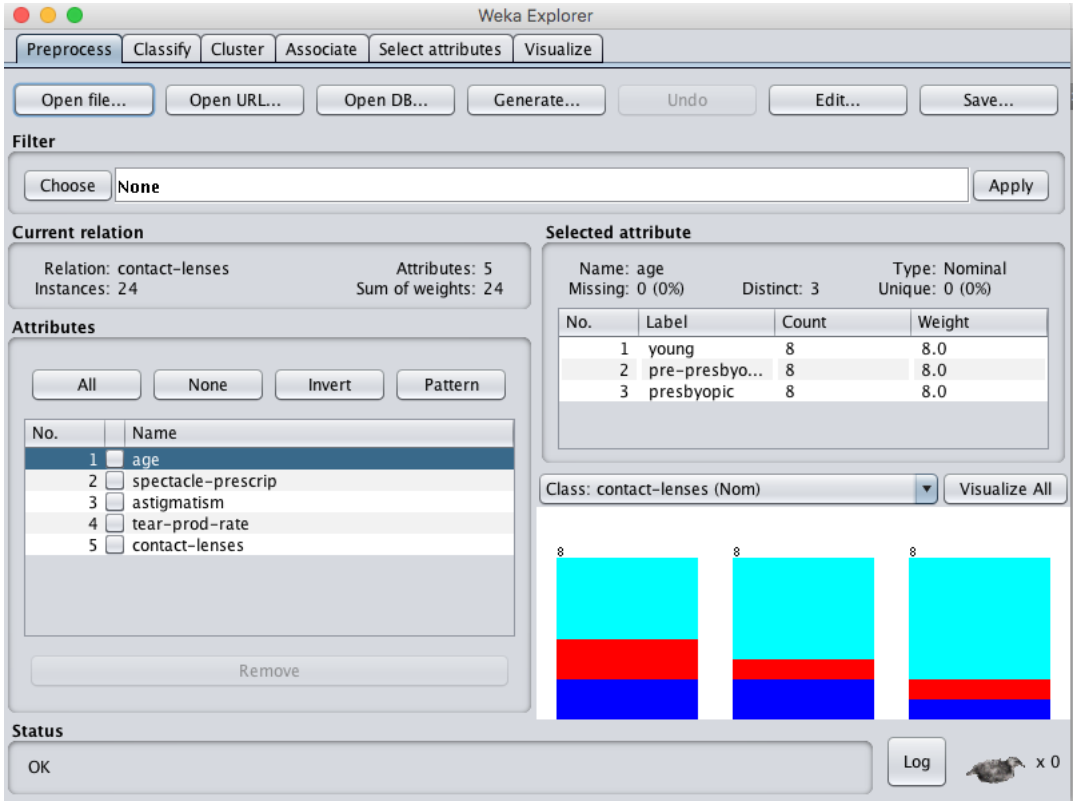


Figure 11.4: Weka Explorer Loaded Contact Lenses Dataset.

- 3. Click the *Choose* button the *unsupervised.attribute.NominalToBinary* filter.

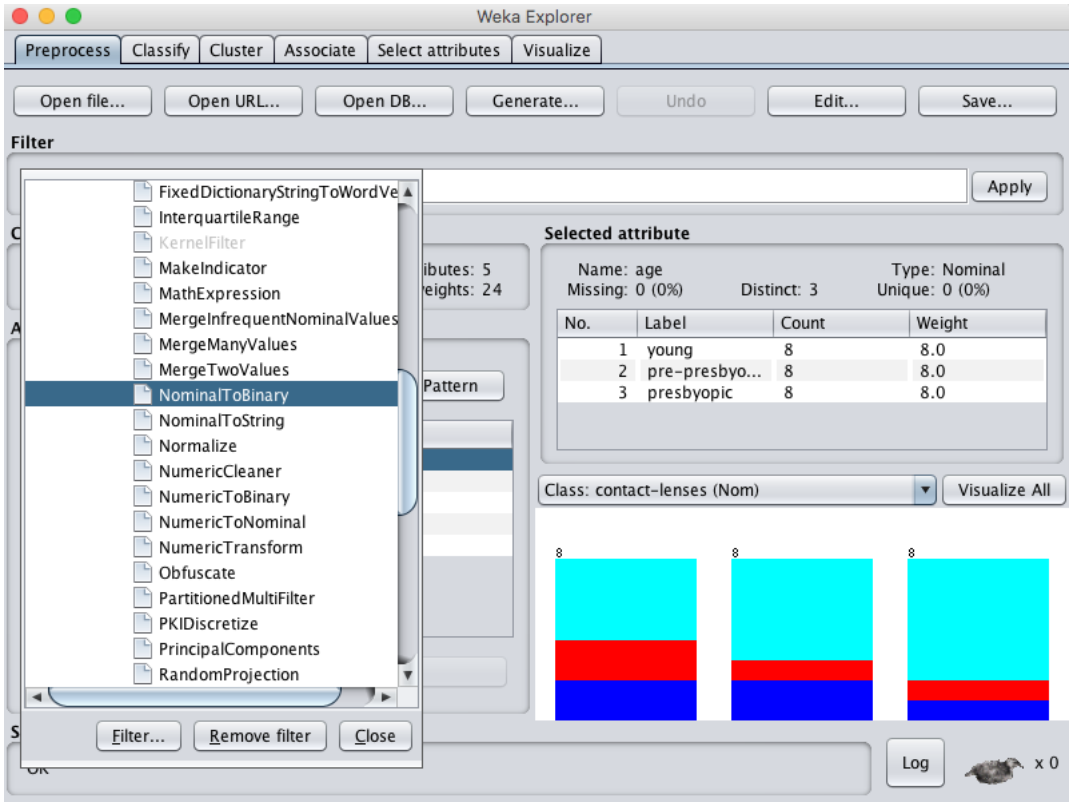


Figure 11.5: Weka Select NominalToBinary Data Filter.

- 4. Click on the filter to configure it. You can select the indices of the attributes to convert to binary values, the default is to convert all attributes. Change it to only the first attribute. Click the *OK* button.

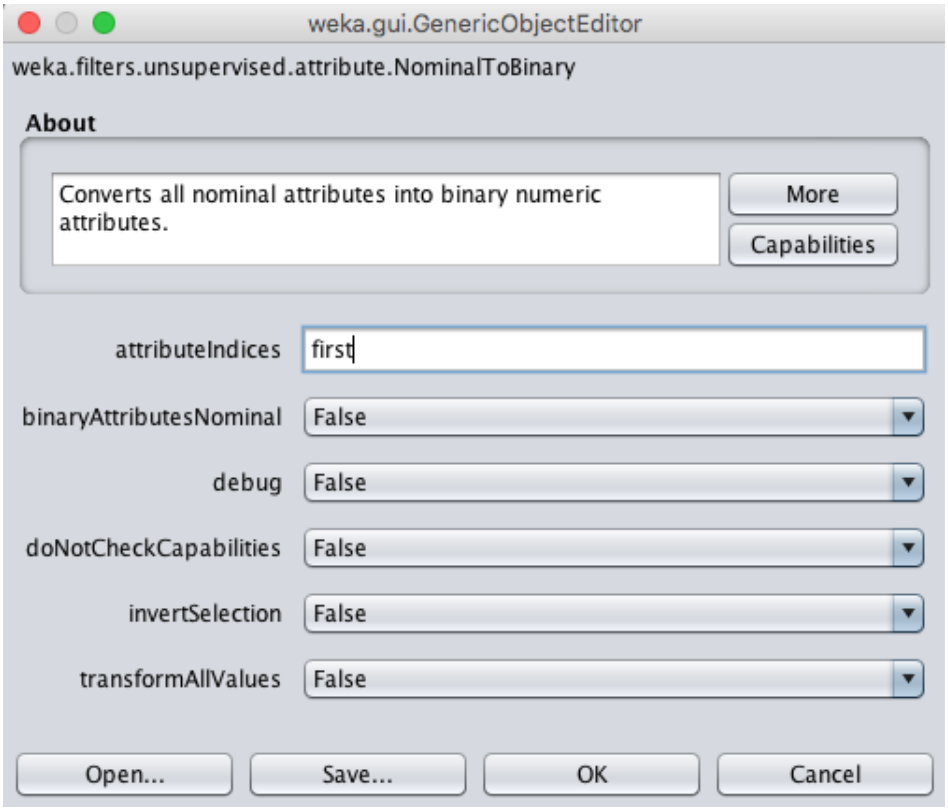


Figure 11.6: Weka NominalToBinary Data Filter Configuration.

- 5. Click the *Apply* button to apply the filter.

Reviewing the list of attributes will show that the age attribute has been removed and replaced with three new binary attributes: `age=young`, `age=pre-presbyopic` and `age=presbyopic`.

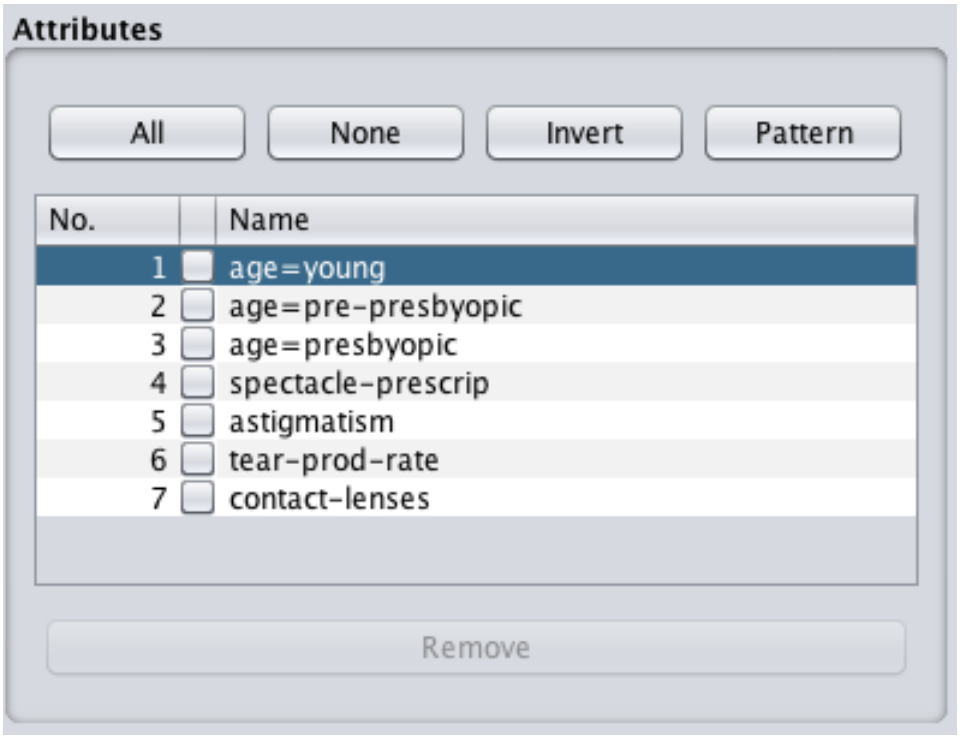


Figure 11.7: Weka Nominal Attribute Converted to Dummy Variables.

Creating dummy variables is useful for techniques that do not support nominal input variables like linear regression and logistic regression. It can also prove useful in techniques like k-nearest neighbors and artificial neural networks.

11.3 Summary

In this lesson you discovered how to transform your machine learning data to meet the expectations of different machine learning algorithms. Specifically, you learned:

- How to convert real valued input attributes to nominal attributes called discretization.
- How to convert a categorical input variable to multiple binary input attributes called dummy variables.
- When to use discretization and dummy variables when modeling data.

11.3.1 Next

Raw data is often not suitable for modeling directly. It may need to be cleaned first. In the next lesson you will learn how to identify, mark, remove and impute missing values in your dataset.

Chapter 12

How To Handle Missing Values In Machine Learning Data

Data is rarely clean and often you can have corrupt or missing values. It is important to identify, mark and handle missing data when developing machine learning models in order to get the very best performance. In this lesson you will discover how to handle missing values in your machine learning data using Weka. After reading this lesson you will know:

- How to mark missing values in your dataset.
- How to remove data with missing values from your dataset.
- How to impute missing values.

Let's get started.

12.1 Mark Missing Values

The problem used for this example is the Pima Indians onset of diabetes dataset. You can learn more about this dataset in Section 8.2.1. The Pima Indians dataset is a good basis for exploring missing data. Some attributes such as blood pressure (`pres`) and Body Mass Index (`mass`) have values of zero, which are impossible. These are examples of corrupt or missing data that must be marked manually. You can mark missing values in Weka using the *NumericCleaner* filter. The recipe below shows you how to use this filter to mark the 11 missing values on the Body Mass Index (`mass`) attribute.

- 1. Open the *Weka Explorer*.
- 2. Load the `data/diabetes.arff` dataset.
- 3. Click the *Choose* button and select the *unsupervised.attribute.NumericCleaner* filter.

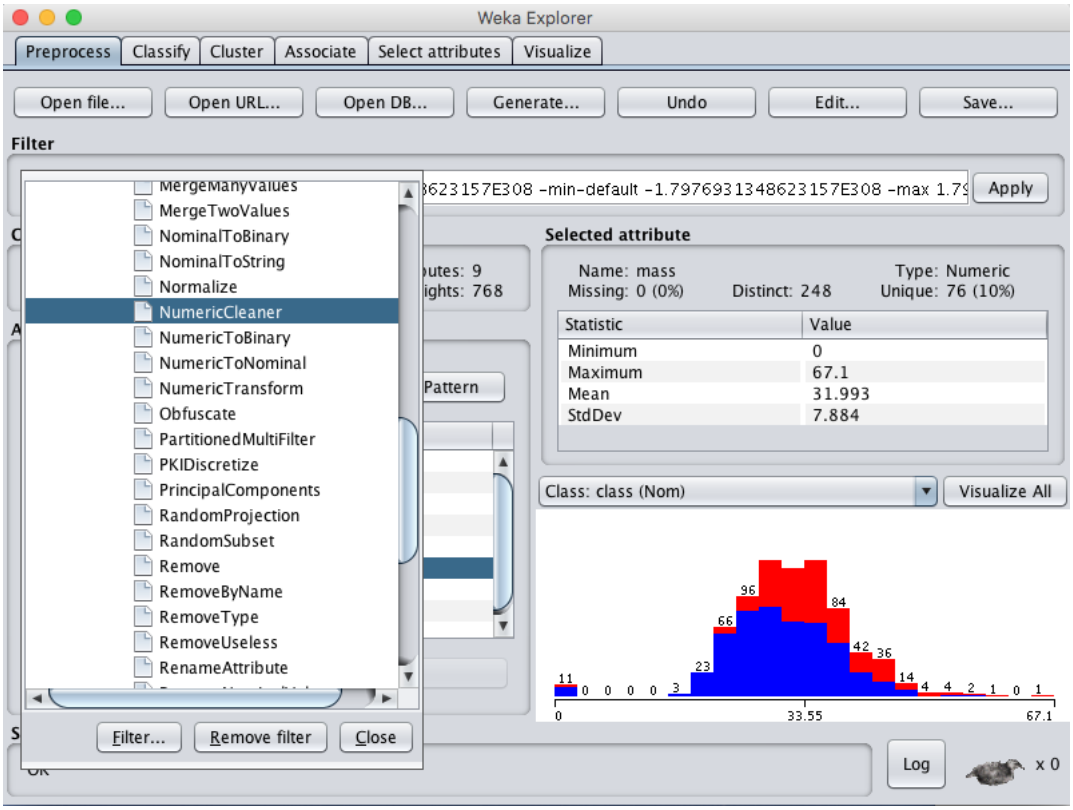


Figure 12.1: Weka Select NumericCleaner Data Filter.

- 4. Click on the filter to configure it.
- 5. Set the *attributeIndicies* to 6, the index of the mass attribute.
- 6. Set *minThreshold* to 0.1E-8 (close to zero), which is the minimum value allowed for the attribute.
- 7. Set *minDefault* to NaN, which is unknown and will replace values below the threshold.
- 8. Click the *OK* button on the filter configuration.
- 9. Click the *Apply* button to apply the filter.
- 10. Click `mass` in the *Attributes* pane and review the details of the *Selected attribute*.

Notice that the 11 attribute values that were formally set to 0 are now marked as *Missing*.

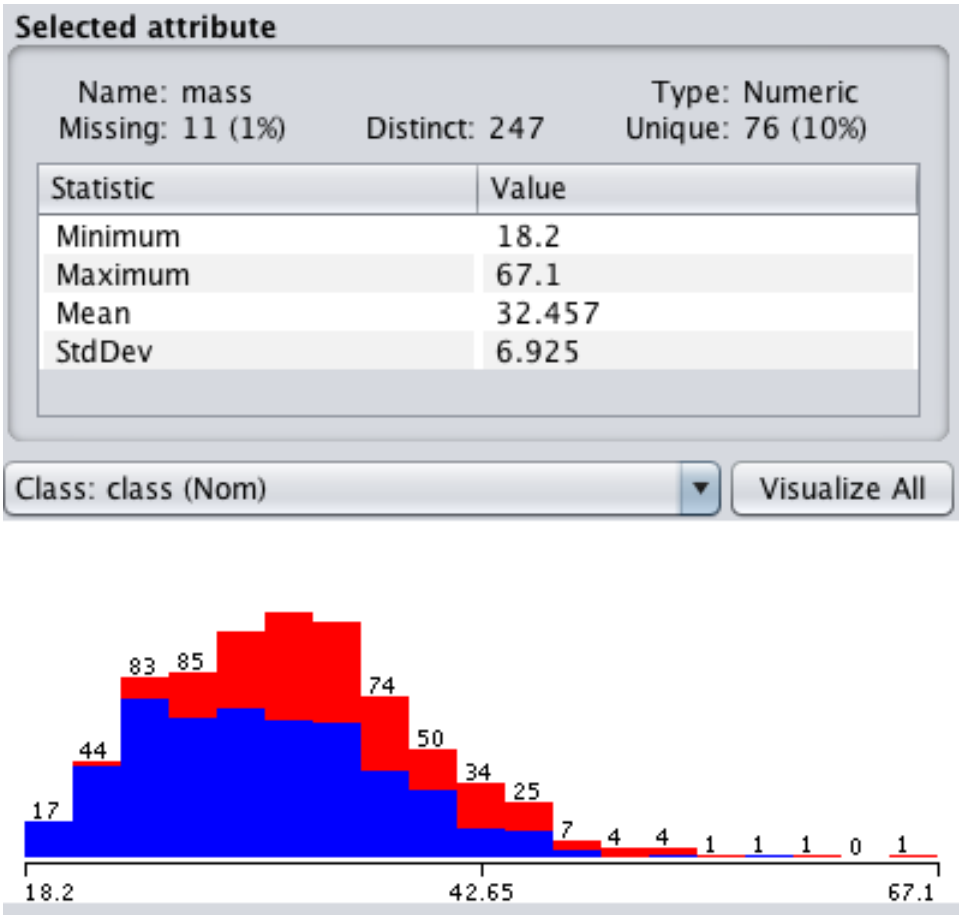


Figure 12.2: Weka Missing Data Marked.

In this example we marked values below a threshold as missing. You could just as easily mark them with a specific numerical value. You could also mark values missing between a upper and lower range of values. Next, let's look at how we can remove instances with missing values from our dataset.

12.2 Remove Missing Data

Now that you know how to mark missing values in your data, you need to learn how to handle them. A simple way to handle missing data is to remove those instances that have one or more missing values. You can do this in Weka using the *RemoveWithValues* filter. Continuing on from the above recipe to mark missing values, you can remove missing values as follows:

- 1. Click the *Choose* button and select the *unsupervised.instance.RemoveWithValues* filter.

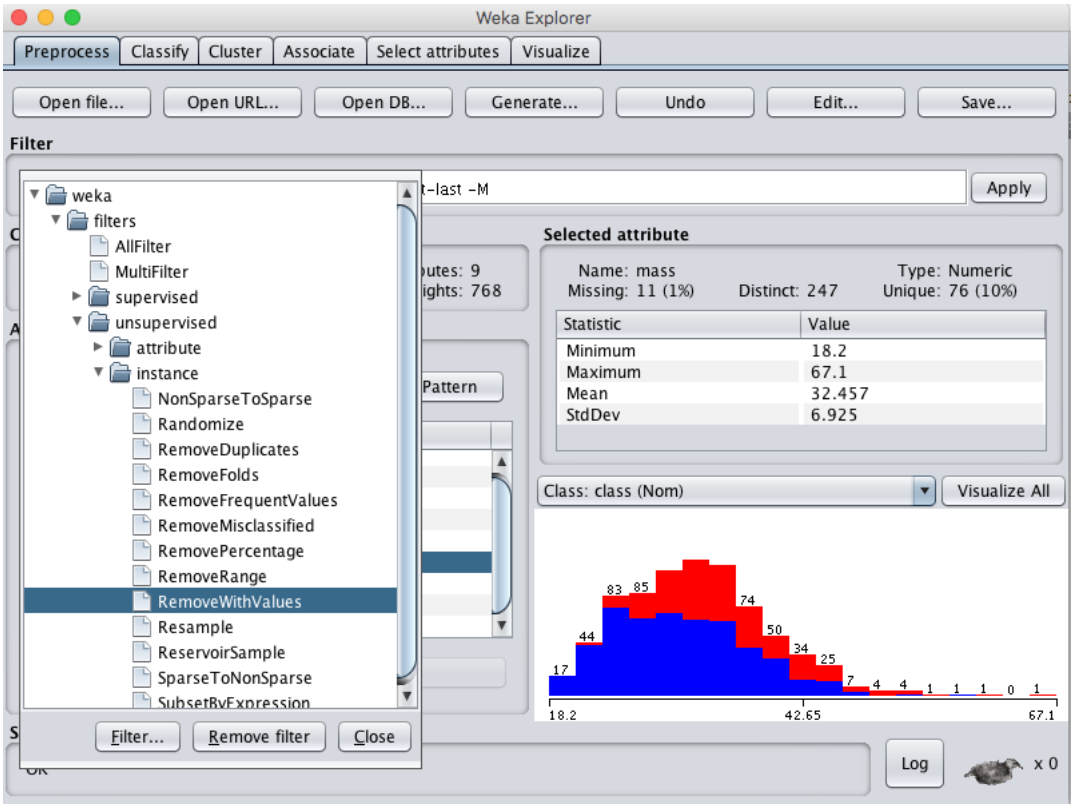


Figure 12.3: Weka Select RemoveWithValues Data Filter.

- 2. Click on the filter to configure it.
- 3. Set the *attributeIndicies* to 6, the index of the mass attribute.
- 4. Set *matchMissingValues* to *True*.
- 5. Click the *OK* button to use the configuration for the filter.
- 6. Click the *Apply* button to apply the filter.
- 7. Click `mass` in the *Attributes* section and review the details of the *Selected attribute*.

Notice that the 11 attribute values that were marked Missing have been removed from the dataset.

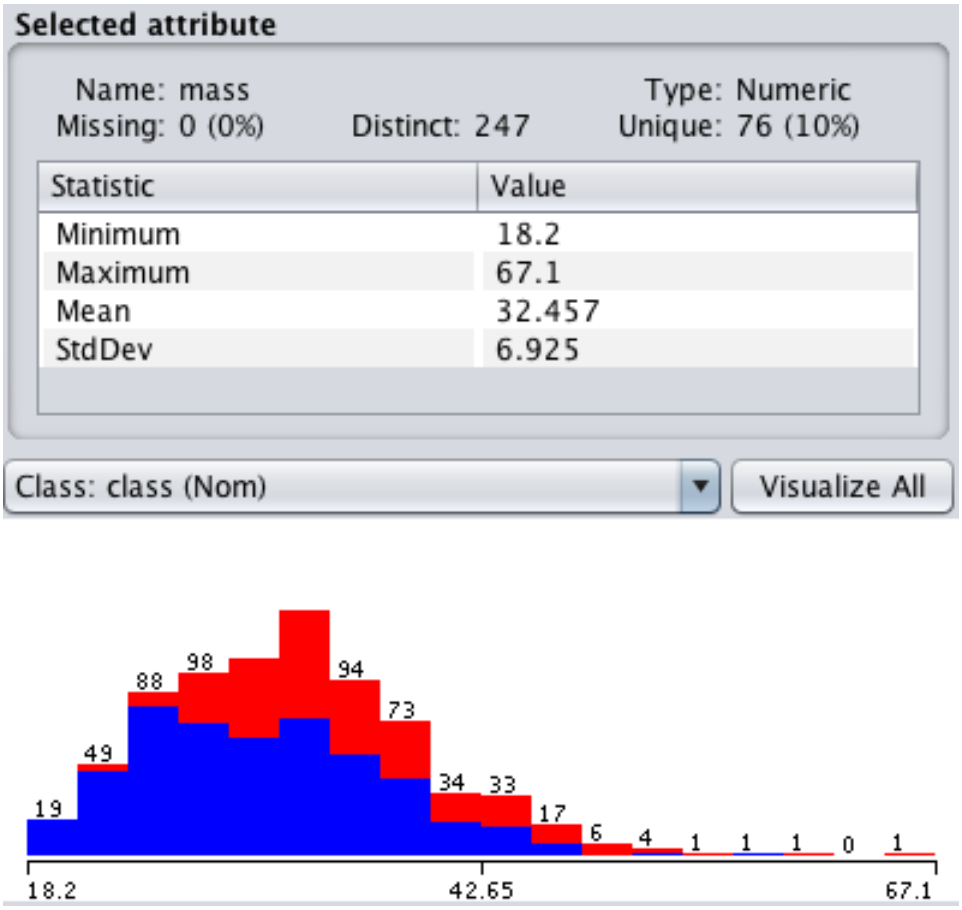


Figure 12.4: Weka Missing Values Removed.

Note, you can undo this operation by clicking the *Undo* button.

12.3 Impute Missing Values

Instances with missing values do not have to be removed, you can replace the missing values with some other value. This is called imputing missing values. It is common to impute missing values with the mean of the numerical distribution. You can do this easily in Weka using the *ReplaceMissingValues* filter. Continuing on from the first recipe above to mark missing values, you can impute the missing values as follows:

- 1. Click the *Choose* button and select the *unsupervised.attribute.ReplaceMissingValues* filter.

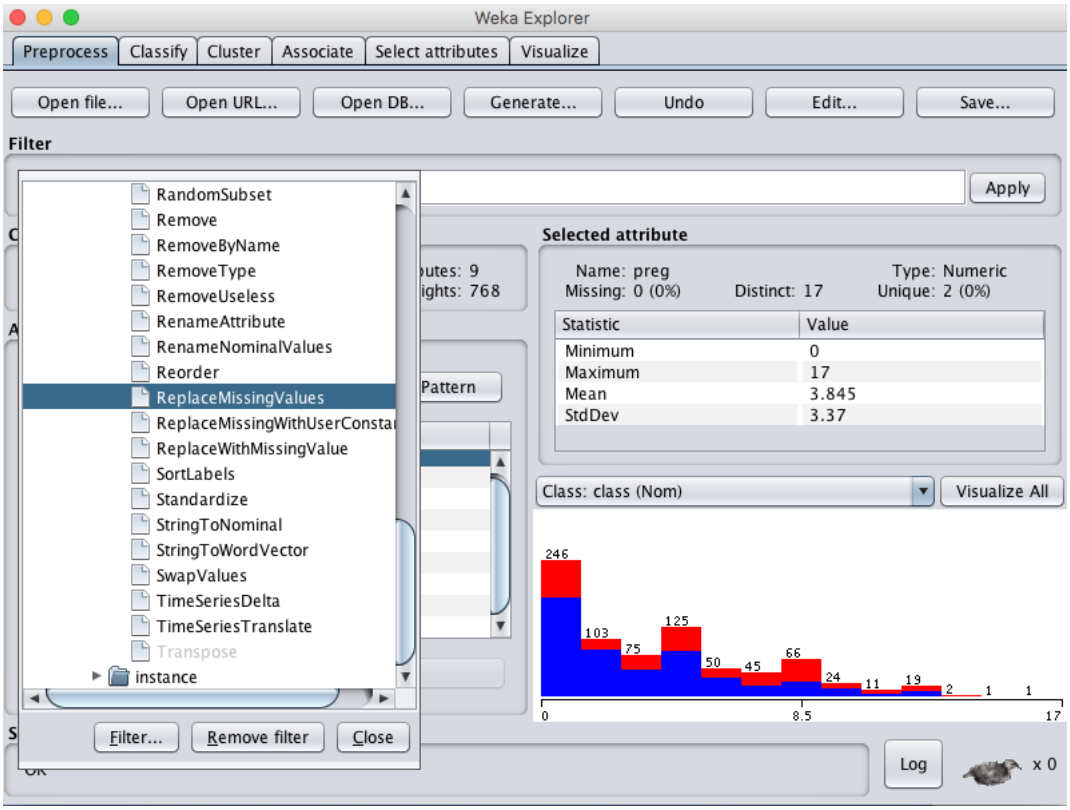


Figure 12.5: Weka ReplaceMissingValues Data Filter.

- 2. Click the *Apply* button to apply the filter to your dataset.
- 3. Click `mass` in the *Attributes* section and review the details of the *Selected attribute*.

Notice that the 11 attribute values that were marked *Missing* have been set to the mean value of the distribution.

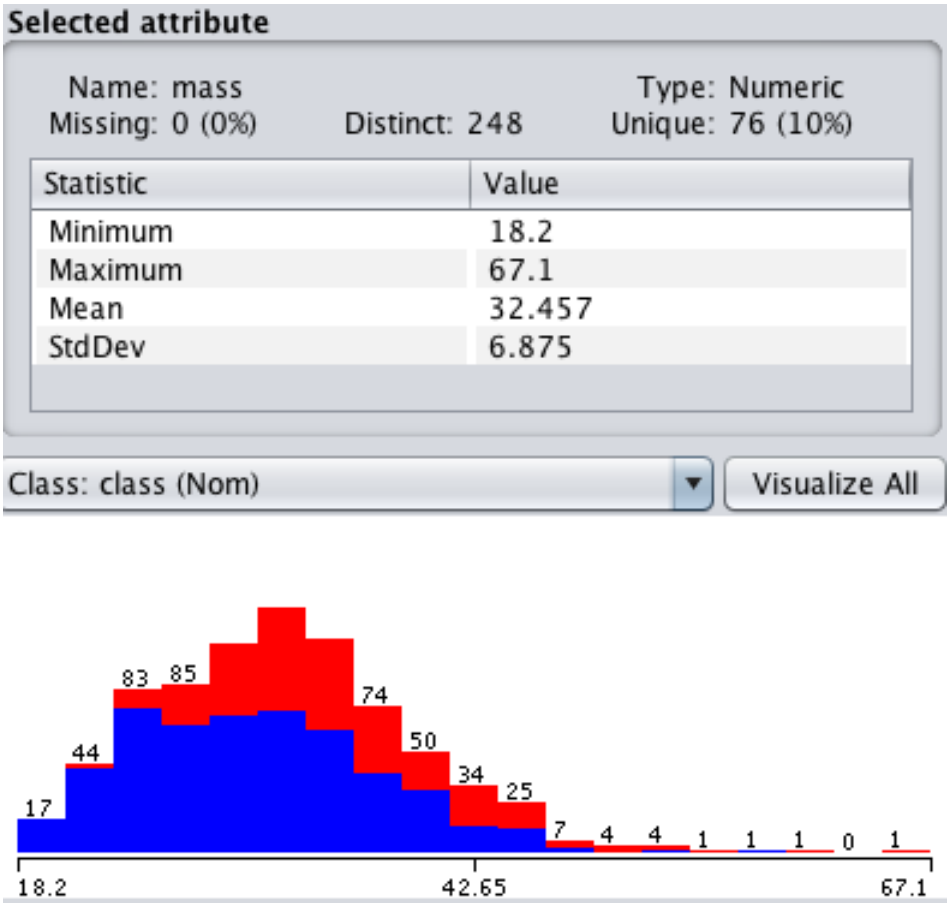


Figure 12.6: Weka Imputed Values.

12.4 Summary

In this lesson you discovered how you can handle missing data in your machine learning dataset using Weka. Specifically, you learned:

- How to mark corrupt values as missing in your dataset.
- How to remove instances with missing values from your dataset.
- How to impute mean values for missing values in your dataset.

12.4.1 Next

Raw data may have many attributes, but not all may be relevant to a model in order to make predictions. In the next lesson you will learn how you can use feature selection to identify only those most relevant features to the output attribute.

Chapter 13

How to Perform Feature Selection With Machine Learning Data

Raw machine learning data contains a mixture of attributes, some of which are relevant to making predictions. How do you know which features to use and which to remove? The process of selecting features in your data to model your problem is called feature selection. In this lesson you will discover how to perform feature selection with your machine learning data in Weka. After reading this lesson you will know:

- About the importance of feature selection when working through a machine learning problem.
- How feature selection is supported on the Weka platform.
- How to use various different feature selection techniques in Weka on your dataset.

Let's get started.

13.1 Feature Selection in Weka

The problem used for this example is the Pima Indians onset of diabetes dataset. You can learn more about this dataset in Section 8.2.1. Many feature selection techniques are supported in Weka. A good place to get started exploring feature selection in Weka is in the *Weka Explorer*.

1. Open the *Weka GUI Chooser*.
2. Click the *Explorer* button to launch the *Weka Explorer*.
3. Open the `data/diabetes.arff` dataset.
4. Click the *Select attributes* tab to access the feature selection methods.

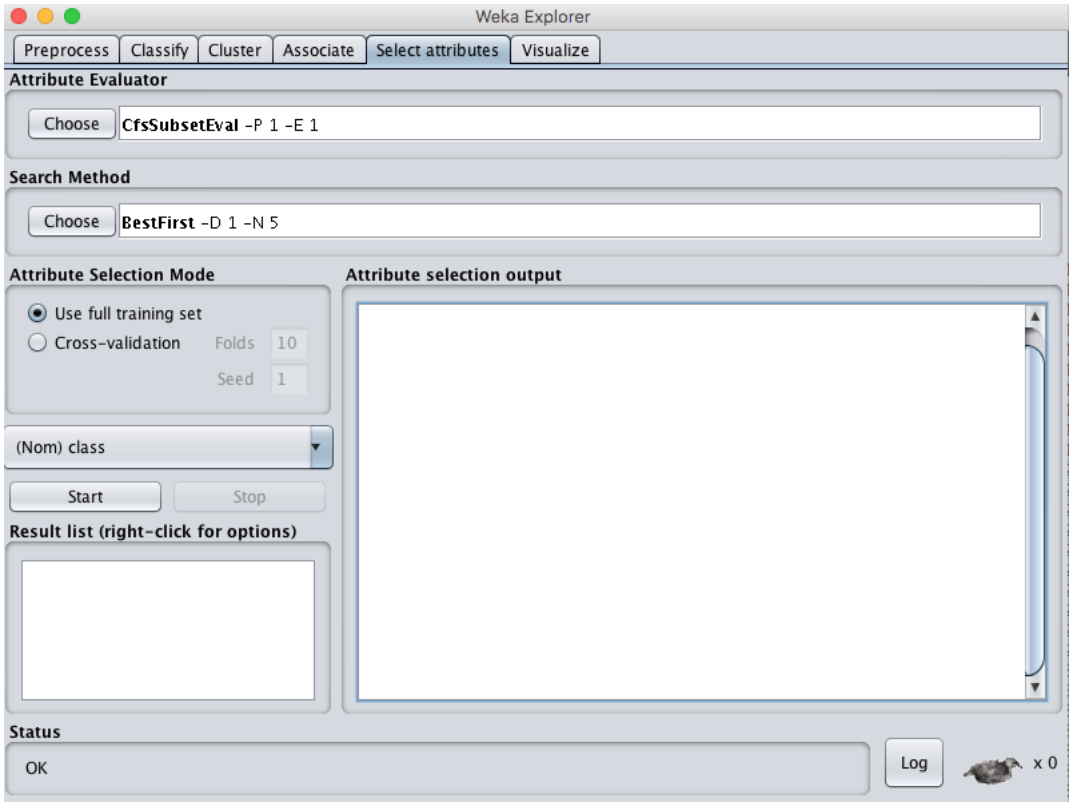


Figure 13.1: Weka Feature Selection.

Feature selection is divided into two parts:

- *Attribute Evaluator*.
- *Search Method*.

Each section has multiple techniques from which to choose. The *Attribute Evaluator* is the technique by which each attribute in your dataset (also called a column or feature) is evaluated in the context of the output variable (e.g. the class). The *Search Method* is the technique by which to try or navigate different combinations of attributes in the dataset in order to arrive on a shortlist of chosen features. Some *Attribute Evaluator* techniques require the use of specific *Search Methods*. For example, the *CorrelationAttributeEval* technique used in the next section can only be used with a *Ranker Search Method*, that evaluates each attribute and lists the results in a rank order. When selecting different *Attribute Evaluators*, the interface may ask you to change the *Search Method* to something compatible with the chosen technique.

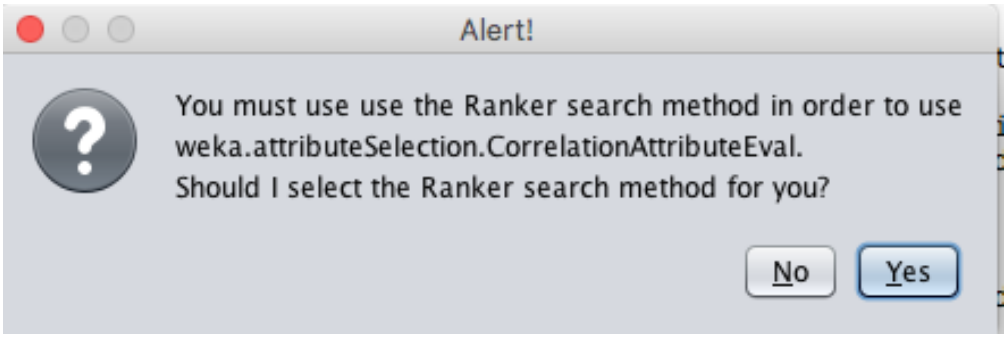


Figure 13.2: Weka Feature Selection Alert.

Both the *Attribute Evaluator* and *Search Method* techniques can be configured. Once chosen, click on the name of the technique to get access to its configuration details.

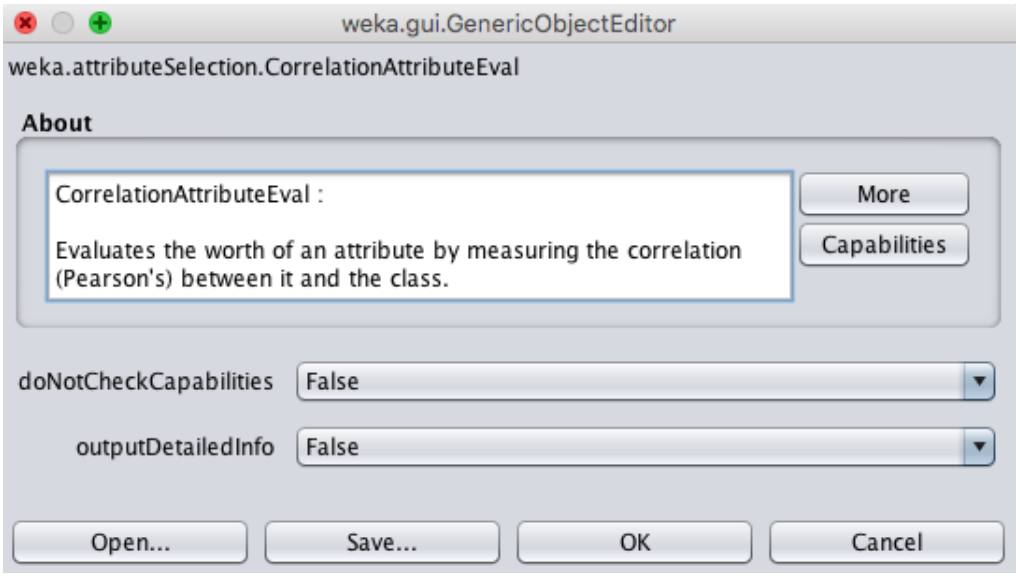


Figure 13.3: Weka Feature Selection Configuration.

Click the *More* button to get more documentation on the feature selection technique and configuration parameters. Hover your mouse cursor over a configuration parameter to get a tooltip containing more details.

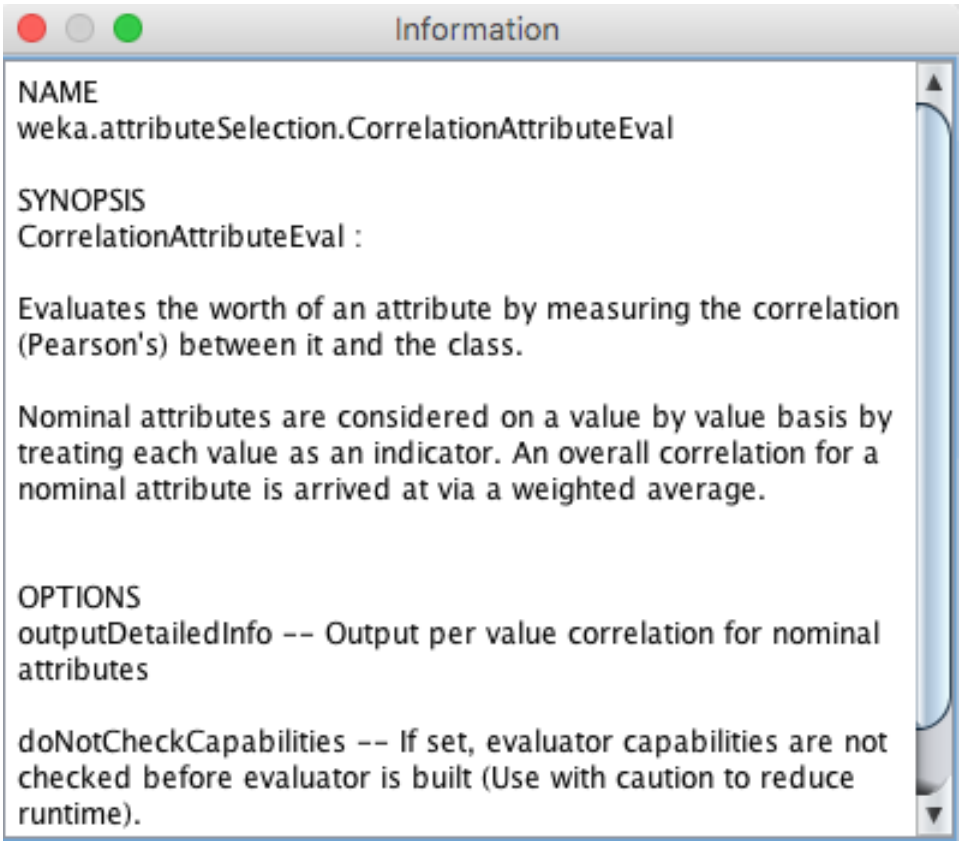


Figure 13.4: Weka Feature Selection More Information.

Now that we know how to access feature selection techniques in Weka, let's take a look at how to use some popular methods on our chosen standard dataset.

13.2 Correlation Based Feature Selection

A popular technique for selecting the most relevant attributes in your dataset is to use correlation. More formally, correlation can be calculated to determine how much two variables change together, either in the same or differing directions on the number line. You can calculate the correlation between each attribute and the output variable and select only those attributes that have a moderate-to-high positive or negative correlation (close to -1 or 1) and drop those attributes with a low correlation (value close to zero).

Weka supports correlation based feature selection with the *CorrelationAttributeEval* technique that requires use of a *Ranker Search Method*. Running this on our Pima Indians dataset suggests that one attribute (`plas`) has the highest correlation with the output class. It also suggests a host of attributes with some modest correlation (`mass`, `age`, `preg`). If we use 0.2 as our cut-off for relevant attributes, then the remaining attributes could possibly be removed (`pedi`, `insu`, `skin` and `pres`).

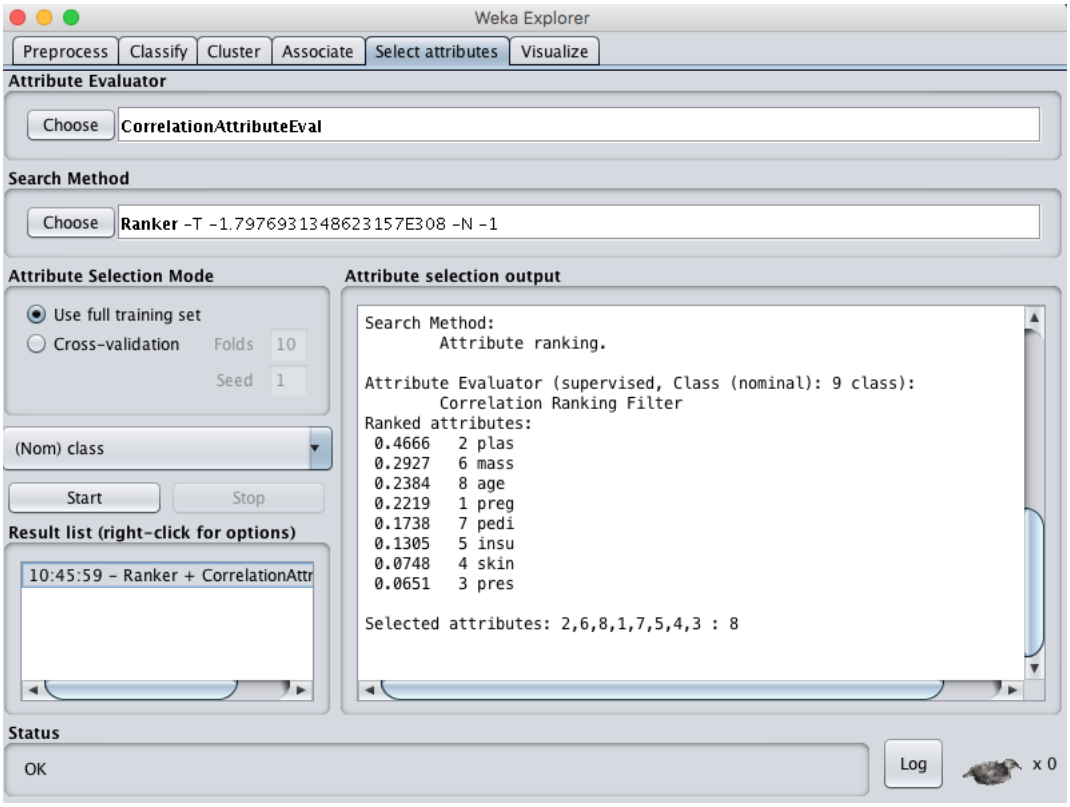


Figure 13.5: Weka Correlation-Based Feature Selection Method.

13.3 Information Gain Based Feature Selection

Another popular feature selection technique is to calculate the information gain. You can calculate the information gain (also called entropy) for each attribute for the output variable. Entry values vary from 0 (no information) to 1 (maximum information). Those attributes that contribute more information will have a higher information gain value and can be selected, whereas those that do not add much information will have a lower score and can be removed.

Weka supports feature selection via information gain using the *InfoGainAttributeEval Attribute Evaluator*. Like the correlation technique above, the *Ranker Search Method* method must be used. Running this technique on our Pima Indians dataset we can see that one attribute contributes more information than all of the others (`plas`). If we use an arbitrary cutoff of 0.05, then we would also select the `mass`, `age` and `insu` attributes and drop the rest from our dataset.

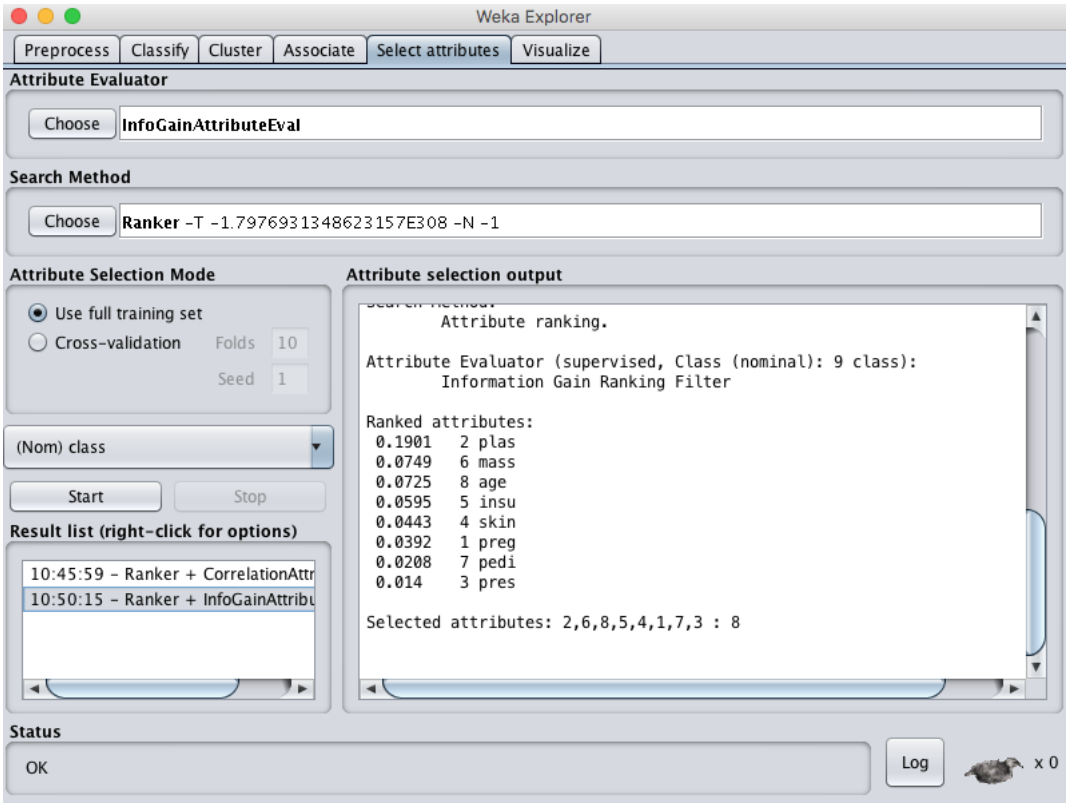


Figure 13.6: Weka Information Gain-Based Feature Selection Method.

13.4 Learner Based Feature Selection

A popular feature selection technique is to use a generic but powerful learning algorithm and evaluate the performance of the algorithm on the dataset with different subsets of attributes selected. The subset that results in the best performance is taken as the selected subset. The algorithm used to evaluate the subsets does not have to be the algorithm that you intend to use to model your problem, but it should be generally quick to train and powerful, like a decision tree method. In Weka this type of feature selection is supported by the WrapperSubsetEval technique and must use a *GreedyStepwise* or *BestFirst Search Method*. The latter, *BestFirst*, is preferred if you can spare the compute time.

- 1. First select the *WrapperSubsetEval* technique.
- 2. Click on the name *WrapperSubsetEval* to open the configuration for the method.
- 3. Click the *Choose* button for the *classifier* parameter and change it to *trees.J48*.

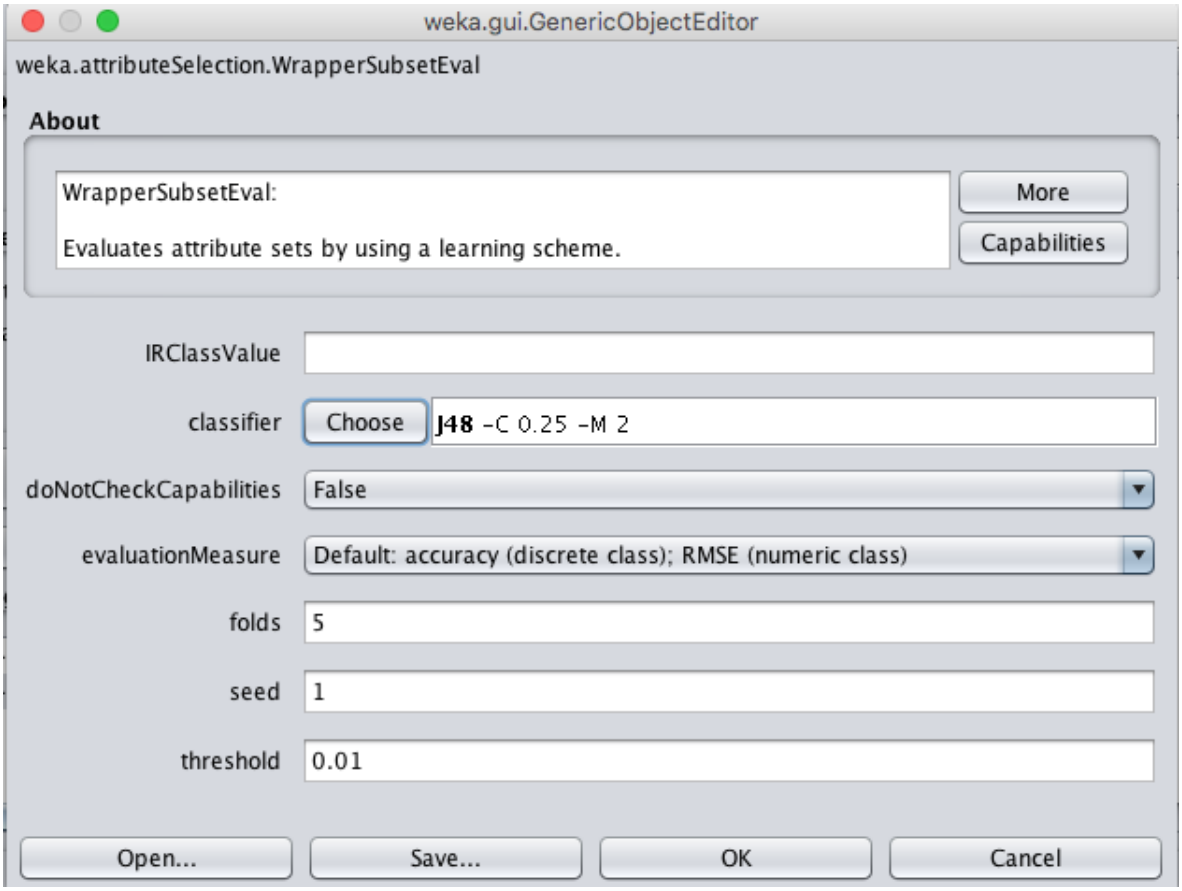


Figure 13.7: Weka Wrapper Feature Selection Configuration.

- 4. Click *OK* to accept the configuration.
- 5. Change the *Search Method* to *BestFirst*.
- 6. Click the *Start* button to evaluate the features.

Running this feature selection technique on the Pima Indians dataset selects 4 of the 8 input variables: `plas`, `pres`, `mass` and `age`.

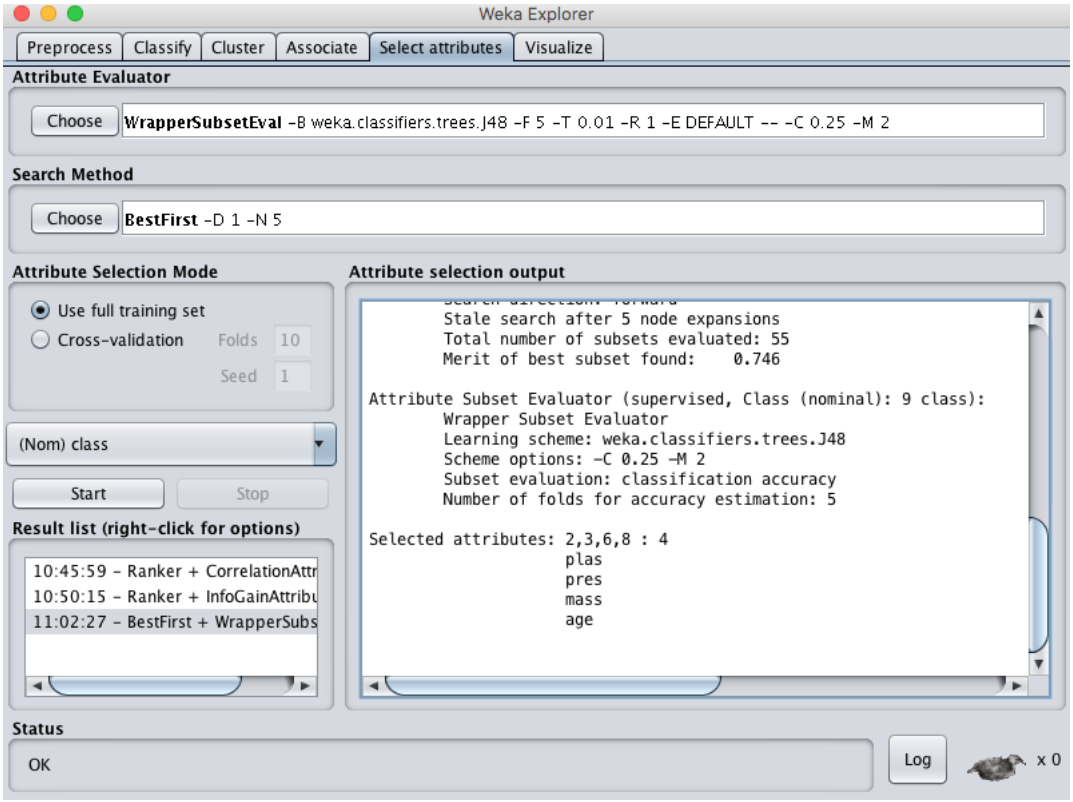


Figure 13.8: Weka Wrapper Feature Selection Method.

13.5 Select Attributes in Weka

Looking back over the three techniques, we can see some overlap in the selected features (e.g. `plas`), but also differences. It is a good idea to evaluate a number of different views of your machine learning dataset. A view of your dataset is nothing more than a subset of features selected by a given feature selection technique. It is a copy of your dataset that you can easily make in Weka. For example, taking the results from the last feature selection technique, let's say we wanted to create a view of the Pima Indians dataset with only the following attributes: `plas`, `pres`, `mass` and `age`:

- 1. Click the *Preprocess* tab.
- 2. In the *Attributes* selection Tick all but the `plas`, `pres`, `mass`, `age` and `class` attributes.

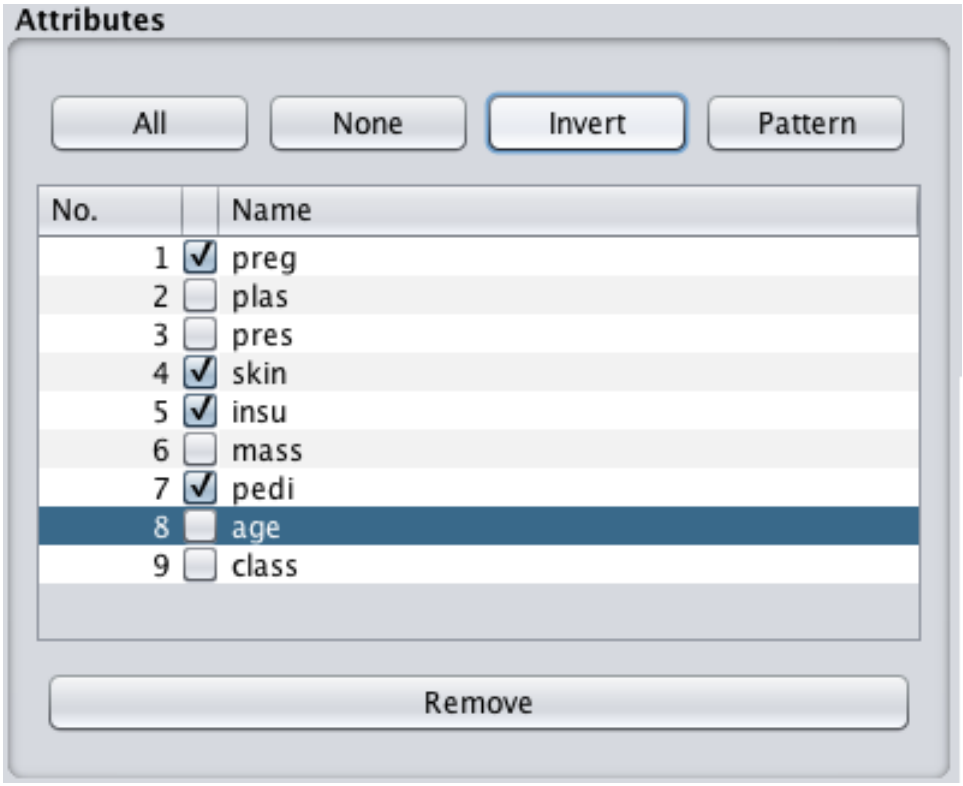


Figure 13.9: Weka Select Attributes To Remove From Dataset.

- 3. Click the *Remove* button.
- 4. Click the *Save* button and enter a filename.

You now have a new view of your dataset to explore.

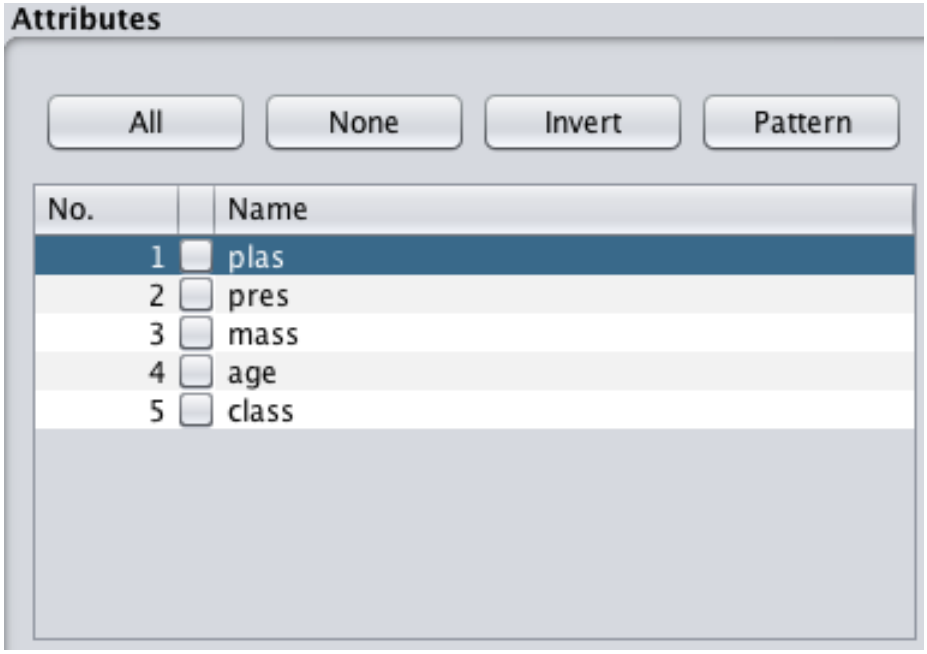


Figure 13.10: Weka Attributes Removed From Dataset.

13.6 What Feature Selection Techniques To Use

You cannot know which views of your data will produce the most accurate models. Therefore, it is a good idea to try a number of different feature selection techniques on your data and in turn create many different views of your data. Select a good generic technique, like a decision tree, and build a model for each view of your data. Compare the results to get an idea of which view of your data results in the best performance. This will give you an idea of the view or more specifically features that best expose the structure of your problem to learning algorithms in general.

13.7 Summary

In this lesson you discovered the importance of feature selection and how to use feature selection on your data with Weka. Specifically, you learned:

- How to perform feature selection using correlation.
- How to perform feature selection using information gain.
- How to perform feature selection by training a model on different subsets of features.

13.7.1 Next

This concludes our lessons working with data. Next you will take a look at the machine learning algorithm supported by Weka and how the Weka platform let's you configure and manage algorithms.

Chapter 14

How to Use Machine Learning Algorithms

A big benefit of using the Weka platform is the large number of supported machine learning algorithms. The more algorithms that you can try on your problem the more you will learn about your problem and likely the closer you will get to discovering the one or few algorithms that perform best. In this lesson you will discover the machine learning algorithms supported by Weka. After reading this lesson you will know:

- The different types of machine learning algorithms supported and key algorithms to try in Weka.
- How algorithms can be configured in Weka and how to save and load good algorithm configurations.
- How to learn more about the machine learning algorithms supported by Weka.

Let's get started.

14.1 Weka Machine Learning Algorithms

Weka has a lot of machine learning algorithms. This is great, it is one of the large benefits of using Weka as a platform for machine learning. A down side is that it can be a little overwhelming to know which algorithms to use, and when. Also, the algorithms have names that may not be familiar to you, even if you know them in other contexts. In this section we will start off by looking at some well known algorithms supported by Weka. What we will learn in this lesson applies to the machine learning algorithms used across the Weka platform, but the Explorer is the best place to learn more about the algorithms as they are all available in one easy place. The problem used for this example is the Pima Indians onset of diabetes dataset. You can learn more about this dataset in Section 8.2.1.

1. Open the *Weka GUI Chooser*.
2. Click the *Explorer* button to open the *Weka Explorer*.
3. Open the `data/diabetes.arff` dataset.

4. Click *Classify* to open the *Classify* tab.

The *Classify* tab of the *Weka Explorer* is where you can learn about the various different algorithms and explore predictive modeling. You can choose a machine learning algorithm by clicking the *Choose* button.

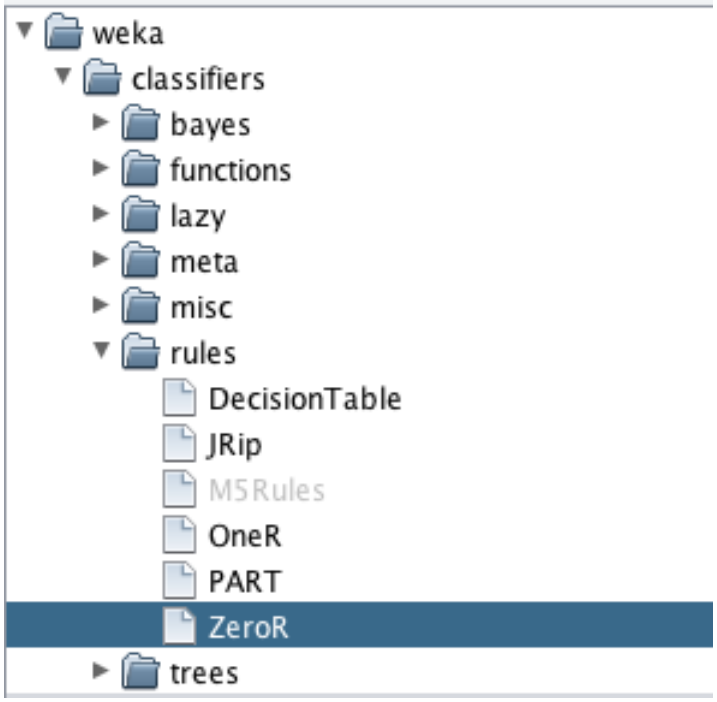


Figure 14.1: Weka Choose a Machine Learning Algorithms.

Clicking on the *Choose* button presents you with a list of machine learning algorithms to choose from. They are divided into a number of main groups:

- **bayes**: Algorithms that use Bayes Theorem in some core way, like Naive Bayes.
- **function**: Algorithms that estimate a function, like Linear Regression.
- **lazy**: Algorithms that use lazy learning, like k-Nearest Neighbors.
- **meta**: Algorithms that use or combine multiple algorithms, like Ensembles.
- **misc**: Implementations that do not neatly fit into the other groups, like running a saved model.
- **rules**: Algorithms that use rules, like One Rule.
- **trees**: Algorithms that use decision trees, like Random Forest.

The tab is called *Classify* and the algorithms are listed under an overarching group called *Classifiers*. Nevertheless, Weka supports both classification (predict a category) and regression (predict a numeric value) predictive modeling problems. The type of problem you are working with is defined by the variable you wish to predict. On the *Classify* tab this is selected below

the test options. By default, Weka selects the last attribute in your dataset. If the attribute is nominal, then Weka assumes you are working on a classification problem. If the attribute is numeric, Weka assumes you are working on a regression problem.

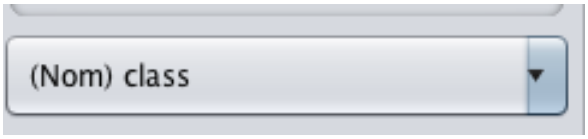


Figure 14.2: Weka Choose an Output Attribute to Predict.

This is important, because the type of problem that you are working on determines what algorithms that you can work with. For example, if you are working on a classification problem, you cannot use regression algorithms like Linear Regression. On the other hand, if you are working on a regression problem, you cannot use classification algorithms like Logistic Regression. Note if you are confused by the word *regression*, that is okay. It is confusing. Regression is a historical word from statistics. It used to mean making a model for a numerical output (to regress). It now means both the name of some algorithms and to predict a numerical value. Weka will gray-out algorithms that are not supported by your chosen problem. Many machine learning algorithms can be used for both classification and regression. So you will have access to a large suite of algorithms regardless of your chosen problem.

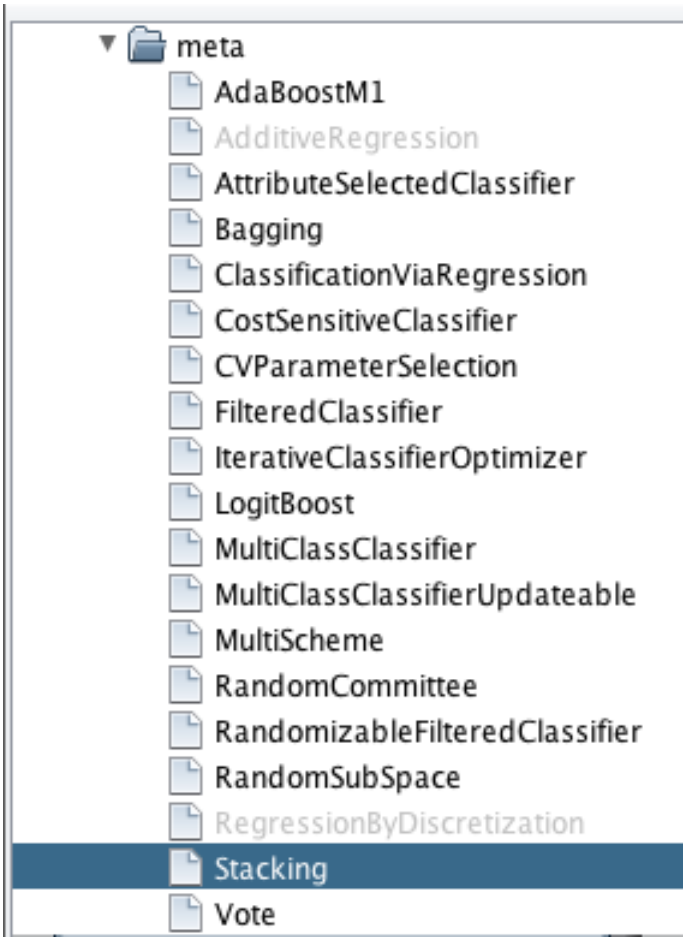


Figure 14.3: Weka Algorithms Unavailable For Some Problem Types.

14.2 Which Algorithm To Use

Generally, when working on a machine learning problem you cannot know which algorithm will be the best for your problem beforehand. If you had enough information to know which algorithm would achieve the best performance, you probably would not be doing applied machine learning. You would be doing something else like statistics. The solution therefore is to try a suite of algorithms on your problem and see what works best. Try a handful of powerful algorithms, then double down on the 1-to-3 algorithms that perform the best. They will given you an idea of the general type of algorithms that perform well or learning strategies that may be better than average at picking out the hidden structure in your data.

Some of the machine learning algorithms in Weka have non-standard names. You may already know the names of some machine learning algorithms, but feel confused by the names of the algorithms in Weka. Below is a list of 10 top machine learning algorithms you should consider trying on your problem, including both their standard name and the name used in Weka.

14.3 Linear Machine Learning Algorithms

Linear algorithms assume that the predicted attribute is a linear combination of the input attributes.

- Linear Regression: *function.LinearRegression*
- Logistic Regression: *function.Logistic*

14.4 Nonlinear Machine Learning Algorithms

Nonlinear algorithms do not make strong assumptions about the relationship between the input attributes and the output attribute being predicted.

- Naive Bayes: *bayes.NaiveBayes*
- Decision Tree (specifically the C4.5 variety): *trees.J48*
- k-Nearest Neighbors (also called KNN: *lazy.IBk*
- Support Vector Machines (also called SVM): *functions.SMO*
- Neural Network: *functions.MultilayerPerceptron*

14.5 Ensemble Machine Learning Algorithms

Ensemble methods combine the predictions from multiple models in order to make more robust predictions.

- Random Forest: *trees.RandomForest*
- Bootstrap Aggregation (also called Bagging): *meta.Bagging*
- Stacked Aggregation (also called Stacking or Blending): *meta.Stacking*

Weka has an extensive array of ensemble methods, perhaps one of the largest available across all of the popular machine learning frameworks. If you are looking for an area to specialize in using Weka, a source of true power in the platform besides ease of use, I would point to the support for ensemble techniques.

14.6 Machine Learning Algorithm Configuration

Once you have chosen a machine learning algorithm, you can configure it. Configuration is optional, but highly recommended. Weka cleverly chooses sensible defaults for each machine learning algorithm meaning that you can select an algorithm and start using it immediately without knowing much about it. To get the best results from an algorithm, you should configure it to behave ideally for your problem. How do you configure an algorithm for your problem?

Again, this is another open question and not knowable beforehand. Given algorithms do have heuristics that can guide you but they are not a silver bullet. The true answer is to systematically test a suite of standard configurations for a given algorithm on your problem. You can configure a machine learning algorithm in Weka by clicking on it's name after you have selected it. This will launch a window that displays all of the configuration details for the algorithm.

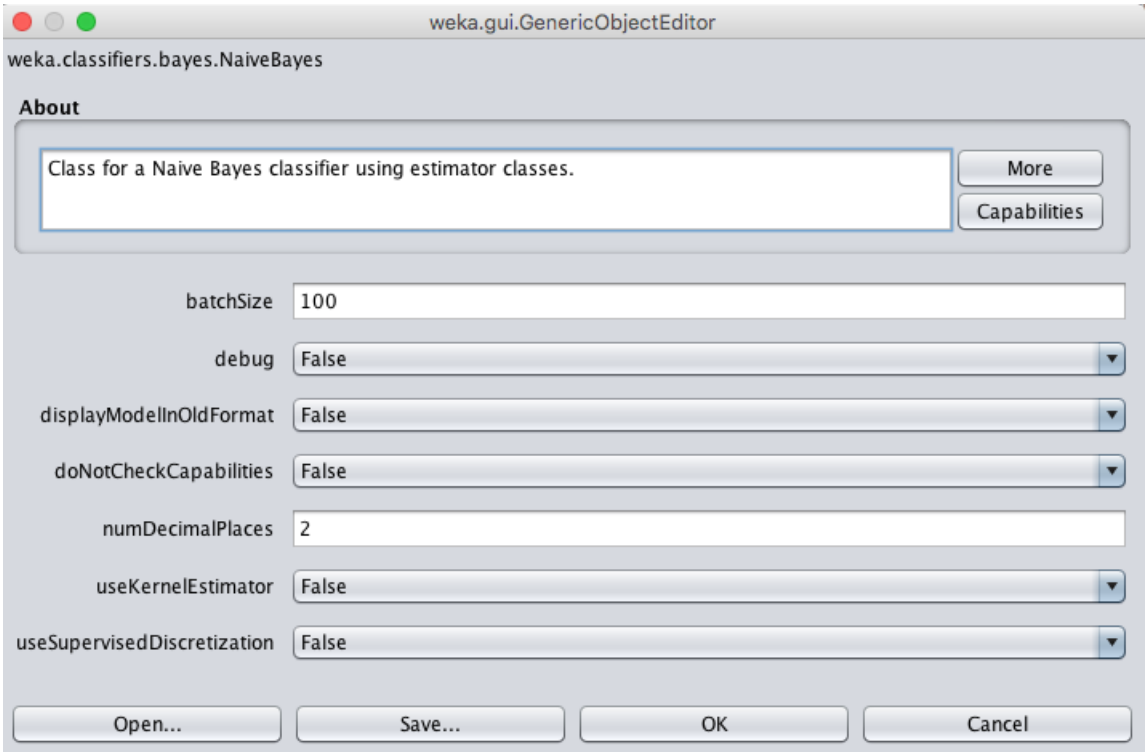


Figure 14.4: Weka Configure a Machine Learning Algorithms.

You can learn more about the meaning of each configuration option by hovering your mouse over each option which will display a tooltip describing the configuration option. Some options give you a limited set of values to choose from, other take integer or real valued numbers. Try both experimentation and research in order to come up with 3-to-5 standard configurations of an algorithm to try on your problem.

A pro-tip that you can use is to save your standard algorithm configurations to a file. Click the *Save* button at the bottom of the algorithm configuration. Enter a filename that clearly labels the algorithm name and the type of configuration you are saving. You can load an algorithm configuration later in the *Weka Explorer*, the *Weka Experiment Environment* and elsewhere in Weka. This is most valuable when you settle on a suite of standard algorithm configurations that you want reuse on problem to problem. You can adopt and use the configuration for the algorithm by clicking the *OK* button on the algorithm configuration window.

14.7 Get More Information on Algorithms

Weka provides more information about each support machine learning algorithm. On the algorithm configuration window, you will notice two buttons to learn more about the algorithm.

14.7.1 More Information

Clicking the *More* button will show a window that summarizes the implementation of the algorithm and all of the algorithms configuration properties.

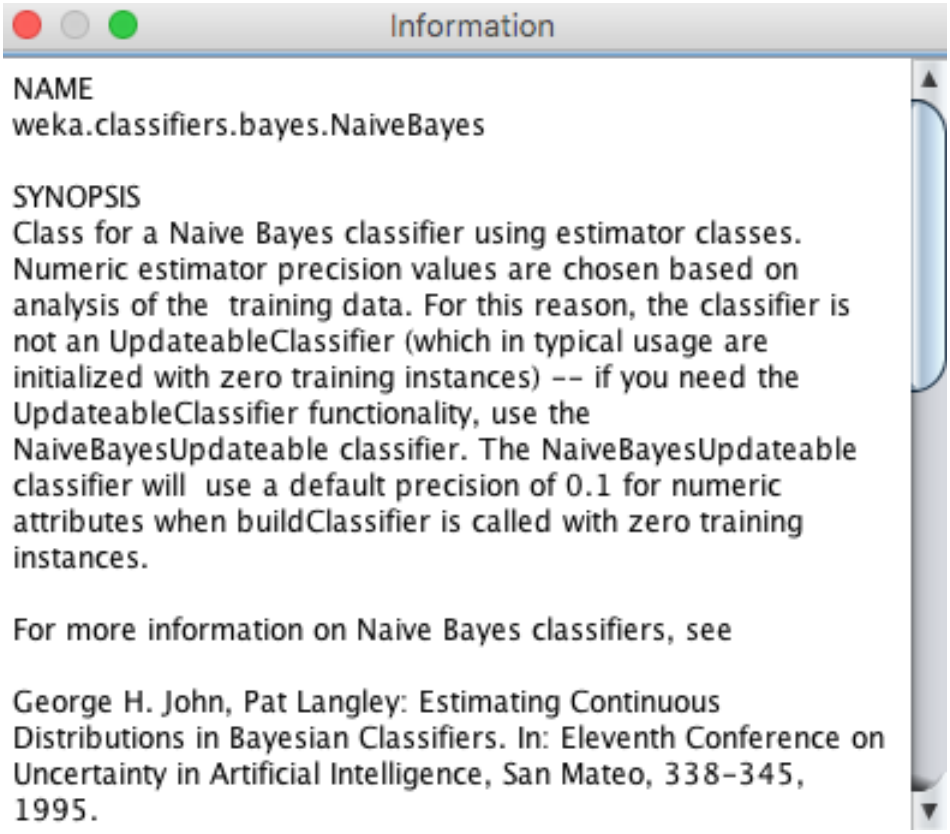


Figure 14.5: Weka More Information About an Algorithm.

This is useful to get a fuller idea of how the algorithm works and how to configure it. It also often includes references to books or papers from which the implementation of the algorithm was based. These can be good resources to track down and review in order to get a better idea for how to get the most from a given algorithm. Reading up on how to better configure an algorithm is not something to do as a beginner because it can feel a little overwhelming, but it is a pro tip that will help you learn more and faster later on when you have more experience with applied machine learning.

14.7.2 Algorithm Capabilities

Clicking on the *Capabilities* button will provide you with a snapshot of the capabilities of the algorithm.

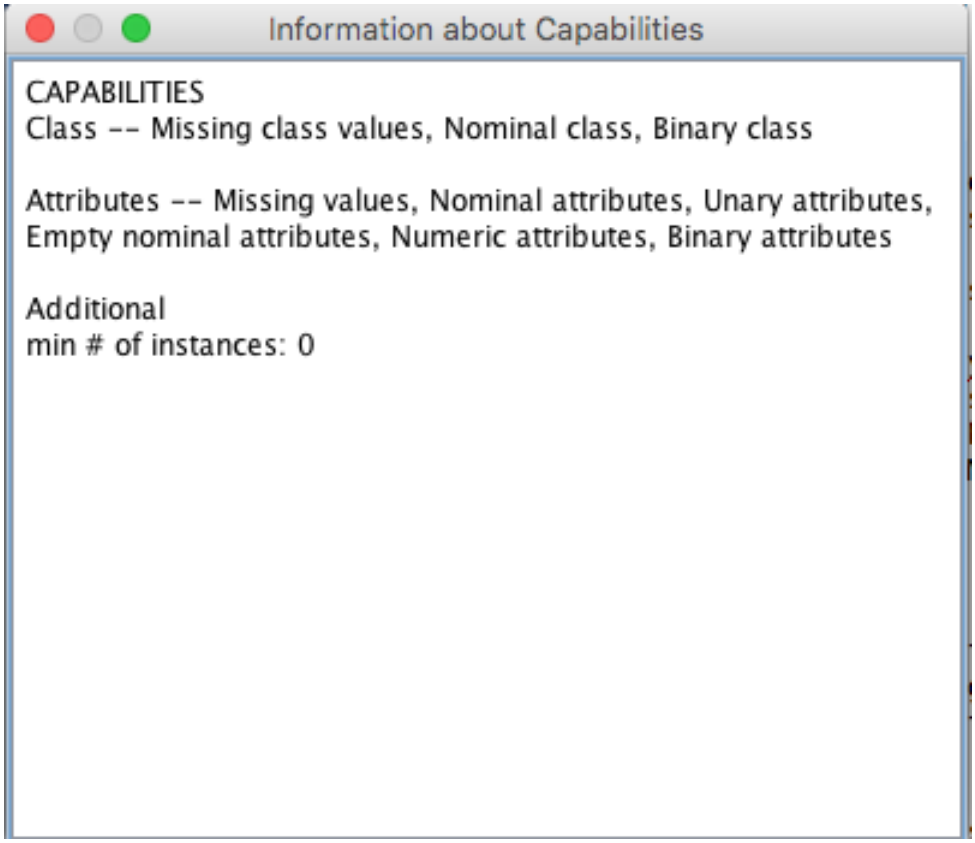


Figure 14.6: Weka Capabilities for an Algorithm.

Most importantly, this is useful to get an idea of how the algorithm handles missing data and any other important expectations it may have on your problem. Reviewing this information can give you ideas on how to create new and different views on your data in order to lift performance for one or more algorithms.

14.8 Summary

In this lesson you discovered the support for machine learning algorithms in the Weka machine learning workbench. Specifically, you learned:

- That Weka has a large selection of machine learning algorithms to choose from for classification and regression problems.
- That you can easily configure each machine learning algorithm and save and load a set of standard configurations.
- That you can dive deeper into the details of a given algorithm and even discover the source from which it was based in order to learn how to get the best performance.

14.8.1 Next

Once you can create models from your data with machine learning algorithms, you need to compare the performance of different models. In the next lesson you will learn about the various

different test options that you can use to compare models on your dataset.

Chapter 15

How To Estimate The Performance of Machine Learning Algorithms

The problem of predictive modeling is to create models that have good performance making predictions on new unseen data. Therefore it is important to use robust techniques to train and evaluate your models on your available training data. The more reliable the estimate of the performance on your model, the further you can push the performance and be confident it will translate to the operational use of your model. In this lesson you will discover the various different ways that you can estimate the performance of your machine learning models in Weka. After reading this lesson you will know:

- How to evaluate your model using the training dataset.
- How to evaluate your model using a random train and test split.
- How to evaluate your model using k-fold cross-validation.

Let's get started.

15.1 Model Evaluation Techniques

There are a number of model evaluation techniques that you can choose from, and the Weka machine learning workbench offers four of them, as follows:

15.1.1 Training Dataset

Prepare your model on the entire training dataset, then evaluate the model on the same dataset. This is generally problematic not least because a perfect algorithm could game this evaluation technique by simply memorizing (storing) all training patterns and achieve a perfect score, which would be misleading.

15.1.2 Supplied Test Set

Split your dataset manually using another program. Prepare your model on the entire training dataset and use the separate test set to evaluate the performance of the model. This is a good approach if you have a large dataset (many tens of thousands of instances).

15.1.3 Percentage Split

Randomly split your dataset into a training and a testing partitions each time you evaluate a model. This can give you a very quick estimate of performance and like using a supplied test set, is preferable only when you have a large dataset.

15.1.4 Cross-Validation

Split the dataset into k-partitions or folds. Train a model on all of the partitions except one that is held out as the test set, then repeat this process creating k-different models and give each fold a chance of being held out as the test set. Then calculate the average performance of all k-models. This is the gold standard for evaluating model performance, but has the cost of creating many more models.

You can see these techniques in the *Weka Explorer* on the *Classify* tab after you have loaded a dataset.

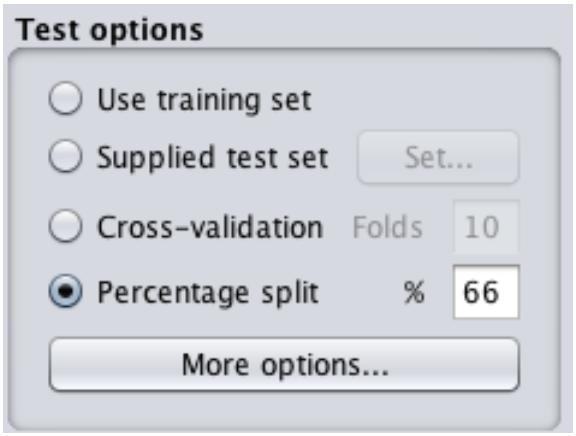


Figure 15.1: Weka Algorithm Evaluation Test Options.

15.2 Which Test Option to Use

Given that there are four different test options to choose from, which one should you use? Each test option has a time and place, summarized as follows:

- **Training Dataset**: Only to be used when you have all of the data and you are interested in creating a descriptive rather than a predictive model. Because you have all of the data, you do not need to make new predictions. You are interested in creating a model to better understand the problem.
- **Supplied Test Set**: When the data is very large, e.g. millions of records and you do not need all of it to train a model. Also useful when the test set has been defined by a third party.
- **Percentage Split**: Excellent to use to get a quick idea of the performance of a model. Not to be used to make decisions, unless you have a very large dataset and are confident (e.g. you have tested) that the splits sufficiently describe the problem. A common split value is 66% to 34% for train and test sets respectively.

- **Cross-Validation**: The default. To be used when you are unsure. Generally provides a more accurate estimate of the performance than the other techniques. Not to be used when you have a very large data. Common values for k are 5 and 10, depending on the size of the dataset.

If in doubt, use k-fold cross-validation where k is set to 10.

15.3 What About The Final Model

Test options are concerned with estimating the performance of a model on unseen data. This is an important concept to internalize. The goal of predictive modeling is to create a model that performs best in a situation that we do not fully understand, the future with new unknown data. We must use these powerful statistical techniques to best estimate the performance of the model in this situation.

That being said, once we have chosen a model, it must be finalized. None of these test options are used for this purpose. The model must be trained on the entire training dataset and saved. This topic of model finalization is beyond the scope of this lesson. Just note, that the performance of the finalized model does not need to be calculated, it is estimated using the test option techniques discussed above.

15.4 Performance Summary

The performance summary is provided in Weka when you evaluate a model. In the *Classify* tab after you have evaluated an algorithm by clicking the *Start* button, the results are presented in the *Classifier output* pane. This pane includes a lot of information, including:

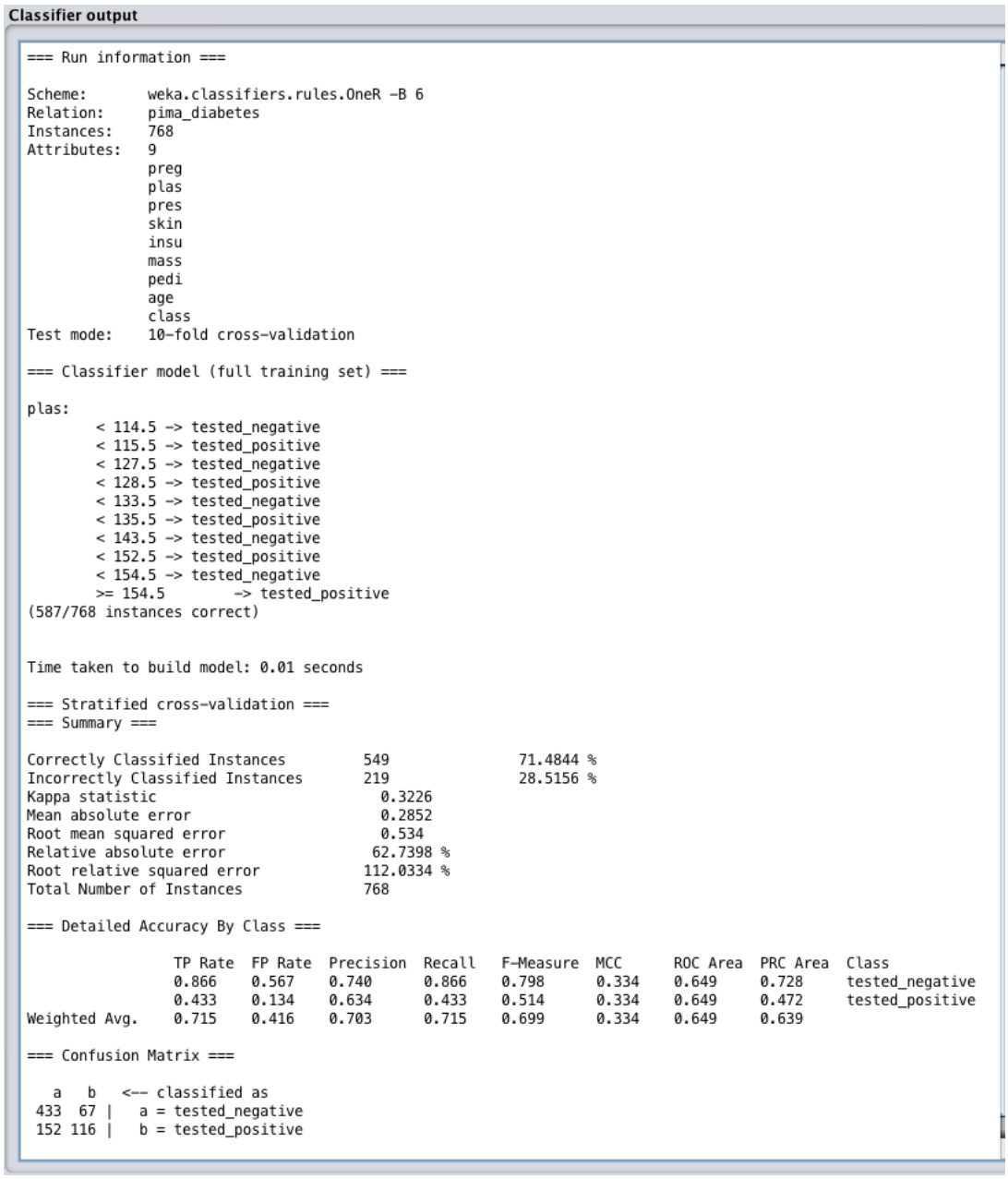
- The run information such as the algorithm and its configuration, the dataset and its properties as well as the test option
- The details of the constructed model, if any.
- The summary of the performance including a host of different measures.

15.4.1 Classification Performance Summary

When evaluating a machine learning algorithm on a classification problem, you are given a vast amount of performance information to digest. This is because classification may be the most studied type of predictive modeling problem and there are so many different ways to think about the performance of classification algorithms. There are three things to note in the performance summary for classification algorithms:

- **Classification accuracy**. This the ratio of the number of correct predictions out of all predictions made, often presented as a percentage where 100% is the best an algorithm can achieve. If your data has unbalanced classes, you may want to look into the Kappa metric which presents the same information taking the class balance into account.

- **Accuracy by class**. Take note of the true-positive and false-positive rates for the predictions for each class which can be instructive of the class breakdown for the problem is uneven or there are more than two classes. This can help you interpret the results if predicting one class is more important than predicting another.
- **Confusion matrix**. A table showing the number of predictions for each class compared to the number of instances that actually belong to each class. This is very useful to get an overview of the types of mistakes the algorithm made.



```
Classifier output

=== Run information ===

Scheme:       weka.classifiers.rules.OneR -B 6
Relation:     pima_diabetes
Instances:    768
Attributes:   9
              preg
              plas
              pres
              skin
              insu
              mass
              pedi
              age
              class
Test mode:    10-fold cross-validation

=== Classifier model (full training set) ===

plas:
        < 114.5 -> tested_negative
        < 115.5 -> tested_positive
        < 127.5 -> tested_negative
        < 128.5 -> tested_positive
        < 133.5 -> tested_negative
        < 135.5 -> tested_positive
        < 143.5 -> tested_negative
        < 152.5 -> tested_positive
        < 154.5 -> tested_negative
        >= 154.5        -> tested_positive
(587/768 instances correct)


Time taken to build model: 0.01 seconds

=== Stratified cross-validation ===
=== Summary ===

Correctly Classified Instances         549               71.4844 %
Incorrectly Classified Instances       219               28.5156 %
Kappa statistic                          0.3226
Mean absolute error                      0.2852
Root mean squared error                  0.534
Relative absolute error                 62.7398 %
Root relative squared error            112.0334 %
Total Number of Instances              768

=== Detailed Accuracy By Class ===

                 TP Rate  FP Rate  Precision  Recall   F-Measure  MCC      ROC Area  PRC Area  Class
                 0.866    0.567    0.740      0.866    0.798      0.334    0.649     0.728     tested_negative
                 0.433    0.134    0.634      0.433    0.514      0.334    0.649     0.472     tested_positive
Weighted Avg.    0.715    0.416    0.703      0.715    0.699      0.334    0.649     0.639

=== Confusion Matrix ===

   a   b   <-- classified as
 433  67 |   a = tested_negative
 152 116 |   b = tested_positive
```

Figure 15.2: Weka Classification Performance Summary.

15.4.2 Regression Performance Summary

When evaluating a machine learning algorithm on a regression problem, you are given a number of different performance measures to review. Of note the performance summary for regression algorithms are two things:

- **Correlation Coefficient**. This is how well the predictions are correlated or change with the actual output value. A value of 0 is the worst and a value of 1 is a perfectly correlated set of predictions.
- **Root Mean Squared Error**. This is the average amount of error made on the test set in the units of the output variable. This measure helps you get an idea on the amount a given prediction may be wrong on average.

```
=== Run information ===

Scheme:       weka.classifiers.rules.ZeroR
Relation:     housing
Instances:    506
Attributes:   14
              CRIM
              ZN
              INDUS
              CHAS
              NOX
              RM
              AGE
              DIS
              RAD
              TAX
              PTRATIO
              B
              LSTAT
              class
Test mode:    split 66.0% train, remainder test

=== Classifier model (full training set) ===

ZeroR predicts class value: 22.532806324110698

Time taken to build model: 0 seconds

=== Evaluation on test split ===

Time taken to test model on training split: 0 seconds

=== Summary ===

Correlation coefficient                  0
Mean absolute error                      6.4347
Root mean squared error                  8.9064
Relative absolute error                100      %
Root relative squared error            100      %
Total Number of Instances              172
```

Figure 15.3: Weka Regression Performance Summary.

15.5 Summary

In this lesson you discovered how to estimate the performance of your machine learning models on unseen data in Weka. Specifically, you learned:

- About the importance of estimating the performance of machine learning models on unseen data as the core problem in predictive modeling.
- About the 4 different test options and when to use each, paying specific attention to train and test splits and k-fold cross-validation.
- About the performance summary for classification and regression problems and to which metrics to pay attention.

15.5.1 Next

When comparing machine learning algorithms on a dataset, you need a common point of reference. In the next lesson you will learn how to create a baseline for performance on classification and regression problems.

Chapter 16

How To Estimate A Baseline Performance For Your Models

It is important to have a performance baseline on your machine learning problem. It will give you a point of reference to which you can compare all other models that you construct. In this lesson you will discover how to develop a baseline of performance for a machine learning problem using Weka. After reading this lesson you will know:

- The importance in establishing a baseline of performance for your machine learning problem.
- How to calculate a baseline performance using the Zero Rule method on a regression problem.
- How to calculate a baseline performance using the Zero Rule method on a classification problem.

Let's get started.

16.1 Importance of Baseline Results

You cannot know which algorithm will perform the best for your problem beforehand so you must try a suite of algorithms and see what works best, then double down on it. As such, it is critically important to develop a baseline of performance when working on a machine learning problem. A baseline provides a point of reference from which to compare other machine learning algorithms.

You can get an idea of both the absolute performance increases you can achieve over the baseline as well as lift ratios that show you relatively how much better you are doing. Without a baseline you do not know how well you are doing on your problem. You have no point of reference to consider whether or not you have or are continuing to add value. The baseline defines the hurdle that all other machine learning algorithms must cross to demonstrate skill on the problem.

16.2 Zero Rule For Baseline Performance

The baseline for both classification and regression problems is called the Zero Rule algorithm. Also called *ZeroR*. Let's take a closer look at how the Zero Rule algorithm can be used on classification and regression problems with some examples.

16.2.1 Baseline Performance For Regression Problems

For a regression predictive modeling problem where a numeric value is predicted, the Zero Rule algorithm predicts the mean of the training dataset. For example, let's demonstrate the Zero Rule algorithm on the Boston House Price prediction problem. You can download the ARFF for the Boston House Price prediction dataset. You can learn more about this dataset in Section 8.4.2.

1. Start the *Weka GUI Chooser*.
2. Click the *Explorer* button to open the *Weka Explorer* interface.
3. Load the `numeric/housing.arff` dataset.
4. Click the *Classify* tab to open the classification tab.
5. Select the *rules.ZeroR* algorithm (it should be selected by default).
6. Select the *Cross-validation* Test options (it should be selected by default).
7. Click the *Start* button to evaluate the algorithm on the dataset.

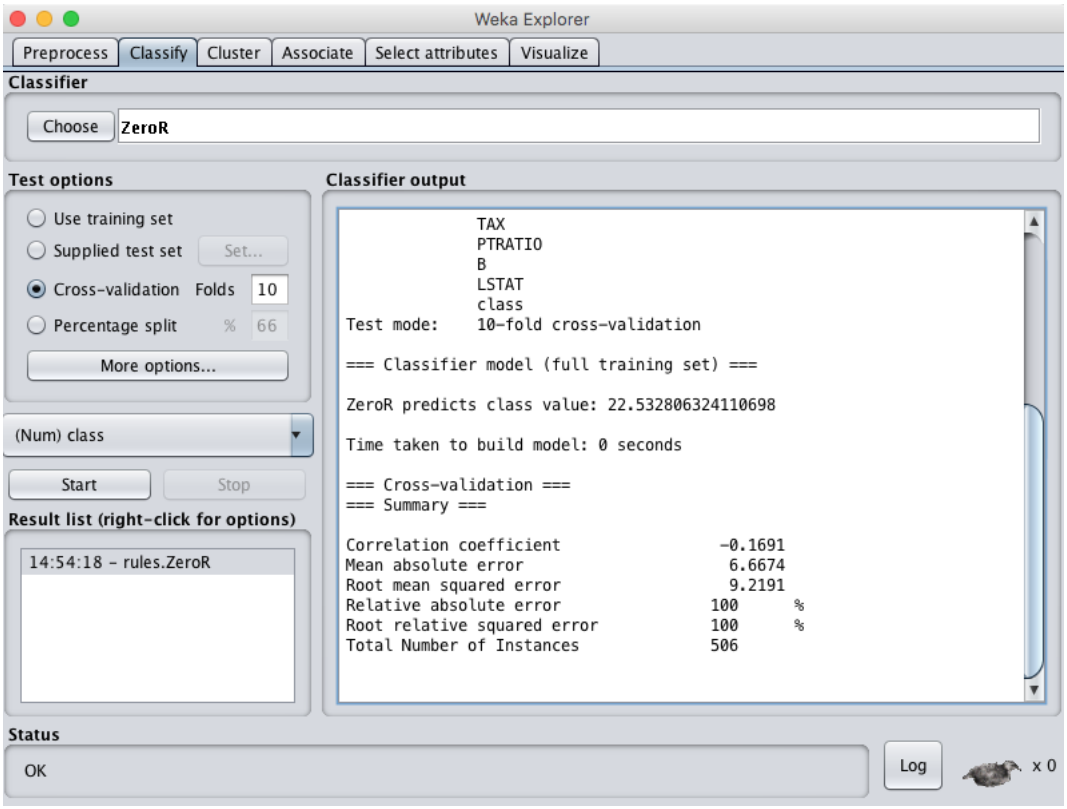


Figure 16.1: Weka Baseline Performance For a Regression Problem.

The *rules.ZeroR* algorithm predicts the mean Boston House price value of 22.5 (in thousands of dollars) and achieves a RMSE of 9.21. For any machine learning algorithm to demonstrate that it has skill on this problem, it must achieve an RMSE better than this value.

16.3 Baseline Performance for Classification Problems

For a classification predictive modeling problem where a categorical value is predicted, the Zero Rule algorithm predicts the class value that has the most observations in the training dataset. For example, let's demonstrate the Zero Rule algorithm on the Pima Indians onset of diabetes problem. You can learn more about this dataset in Section 8.2.1.

1. Start the *Weka GUI Chooser*.
2. Click the *Explorer* button to open the *Weka Explorer* interface.
3. Load the `data/diabetes.arff` dataset.
4. Click the *Classify* tab to open the classification tab.
5. Select the *rules.ZeroR* algorithm (it should be selected by default).
6. Select the *Cross-validation* Test options (it should be selected by default).
7. Click the *Start* button to evaluate the algorithm on the dataset.

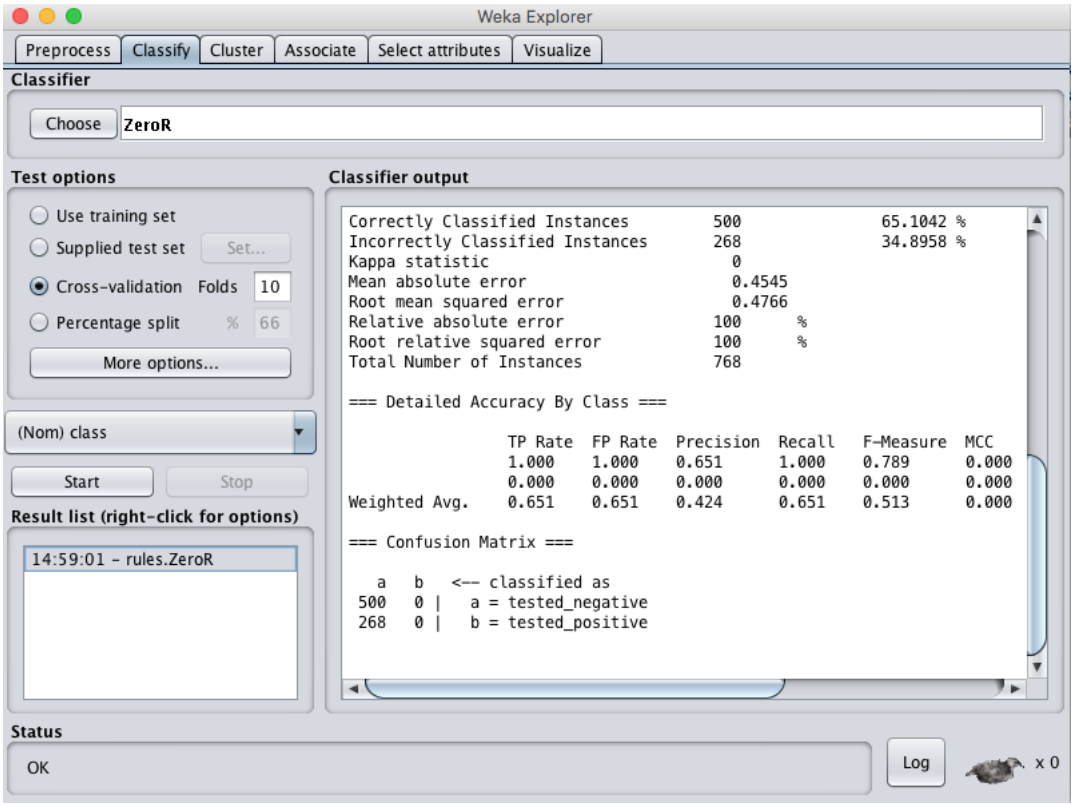


Figure 16.2: Weka Baseline Performance For a Classification Problem.

The *rules.ZeroR* algorithm predicts the *tested_negative* value for all instances as it is the majority class, and achieves an accuracy of 65.1%. For any machine learning algorithm to demonstrate that it has skill on this problem, it must achieve an accuracy better than this value.

16.4 Summary

In this lesson you have discovered how to calculate a baseline performance for your machine learning problems using Weka. Specifically, you learned:

- The importance of calculating a baseline of performance on your problem.
- How to calculate a baseline performance for a regression problem using the Zero Rule algorithm.
- How to calculate a baseline performance for a classification problem using the Zero Rule algorithm.

16.4.1 Next

Classification may be the most studied form of predictive modeling and Weka has more classification algorithms than any other type of algorithm. In the next lesson you will learn about 5 top classification algorithms that you can use on your problems.

Chapter 17

How To Use Top Classification Machine Learning Algorithms

Weka makes a large number of classification algorithms available. This is one of the key benefits of using the Weka platform to work through your machine learning problems. In this lesson you will discover how to use 5 top machine learning algorithms in Weka. After reading this lesson you will know:

- About 5 top machine learning algorithms that you can use on your classification problems.
- How to use 5 top classification algorithms in Weka.
- The key configuration parameters for 5 top classification algorithms.

Let's get started.

17.1 Classification Algorithm Tour Overview

We are going to take a tour of 5 top classification algorithms in Weka. Each algorithm that we cover will be briefly described in terms of how it works, key algorithm parameters will be highlighted and the algorithm will be demonstrated in the *Weka Explorer* interface. The 5 algorithms that we will review are:

1. Logistic Regression.
2. Naive Bayes.
3. Decision Tree.
4. k-Nearest Neighbors.
5. Support Vector Machines.

These are 5 algorithms that you can try on your classification problem as a starting point. A standard machine learning classification problem will be used to demonstrate each algorithm. Specifically, the Ionosphere binary classification problem. You can learn more about this dataset in Section 8.2.3.

1. Open the *Weka GUI Chooser*.
2. Click the *Explorer* button to open the *Weka Explorer*.
3. Load the `data/ionosphere.arff` dataset.
4. Click *Classify* to open the *Classify* tab.

17.2 Logistic Regression

Logistic regression is a binary classification algorithm. It assumes the input variables are numeric and have a Gaussian (bell curve) distribution. This last point does not have to be true, as logistic regression can still achieve good results if your data is not Gaussian. In the case of the Ionosphere dataset, some input attributes have a Gaussian-like distribution, but many do not.

The algorithm learns a coefficient for each input value, which are linearly combined into a regression function and transformed using a logistic (s-shaped) function. Logistic regression is a fast and simple technique, but can be very effective on some problems. The logistic regression only supports binary classification problems, although the Weka implementation has been adapted to support multiclass classification problems. Choose the logistic regression algorithm:

1. Click the *Choose* button and select *Logistic* under the *functions* group.
2. Click on the name of the algorithm to review the algorithm configuration.

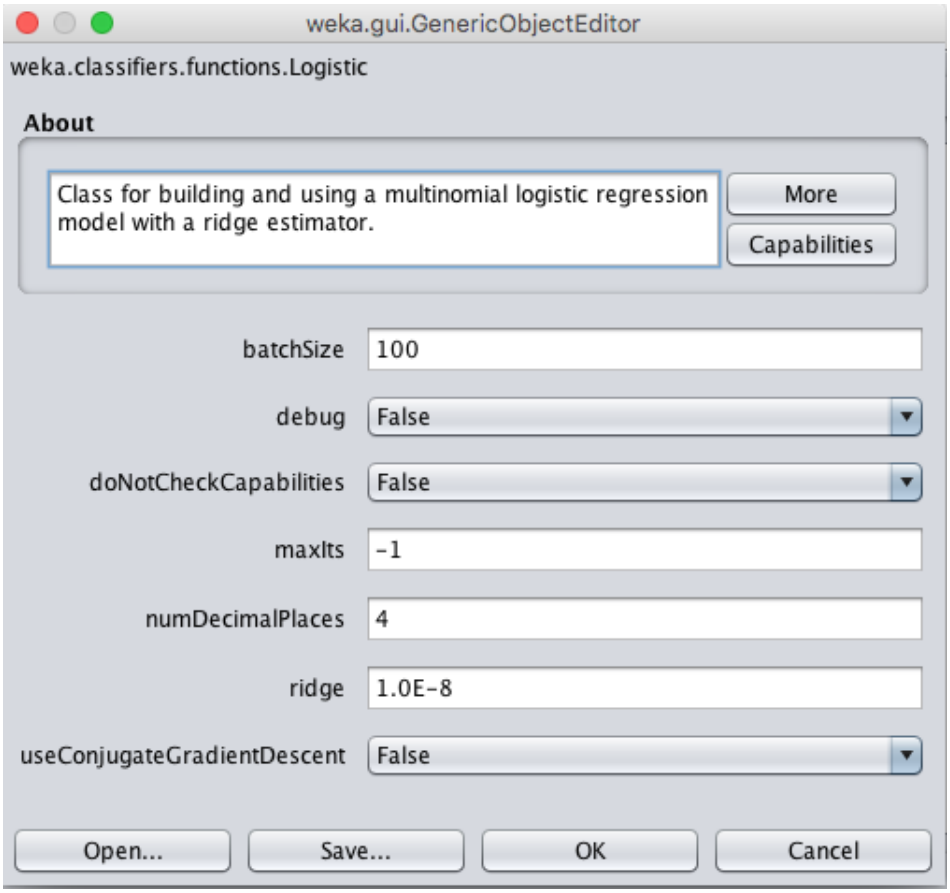


Figure 17.1: Weka Configuration for the Logistic Regression Algorithm.

The algorithm can run for a fixed number of iterations (*maxIts*), but by default will run until it is estimated that the algorithm has converged. The implementation uses a ridge estimator which is a type of regularization. This method seeks to simplify the model during training by minimizing the coefficients learned by the model. The *ridge* parameter defines how much pressure to put on the algorithm to reduce the size of the coefficients. Setting this to 0 will turn off this regularization.

1. Click *OK* to close the algorithm configuration.

2. Click the *Start* button to run the algorithm on the Ionosphere dataset.

You can see that with the default configuration that logistic regression achieves an accuracy of 88%.

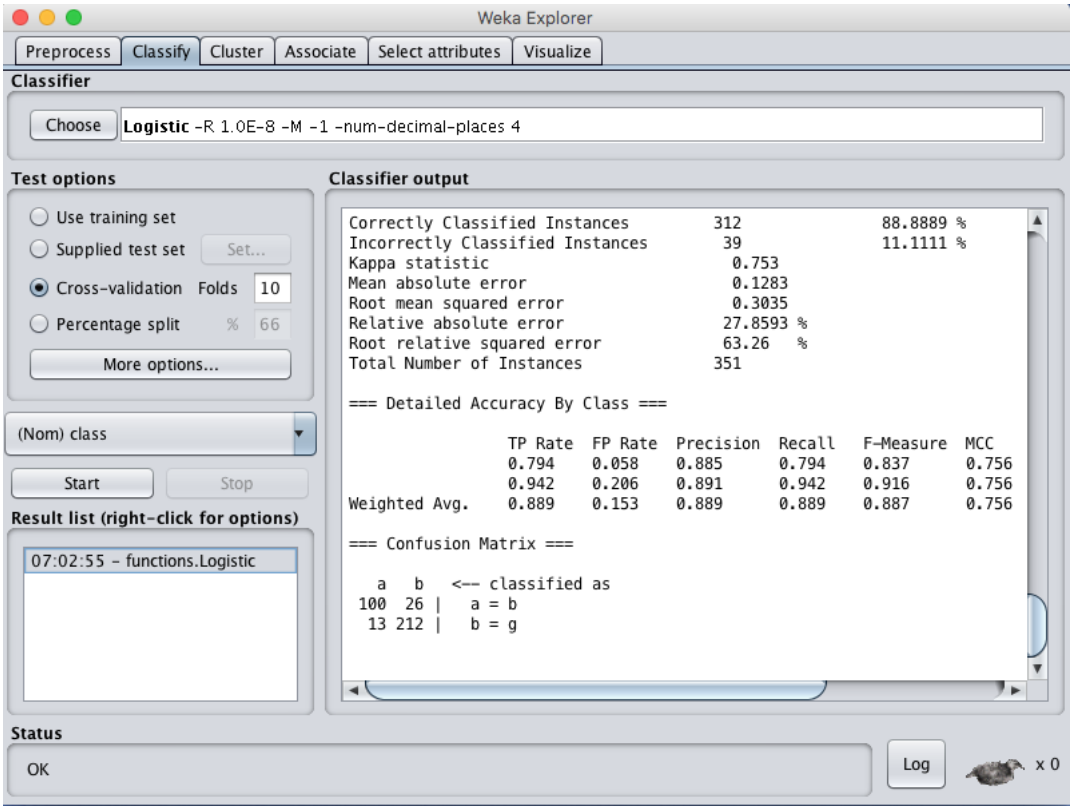


Figure 17.2: Weka Configuration for the Logistic Regression Algorithm.

17.3 Naive Bayes

Naive Bayes is a classification algorithm. Traditionally it assumes that the input values are nominal, although numerical inputs are supported by assuming a distribution. Naive Bayes uses a simple implementation of Bayes Theorem (hence naive) where the prior probability for each class is calculated from the training data and assumed to be independent of each other (technically called conditionally independent).

This is an unrealistic assumption because we expect the variables to interact and be dependent, although this assumption makes the probabilities fast and easy to calculate. Even under this unrealistic assumption, Naive Bayes has been shown to be a very effective classification algorithm. Naive Bayes calculates the posterior probability for each class and makes a prediction for the class with the highest probability. As such, it supports both binary classification and multiclass classification problems. Choose the Naive Bayes algorithm:

1. Click the *Choose* button and select *NaiveBayes* under the *bayes* group.

2. Click on the name of the algorithm to review the algorithm configuration.

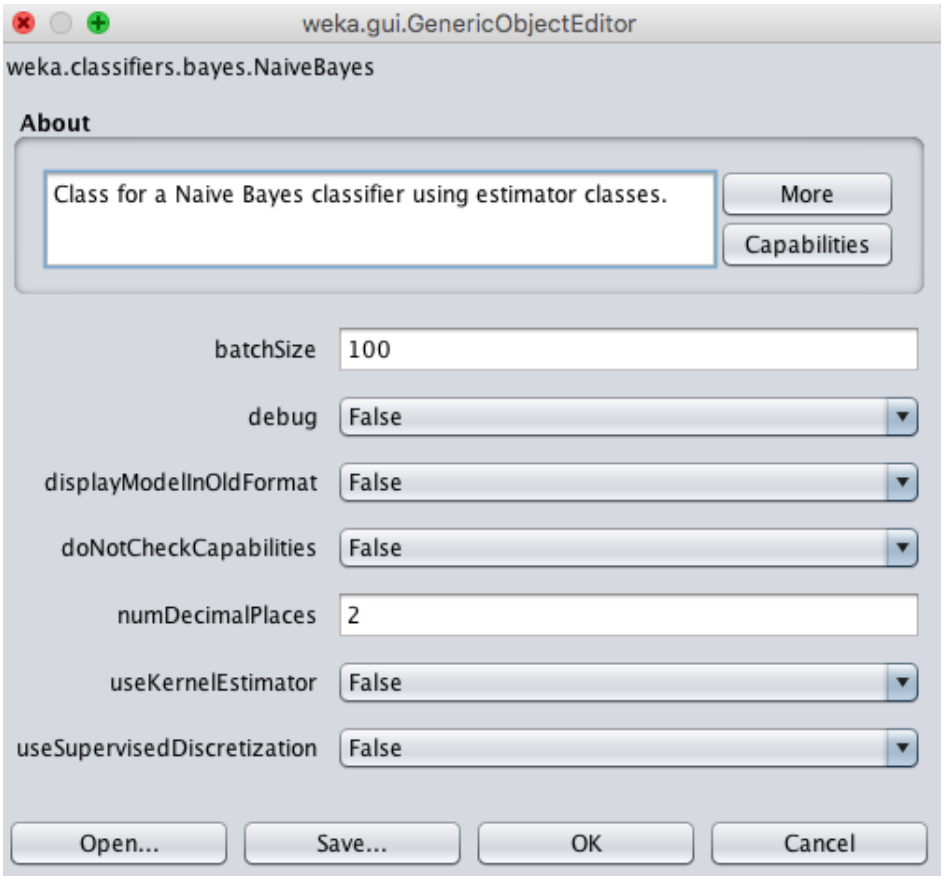


Figure 17.3: Weka Configuration for the Naive Bayes Algorithm.

By default a Gaussian distribution is assumed for each numerical attributes. You can change the algorithm to use a kernel estimator with the *useKernelEstimator* argument that may better match the actual distribution of the attributes in your dataset. Alternately, you can automatically convert numerical attributes to nominal attributes with the *useSupervisedDiscretization* parameter.

1. Click *OK* to close the algorithm configuration.

2. Click the *Start* button to run the algorithm on the Ionosphere dataset.

You can see that with the default configuration that Naive Bayes achieves an accuracy of 82%.

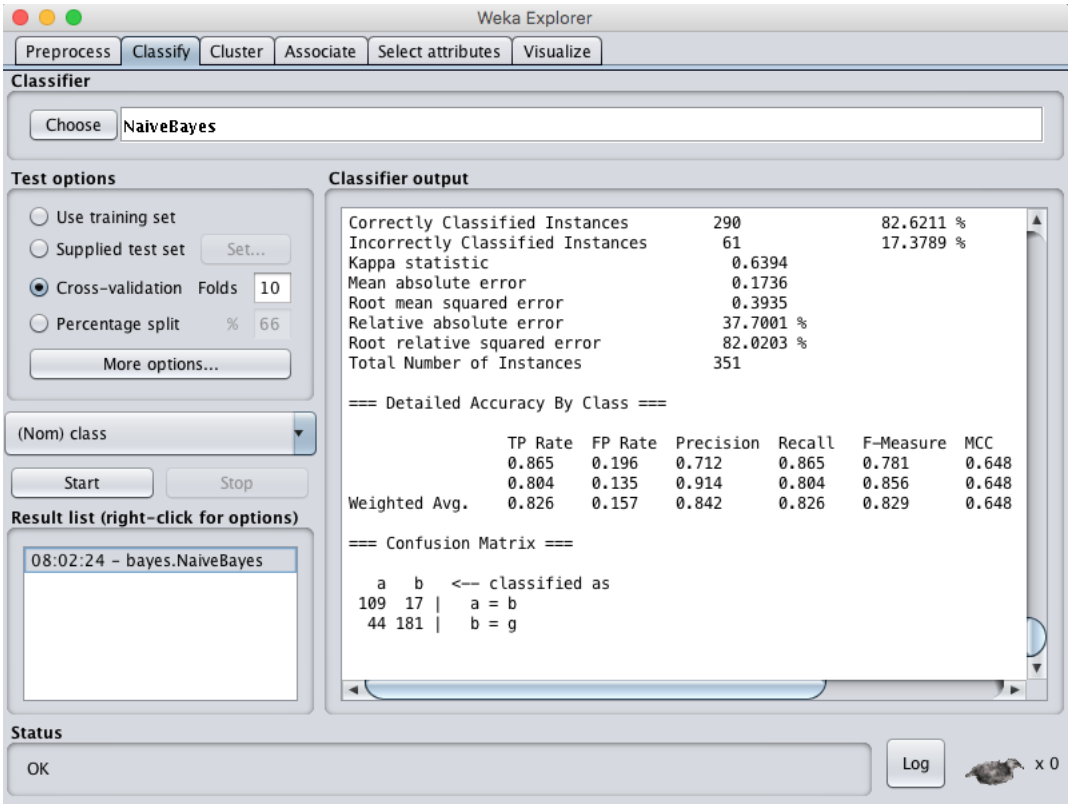


Figure 17.4: Weka Classification Results for the Naive Bayes Algorithm.

There are a number of other flavors of Naive Bayes algorithms that you could work with.

17.4 Decision Tree

Decision trees can support classification and regression problems. Decision trees are more recently referred to as Classification And Regression Trees (CART). They work by creating a tree to evaluate an instance of data. The tree is inverted, starting at the top or root of the tree and moving down to the leaves until a prediction can be made. The process of creating a decision tree works by greedily selecting the best split point in order to make predictions and repeating the process until the tree is a fixed depth. After the tree is constructed, it is pruned in order to improve the model's ability to generalize to new data. Choose the decision tree algorithm:

1. Click the *Choose* button and select *REPTree* under the *trees* group.

2. Click on the name of the algorithm to review the algorithm configuration.

Figure 17.5: Weka Configuration for the Decision Tree Algorithm.

The depth of the tree is defined automatically, but a depth can be specified in the *maxDepth* parameter. You can also choose to turn of pruning by setting the *noPruning* parameter to True, although this may result in worse performance. The *minNum* parameter defines the minimum number of instances supported by the tree in a leaf node when constructing the tree from the training data.

1. Click *OK* to close the algorithm configuration.

2. Click the *Start* button to run the algorithm on the Ionosphere dataset.

You can see that with the default configuration that the decision tree algorithm achieves an accuracy of 89%.

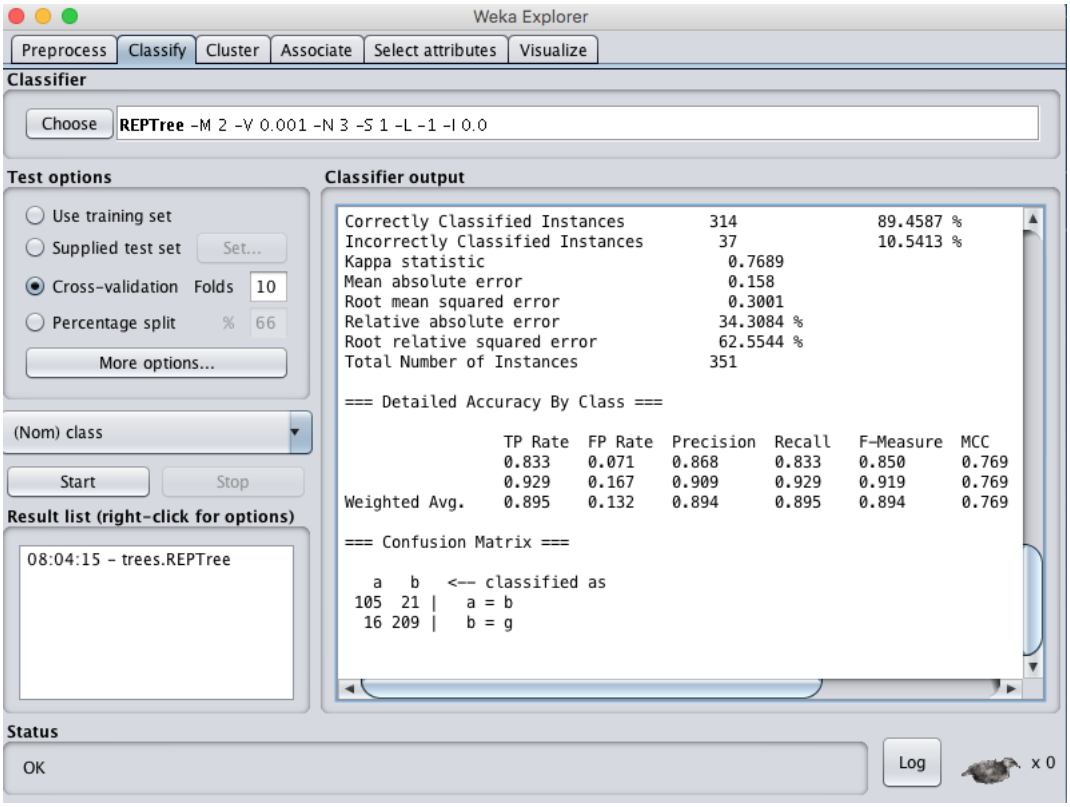


Figure 17.6: Weka Classification Results for the Decision Tree Algorithm.

Another more advanced decision tree algorithm that you can use is the C4.5 algorithm, called *trees.J48* in Weka. You can review a visualization of a decision tree prepared on the entire training data set by right clicking on the *Result list* and clicking *Visualize Tree*.

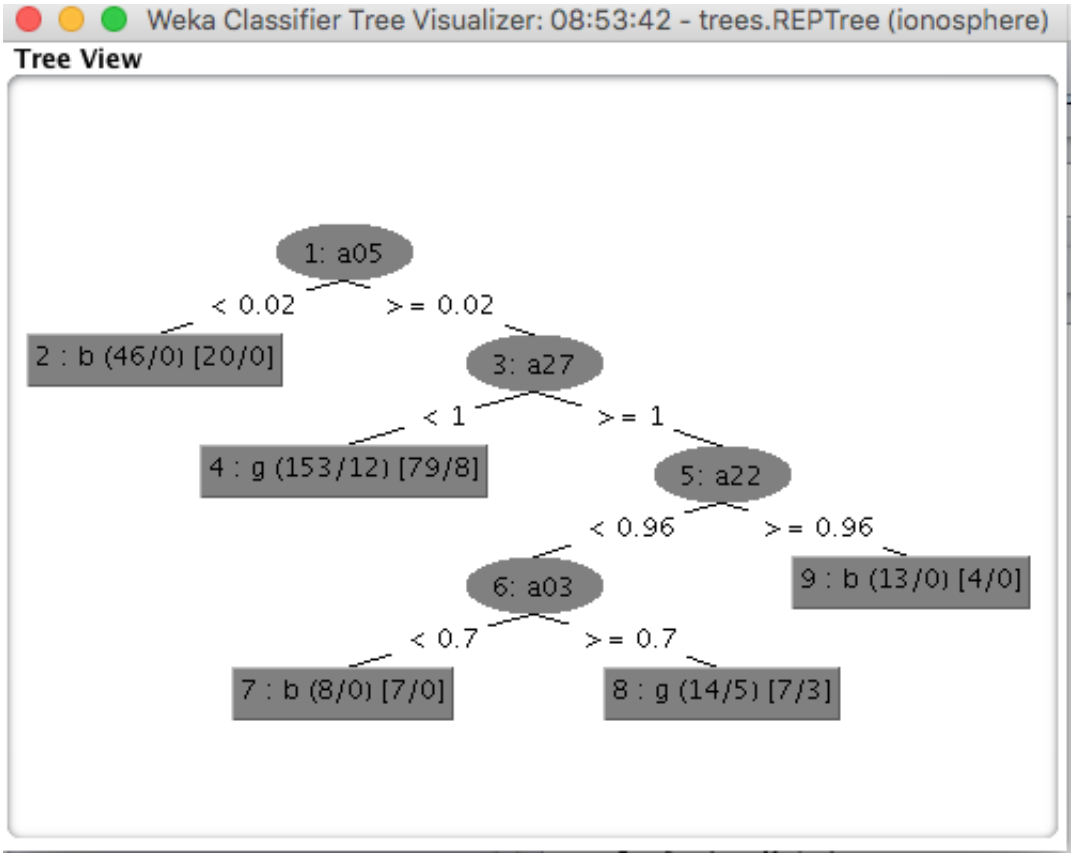


Figure 17.7: Weka Visualization of a Decision Tree.

17.5 k-Nearest Neighbors

The *k*-nearest neighbors algorithm supports both classification and regression. It is also called KNN for short. It works by storing the entire training dataset and querying it to locate the *k* most similar training patterns when making a prediction. As such, there is no model other than the raw training dataset and the only computation performed is the querying of the training dataset when a prediction is requested.

It is a simple algorithm, but one that does not assume very much about the problem other than that the distance between data instances is meaningful in making predictions. As such, it often achieves very good performance. When making predictions on classification problems, KNN will take the mode (most common class) of the *k* most similar instances in the training dataset. Choose the *k*-Nearest Neighbors algorithm:

1. Click the *Choose* button and select *IBk* under the *lazy* group.
2. Click on the name of the algorithm to review the algorithm configuration.

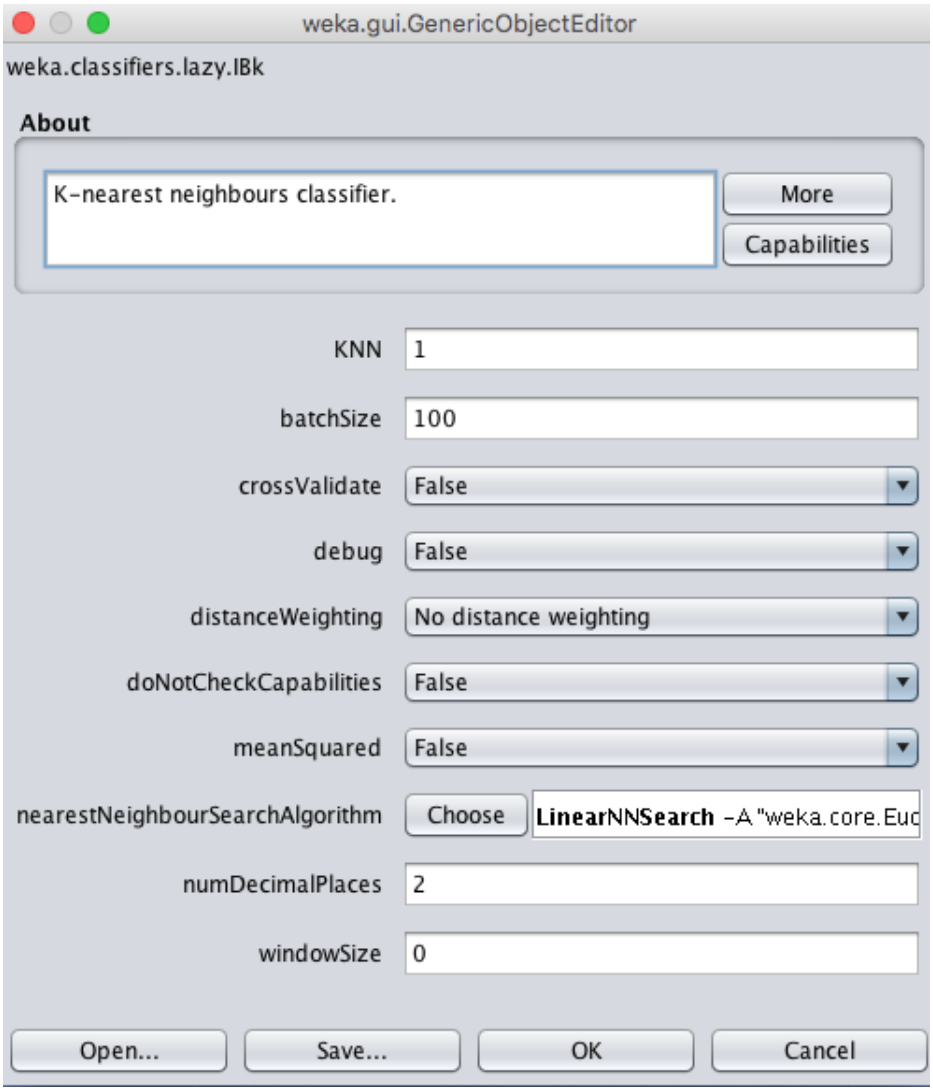


Figure 17.8: Weka Configuration for the k-Nearest Neighbors Algorithm.

The size of the neighborhood is controlled by the k parameter, called *IBk* in Weka. For example, if k is set to 1, then predictions are made using the single most similar training instance to a given new pattern for which a prediction is requested. Common values for k are 3, 7, 11 and 21, larger for larger dataset sizes. Weka can automatically discover a good value for k using cross-validation inside the algorithm by setting the crossValidate parameter to True. Another important parameter is the distance measure used. This is configured in the *nearestNeighbourSearchAlgorithm* which controls the way in which the training data is stored and searched. The default is a *LinearNNSearch*. Clicking the name of this search algorithm will provide another configuration window where you can choose a *distanceFunction* parameter. By default, *Euclidean* distance is used to calculate the distance between instances, which is good for numerical data with the same scale. *Manhattan* distance is good to use if your attributes differ in measures or type.

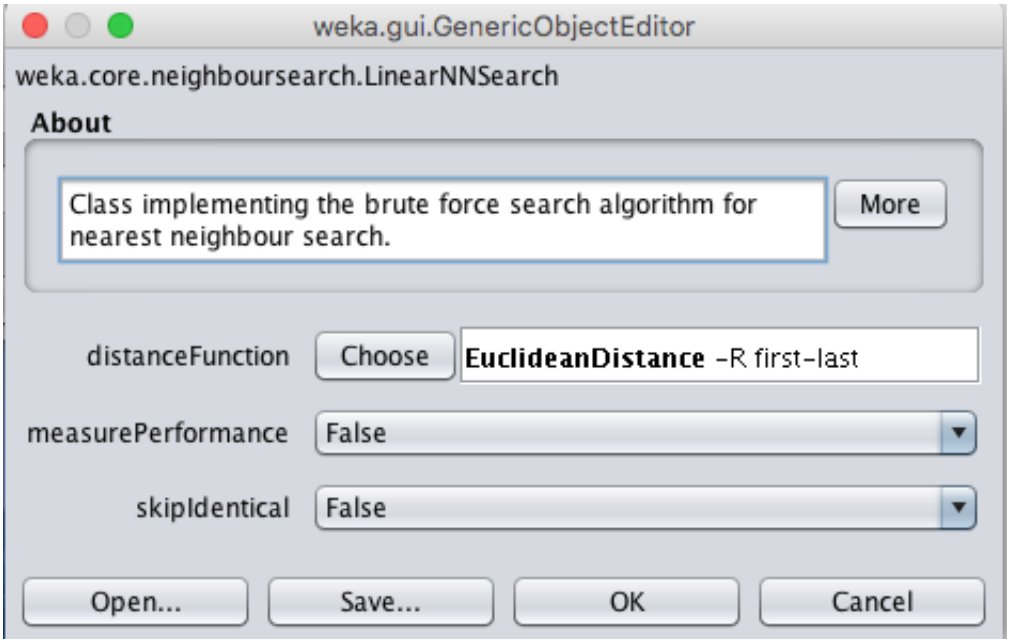


Figure 17.9: Weka Configuration for the Search Algorithm in the k-Nearest Neighbors Algorithm.

It is a good idea to try a suite of different k values and distance measures on your problem and see what works best.

1. Click *OK* to close the algorithm configuration.

2. Click the *Start* button to run the algorithm on the Ionosphere dataset.

You can see that with the default configuration that the KNN algorithm achieves an accuracy of 86%.

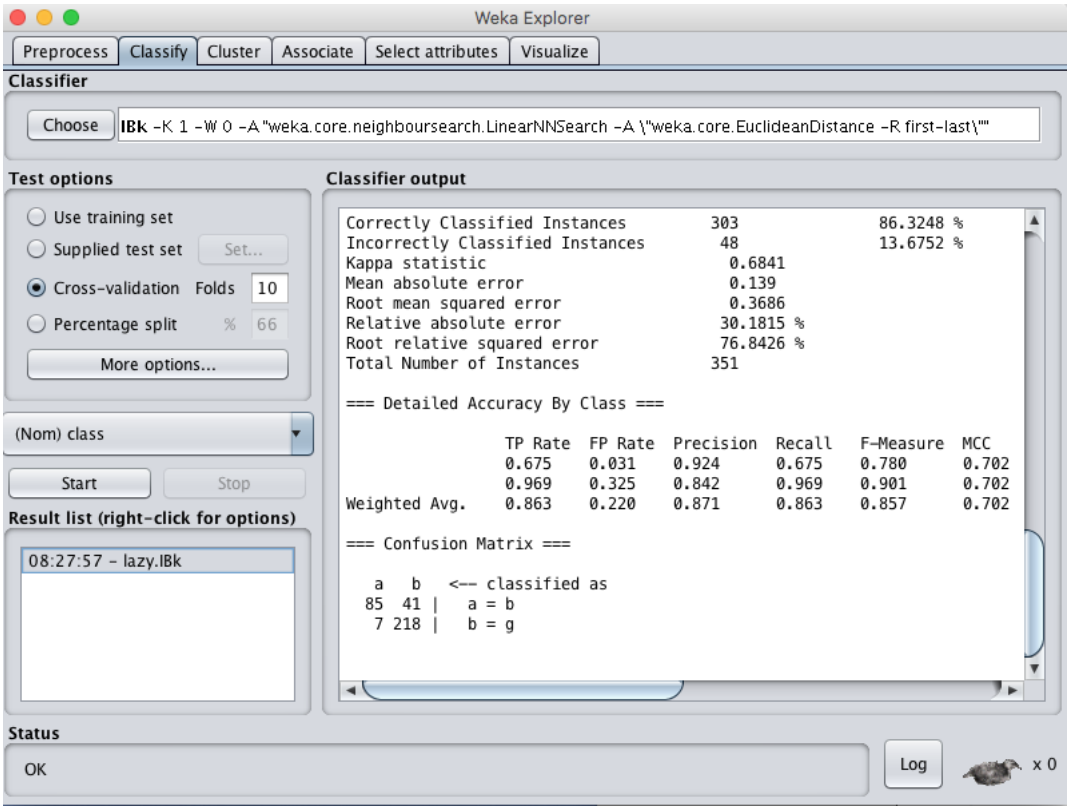


Figure 17.10: Weka Classification Results for k-Nearest Neighbors.

17.6 Support Vector Machines

Support Vector Machines were developed for binary classification problems, although extensions to the technique have been made to support multiclass classification and regression problems. The algorithm is often referred to as SVM for short. SVM was developed for numerical input variables, although will automatically convert nominal values to numerical values. Input data is also normalized before being used. SVM work by finding a line that best separates the data into the two groups. This is done using an optimization process that only considers those data instances in the training dataset that are closest to the line that best separates the classes. The instances are called support vectors, hence the name of the technique.

In almost all problems of interest, a line cannot be drawn to neatly separate the classes, therefore a margin is added around the line to relax the constraint, allowing some instances to be misclassified but allowing a better result overall. Finally, few datasets can be separated with just a straight line. Sometimes a line with curves or even polygonal regions need to be marked out. This is achieved with SVM by projecting the data into a higher dimensional space in order to draw the lines and make predictions. Different kernels can be used to control the projection and the amount of flexibility in separating the classes. Choose the SVM algorithm:

1. Click the *Choose* button and select *SMO* under the *function* group.

2. Click on the name of the algorithm to review the algorithm configuration.

SMO refers to the specific efficient optimization algorithm used inside the SVM implementation, which stands for Sequential Minimal Optimization.

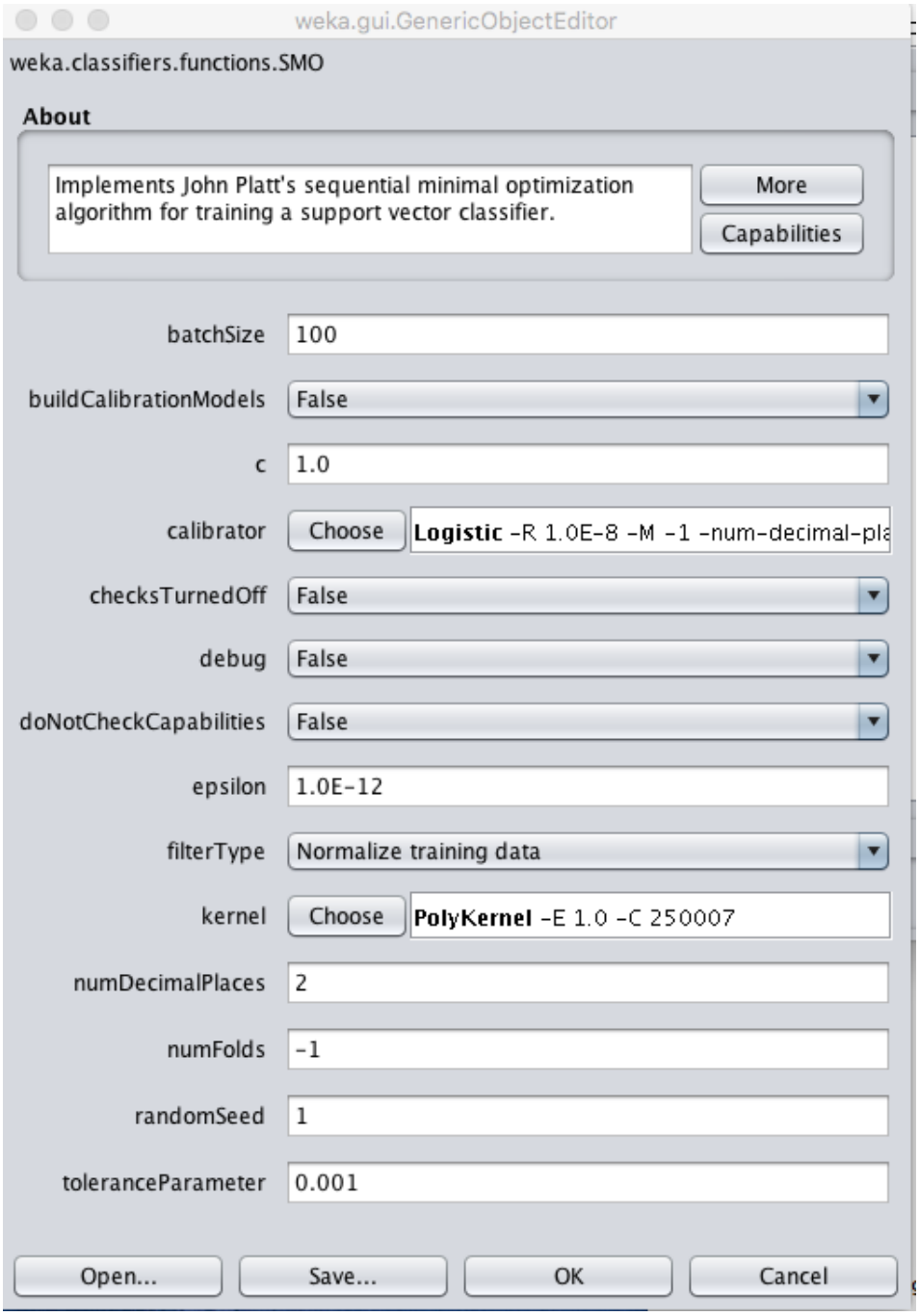


Figure 17.11: Weka Configuration for the Support Vector Machines Algorithm.

The *C* parameter, called the complexity parameter in Weka controls how flexible the process for drawing the line to separate the classes can be. A value of 0 allows no violations of the margin, whereas the default is 1. A key parameter in SVM is the type of Kernel to use. The simplest kernel is a *Linear* kernel that separates data with a straight line or hyperplane. The default in Weka is a *Polynomial* kernel that will separate the classes using a curved or wiggly line, the higher the polynomial, the more wiggly (the exponent value).

A popular and powerful kernel is the *RBF* kernel or *RadialBasisFunction* kernel that is capable of learning closed polygons and complex shapes to separate the classes. It is a good

idea to try a suite of different kernels and C (complexity) values on your problem and see what works best.

1. Click *OK* to close the algorithm configuration.
2. Click the *Start* button to run the algorithm on the Ionosphere dataset.

You can see that with the default configuration that the SVM algorithm achieves an accuracy of 88%.

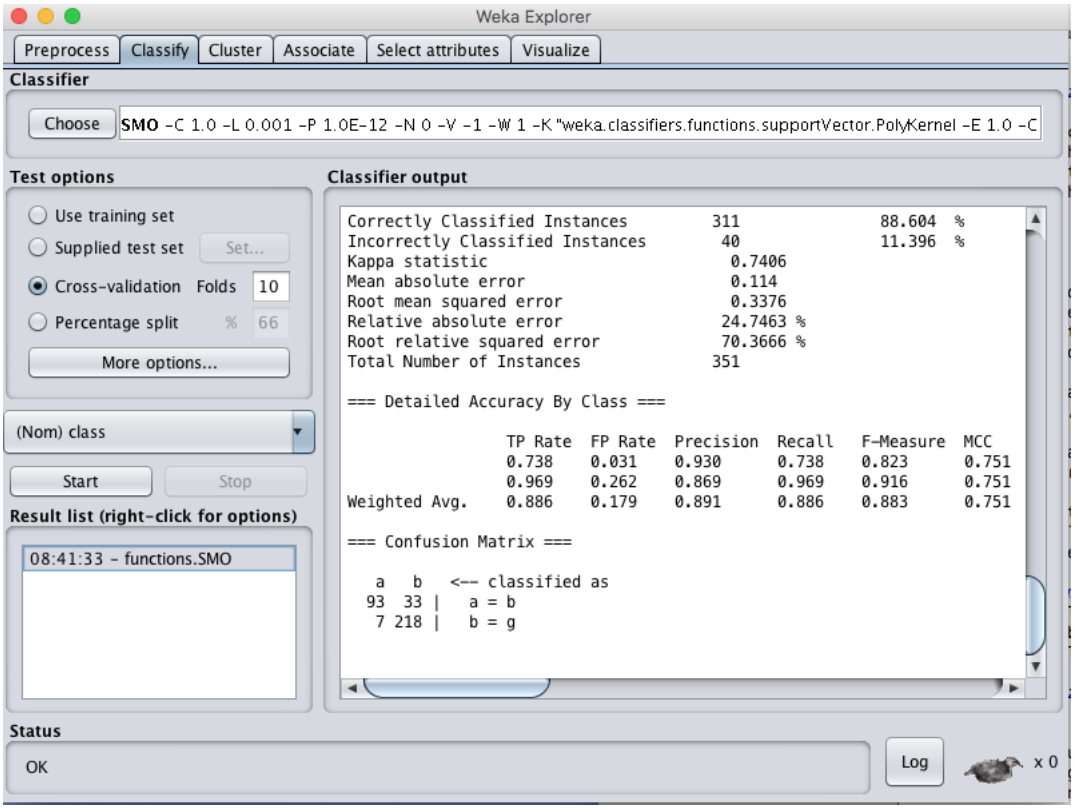


Figure 17.12: Weka Classification Results for the Support Vector Machine Algorithm.

17.7 Summary

In this lesson you discovered how to use top classification machine learning algorithms in Weka. Specifically, you learned:

- 5 top classification algorithms you can try on your own problems.
- The key configuration parameters to tune for each algorithm.
- How to use each algorithm in Weka.

17.7.1 Next

There are a lot of machine learning algorithms to choose from. In the next lesson you will learn about 5 top algorithms that you can use on regression problems.

Chapter 18

How To Use Top Regression Machine Learning Algorithms

Weka has a large number of regression algorithms available on the platform. The large number of machine learning algorithms supported by Weka is one of the biggest benefits of using Weka for applied machine learning. In this lesson you will discover how to use top regression machine learning algorithms in Weka. After reading this lesson you will know:

- About 5 top regression algorithms supported by Weka.
- How to use regression machine learning algorithms for predictive modeling in Weka.
- About the key configuration options of regression algorithms in Weka.

Let's get started.

18.1 Regression Algorithms Overview

We are going to take a tour of 5 top regression algorithms in Weka. Each algorithm that we cover will be briefly described in terms of how it works, key algorithm parameters will be highlighted and the algorithm will be demonstrated in the *Weka Explorer* interface. The 5 algorithms that we will review are:

1. Linear Regression.
2. k-Nearest Neighbors.
3. Decision Tree.
4. Support Vector Machines.
5. Multilayer Perceptron.

These are 5 algorithms that you can try on your regression problem as a starting point. A standard machine learning regression problem will be used to demonstrate each algorithm. Specifically, the Boston House Price Dataset. You can learn more about this dataset in Section 8.4.2.

1. Open the *Weka GUI Chooser*.
2. Click the *Explorer* button to open the *Weka Explorer*.
3. Load the `numeric/housing.arff` dataset.
4. Click *Classify* to open the *Classify* tab.

Let's start things off by looking at the linear regression algorithm.

18.2 Linear Regression

Linear regression only supports regression type problems. It works by estimating coefficients for a line or hyperplane that best fits the training data. It is a very simple regression algorithm, fast to train and can have great performance if the output variable for your data is a linear combination of your inputs. It is good idea to evaluate linear regression on your problem before moving onto more complex algorithms in case it performs well. Choose the linear regression algorithm:

1. Click the *Choose* button and select *LinearRegression* under the *functions* group.
2. Click on the name of the algorithm to review the algorithm configuration.

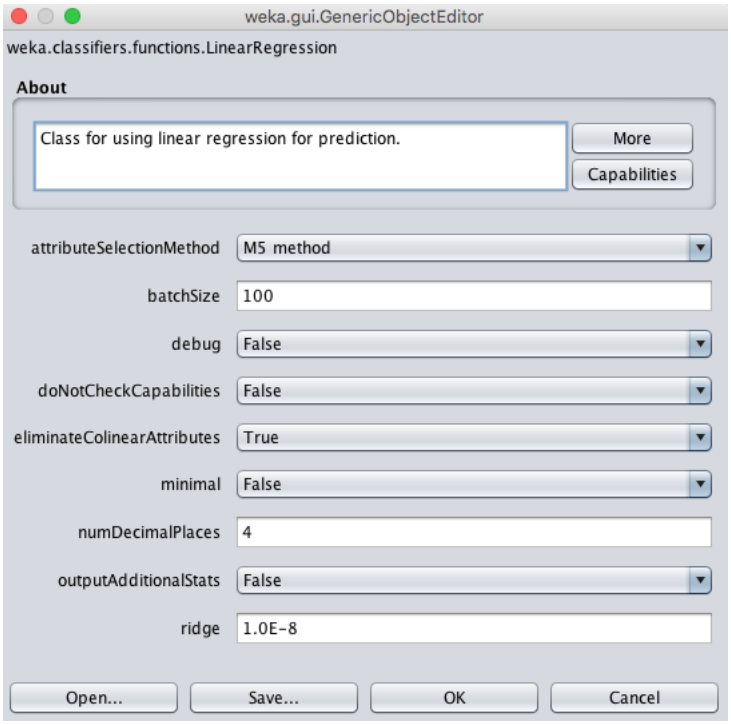


Figure 18.1: Weka Configuration of Linear Regression.

The performance of linear regression can be reduced if your training data has input attributes that are highly correlated. Weka can detect and remove highly correlated input attributes

automatically by setting *eliminateColinearAttributes* to *True*, which is the default. Additionally, attributes that are unrelated to the output variable can also negatively impact performance. Weka can automatically perform feature selection to only select those relevant attributes by setting the *attributeSelectionMethod*. This is enabled by default and can be disabled.

Finally, the Weka implementation uses a ridge regularization technique in order to reduce the complexity of the learned model. It does this by minimizing the square of the absolute sum of the learned coefficients, which will prevent any specific coefficient from becoming too large (a sign of complexity in regression models).

1. Click *OK* to close the algorithm configuration.

2. Click the *Start* button to run the algorithm on the Boston house price dataset.

You can see that with the default configuration that linear regression achieves an RMSE of 4.9.

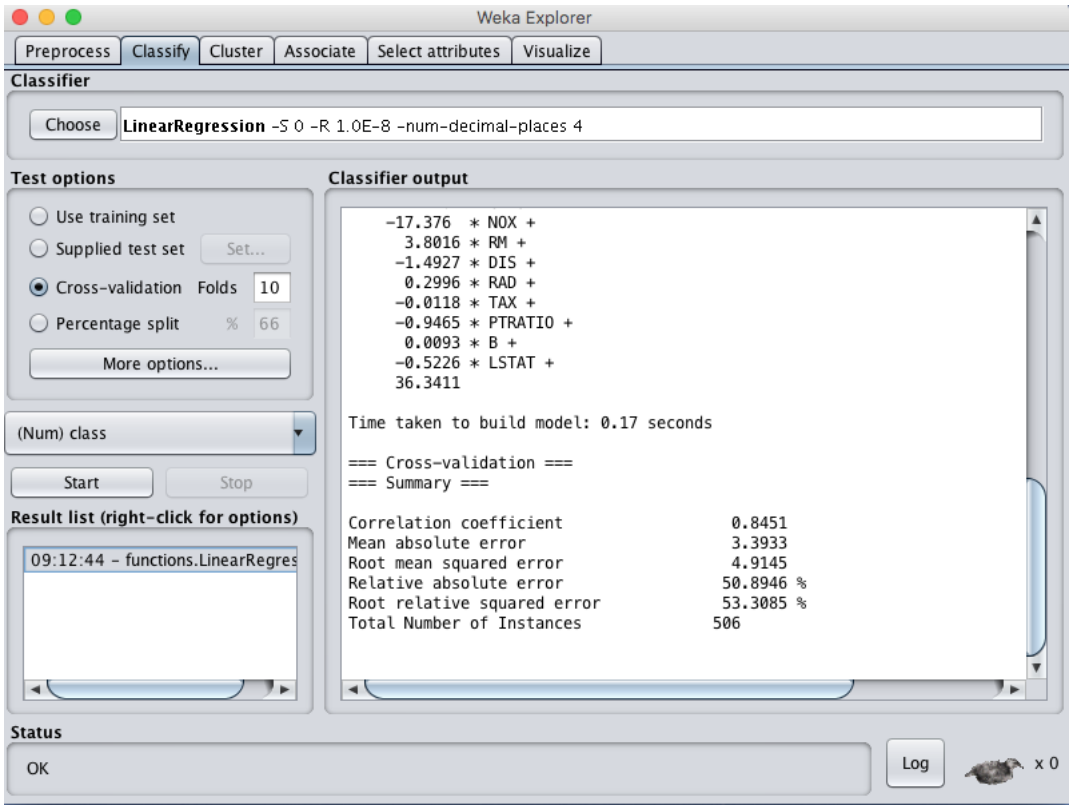


Figure 18.2: Weka Results for Linear Regression.

18.3 k-Nearest Neighbors

The k-Nearest Neighbors algorithm supports both classification and regression. It is also called KNN for short. It works by storing the entire training dataset and querying it to locate the k most similar training patterns when making a prediction. As such, there is no model other than the raw training dataset and the only computation performed is the querying of the training dataset when a prediction is requested.

It is a simple algorithm, but one that does not assume very much about the problem other than that the distance between data instances is meaningful in making predictions. As such, it often achieves very good performance. When making predictions on regression problems, KNN will take the mean of the k most similar instances in the training dataset. Choose the KNN algorithm:

1. Click the *Choose* button and select *IBk* under the *lazy* group.
2. Click on the name of the algorithm to review the algorithm configuration.

In Weka KNN is called *IBk* which stands for Instance-Based k.

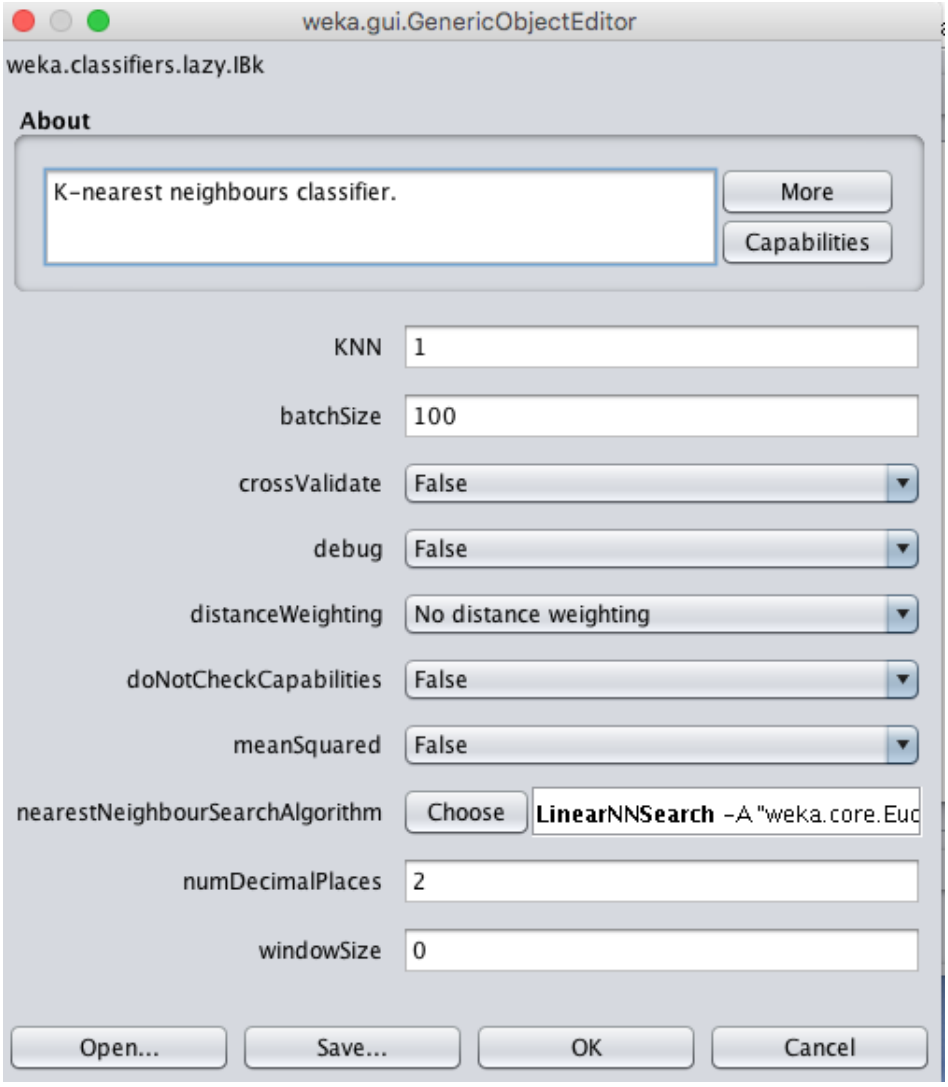


Figure 18.3: Weka k-Nearest Neighbors Configuration.

The size of the neighborhood is controlled by the k parameter, called *IBk* in Weka. For example, if set to 1, then predictions are made using the single most similar training instance to a given new pattern for which a prediction is requested. Common values for k are 3, 7, 11 and 21, larger for larger dataset sizes. Weka can automatically discover a good value for k using cross-validation inside the algorithm by setting the *crossValidate* parameter to *True*.

Another important parameter is the distance measure used. This is configured in the *nearestNeighbourSearchAlgorithm* which controls the way in which the training data is stored and searched. The default is a *LinearNNSearch*. Clicking the name of this search algorithm will provide another configuration window where you can choose a *distanceFunction* parameter. By default, *Euclidean* distance is used to calculate the distance between instances, which is good for numerical data with the same scale. *Manhattan* distance is good to use if your attributes differ in measures or type. It is a good idea to try a suite of different k values and distance measures on your problem and see what works best.

1. Click *OK* to close the algorithm configuration.

2. Click the *Start* button to run the algorithm on the Boston house price dataset.

You can see that with the default configuration that KNN algorithm achieves an RMSE of 4.6.

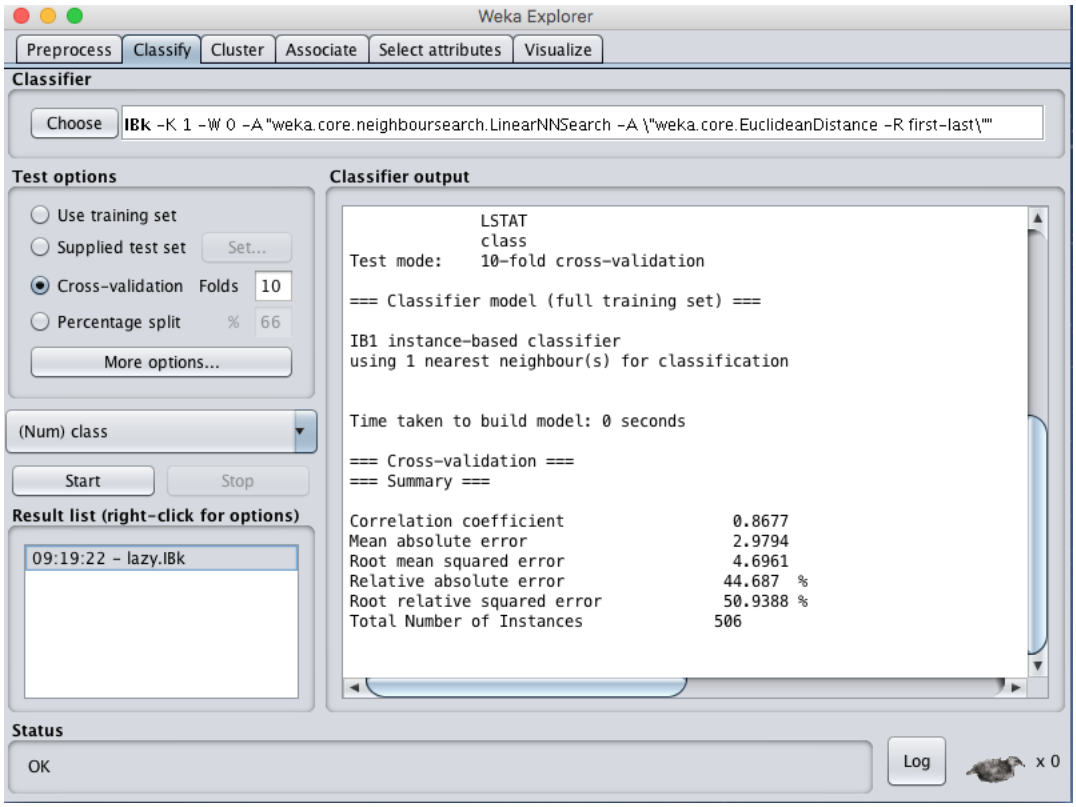


Figure 18.4: Weka Regression Results for the k-Nearest Neighbors Algorithm.

18.4 Decision Tree

Decision trees can support classification and regression problems. Decision trees are more recently referred to as Classification And Regression Trees or CART. They work by creating a tree to evaluate an instance of data. The tree is inverted, starting at the top or root of the tree and moving down to the leaves until a prediction can be made. The process of creating a decision tree works by greedily selecting the best split point in order to make predictions and

repeating the process until the tree is a fixed depth. After the tree is constructed, it is pruned in order to improve the model's ability to generalize to new data. Choose the decision tree algorithm:

1. Click the *Choose* button and select *REPTree* under the *trees* group.
2. Click on the name of the algorithm to review the algorithm configuration.

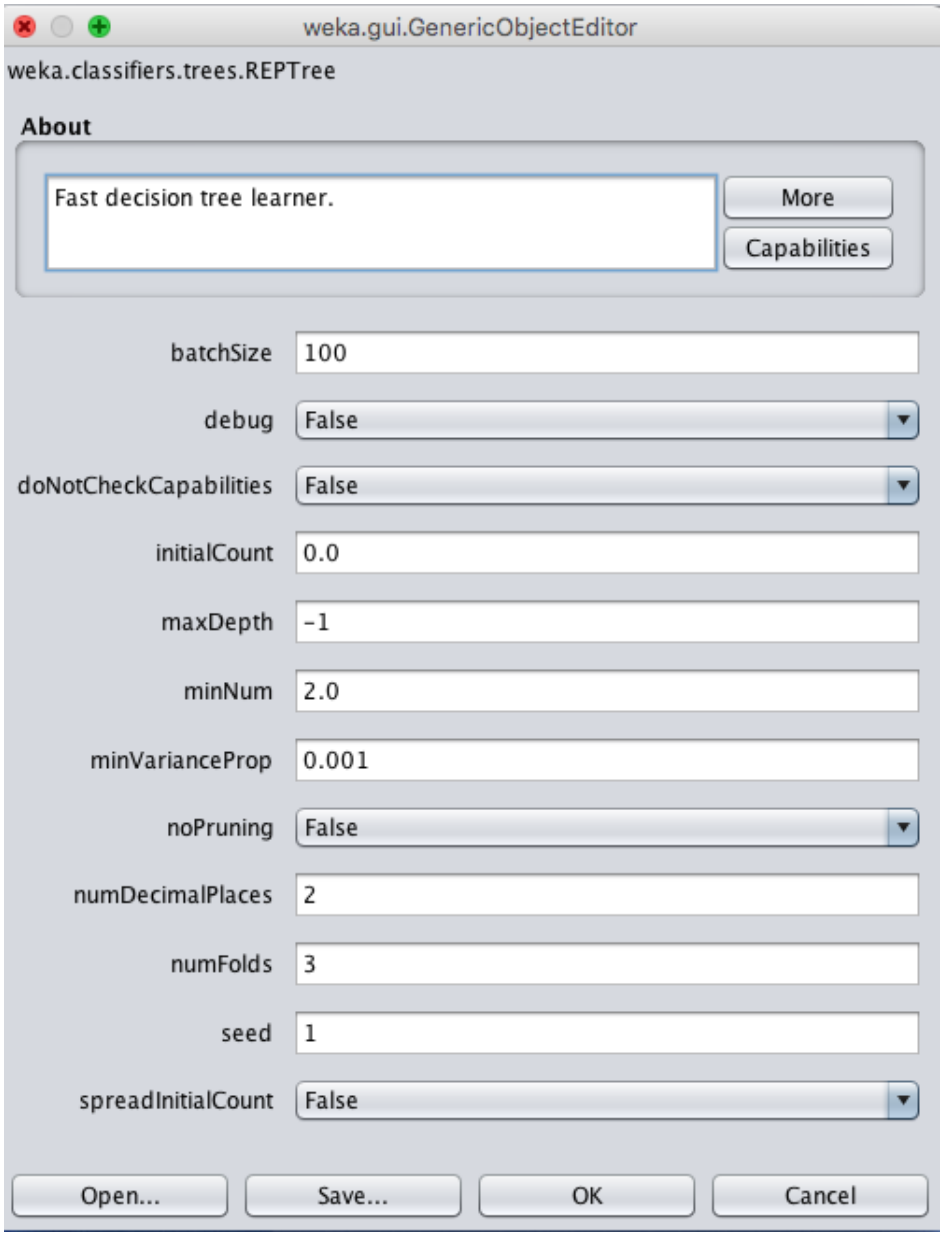


Figure 18.5: Weka Configuration for Decision Tree Algorithm.

The depth of the tree is defined automatically, but can specify a depth in the *maxDepth* parameter. You can also choose to turn off pruning by setting the *noPruning* parameter to *True*, although this may result in worse performance. The *minNum* parameter defines the minimum

number of instances supported by the tree in a leaf node when constructing the tree from the training data.

1. Click *OK* to close the algorithm configuration.

2. Click the *Start* button to run the algorithm on the Boston house price dataset.

You can see that with the default configuration that decision tree algorithm achieves an RMSE of 4.8.

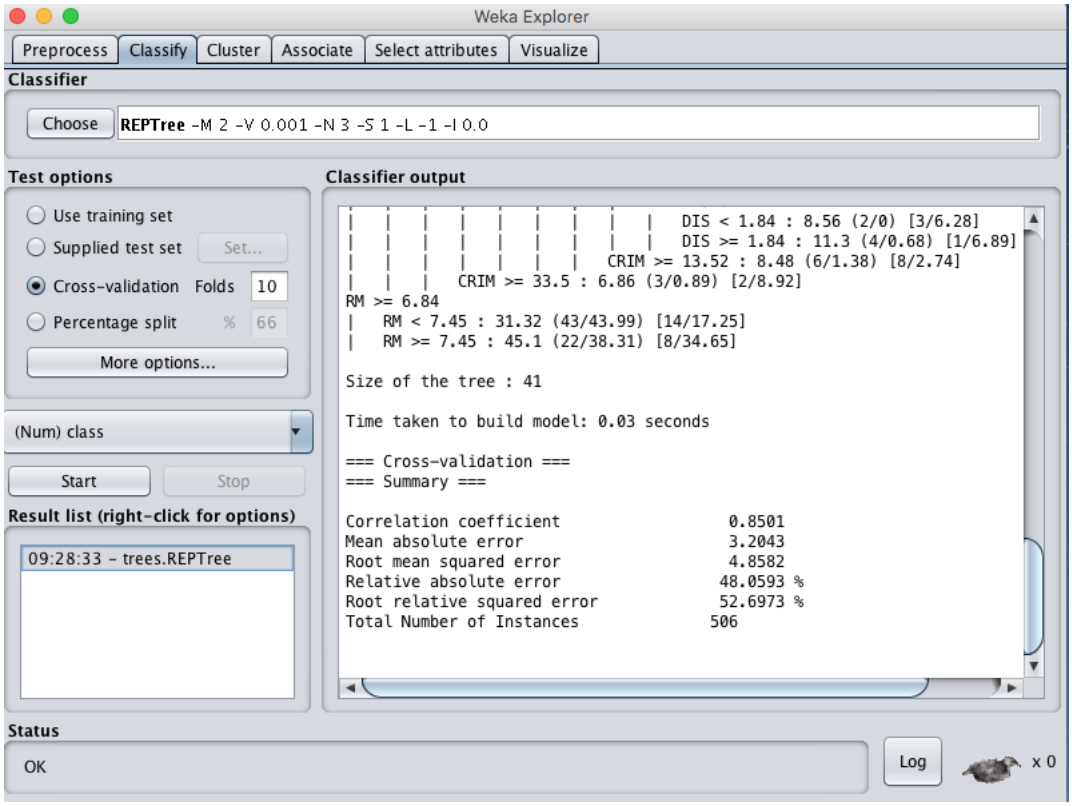


Figure 18.6: Weka Regression Results for the Decision Tree Algorithm.

18.5 Support Vector Regression

Support Vector Machines were developed for binary classification problems, although extensions to the technique have been made to support multiclass classification and regression problems. The adaptation of SVM for regression is called Support Vector Regression or SVR for short. SVM was developed for numerical input variables, although will automatically convert nominal values to numerical values. Input data is also normalized before being used.

Unlike SVM that finds a line that best separates the training data into classes, SVR works by finding a line of best fit that minimizes the error of a cost function. This is done using an optimization process that only considers those data instances in the training dataset that are closest to the line with the minimum cost. These instances are called support vectors, hence the name of the technique. In almost all problems of interest, a line cannot be drawn to best fit the

data, therefore a margin is added around the line to relax the constraint, allowing some bad predictions to be tolerated but allowing a better result overall.

Finally, few datasets can be fit with just a straight line. Sometimes a line with curves or even polygonal regions need to be marked out. This is achieved by projecting the data into a higher dimensional space in order to draw the lines and make predictions. Different kernels can be used to control the projection and the amount of flexibility. Choose the SVR algorithm:

1. Click the *Choose* button and select *SMOreg* under the *function* group.
2. Click on the name of the algorithm to review the algorithm configuration.

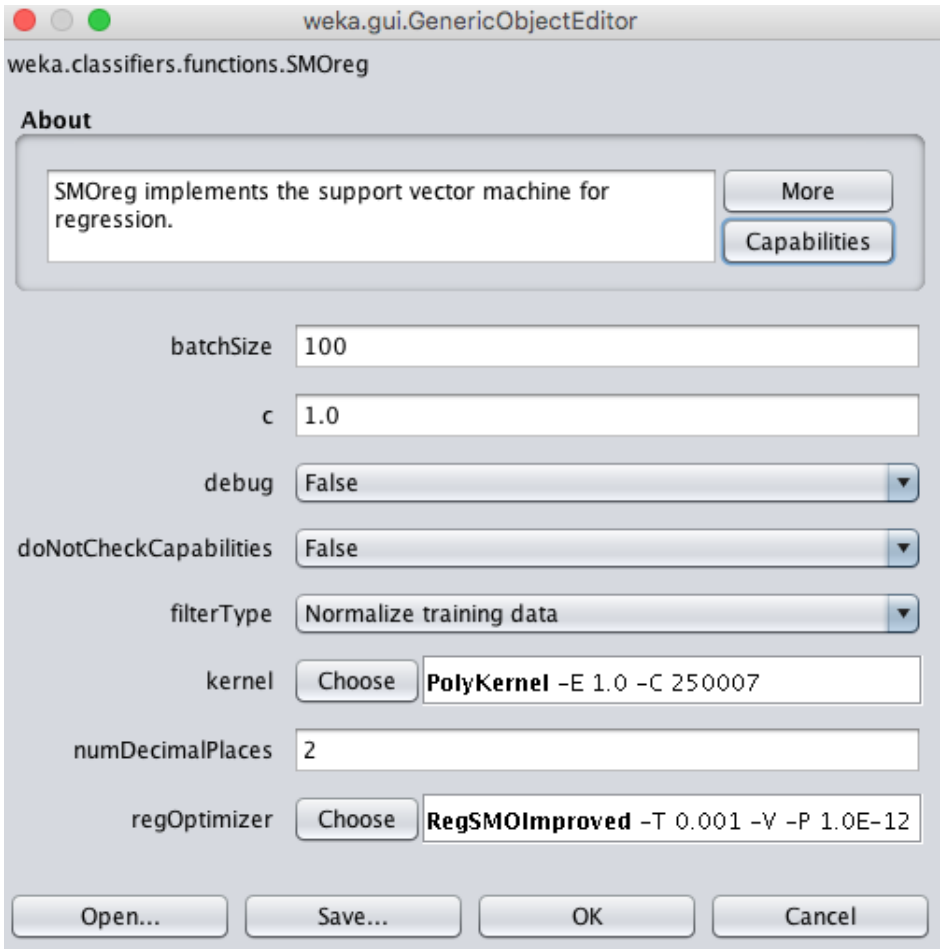


Figure 18.7: Weka Configuration for the Support Vector Regression Algorithm.

The *C* parameter, called the complexity parameter in Weka controls how flexible the process for drawing the line to fit the data can be. A value of 0 allows no violations of the margin, whereas the default is 1. A key parameter in SVM is the type of *Kernel* to use. The simplest kernel is a *Linear* kernel that separates data with a straight line or hyperplane. The default in Weka is a *Polynomial* kernel that will fit the data using a curved or wiggly line, the higher the polynomial, the more wiggly (the exponent value). The *Polynomial* kernel has a default *exponent* of 1, which makes it equivalent to a linear kernel. A popular and powerful kernel is the *RBF* kernel or Radial Basis Function Kernel that is capable of learning closed polygons and

complex shapes to fit the training data. It is a good idea to try a suite of different kernels and C (complexity) values on your problem and see what works best.

1. Click *OK* to close the algorithm configuration.

2. Click the *Start* button to run the algorithm on the Boston house price dataset.

You can see that with the default configuration that SVR algorithm achieves an RMSE of 5.1.

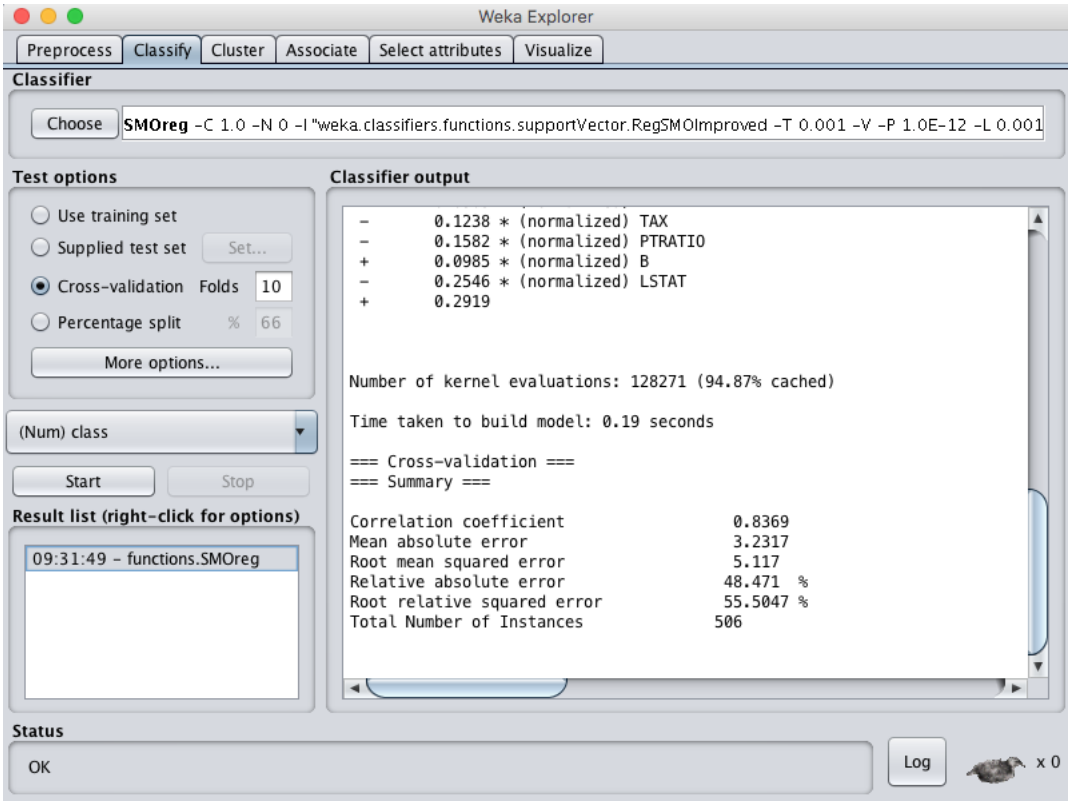


Figure 18.8: Weka Regression Results for the Support Vector Regression Algorithm.

18.6 Multilayer Perceptron

The Multilayer Perceptron algorithms supports both regression and classification problems. It is also called artificial neural networks or simply neural networks for short. Neural networks are a complex algorithm to use for predictive modeling because there are so many configuration parameters that can only be tuned effectively through intuition and a lot of trial and error.

It is an algorithm inspired by a model of biological neural networks in the brain where small processing units called neurons are organized into layers that if configured well are capable of approximating any function. In classification we are interested in approximating the underlying function to best discriminate between classes. In regression problems we are interested in approximating a function that best fits the real value output. Choose the Multilayer Perceptron algorithm:

1. Click the *Choose* button and select *MultilayerPerceptron* under the *function* group.

2. Click on the name of the algorithm to review the algorithm configuration.

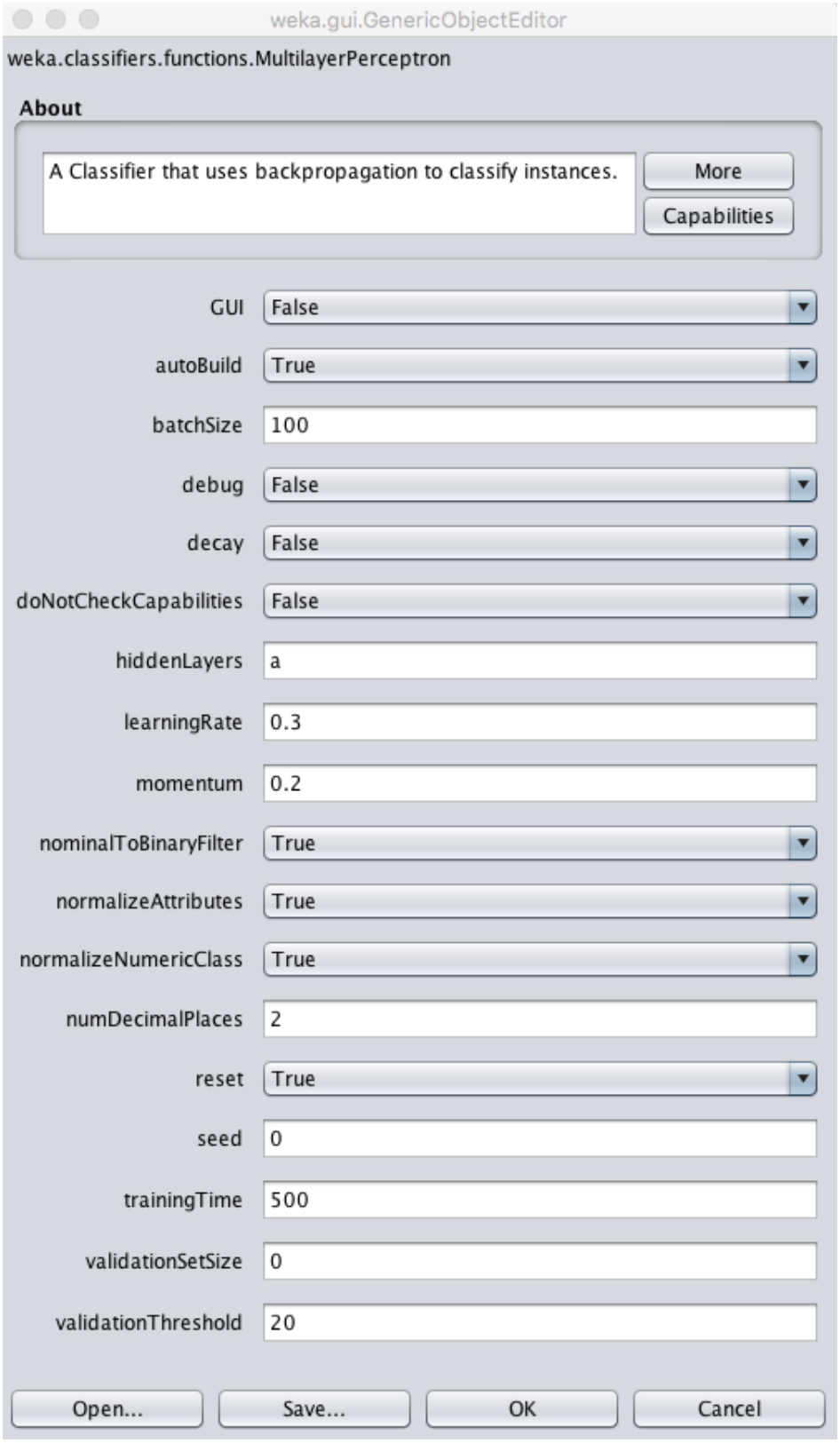


Figure 18.9: Weka Configuration for the Multilayer Perceptron Algorithm.

You can manually specify the structure of the neural network that is used by the model, but this is not recommended for beginners. The default will automatically design the network and train it on your dataset. The default will create a single hidden layer network. You can specify the number of hidden layers in the hiddenLayers parameter, set to automatic *a* by default. You can also use a GUI to design the network structure. This can be fun, but it is recommended that you use the GUI with a simple train and test split of your training data, otherwise you will be asked to design a network for each of the 10-folds of cross-validation.

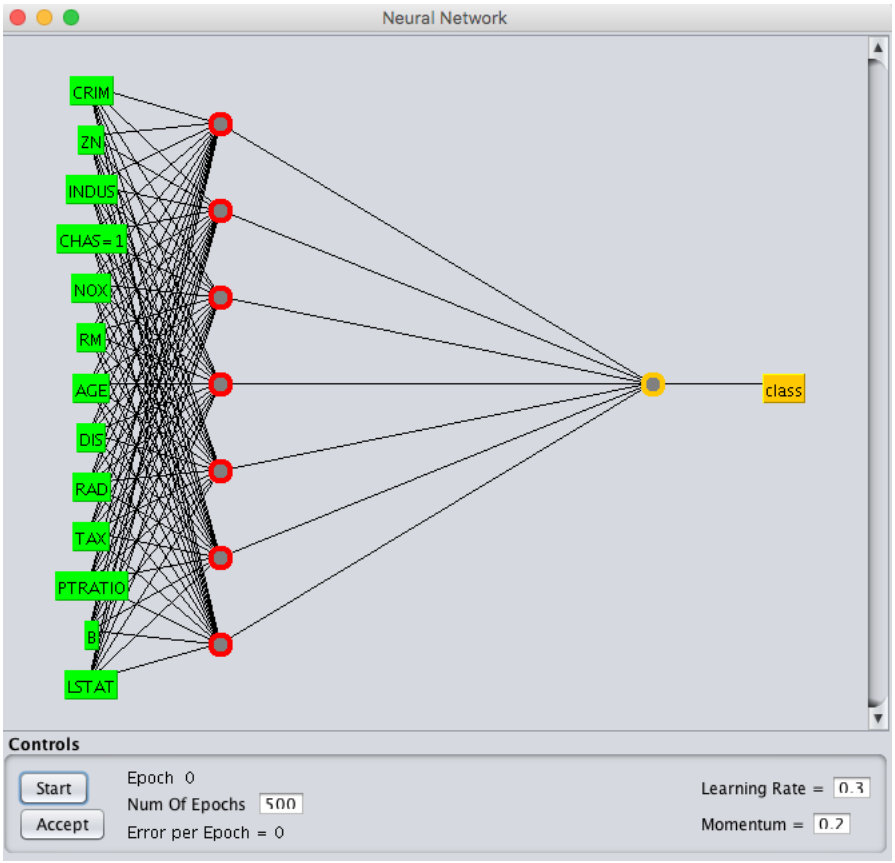


Figure 18.10: Weka GUI Designer for the Multilayer Perceptron Algorithm.

You can configure the learning process by specifying how much to update the model each epoch by setting the *learningRate*. Common values are small such as values between 0.3 (the default) and 0.1. The learning process can be further tuned with a *momentum* (set to 0.2 by default) to continue updating the weights even when no changes need to be made, and a decay (set *decay* to *True*) which will reduce the learning rate over time to perform more learning at the beginning of training and less at the end.

1. Click *OK* to close the algorithm configuration.

2. Click the *Start* button to run the algorithm on the Boston house price dataset.

You can see that with the default configuration that Multilayer Perceptron algorithm achieves an RMSE of 4.7.

18.7 Summary

In this lesson you discovered regression algorithms in Weka. Specifically you learned:

- About 5 top regression algorithms you can use for predictive modeling.
- How to run regression algorithms in Weka.
- About key configuration options for regression algorithms in Weka.

18.7.1 Next

Another key benefit of using the Weka platform for applied machine learning is the large number of ensemble methods that it supports. In the next lesson you will discover 5 top ensemble machine learning algorithms that you can use on your problem.

Chapter 19

How to Use Top Ensemble Machine Learning Algorithms

Ensemble algorithms are a powerful class of machine learning algorithm that combine the predictions from multiple models. A benefit of using Weka for applied machine learning is that it makes available so many different ensemble machine learning algorithms. In this lesson you will discover the how to use ensemble machine learning algorithms in Weka. After reading this lesson you will know:

- About 5 top ensemble machine learning algorithms.
- How to use top ensemble algorithms in Weka.
- About key configuration parameters for ensemble algorithms in Weka.

Let's get started.

19.1 Ensemble Algorithms Overview

We are going to take a tour of 5 top ensemble machine learning algorithms in Weka. Each algorithm that we cover will be briefly described in terms of how it works, key algorithm parameters will be highlighted and the algorithm will be demonstrated in the *Weka Explorer* interface. The 5 algorithms that we will review are:

1. Bagging.
2. Random Forest.
3. AdaBoost.
4. Voting.
5. Stacking.

These are 5 algorithms that you can try on your problem in order to lift performance. A standard machine learning classification problem will be used to demonstrate each algorithm. Specifically, the Ionosphere binary classification problem. You can learn more about this dataset in Section 8.2.3.

1. Open the *Weka GUI Chooser*.
2. Click the *Explorer* button to open the *Weka Explorer*.
3. Load the `data/ionosphere.arff` data
4. Click *Classify* to open the *Classify* tab.

19.2 Bootstrap Aggregation

Bootstrap Aggregation or Bagging for short is an ensemble algorithm that can be used for classification or regression. Bootstrap is a statistical estimation technique where a statistical quantity like a mean is estimated from multiple random samples of your data (with replacement). It is a useful technique when you have a limited amount of data and you are interested in a more robust estimate of a statistical quantity.

This sample principle can be used with machine learning models. Multiple random samples of your training data are drawn with replacement and used to train multiple different machine learning models. Each model is then used to make a prediction and the results are averaged to give a more robust prediction. It is a technique that is best used with models that have a low bias and a high variance, meaning that the predictions they make are highly dependent on the specific data from which they were trained. The most used algorithm for bagging that fits this requirement of high variance are decision trees.

Choose the bagging algorithm:

1. Click the *Choose* button and select *Bagging* under the *meta* group.
2. Click on the name of the algorithm to review the algorithm configuration.

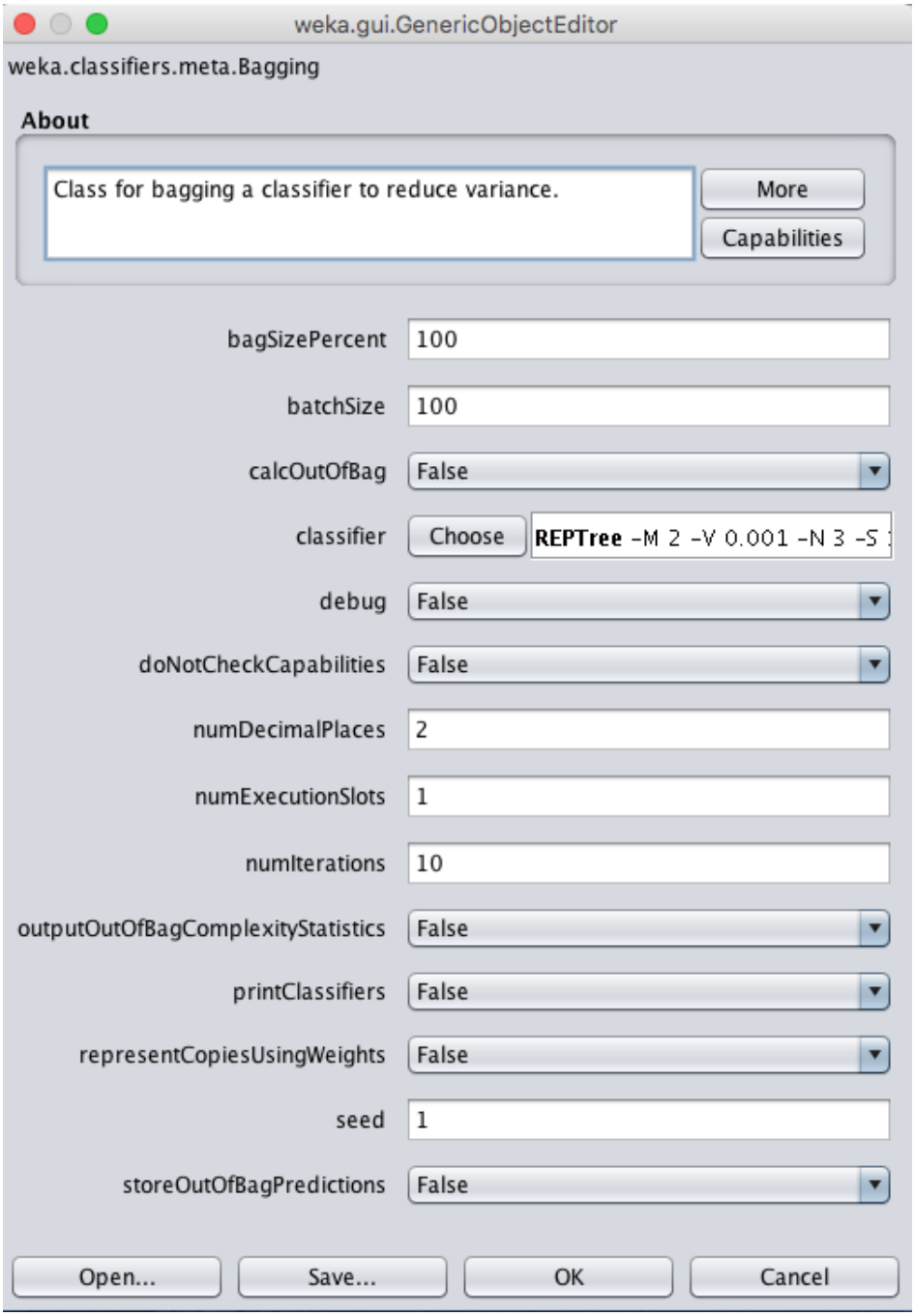


Figure 19.1: Weka Configuration for the Bagging Algorithm.

A key configuration parameter in bagging is the type of model being bagged. The default is the *REPTree* which is the Weka implementation of a standard decision tree, also called a Classification and Regression Tree or CART for short. This is specified in the *classifier* parameter. The size of each random sample is specified in the *bagSizePercent*, which is a size as a percentage of the raw training dataset. The default is 100% which will create a new random sample the same size as the training dataset, but will have a different composition. This is because the random sample is drawn with replacement, which means that each time an instance is randomly drawn from the training dataset and added to the sample, it is also added back

into the training dataset (replaced) meaning that it can be chosen again and added twice or more times to the sample.

Finally, the number of bags (and number of classifiers) can be specified in the *numIterations* parameter. The default is 10, although it is common to use values in the hundreds or thousands. Continue to increase the value of *numIterations* until you no longer see an improvement in the model, or you run out of memory.

1. Click *OK* to close the algorithm configuration.

2. Click the *Start* button to run the algorithm on the Ionosphere dataset.

You can see that with the default configuration that bagging achieves an accuracy of 91%.

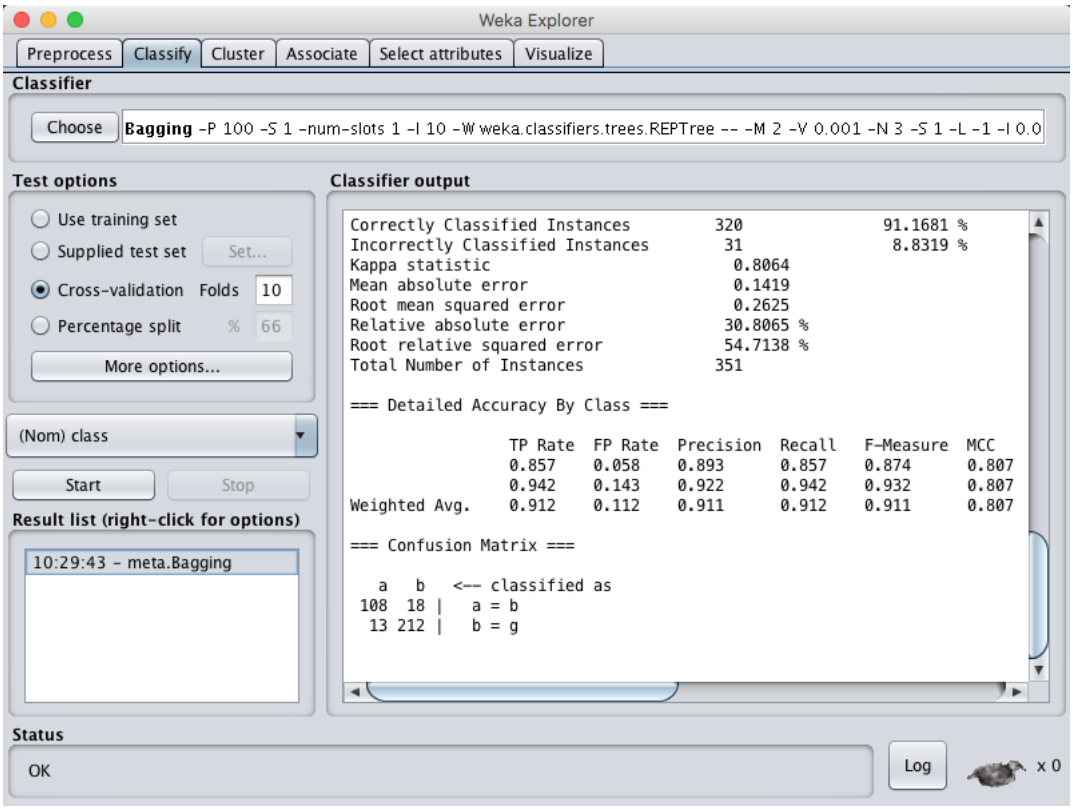


Figure 19.2: Weka Classification Results for the Bagging Algorithm.

19.3 Random Forest

Random Forest is an extension of bagging for decision trees that can be used for classification or regression. A down side of bagged decision trees is that decision trees are constructed using a greedy algorithm that selects the best split point at each step in the tree building process. As such, the resulting trees end up looking very similar which reduces the variance of the predictions from all the bags which in turn hurts the robustness of the predictions made.

Random Forest is an improvement upon bagged decision trees that disrupts the greedy splitting algorithm during tree creation so that split points can only be selected from a random subset of the input attributes. This simple change can have a big effect decreasing the similarity

between the bagged trees and in turn the resulting predictions. Choose the random forest algorithm:

1. Click the *Choose* button and select *RandomForest* under the *trees* group.
2. Click on the name of the algorithm to review the algorithm configuration.

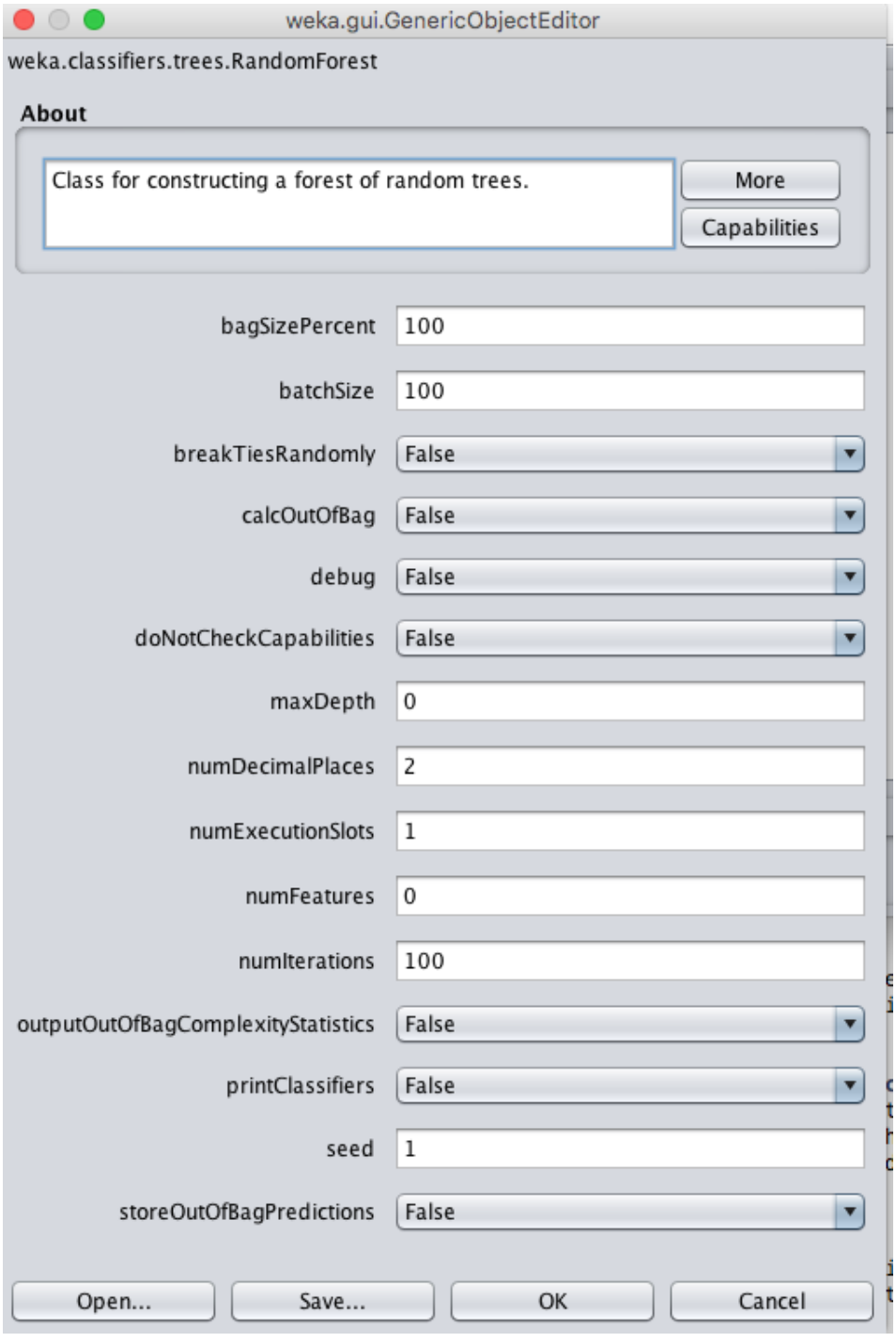


Figure 19.3: Weka Configuration for the Random Forest Algorithm.

In addition to the parameters listed above for bagging, a key parameter for random forest is the number of attributes to consider in each split point. In Weka this can be controlled by the *numFeatures* parameter, which by default is set to 0, which selects the value automatically based on a rule of thumb.

1. Click *OK* to close the algorithm configuration.

2. Click the *Start* button to run the algorithm on the Ionosphere dataset.

You can see that with the default configuration that random forests achieves an accuracy of 92%.

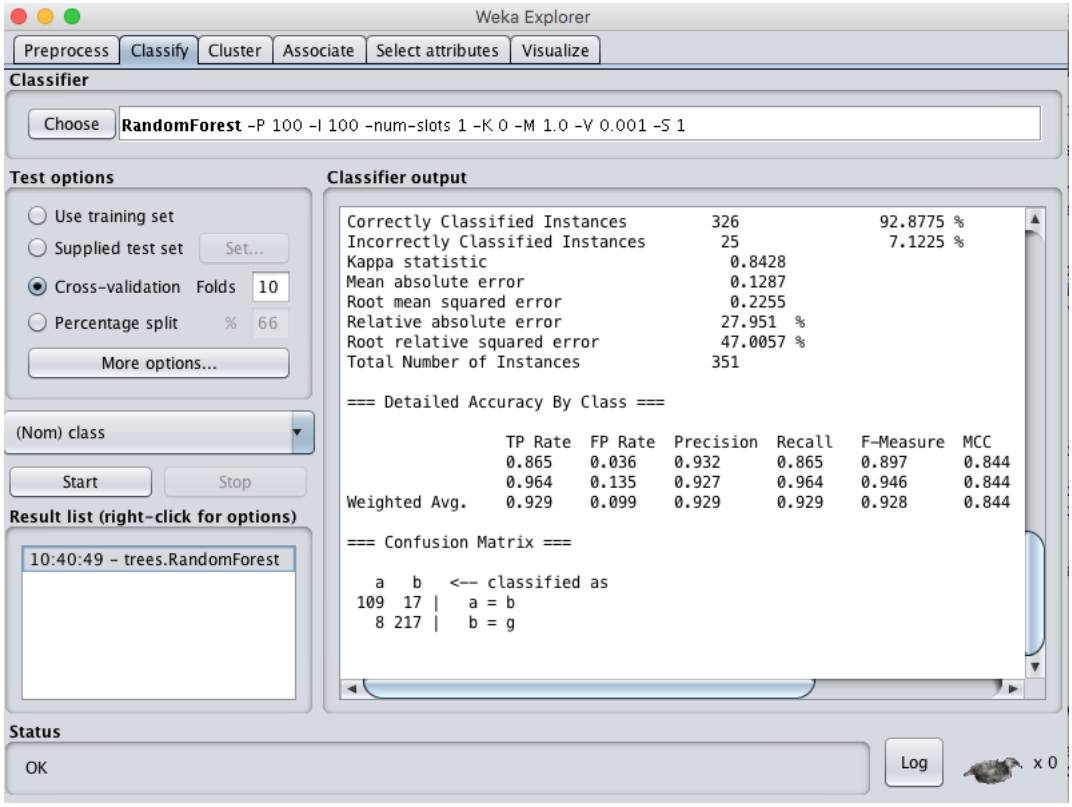


Figure 19.4: Weka Classification Results for the Random Forest Algorithm.

19.4 AdaBoost

AdaBoost is an ensemble machine learning algorithm for classification problems. It is part of a group of ensemble methods called boosting, that add new machine learning models in a series where subsequent models attempt to fix the prediction errors made by prior models. AdaBoost was the first successful implementation of this type of model. AdaBoost was designed to use short decision tree models, each with a single decision point. Such short trees are often referred to as decision stumps.

The first model is constructed as per normal. Each instance in the training dataset is weighted and the weights are updated based on the overall accuracy of the model and whether an instance was classified correctly or not. Subsequent models are trained and added until a

minimum accuracy is achieved or no further improvements are possible. Each model is weighted based on its skill and these weights are used when combining the predictions from all of the models on new data. Choose the AdaBoost algorithm:

1. Click the *Choose* button and select *AdaBoostM1* under the *meta* group.
2. Click on the name of the algorithm to review the algorithm configuration.

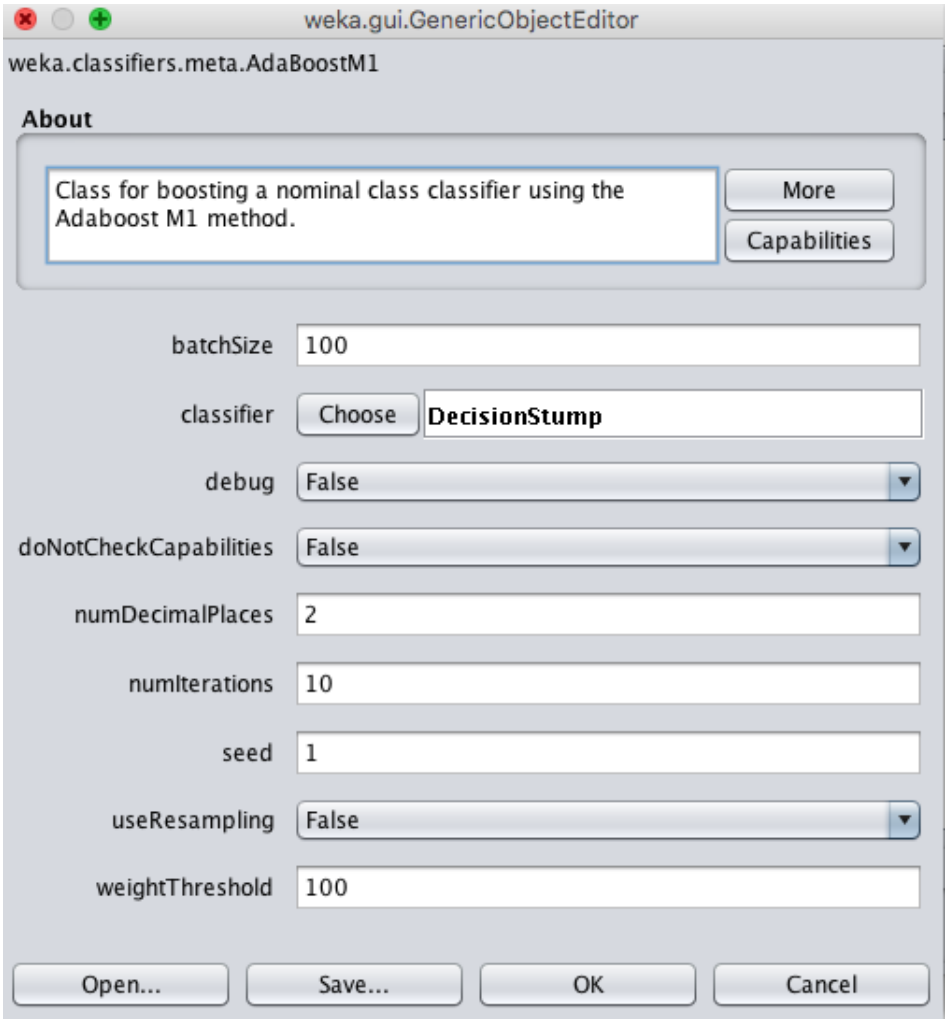


Figure 19.5: Weka Configuration for the AdaBoost Algorithm.

The weak learner within the AdaBoost model can be specified by the *classifier* parameter. The default is the decision stump algorithm, but other algorithms can be used. A key parameter in addition to the weak learner is the number of models to create and add in series. This can be specified in the *numIterations* parameter and defaults to 10.

1. Click *OK* to close the algorithm configuration.
2. Click the *Start* button to run the algorithm on the Ionosphere dataset.

You can see that with the default configuration that AdaBoost achieves an accuracy of 90%.

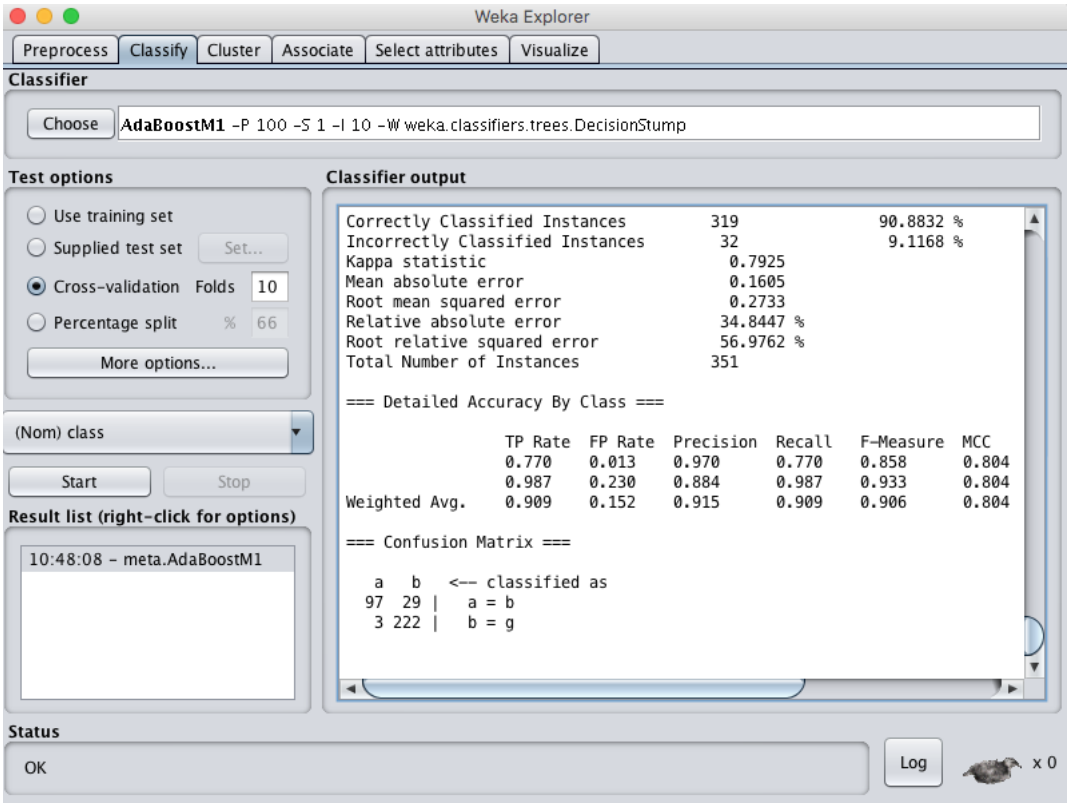


Figure 19.6: Weka Classification Results for the AdaBoost Algorithm.

19.5 Voting

Voting is perhaps the simplest ensemble algorithm, and is often very effective. It can be used for classification or regression problems. Voting works by creating two or more submodels. Each submodel makes predictions which are combined in some way, such as by taking the mean or the mode of the predictions, allowing each submodel to vote on what the outcome should be. Choose the Vote algorithm:

1. Click the *Choose* button and select *Vote* under the *meta* group.

2. Click on the name of the algorithm to review the algorithm configuration.

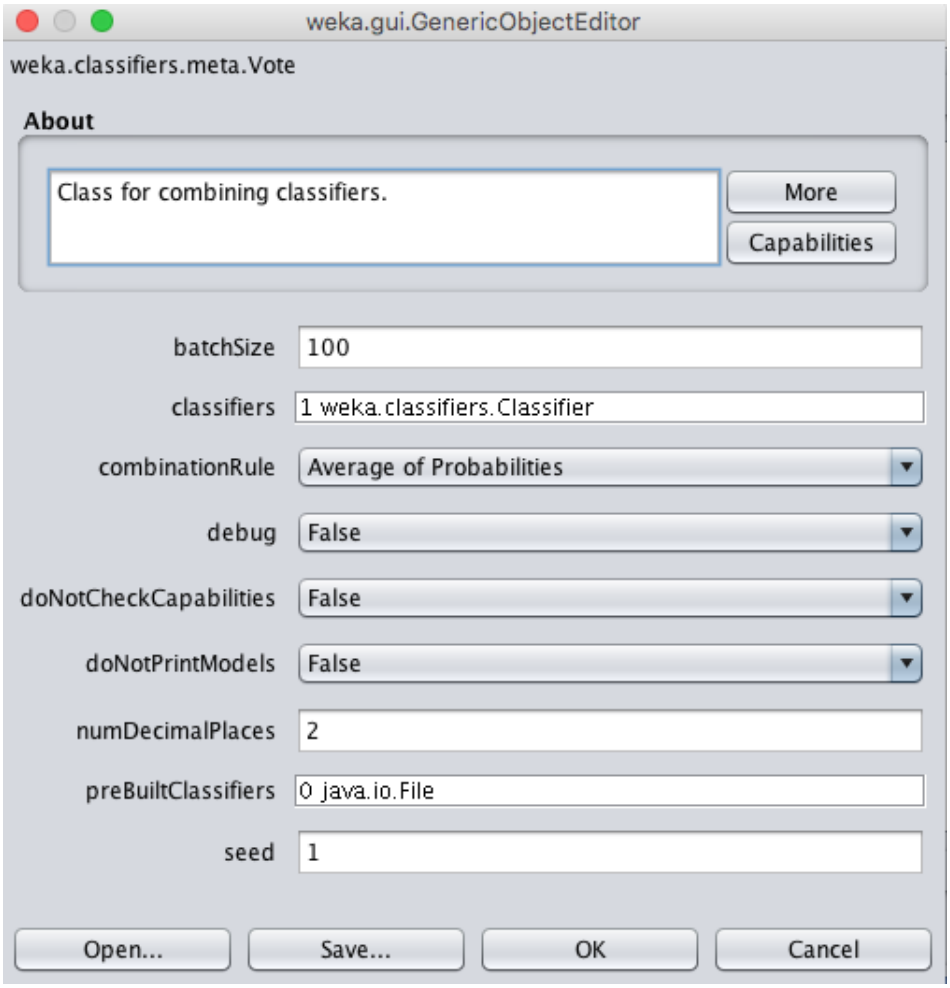


Figure 19.7: Weka Configuration for the Voting Ensemble Algorithm.

The key parameter of a Vote ensemble is the selection of submodels. Models can be specified in Weka in the *classifier* parameter. Clicking this parameter let's you add a number of classifiers.

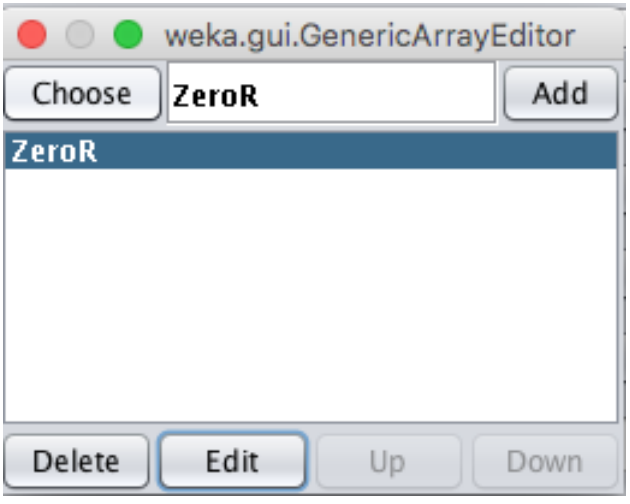


Figure 19.8: Weka Algorithm Selection for the Voting Ensemble Algorithm.

Clicking the *Edit* button with a classifier selected let's you configure the details of that classifier. An objective in selecting submodels is to select models that make quite different predictions (uncorrelated predictions). As such, it is a good rule of thumb to select very different model types, such as trees, instance-based methods, functions and so on. Another key parameter to configure for voting is how the predictions of the submodels are combined. This is controlled by the *combinationRule* parameter which is set to take the average of the probabilities by default.

1. Click *OK* to close the algorithm configuration.
2. Click the *Start* button to run the algorithm on the Ionosphere dataset.

You can see that with the default configuration that Vote achieves an accuracy of 64%. Obviously, this technique achieved poor results because only the *ZeroR* submodel was selected. Try selecting a collection of 5-to-10 different submodels.

19.6 Stacked Generalization

Stacked Generalization or Stacking for short is a simple extension to Voting ensembles that can be used for classification and regression problems. In addition to selecting multiple submodels, stacking allows you to specify another model to learn how to best combine the predictions from the submodels. Because a meta model is used to best combine the predictions of submodels, this technique is sometimes called blending, as in blending predictions together. Choose the Stacking algorithm:

1. Click the *Choose* button and select *Stacking* under the *meta* group.
2. Click on the name of the algorithm to review the algorithm configuration.

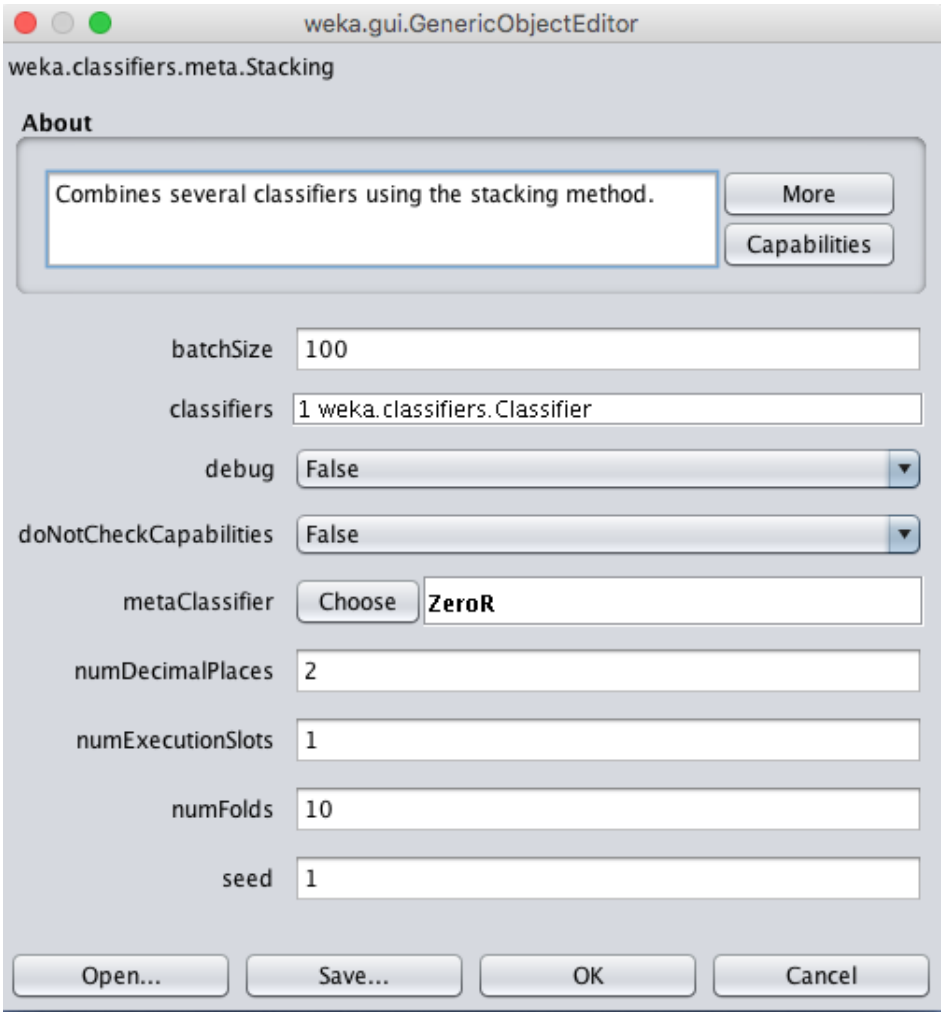


Figure 19.9: Weka Configuration for the Stacking Ensemble Algorithm.

As with the *Vote* classifier, you can specify the submodels in the classifiers parameter. The model that will be trained to learn how to best combine the predictions from the submodel can be specified in the *metaClassifier* parameter, which is set to *ZeroR* (majority vote or mean) by default. It is common to use a linear algorithm like linear regression or logistic regression for regression and classification type problems respectively. This is to achieve an output that is a simple linear combination of the predictions of the submodels.

1. Click *OK* to close the algorithm configuration.

2. Click the *Start* button to run the algorithm on the Ionosphere dataset.

You can see that with the default configuration that Stacking achieves an accuracy of 64%. Again, the same as voting, Stacking achieved poor results because only the *ZeroR* submodel was selected. Try selecting a collection of 5-to-10 different submodels and a good model to combine the predictions.

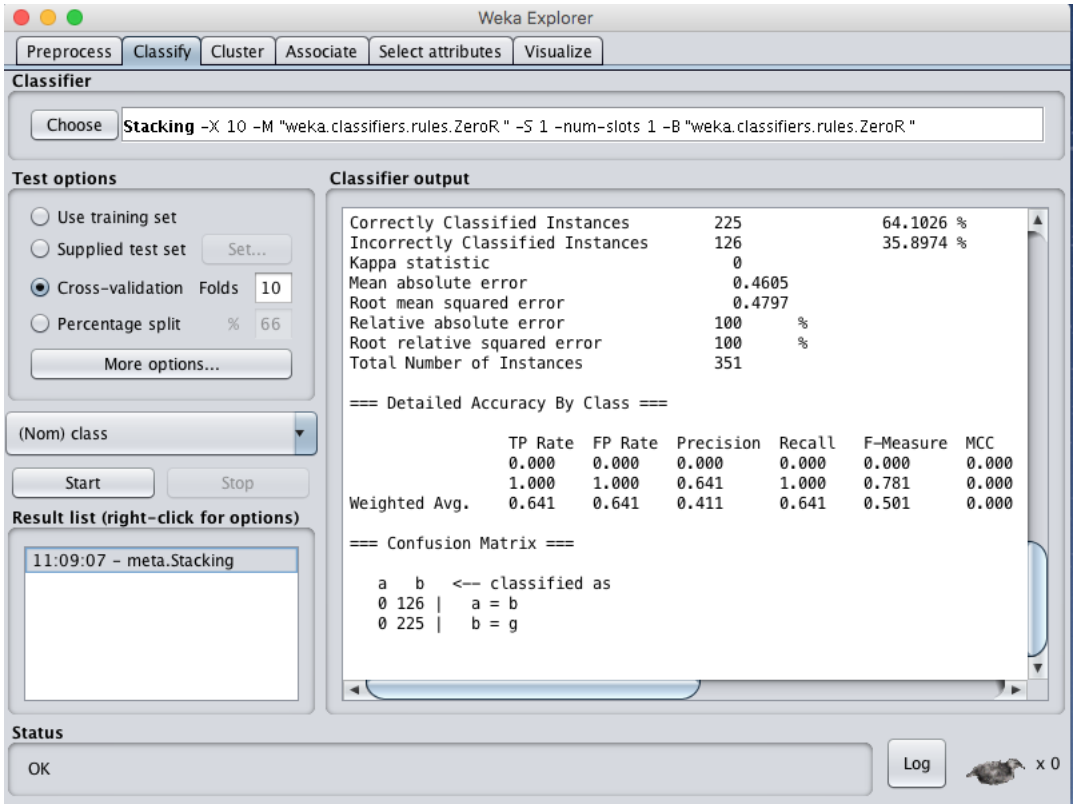


Figure 19.10: Weka Classification Results for the Stacking Ensemble Algorithm.

19.7 Summary

In this lesson you discovered how to use ensemble machine learning algorithms in Weka. Specifically you learned:

- About 5 ensemble machine learning algorithms that you can use on your problem.
- How to use ensemble machine learning algorithms in Weka.
- About the key configuration parameters for ensemble machine learning algorithms in Weka.

19.7.1 Next

Now that you know how to use a large number of machine learning algorithms, you must know how to effectively compare their performance. In the next lesson you will discover how you can design controlled experiments to compare algorithm performance and statistical methods to determine whether the differences in the results are significant.

Chapter 20

How To Compare the Performance of Machine Learning Algorithms

What algorithm should you use for a given machine learning problem? This is the challenge of applied machine learning. There is no quick answer to this question, but there is a reliable process that you can use. In this lesson you will discover how to find good and even best machine learning algorithms for a data set by directly comparing them in Weka. After reading this lesson you will know:

- The process for discovering good and even best machine learning algorithms for a problem.
- How to design an experiment in Weka to compare the performance of different machine learning algorithms.
- How to analyze the results of experiments in Weka.

Let's get started.

20.1 Best Machine Algorithm For A Problem

The most common question in applied machine learning is: *What algorithm is best for my problem?* The answer cannot be known beforehand. If you understood your problem well enough to know which algorithm was best, you would not need to use machine learning. You would simply solve your problem. Machine learning techniques are for those difficult problems where a solution must be learned from data. Where traditional techniques cannot be used. There are a lot of rules of thumb about choosing algorithms. For example, if an algorithm expects data of a specific distribution and your data has that distribution, then perhaps the algorithm is a good fit for your problem. There are two issues with this:

1. Many algorithms may have expectations that make them suitable for your problem.
2. Sometimes good and even best results can be achieved when the expectations of an algorithm are violated.

Rules of thumb are great for a starting point, but not the final choice of algorithm. The best machine learning algorithm for your problem is found empirically. By trial and error. This is done by evaluating a suite of algorithms of very different types on your problem, finding what works well and doubling down on those 2-to-3 that show promise. I call this approach spot-checking. The process is as follows:

- Design a test harness including the training dataset and any data preparation and the test options, such as 10-fold cross-validation. For bonus points repeat the experiment with a number of different presentation or views of your dataset. This will help to best expose the structure of the problem to algorithms that are receptive to learning it.
- Select a suite of diverse algorithms with very different assumptions about the problem, such as linear models, trees, instance-based methods, probabilistic methods, neural networks and more. For bonus points allow each algorithm to put its best foot forward, including variations of each algorithm with different commonly used configuration schemes.
- Evaluate the suite of algorithms on your training dataset.
- Analyze the results and identify 2-to-3 different algorithms on which to investigate further.

This process will always lead you to algorithms that perform well on your machine learning problem. In practice, you really just need a model that does as well as possible, given the time you have available. Finding the very best model for your problem is a matter of how much time you are willing to invest trying different algorithms and tuning those algorithms that perform well.

20.2 Compare Algorithm Performance in Weka

You can spot-check machine learning algorithms on your problem in Weka using the Experiment Environment. In this tutorial you are going to design, run and analyze your first machine learning experiment. We are going to use the Pima Indians Onset of Diabetes dataset. You can learn more about this dataset in Section 8.2.1. We are going to evaluate 5 different machine learning algorithms on the raw dataset:

- Logistic Regression (*Logistic*).
- Naive Bayes (NaiveBayes).
- Classification and Regression Trees or CART (*REPTree*).
- k-Nearest Neighbors or KNN (*IBk*).
- Support Vector Machines or SVM (*SMO*).

Each algorithm will be evaluated using the default algorithm configuration. Natural extensions of this experiment that you can investigate yourself are to:

- Create multiple different views of the dataset on which to evaluate the algorithms, such as normalized, standardized and more.

- Add more algorithms to the suite to evaluate.
- Add more variations of each algorithm with common or standard algorithm configurations.

This tutorial is divided into 3 parts:

1. Design The Experiment.
2. Run The Experiment.
3. Review Experiment Results.

20.3 Design The Experiment

The *Weka Experiment Environment* is a tool that you can use to run controlled experiments on datasets with machine learning algorithms. The *Weka Experiment Environment* allows you to define one or more datasets to work on and one or more algorithms to work on the dataset. You can then run and monitor the experiment. Finally, all of the results are collected and presented for you to analyze. In this section we are going to define an experiment with the Pima Indians onset of diabetes dataset, 10-fold cross-validation (the default) and 5 common classification algorithms.

Each algorithm will be evaluated on the dataset 10 times (10 runs of 10-fold cross-validation) with different random number seeds. This will result is 10 slightly different results for each evaluated algorithm, a small population that we can interpret using statistical methods later.

- 1. Open the *Weka GUI Chooser*.

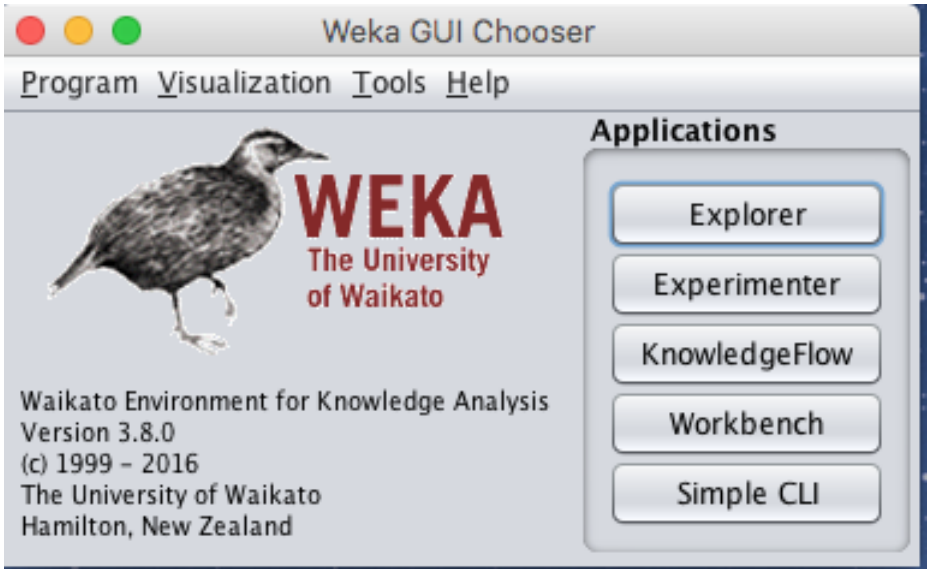


Figure 20.1: Weka GUI Chooser.

- 2. Click the *Experimenter* button to open the *Weka Experiment Environment* interface.

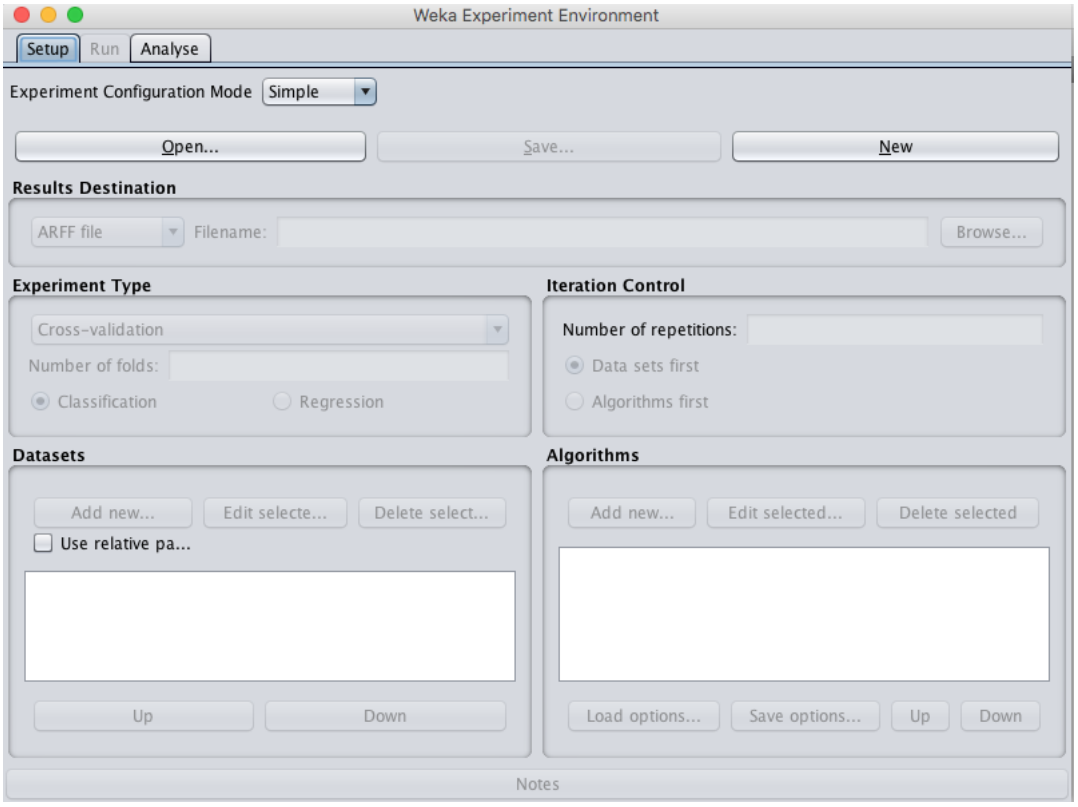


Figure 20.2: Weka Experiment Environment Setup Tab.

- 3. On the *Setup* tab, click the *New* button to start a new experiment.
- 4. In the *Dataset* pane, click the *Add new...* button and choose `data/diabetes.arff`.
- 5. In the *Algorithms* pane, click the *Add new...* button, click the *Choose* button and select the *Logistic* algorithm under the *functions* group. Click the *OK* button to add it.

Repeat and add the following 4 additional algorithms:

- *NaiveBayes* under the *bayes* group.
- *REPTree* under the *trees* group.
- *IBk* under the *lazy* group.
- *SMO* under the *functions* group.

You can save this experiment definition by clicking the *Save* button at the top of the *Setup* pane. This is useful if you want to create new variations on an experiment in the future. You can load the definition using the *Open* button. Finally, you can save the experiment results to an ARFF file. This is useful if you want to load and analyze the results at a later time. You can specify the file in which to save the experiment results in the *Results Destination* pane.

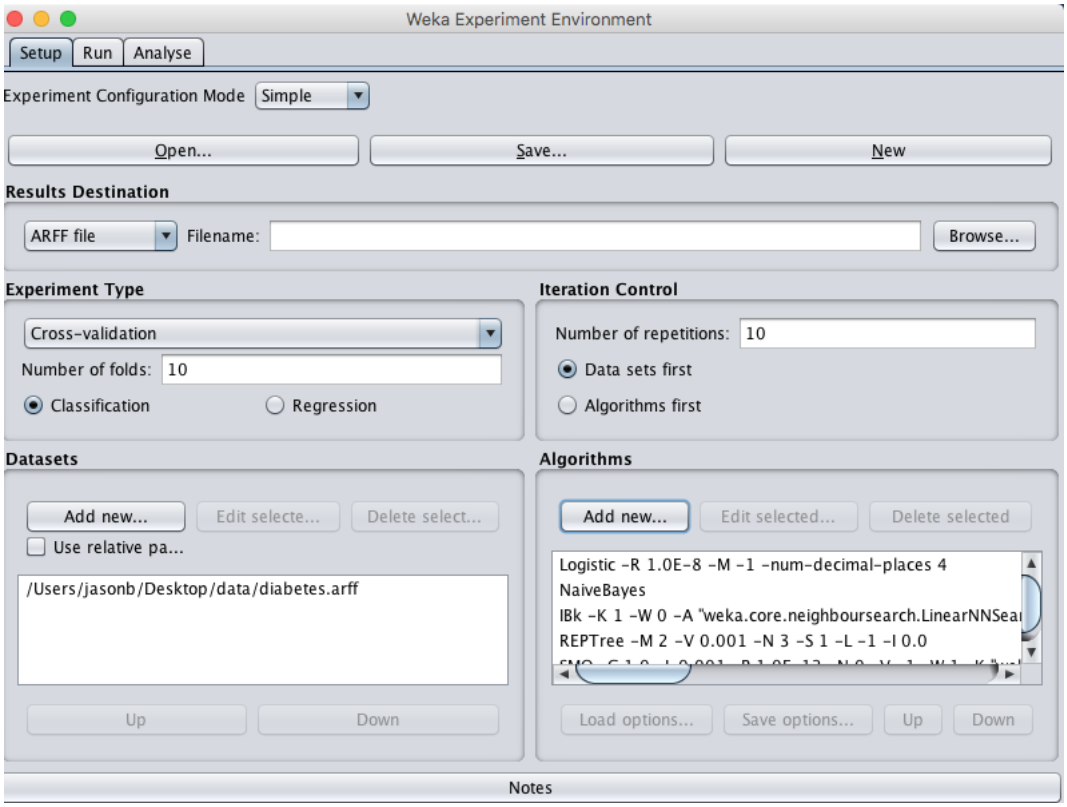


Figure 20.3: Weka Experiment Environment Configured Experiment.

20.4 Run The Experiment

Now it is time to run the experiment.

- 1. Click the *Run* tab.

There few options. All you can do is start an experiment or stop a running experiment.

- 2. Click the *Start* button and run the experiment. It should complete in a few seconds. This is because the dataset is small.

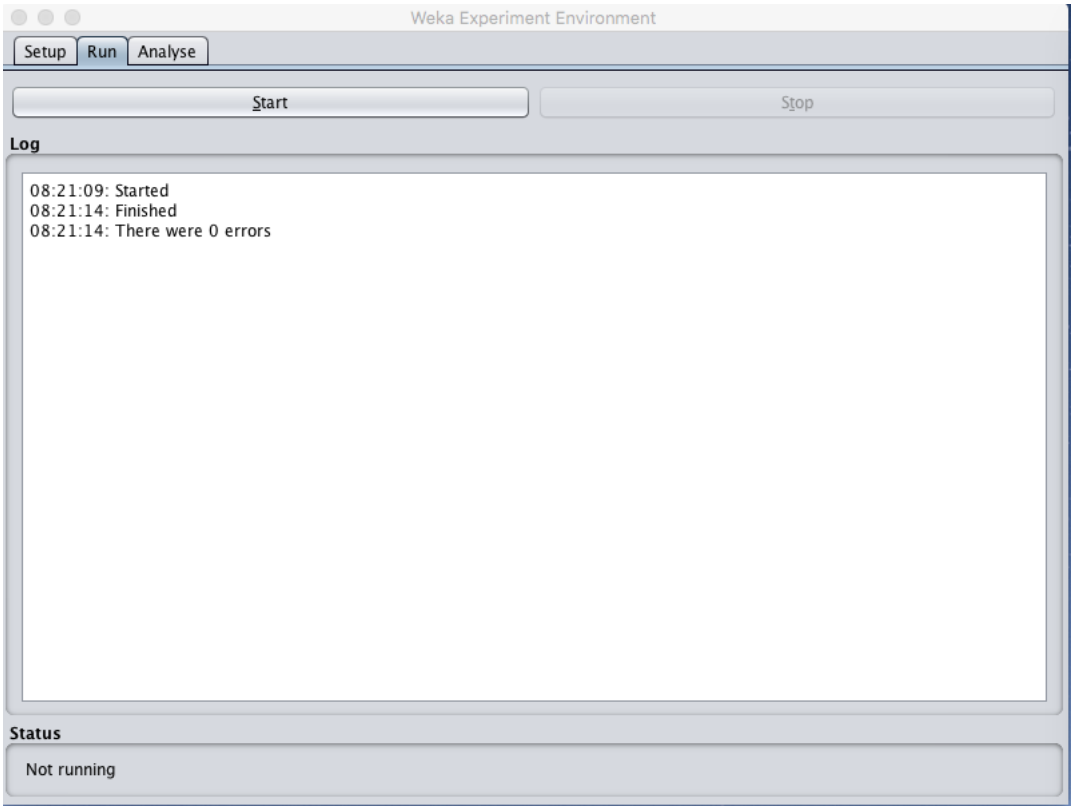


Figure 20.4: Weka Experiment Environment Run Experiment.

20.5 Review Experiment Results

The third tab of the *Weka Experiment Environment* is for analyzing experimental results.

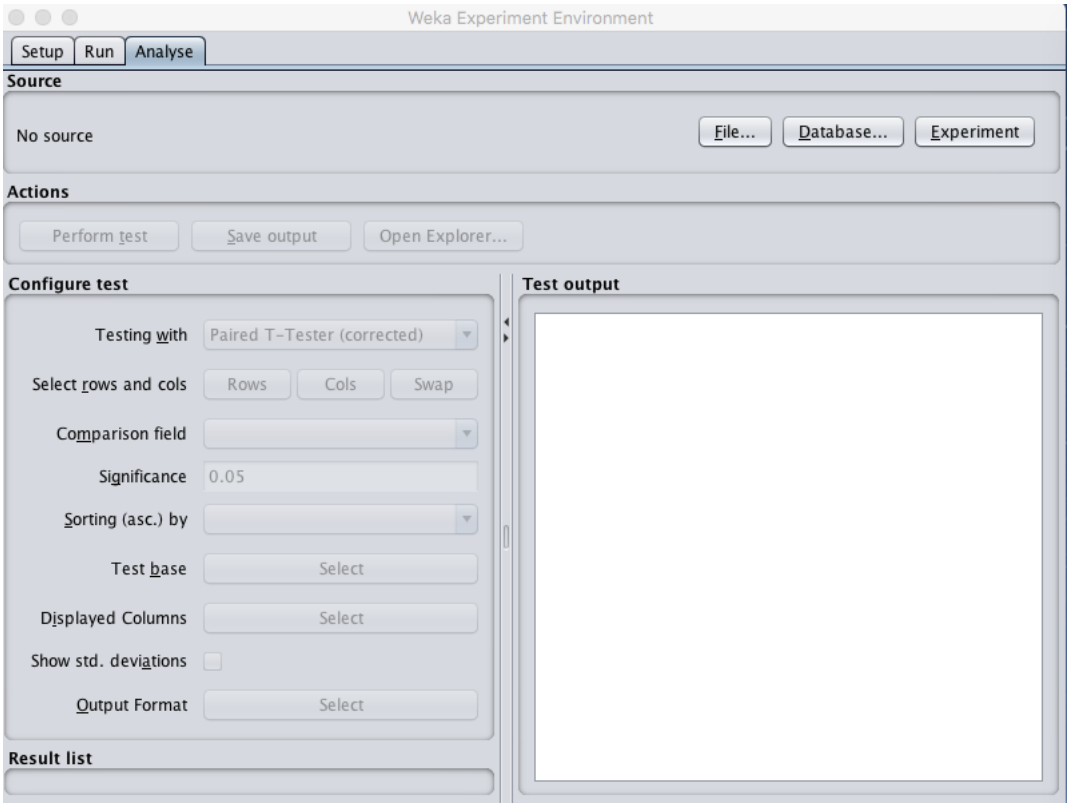


Figure 20.5: `Weka Experiment Environment Analyse` Tab.

You can load results from:

- A file, if you configured your experiment to save results to a file on the *Setup* tab.
- A Database, if you configured your experiment to save results to a database on the *Setup* tab.
- A Experiment, if you just ran an experiment in the Experiment Environment (which we just did).

Load the results from the experiment we just executed by clicking the *Experiment* button in the *Source* pane. You will see that 500 results were loaded. This is because we had 5 algorithms that were each evaluated 100 times, 10-fold cross-validation multiplied by 10 repeats.

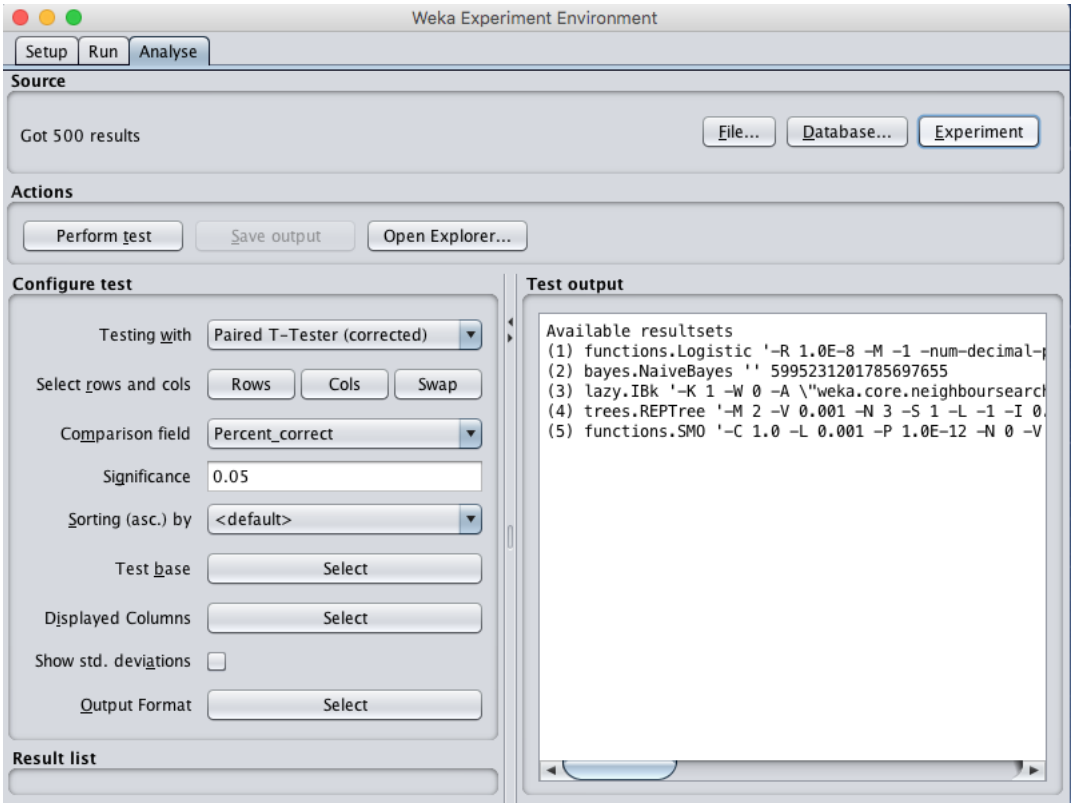


Figure 20.6: Weka Experiment Environment Load Results.

Results were collected for many different performance measures, such as classification accuracy. The *Weka Experiment Environment* allows us to performs statistical tests on the different performance measures to allow us to draw conclusions from the experiment. For example, we are interested in two questions from our experiments:

- Which algorithm evaluated in the experiment had the best performance? This is useful to know if we wanted to create a good performance model immediately.
- What is the rank of algorithms by performance? This is useful to know if we want to further investigate and tune the 2-to-3 algorithms that performed the best on the problem.

We can configure the result summary to display in the *Configure test* pane. The type of statistical test can be selected in the *Testing with* option, by default this is set to *Paired T-Tester (corrected)*. This is just fine and will compare each algorithm in pairs and make some reasonable assumptions about the distribution of the results collected, such as that they are drawn from a Gaussian distribution. The significance level is set in the *Significance* parameter and is default to 0.05 (5%), which again, is just fine.

We do not need get caught up in the technical details of statistical significance testing. These useful defaults will inform us whether the differences between any of the pairwise algorithm performance comparisons we review are statistically significant with a confidence of 95%. We can choose the performance measure by which to compare the algorithms in the *Comparison field* option. The default is the *Percent_correct* metric (accuracy) which is exactly what we are interested in as a first pass.

We can compare all of the algorithm results to one base result. This can be specified by the *Test base* option. The default is the first algorithm evaluated in the list, which in this case is Logistic regression. We can see this by clicking the *Select* button next to *Test base*.

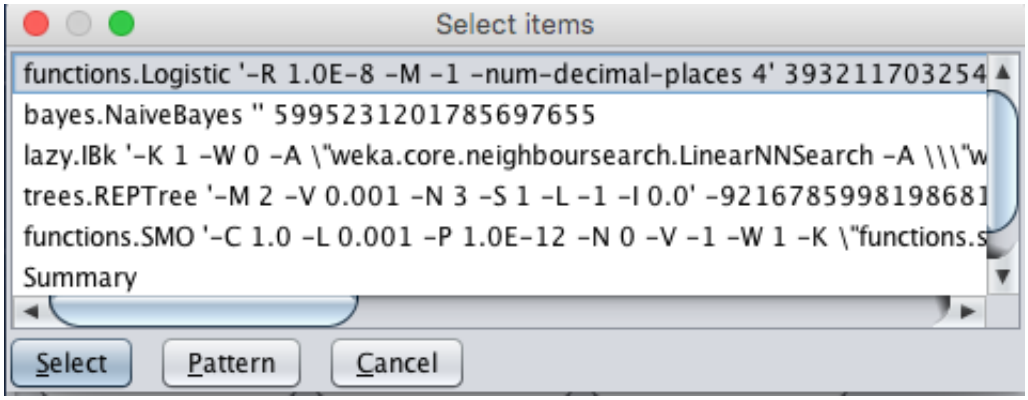


Figure 20.7: Weka Experiment Environment Test Base.

Click the *Perform test* button in the *Actions* pane to perform the statistical test and produce some output we can review. You should see results like those listed below.

```
Dataset                 (1) function | (2) bayes (3) lazy. (4) trees (5) funct
----------------------------------------------------------------------------
pima_diabetes          (100)  77.47 | 75.75    70.62 *  74.46 *  76.80
----------------------------------------------------------------------------
                            (v/ /*) | (0/1/0)  (0/0/1)  (0/0/1)  (0/1/0)


Key:
(1) functions.Logistic
(2) bayes.NaiveBayes
(3) lazy.IBk
(4) trees.REPTree
(5) functions.SMO
```

Listing 20.1: Algorithm Comparison Results.

We can see that *Logistic*, our base for comparison marked as (1) has the accuracy of 77.47% on the problem. This result is compared to the other 4 algorithms and indicated with a number and mapped in a legend at the bottom below the table of results. Note the * next to the *IBk* and *REPTree* results. This indicates that the results are significantly different from the *Logistic* results, but the scores are lower. *NaiveBayes* and *SMO* do not have any character next to their results in the table, indicating that the results are not significantly different from *Logistic*. If an algorithm had results larger than the base algorithm and the difference was significant, a little v would appear next to the results.

If we had to build a model immediately, we might choose *Logistic*, but we might also choose *NaiveBayes* or *SMO* as their results were not significantly different. Logistic regression is a good model to choose because it is simple, we understood and fast to train. We would probably not choose *IBk* or the decision tree, at least not their default configurations because we know *Logistic* can do better and that result is statistically significant.

20.6 Debugging Errors With Experiments

You can sometimes get errors when running your experiments. The log in the *Run* tab will report `there was 1 error` and no more information.

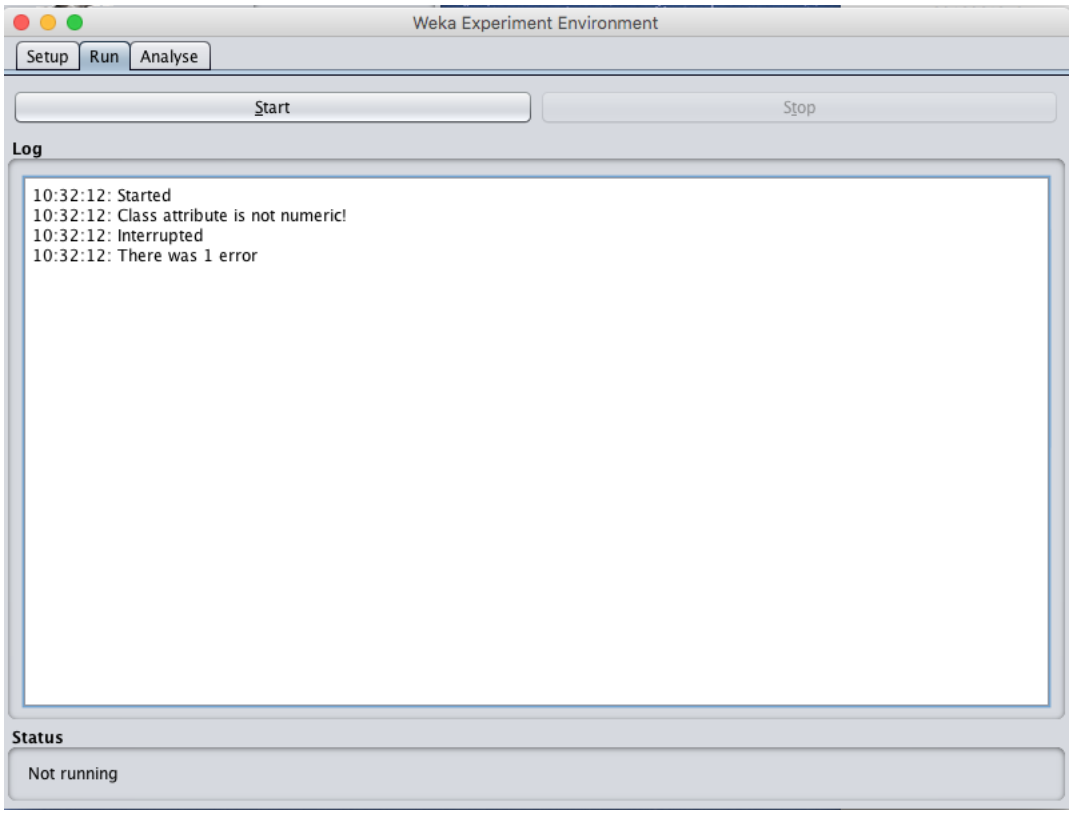


Figure 20.8: Weka Error With an Experiment.

You can easily find out what went wrong by reviewing the Weka log.

- In the *Weka GUI Chooser*, click the *Program* menu and *LogMenu*.

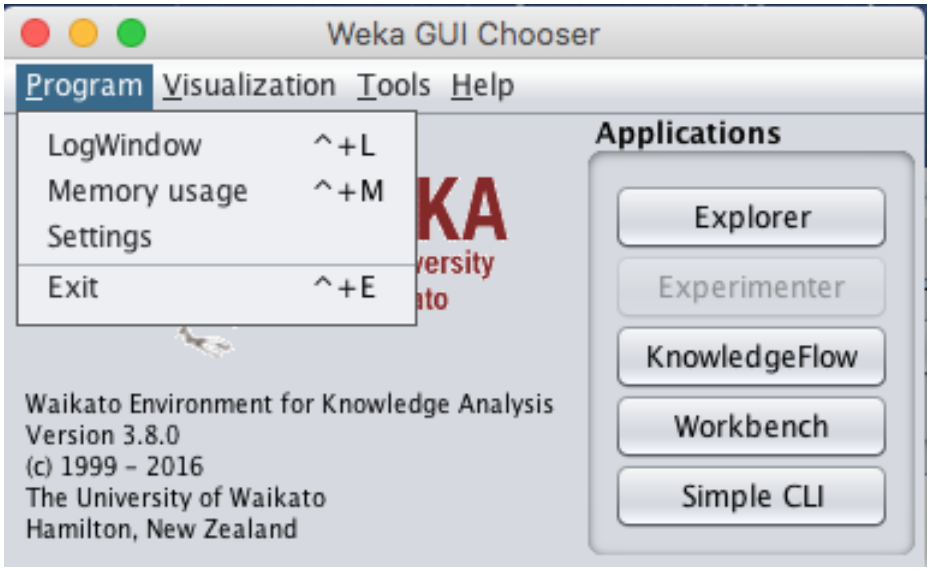


Figure 20.9: Weka Open Log.

This will open the *Weka Log*. Scroll through the log from top to bottom looking for the error that corresponded to your experiment. It may be the last error in the log (at the bottom).

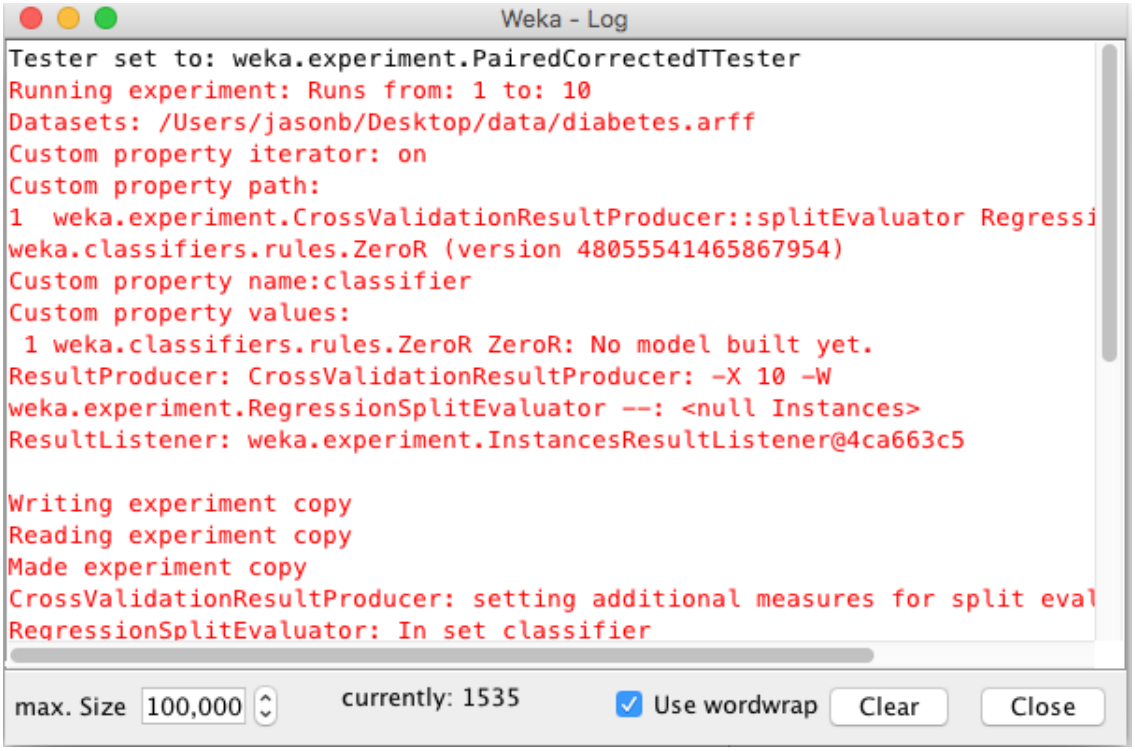


Figure 20.10: Weka Log Window.

Often, the experiment error will be caused by one of two reasons:

- You chose the wrong type of problem in the *Experiment Type* pane on the *Setup* tab of the *Weka Experiment Environment*. For example your problem may be a *Regression* type problem and you chose *Classification*.

- You added the wrong type of algorithm to the experiment in the *Algorithms* pane on the *Setup* tab of the *Weka Experiment Environment*. For example, you may have added an algorithm that only supports regression (like LinearRegression) to a classification type problem.

When reviewing the errors in the Weka log, keep an eye out for messages that suggests one of the above types of problems.

20.7 Summary

In this lesson you discovered how to find good and even the best machine learning algorithms for your data problem. Specifically you learned:

- A process to spot-check a suite of different algorithms on a problem in order to find a subset of algorithms that do well for further investigation.
- How to design and run an experiment to compare the performance of machine learning algorithms on a dataset.
- How to interpret the results of a machine learning experiment in order to answer questions about your problem.

20.7.1 Next

Machine learning algorithms have a variety of parameters to configure. In the next lesson you will discover how you can design controlled experiments to tune the parameters of machine learning algorithms for performance on your datasets.

Chapter 21

How to Tune the Parameters of Machine Learning Algorithms

You can get the most from a machine learning algorithm by tuning its parameters, called hyperparameters. In this lesson you will discover how to tune machine learning algorithms with controlled experiments in Weka. After reading this lesson you will know:

- The importance of improving the performance of machine learning models by algorithm tuning.
- How to design a controlled experiment to tune the hyperparameters of a machine learning algorithm.
- How to interpret the results from tuning an experiment using statistical significance.

Let's get started.

21.1 Improve Performance By Tuning

Machine learning algorithms can be configured to elicit different behavior. This is useful because it allows their behavior to be adapted to the specifics of your machine learning problem. This is also a difficulty because you must choose how to configure an algorithm without knowing beforehand which configuration is best for your problem. Because of this, you must tune the configuration parameters of each machine learning algorithm to your problem. This is often called algorithm tuning or algorithm hyperparameter optimization. It is an empirical process of trial and error.

You can tinker with an algorithm in order to discover the combination of parameters that result in the best performance for your problem, but this can be difficult because you must record all of the results and compare them manually. A more robust approach is to design a controlled experiment to evaluate a number of predefined algorithm configurations and provide tools to review compare the results. The *Weka Experiment Environment* provides an interface that allows you to design, execute and analyze the results from these types of experiments.

21.2 Algorithm Tuning Experiment Overview

In this tutorial we are going to define an experiment to investigate the parameters of the k-Nearest Neighbors (KNN) machine learning algorithm. We are going to investigate two parameters of the KNN algorithm:

1. The value of k, which is the number of neighbors to query in order to make a prediction.
2. The distance metric, which is the way that neighbors are determined in query in order to make predictions.

We are going to use the Pima Indians Onset of Diabetes dataset. You can learn more about this dataset in Section 8.2.1. This tutorial is divided into 3 parts:

1. Design The Experiment
2. Run The Experiment
3. Review Experiment Results

You can use this experimental design as the basis for tuning the parameters of different machine learning algorithms on your own datasets.

21.3 Design The Experiment

In this section we are going to define the experiment. We will select the dataset that will be used to evaluate the different algorithm configurations. We will also add multiple instances of the KNN algorithm (called *IBk* in Weka) each with a different algorithm configuration.

- 1. Open the *Weka GUI Chooser*.
- 2. Click the *Experimenter* button to open the *Weka Experiment Environment* interface.

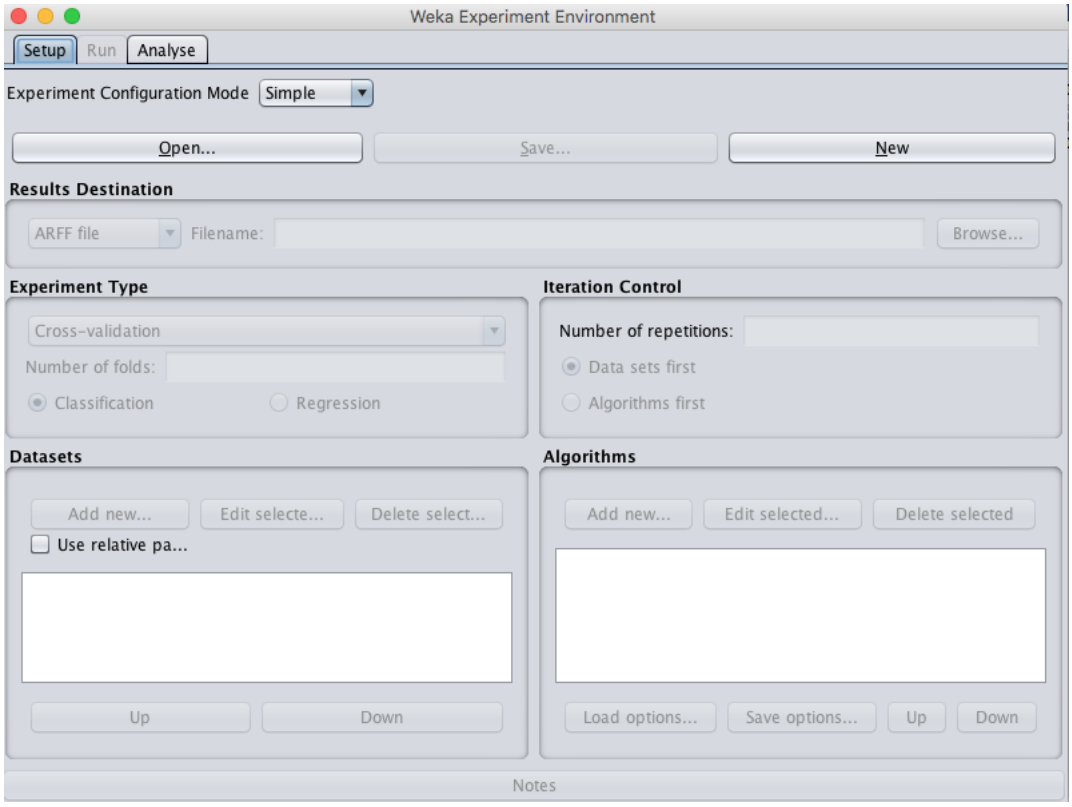


Figure 21.1: Weka Experiment Environment.

- 3. On the *Setup* tab, click the *New* button to start a new experiment.
- 4. In the *Dataset* pane, click the *Add new...* button and choose `data/diabetes.arff`.
- 5. In the *Algorithms* pane, click the *Add new...* button, click the *Choose* button and select the *IBk* algorithm under the *lazy* group. Click the *OK* button to add it.

This has added KNN with *k=1* and *distance=Euclidean*. Repeat this process and *IBk* with configuration parameter configurations listed below. The k parameter can be specified in the K parameter on the algorithm configuration.

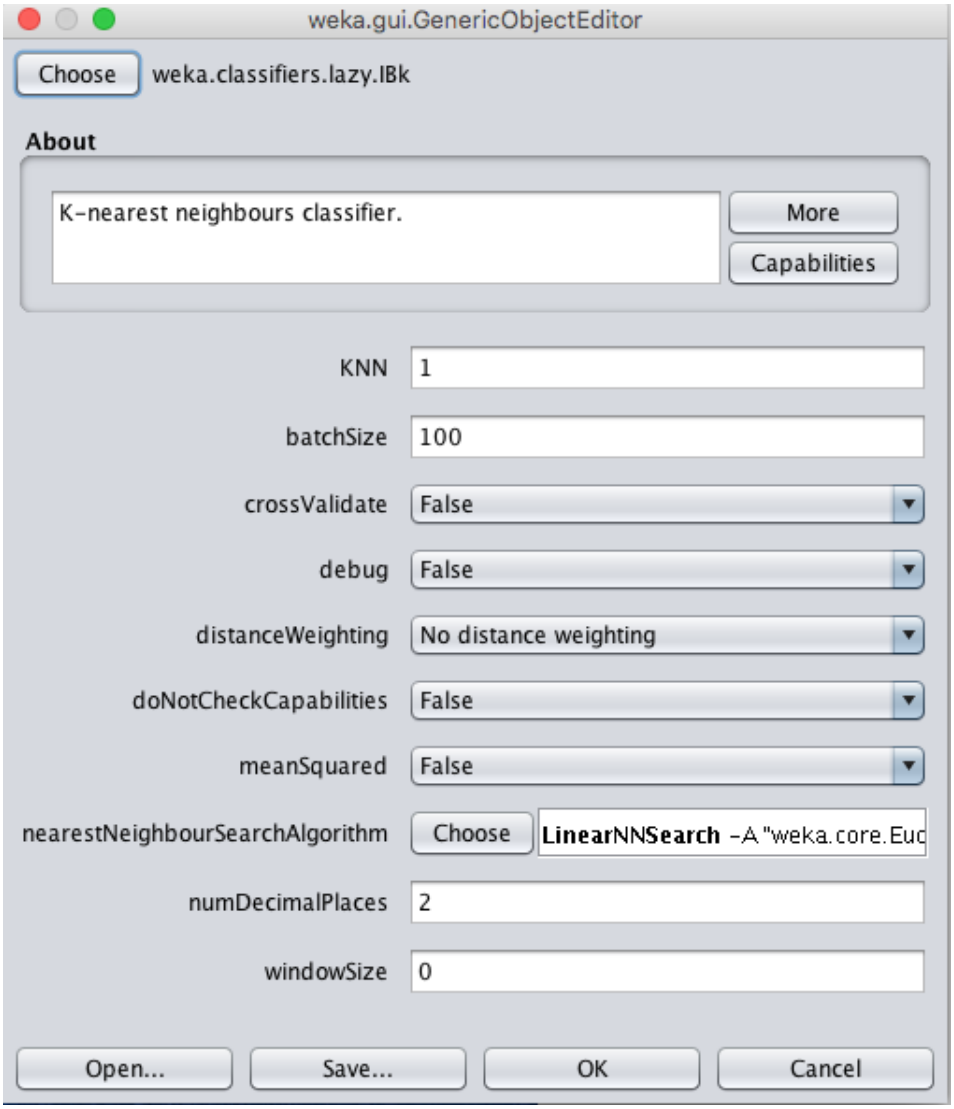


Figure 21.2: Weka Experiment Environment.

The distance measure can be changed by clicking the name of the technique for the *nearestNeighbourSearchAlgorithm* parameter to open the configuration properties for the search method, then clicking the *Choose* button for the *distanceFunction* parameter.

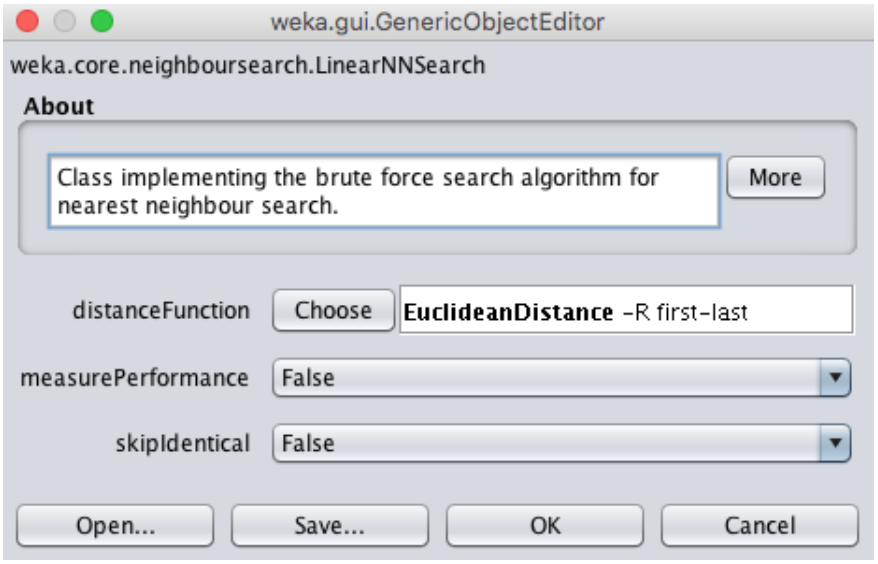


Figure 21.3: Weka Choose the Distance Function for the k-Nearest Neighbors Algorithm.

Note that you can edit the configuration for an algorithm added to the experiment by clicking the *Edit selected...* button in the *Algorithms* pane.

- *IBk*, *k=3*, *distanceFunction=Euclidean*
- *IBk*, *k=7*, *distanceFunction=Euclidean*
- *IBk*, *k=1*, *distanceFunction=Manhattan*
- *IBk*, *k=3*, *distanceFunction=Manhattan*
- *IBk*, *k=7*, *distanceFunction=Manhattan*

Euclidean distance is a distance measure that is best used when all of the input attributes in the dataset have the same scale. Manhattan distance is a measure that is best used when the input attributes have varying scales, such as in the case of the Pima Indians onset of diabetes dataset. We would expect that KNN with a Manhattan distance measure would achieve a better overall score from this experiment. The experiment uses 10-fold cross-validation (the default). Each configuration will be evaluated on the dataset 10 times (10 runs of 10-fold cross-validation) with different random number seeds. This will result is 10 slightly different results for each evaluated algorithm configuration, a small population that we can interpret using statistical methods later.

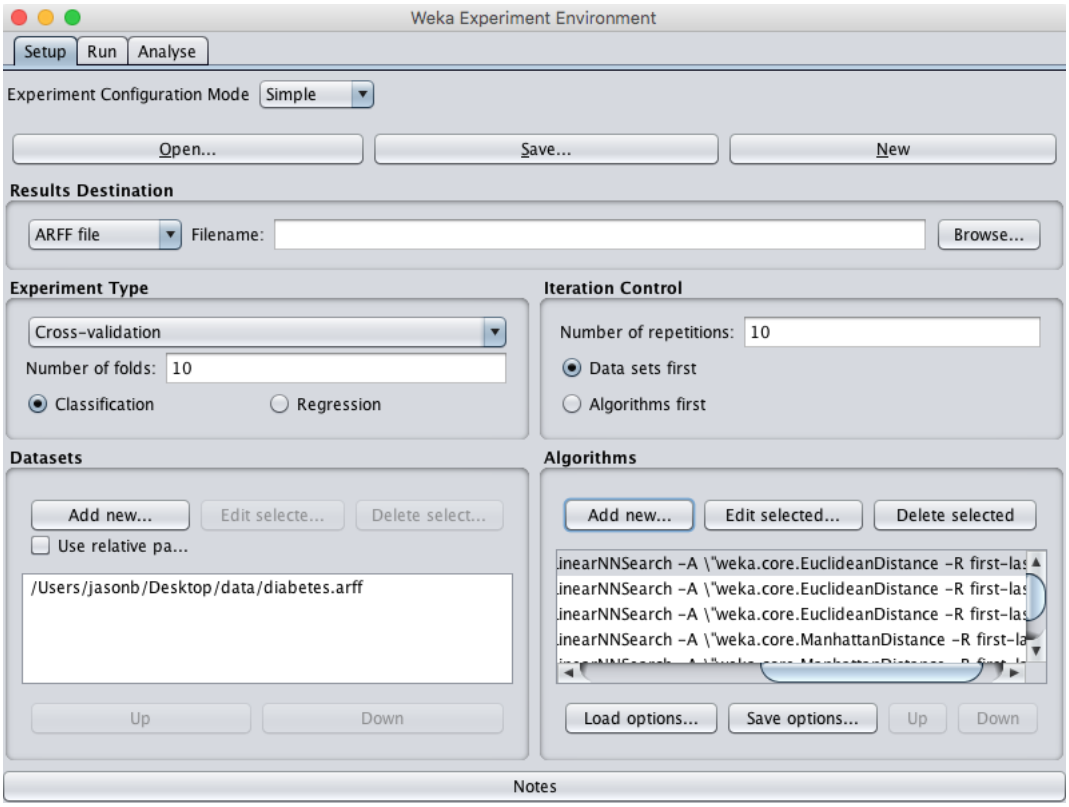


Figure 21.4: Weka Configured Algorithm Tuning Experiment.

21.4 Run The Experiment

Now it is time to run the experiment.

- 1. Click the *Run* tab.

There are few options. All you can do is start an experiment or stop a running experiment.

- 2. Click the *Start* button and run the experiment. It should complete in a few seconds. This is because the dataset is small.

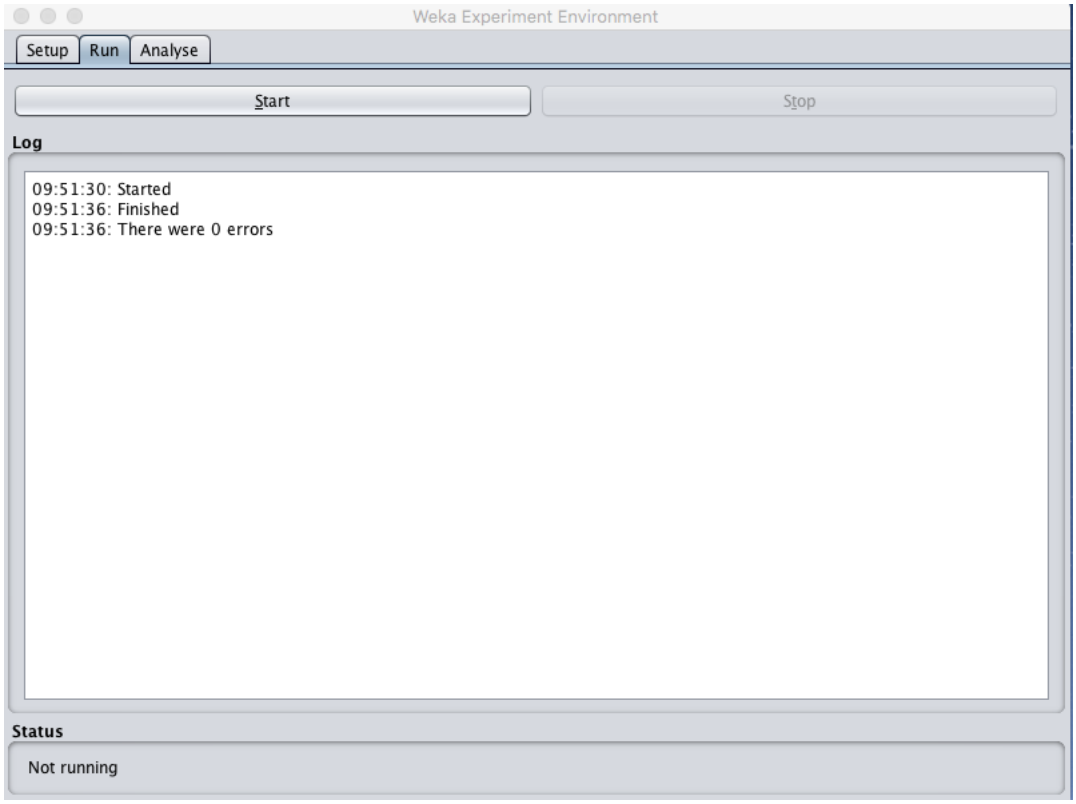


Figure 21.5: Weka Run the Algorithm Tuning Experiment.

21.5 Review Experiment Results

Load the results from the experiment we just executed by clicking the *Experiment* button in the *Source* pane.

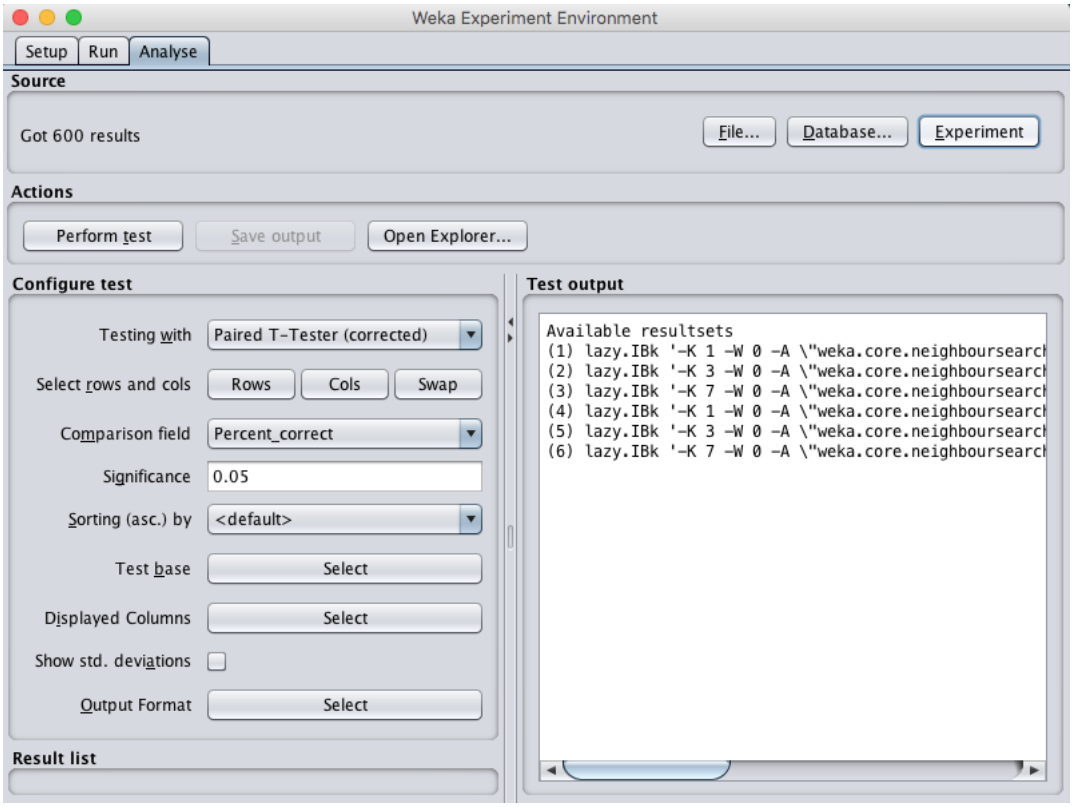


Figure 21.6: Weka Load the Algorithm Tuning Experiment Results.

You will see that 600 results were loaded. This is because we had 6 algorithm configurations that were each evaluated 100 times, or 10-fold cross-validation multiplied by 10 repeats. We are going to compare each algorithm configuration based on the percentage correct using pairwise statistical significance tests. All of the default configurations. The base for comparison is the first algorithm in the list, *IBk* with *k=1* and *distanceMeasure=Euclidean*, the first algorithm added to the experiment, also the default selection.

- Click the *Perform test* button in the *Actions* pane.

You will see a table of results like the one listed below:

```
Dataset                 (1) lazy.IBk | (2) lazy. (3) lazy. (4) lazy. (5) lazy. (6) lazy.
----------------------------------------------------------------------------------------
pima_diabetes          (100)   70.62 | 73.86 v  74.45 v  69.68    71.90    73.25
----------------------------------------------------------------------------------------
                             (v/ /*) | (1/0/0)  (1/0/0)  (0/1/0)  (0/1/0)  (0/1/0)


Key:
(1) lazy.IBk '-K 1 -W 0 -A \"weka.core.neighboursearch.LinearNNSearch -A
    \\\"weka.core.EuclideanDistance -R first-last\\\"\"' -3080186098777067172
(2) lazy.IBk '-K 3 -W 0 -A \"weka.core.neighboursearch.LinearNNSearch -A
    \\\"weka.core.EuclideanDistance -R first-last\\\"\"' -3080186098777067172
(3) lazy.IBk '-K 7 -W 0 -A \"weka.core.neighboursearch.LinearNNSearch -A
    \\\"weka.core.EuclideanDistance -R first-last\\\"\"' -3080186098777067172
(4) lazy.IBk '-K 1 -W 0 -A \"weka.core.neighboursearch.LinearNNSearch -A
    \\\"weka.core.ManhattanDistance -R first-last\\\"\"' -3080186098777067172
```

```
(5) lazy.IBk '-K 3 -W 0 -A \"weka.core.neighboursearch.LinearNNSearch -A
    \\\"weka.core.ManhattanDistance -R first-last\\\"\"' -3080186098777067172
(6) lazy.IBk '-K 7 -W 0 -A \"weka.core.neighboursearch.LinearNNSearch -A
    \\\"weka.core.ManhattanDistance -R first-last\\\"\"' -3080186098777067172
```

Listing 21.1: KNN Algorithm Tuning Experiment Results.

The results are fascinating. We can see that in general, the Euclidean distance measure achieved better results than Manhattan but the difference between the base algorithm (*k=1* with *distanceMeasure=Euclidean*) and all three Manhattan distance configurations was not significant (no * next to the result values). We can also see that that using Euclidean distance with *k=3* or *k=7* does result in an accuracy that is better than the base algorithm and that this difference is statistically significant (a little v next to the results). Is algorithm (3) with *k=7* and *distanceMeasure=Euclidean* significantly better than the other results? We can perform this comparison easily by selecting this algorithm as the basis for comparison:

1. Click the *Select* button next to *Test base*.
2. Click the third algorithm down with *k=7* and *distanceMeasure=Euclidean*.
3. Click the *Select* button to choose it as the base.
4. Click the *Perform Test* button to create a report of the results.

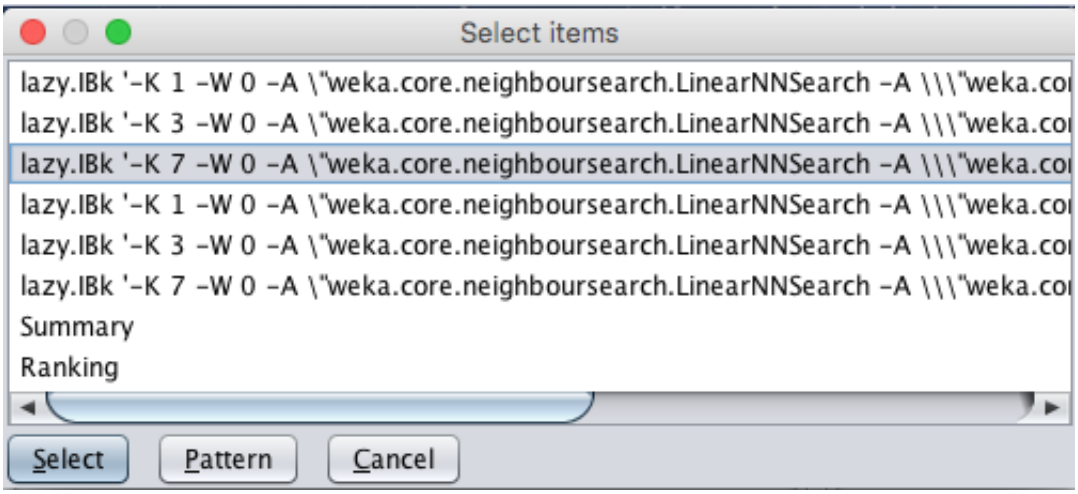


Figure 21.7: Weka Choose New Test Base For Analysis of Results.

You will see a table of results like the one listed below.

```
Dataset                 (3) lazy.IBk | (1) lazy. (2) lazy. (4) lazy. (5) lazy. (6) lazy.
----------------------------------------------------------------------------------------
pima_diabetes          (100)  74.45 | 70.62 *  73.86    69.68 *  71.90    73.25
----------------------------------------------------------------------------------------
                            (v/ /*) | (0/0/1)  (0/1/0)  (0/0/1)  (0/1/0)  (0/1/0)


Key:
```

```
(1) lazy.IBk '-K 1 -W 0 -A \"weka.core.neighboursearch.LinearNNSearch -A
    \\\"weka.core.EuclideanDistance -R first-last\\\"\"' -3080186098777067172
(2) lazy.IBk '-K 3 -W 0 -A \"weka.core.neighboursearch.LinearNNSearch -A
    \\\"weka.core.EuclideanDistance -R first-last\\\"\"' -3080186098777067172
(3) lazy.IBk '-K 7 -W 0 -A \"weka.core.neighboursearch.LinearNNSearch -A
    \\\"weka.core.EuclideanDistance -R first-last\\\"\"' -3080186098777067172
(4) lazy.IBk '-K 1 -W 0 -A \"weka.core.neighboursearch.LinearNNSearch -A
    \\\"weka.core.ManhattanDistance -R first-last\\\"\"' -3080186098777067172
(5) lazy.IBk '-K 3 -W 0 -A \"weka.core.neighboursearch.LinearNNSearch -A
    \\\"weka.core.ManhattanDistance -R first-last\\\"\"' -3080186098777067172
(6) lazy.IBk '-K 7 -W 0 -A \"weka.core.neighboursearch.LinearNNSearch -A
    \\\"weka.core.ManhattanDistance -R first-last\\\"\"' -3080186098777067172
```

Listing 21.2: KNN Algorithm Tuning Experiment Results with new Test Base.

We can see that difference of *k=7* and Euclidean distance is statistically significant when compared with *k=1* with Euclidean or Manhattan distance, but not *k=3* or 7 regardless of the distance measure. This is helpful. It shows us that we could probably just as easily use Manhattan or Euclidean distance with *k=7* or *k=3* and achieve similar results.

The chosen base algorithm does have the highest accuracy at 74.45%. It is possible that this difference is statistically significant at a lower significance level, such as 20% (by setting *significance* to 0.20), but the variance would be so high (better 4 in 5 random instances) that the result might not be reliable enough. These results are useful in another way other than seeking out the *best*. We can perhaps see a trend of increasing accuracy with increasing values of *k*. We could design a follow-up experiment where the distanceMeasure is fixed with Euclidean and larger *k* values are considered.

21.6 Summary

In this lesson you discovered how to design controlled experiments to tune machine learning algorithm hyperparameters. Specifically you learned:

- About the need for algorithm parameters and the importance of tuning them empirically to your problem.
- How to design and execute a controlled experiment to tune the parameters of a machine learning algorithm in Weka.
- How to interpret the results of a controlled Experiment using statistical significance.

21.6.1 Next

After you discover the best performing algorithm for your problem, you need to finalize it. In the next lesson you will discover how you can train your final model, save it to file, then later load it and use it to make predictions on new data.

Chapter 22

How to Save Your Machine Learning Model and Make Predictions

After you have found a well performing machine learning model and tuned it, you must finalize your model so that you can use it to make predictions on new data. In this lesson you will discover how to finalize your machine learning model, save it to file and load it later in order to make predictions on new data. After reading this lesson you will know:

- How to train a final version of your machine learning model in Weka.
- How to save your finalized model to file.
- How to load your finalized model later and use it to make predictions on new data.

Let's get started.

22.1 Tutorial Overview

This tutorial is broken down into 4 parts:

1. Finalize Model where you will discover how to train a finalized version of your model.
2. Save Model where you will discover how to save a model to file.
3. Load Model where you will discover how to load a model from file.
4. Make Predictions where you will discover how to make predictions for new data.

The tutorial provides a template that you can use to finalize your own machine learning algorithms on your data problems. We are going to use the Pima Indians Onset of Diabetes dataset. You can learn more about this dataset in Section 8.2.1. We are going to finalize a logistic regression model on this dataset, both because it is a simple algorithm that is well understood and because it does very well on this problem.

22.2 Finalize a Machine Learning Model

Perhaps the most neglected task in a machine learning project is how to finalize your model. Once you have gone through all of the effort to prepare your data, compare algorithms and tune them on your problem, you actually need to create the final model that you intend to use to make new predictions. Finalizing a model involves training the model on the entire training dataset that you have available.

- 1. Open the *Weka GUI Chooser*.
- 2. Click the *Explorer* button to open the *Weka Explorer* interface.
- 3. Load the Pima Indians onset of diabetes dataset from the `data/diabetes.arff` file.

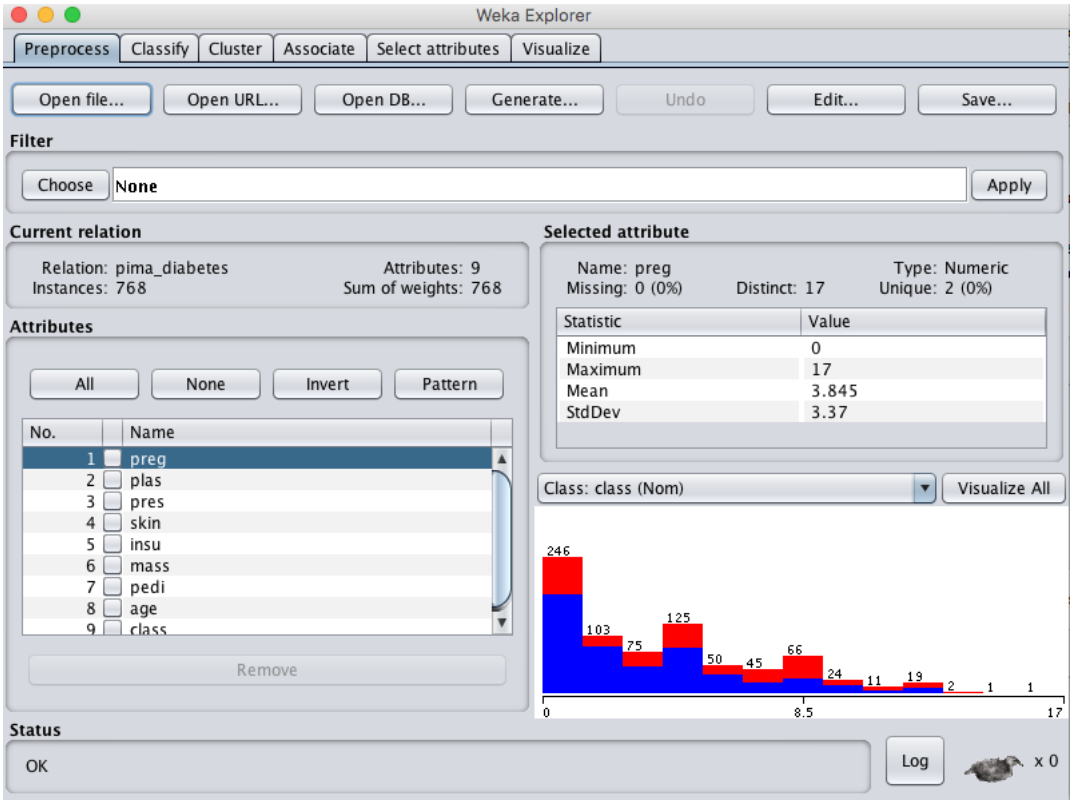


Figure 22.1: Weka Load Pima Indians Onset of Diabetes Dataset.

- 4. Click the *Classify* tab to open up the classifiers.
- 5. Click the *Choose* button and choose *Logistic* under the *functions* group.
- 6. Select *Use training set* under *Test options*.
- 7. Click the *Start* button.

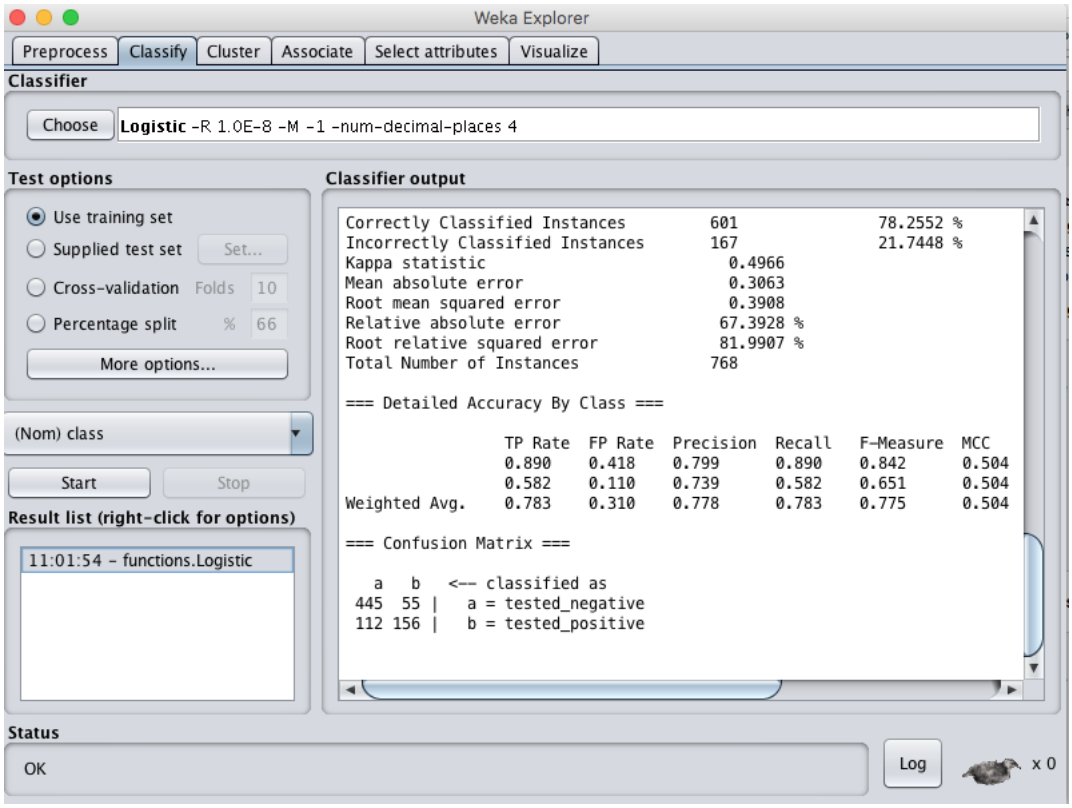


Figure 22.2: Weka Train Logistic Regression Model.

This will train the chosen Logistic regression algorithm on the entire loaded dataset. It will also evaluate the model on the entire dataset, but we are not interested in this evaluation. It is assumed that you have already estimated the performance of the model on unseen data using cross-validation as a part of selecting the algorithm you wish to finalize. It is this estimate you prepared previously that you can report when you need to inform others about the skill of your model. Now that we have finalized the model, we need to save it to file.

22.3 Save Finalized Model To File

Continuing on from the previous section, we need to save the finalized model to a file on your disk. This is so that we can load it up at a later time, or even on a different computer in the future and use it to make predictions. We won't need the training data in the future, just the model of that data. You can easily save a trained model to file in the *Weka Explorer* interface.

- 1. Right click on the result item for your model in the *Result list* on the *Classify* tab.
- 2. Click *Save model* from the right click menu.

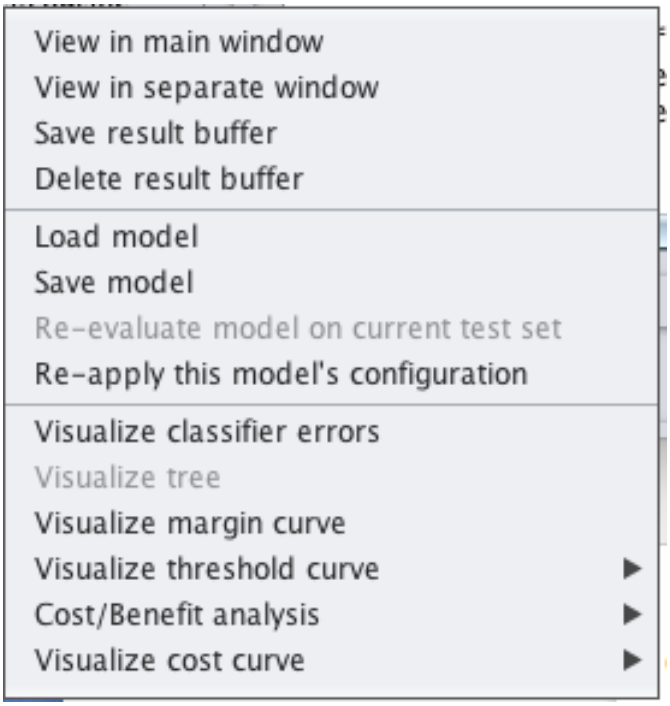


Figure 22.3: Weka Save Model to File.

- 3. Select a location and enter a filename such as *logistic*, click the *Save button*.

Your model is now saved to the file `logistic.model`. It is in a binary format (not text) that can be read again by the Weka platform. As such, it is a good idea to note down the version of Weka you used to create the model file, just in case you need the same version of Weka in the future to load the model and make predictions. Generally, this will not be a problem, but it is a good safety precaution. You can close the *Weka Explorer* now. The next step is to discover how to load up the saved model.

22.4 Load a Finalized Model

You can load saved Weka models from file. The *Weka Explorer* interface makes this easy.

- 1. Open the *Weka GUI Chooser*.
- 2. Click the *Explorer* button to open the *Weka Explorer* interface.
- 3. Load any old dataset, it does not matter. We will not be using it, we just need to load a dataset to get access to the *Classify* tab. If you are unsure, load the `data/diabetes.arff` file again.
- 4. Click the *Classify* tab to open up the classifiers.
- 5. Right click on the *Result list* and click *Load model*, select the model saved in the previous section *logistic.model*.

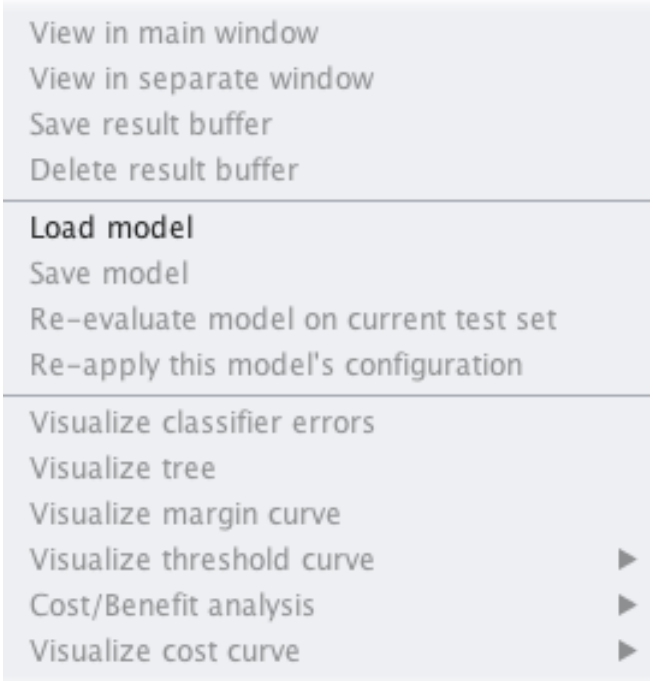


Figure 22.4: Weka Load Model From File.

The model will now be loaded into the *Weka Explorer*. We can now use the loaded model to make predictions for new data.

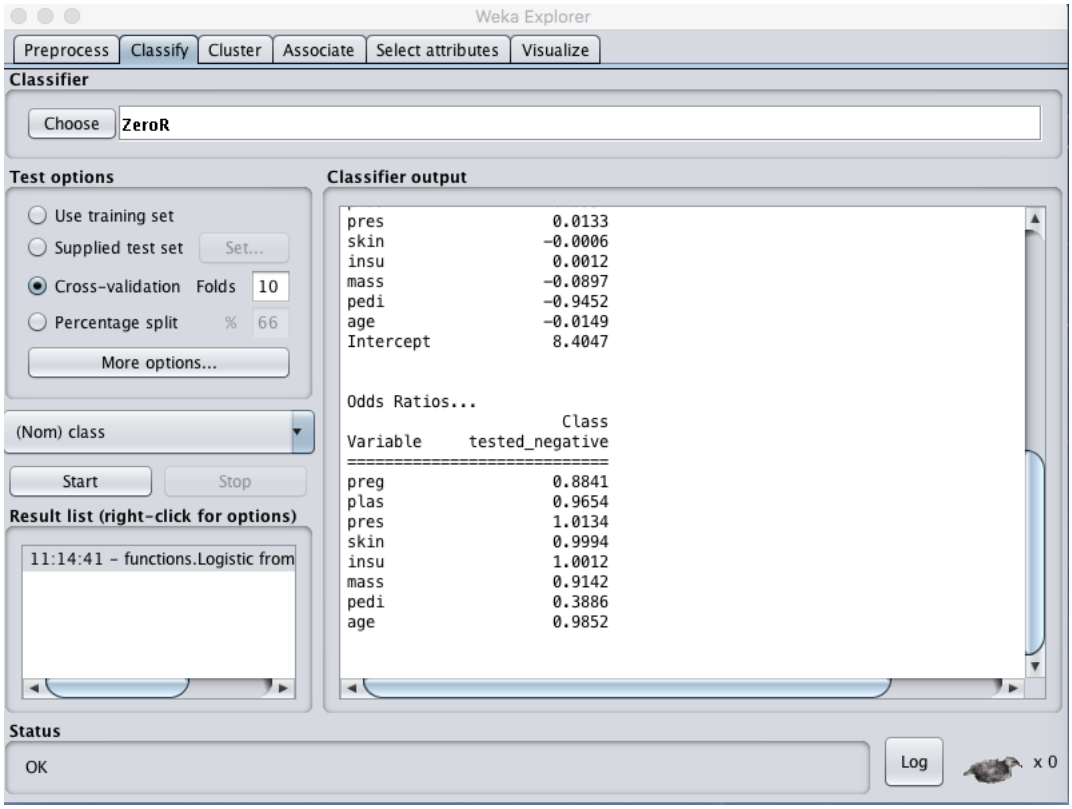


Figure 22.5: Weka Model Loaded From File Ready For Use.

22.5 Make Predictions on New Data

We can now make predictions on new data. First, let's create some pretend new data. Make a copy of the file *data/diabetes.arff* and save it as `data/diabetes-new-data.arff`.

- Open the file in a text editor.
- Find the start of the actual data in the file with the `@data` on line 95.
- We only want to keep 5 records. Move down 5 lines, then delete all the remaining lines of the file.

The class value (output variable) that we want to predict is on the end of each line. Delete each of the 5 output variables and replace them with question mark symbols (`?`).

```
85  @relation pima_diabetes
86  @attribute 'preg' numeric
87  @attribute 'plas' numeric
88  @attribute 'pres' numeric
89  @attribute 'skin' numeric
90  @attribute 'insu' numeric
91  @attribute 'mass' numeric
92  @attribute 'pedi' numeric
93  @attribute 'age' numeric
94  @attribute 'class' { tested_negative, tested_positive}
95  @data
96  6,148,72,35,0,33.6,0.627,50,?
97  1,85,66,29,0,26.6,0.351,31,?
98  8,183,64,0,0,23.3,0.672,32,?
99  1,89,66,23,94,28.1,0.167,21,?
100 0,137,40,35,168,43.1,2.288,33,?
101
```

Figure 22.6: Weka Dataset For Making New Predictions.

We now have *unseen* data with no known output for which we would like to make predictions. Continue on from the previous part of the tutorial where we already have the model loaded.

- 1. On the *Classify* tab, select the *Supplied test set* option in the *Test options* pane.

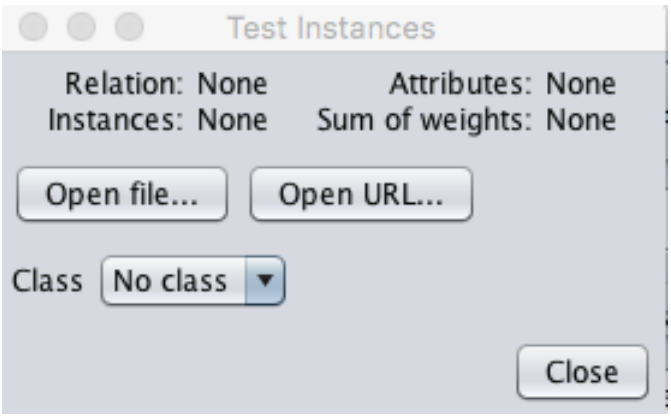


Figure 22.7: Weka Select New Dataset On Which To Make New Predictions.

- 2. Click the *Set* button, click the *Open file* button on the options window and select the mock new dataset we just created with the name *diabetes-new-data.arff*. Click *Close* on the window.
- 3. Click the *More options...* button to bring up options for evaluating the classifier.
- 4. Uncheck the information we are not interested in, specifically:
- *Output model*
- *Output per-class stats*
- *Output confusion matrix*
- *Store predictions for visualization*

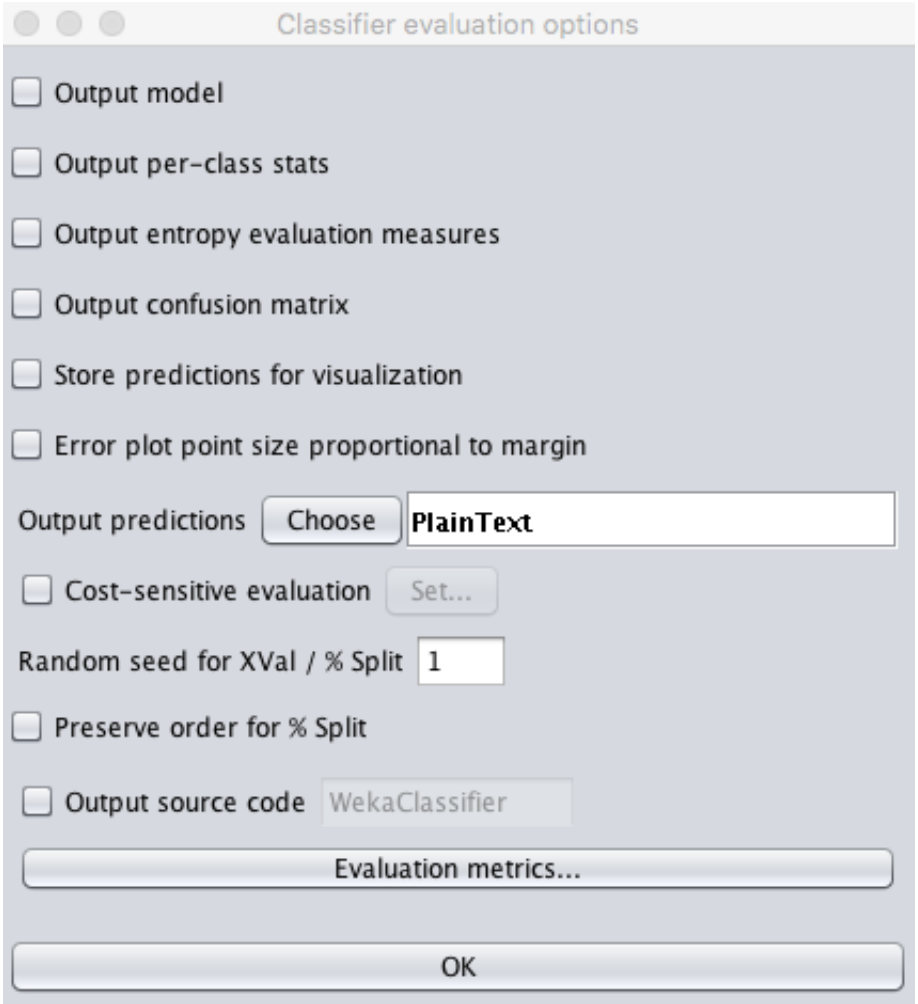


Figure 22.8: Weka Customized Test Options For Making Predictions.

- 5. For the *Output predictions* option click the *Choose* button and select *PlainText*.

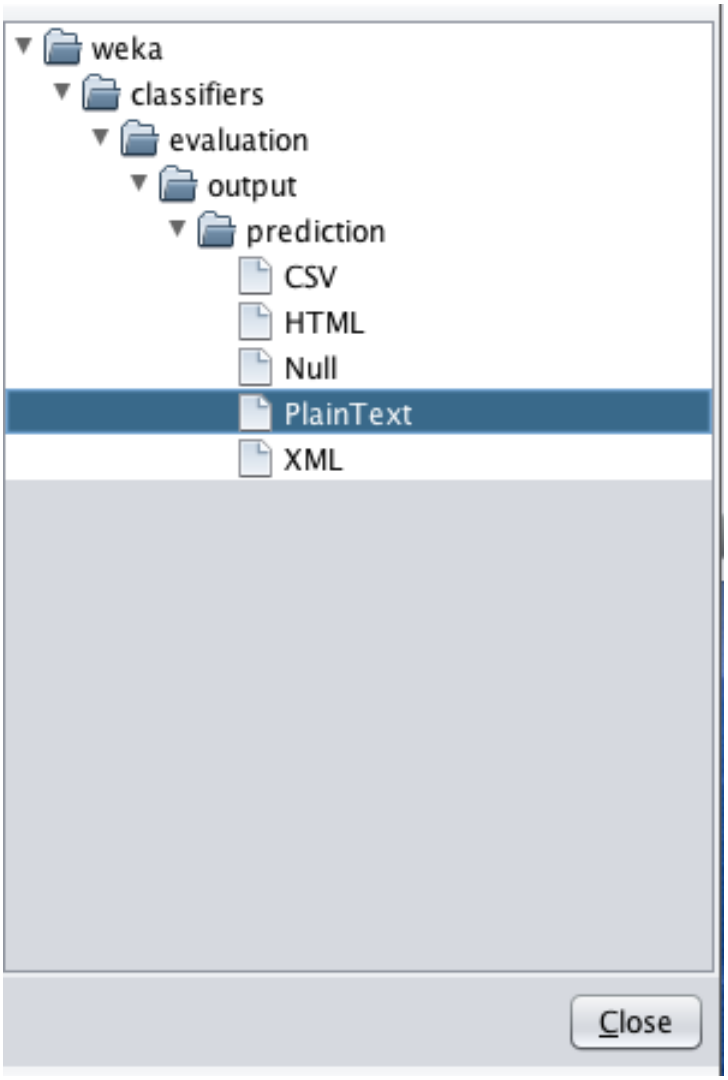


Figure 22.9: Weka Output Predictions in Plain Text Format.

- 6. Click the *OK* button to confirm the Classifier evaluation options.
- 7. Right click on the list item for your loaded model in the *Results list* pane.
- 8. Select *Re-evaluate model on current test set*.

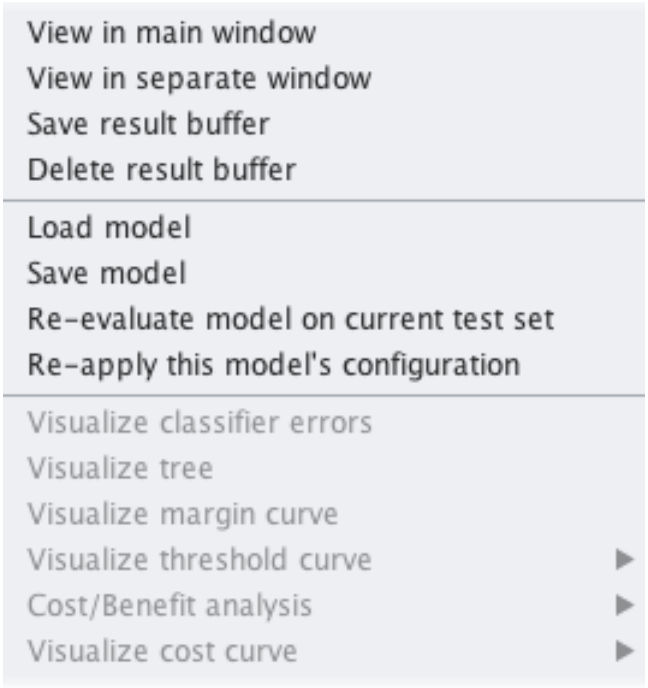


Figure 22.10: Weka Revaluate Loaded Model On Test Data And Make Predictions.

The predictions for each test instance are then listed in the *Classifier Output* pane. Specifically the middle column of the results with predictions like *tested_positive* and *tested_negative*. You could choose another output format for the predictions, such as CSV, that you could later load into a spreadsheet like Excel. For example, below is an example of the same predictions in CSV format.

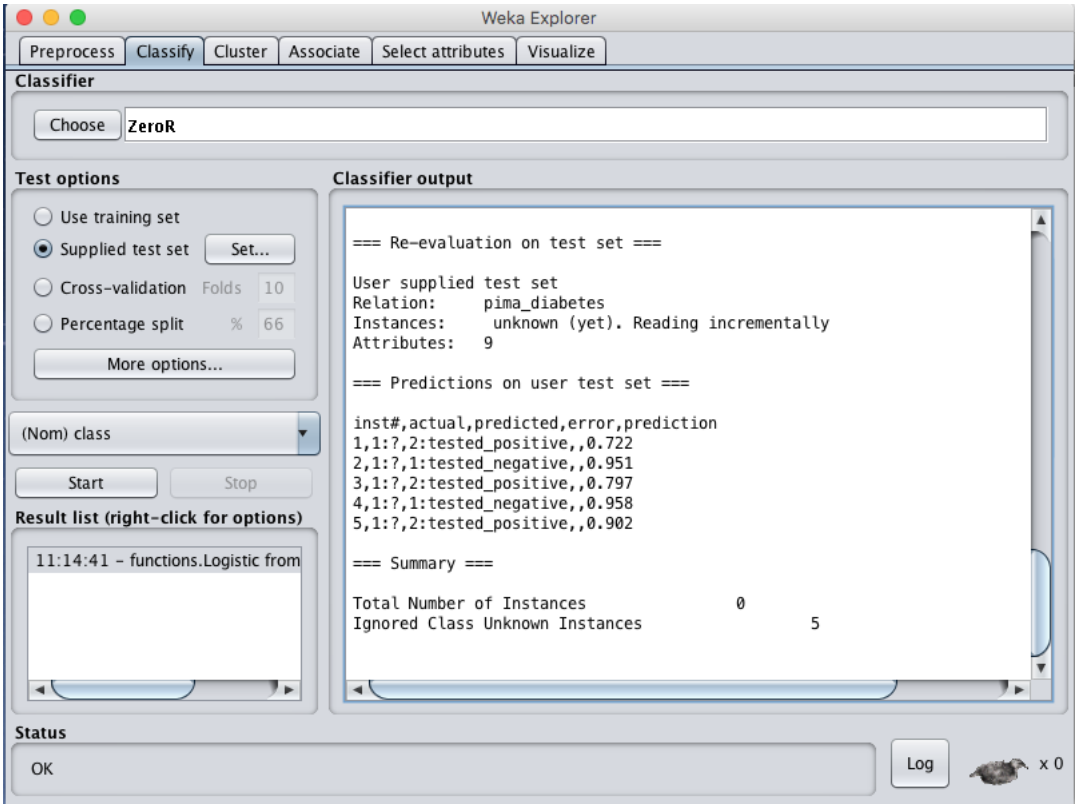


Figure 22.11: Weka Predictions Made on New Data By a Loaded Model.

22.6 Summary

In this lesson you discovered how to finalize your model and make predictions for new unseen data. You can see how you can use this process to make predictions on new data yourself. Specifically, you learned:

- How to train a final instance of your machine learning model.
- How to save a finalized model to file for later use.
- How to load a model from file and use it to make predictions on new data.

22.6.1 Next

This was the last lesson and concludes Part II of this book. Next in Part III you will take on your first end-to-end machine learning project on a multiclass classification problem.

Part III

Projects

Chapter 23

How To Work Through a Multiclass Classification Project

The Weka machine learning workbench is so easy to use that working through a machine learning project can be a lot of fun. In this section you will complete your first machine learning project using Weka, end-to-end. This gentle introduction to working through a project will tie together the key steps you need to complete when working through machine learning project in Weka. After completing this project, you will know:

- How to analyze a dataset and hypothesize data preparation and modeling algorithms that could be used.
- How to spot-check a suite of standard machine learning algorithms on a problem
- How to present final results.

Let's get started.

23.1 Tutorial Overview

This tutorial will gently walk you through the key steps required to complete a machine learning project. We will work through the following process:

- Load the dataset.
- Analyze the dataset.
- Evaluate algorithms.
- Present results.

You can use this as a template for the minimum steps in the process to work through your own machine learning project using Weka.

23.2 Load Dataset

In this tutorial, we will use the Iris Flowers Classification dataset. Each instance in the iris dataset describes measurements of iris flowers and the task is to predict which species of 3 iris flower the observation belongs. You can learn more about this dataset in Section 8.3.1.

- 1. Open the *Weka GUI Chooser*.

Figure 23.1: Weka GUI Chooser.

- 2. Click the *Explorer* button to open the *Weka Explorer*.
- 3. Click the *Open file...* button, navigate to the `data/` directory and select `iris.arff`. Click the *Open button*.

The dataset is now loaded into Weka.

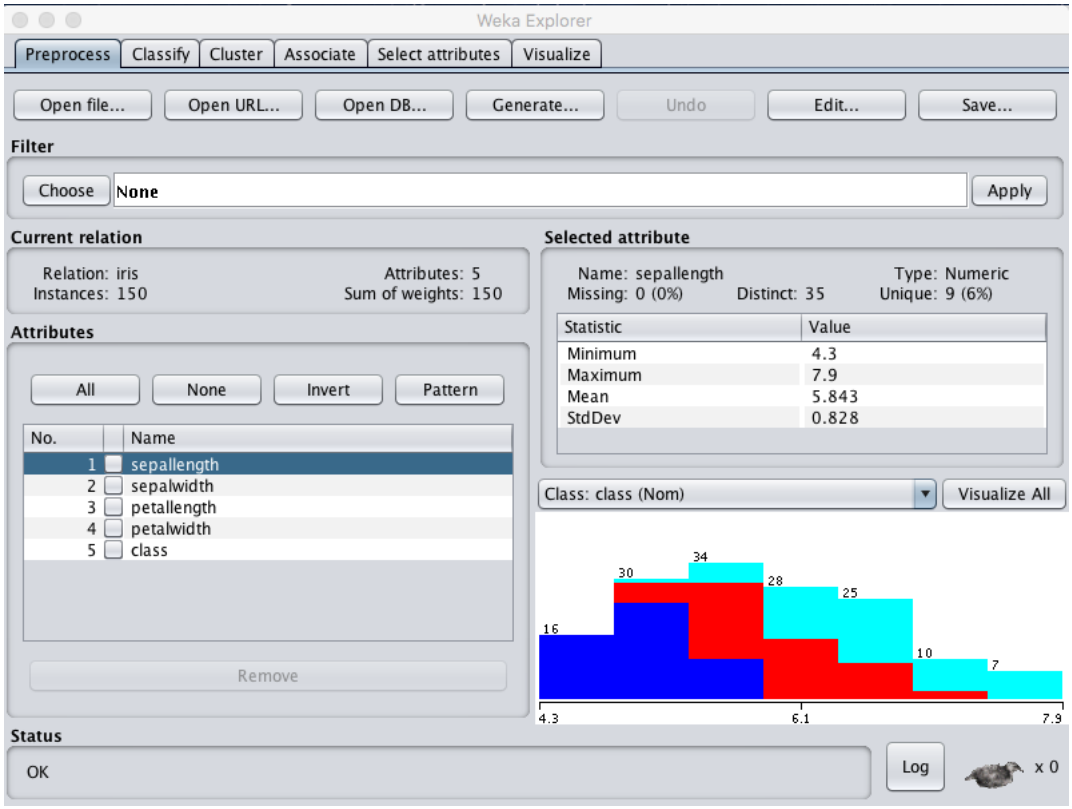


Figure 23.2: Weka Load Iris Flowers Dataset.

23.3 Analyze the Dataset

It is important to review your data before you start modeling. Reviewing the distribution of each attribute and the interactions between attributes may shed light on specific data transforms and specific modeling techniques that we could use.

23.3.1 Summary Statistics

Review the details about the dataset in the *Current relation* pane. We can notice a few things:

- The dataset is called iris.
- There are 150 instances. If we use 10-fold cross-validation later to evaluate the algorithms, then each fold will be comprised of 15 instances, which is quite small. We may want to think about using 5-folds of 30 instances instead.
- There are 5 attributes, 4 inputs and 1 output variable.

There are a small number of attributes and we could investigate further using feature selection methods.

- Click on each attribute in the *Attributes* pane and review the summary statistics in the *Selected attribute* pane.

We can notice a few facts about our data:

- There are no missing values for any of the attributes.
- All inputs are numeric and have values in the same range between about 0 and about 8.
- The last attribute is the output variable called class, it is nominal and has three values.
- The classes are balanced, meaning that there is an equal number of instances in each class. If they were not balanced we may want to think about balancing them.

We may see some benefit from either normalizing or standardizing the data.

23.3.2 Attribute Distributions

- Click the *Visualize All* button and let's review the graphical distribution of each attribute.

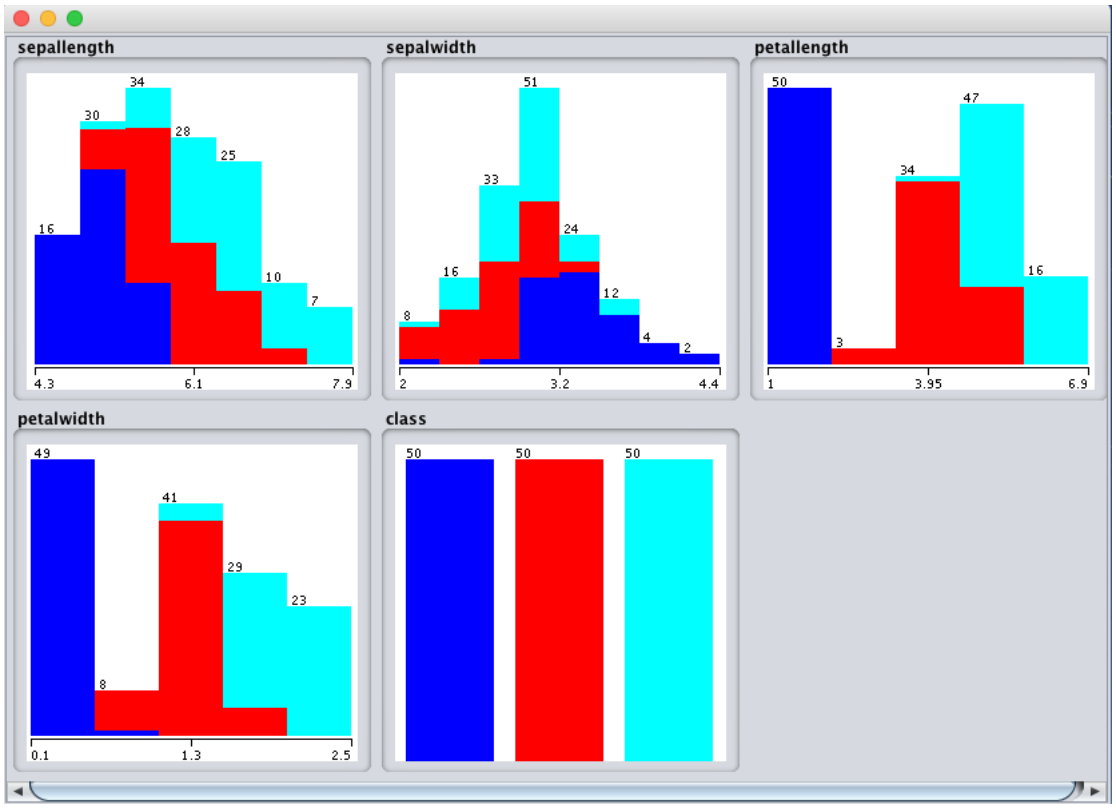


Figure 23.3: Weka Univariate Attribute Distribution Plots.

We can notice a few things about the shape of the data:

- We can see overlap but differing distributions for each of the class values on each of the attributes. This is a good sign as we can probably separate the classes.
- It looks like *sepalwidth* has a Gaussian-like distribution. If we had a lot more data, perhaps it would be even more Gaussian.

- It looks like the other 3 input attributes have nearly-Gaussian distributions with a skew or a large number of observations at the low end of the distribution. Again, it makes me think that the data may be Gaussian if we had an order of magnitude more examples.
- We also get a visual indication that the classes are balanced.

23.3.3 Attribute Interactions

- Click the *Visualize* tab and let's review some interactions between the attributes.
- Increase the window size so all plots are visible.
- Increase the *PointSize* to 3 to make the dots easier to see.
- Click the *Update* button to apply the changes.

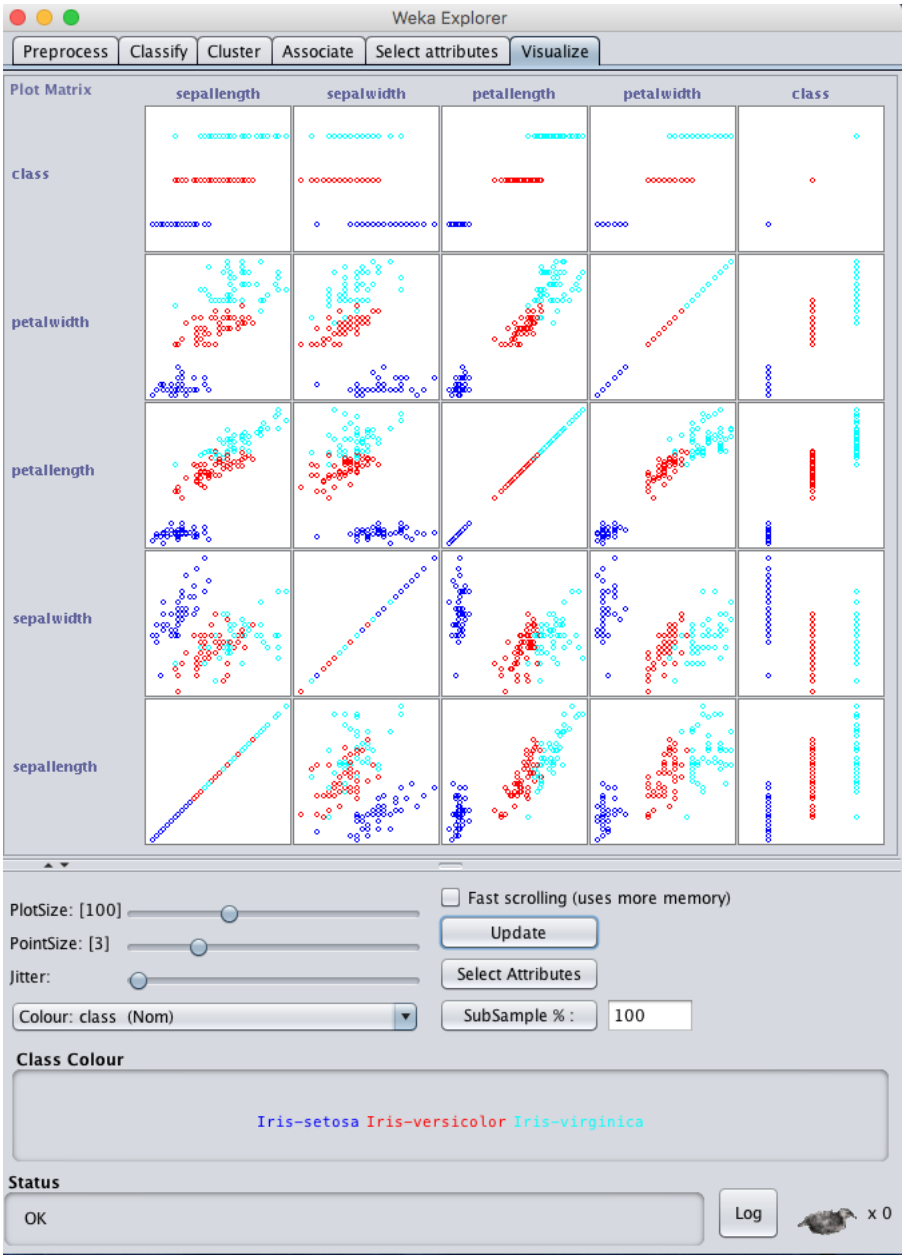


Figure 23.4: Weka Attribute Scatter Plot Matrix.

Looking across the graphs for the input variables, we can see good separation between the classes on the scatter plots. For example, *petalwidth* versus *sepallength* and *petalwidth* versus *sepalwidth* are good examples. This suggest that linear methods and maybe decision trees and instance-based methods may do well on this problem. It also suggest that we probably do not need to spend too much time tuning or using advanced modeling techniques and ensembles. It may be a straightforward modeling problem.

23.4 Evaluate Algorithms

Let's design a small experiment to evaluate a suite of standard classification algorithms on the problem.

- 1. Close the *Weka Explorer*.
- 2. Click the *Experimenter* button on the *Weka GUI Chooser* to launch the *Weka Experiment Environment*.

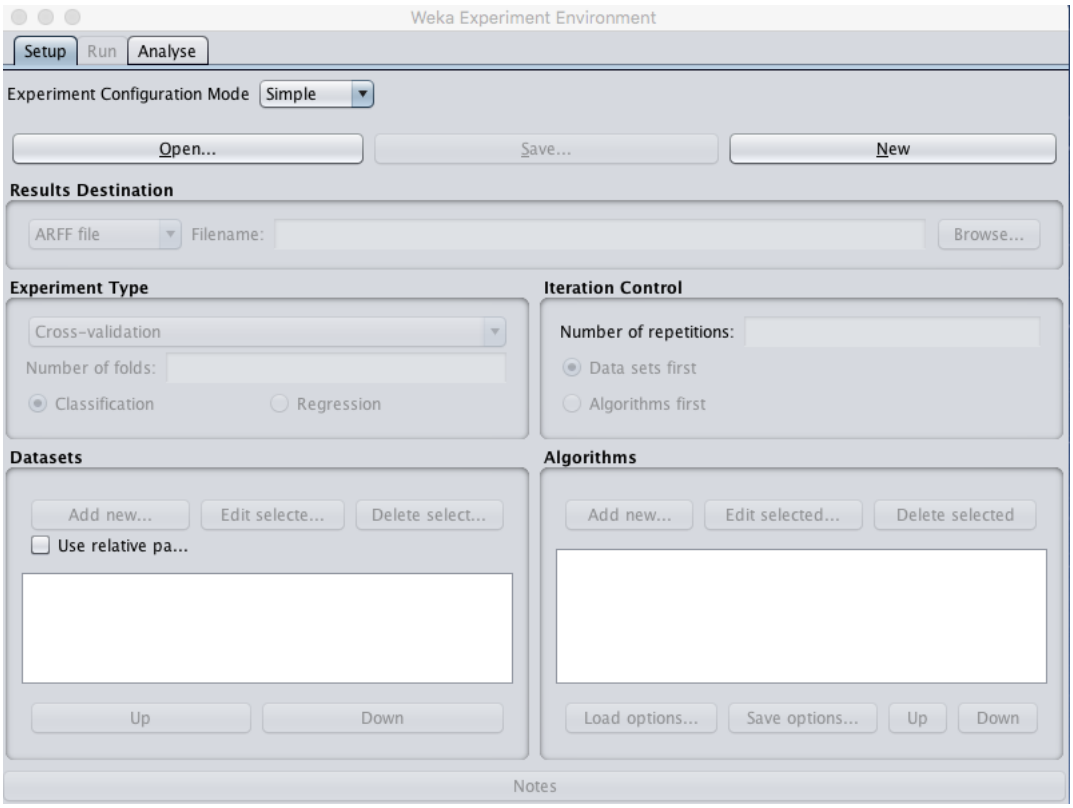


Figure 23.5: Weka Experiment Environment.

- 3. Click *New* to start a new experiment.
- 4. In the *Experiment Type* pane change the *Number of folds* from *10* to *5*.
- 5. In the *Datasets* pane click *Add new...* and select `data/iris.arff` in your Weka installation directory.
- 6. In the *Algorithms* pane click *Add new...* and add the following 8 multiclass classification algorithms:
  - *rules.ZeroR*
  - *bayes.NaiveBayes*
  - *functions.Logistic*
  - *functions.SMO*
  - *lazy.IBk*
  - *rules.PART*
  - *trees.REPTree*

- *trees.J48*

- 7. Select *IBk* in the list of algorithms and click the *Edit selected...* button.
- 8. Change *KNN* from *1* to *3* and click the *OK* button to save the settings.

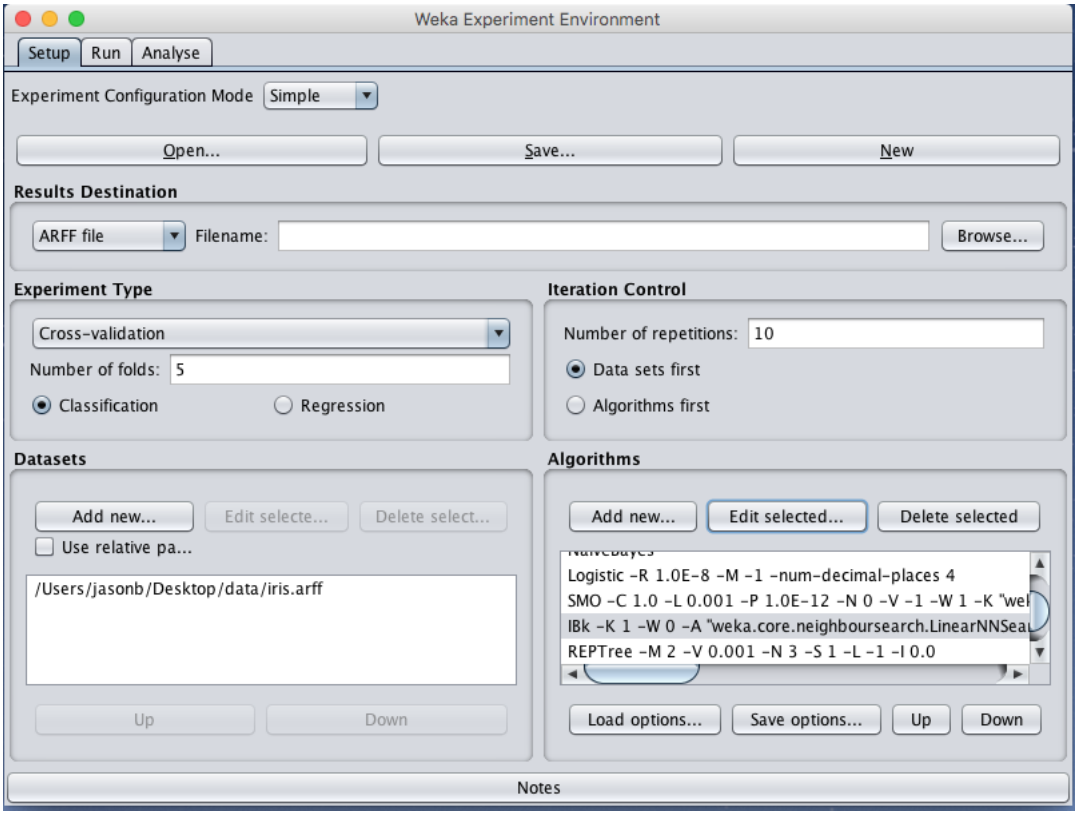


Figure 23.6: Weka Designed Algorithm Comparison Experiment.

- 9. Click on *Run* tab and click the *Start* button to run the experiment. The experiment should complete in just a few seconds.

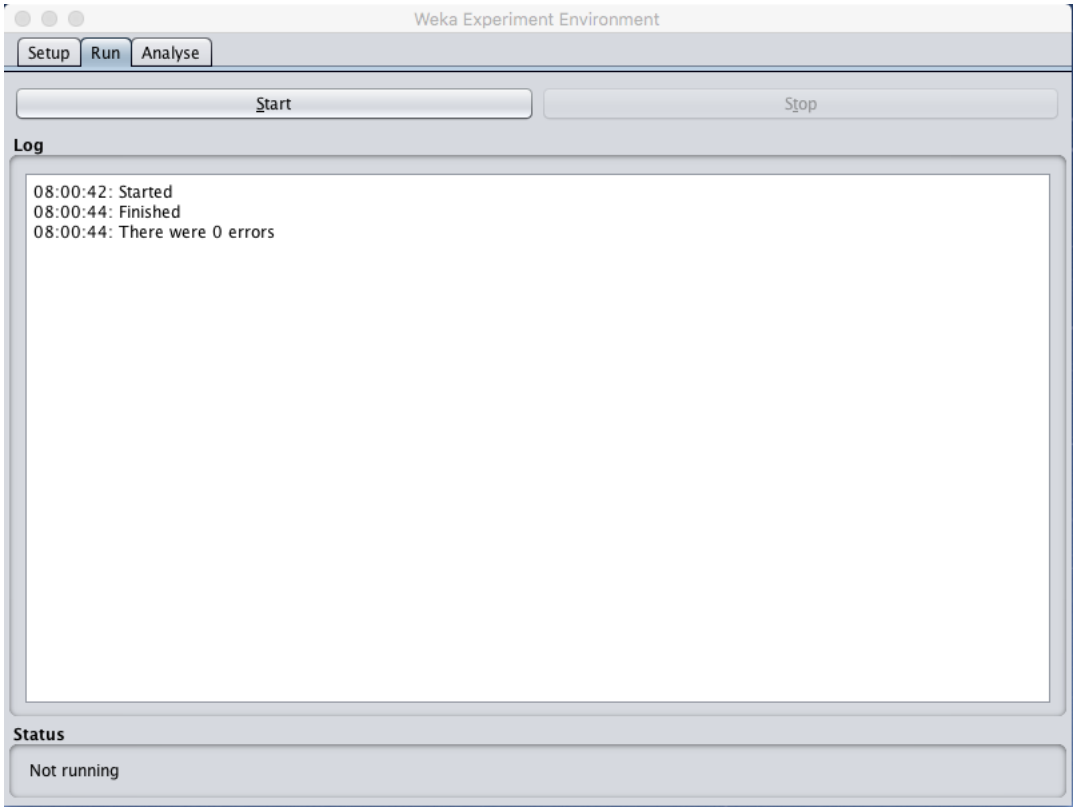


Figure 23.7: Weka Execute Weka Algorithm Comparison Experiment.

- 10. Click on *Analyse* tab. Click the *Experiment* button to load the results from the experiment.

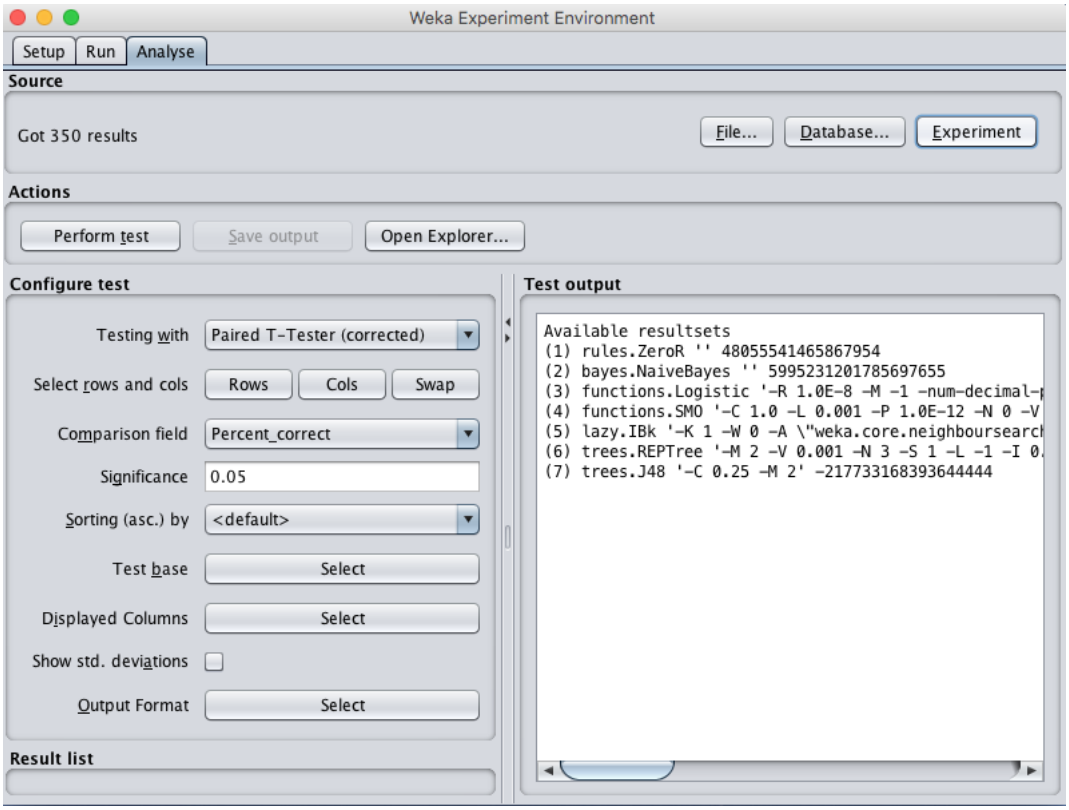


Figure 23.8: Weka Load Algorithm Comparison Experiment Results.

- 11. Click the *Perform test* button to perform a pairwise test comparing all of the results to the results for *ZeroR*.

```
Dataset          (1) rules.Ze | (2) bayes (3) funct (4) funct (5) lazy. (6) trees (7) trees
-------------------------------------------------------------------------------------------
iris             (50)   33.33 | 95.47 v  96.33 v  96.33 v  95.20 v  94.27 v  94.53 v
-------------------------------------------------------------------------------------------
                      (v/ /*) | (1/0/0)  (1/0/0)  (1/0/0)  (1/0/0)  (1/0/0)  (1/0/0)


Key:
(1) rules.ZeroR
(2) bayes.NaiveBayes
(3) functions.Logistic
(4) functions.SMO
(5) lazy.IBk
(6) trees.REPTree
(7) trees.J48
```

Listing 23.1: Results from Algorithm Comparison Experiment.

We can see that all of the models have skill. Each model has a score that is better than *ZeroR* and the difference is statistically significant. The results suggest both Logistic Regression and SVM achieved the highest accuracy. If we were to pick between the two, we would choose Logistic Regression if for no other reason that it is a much simpler model. Let's compare all of the results to the Logistic Regression results as the test base.

- 12. Click *Select* for the *Test base*, select *functions.Logistic* and click the *Select* button to choose the new test base. Click the *Perform test* button again to perform the new analysis.

```
Dataset         (3) function | (1) rules (2) bayes (4) funct (5) lazy. (6) trees (7) trees
-------------------------------------------------------------------------------------------
iris            (50)  96.33 |  33.33 *  95.47    96.33    95.20    94.27    94.53
-------------------------------------------------------------------------------------------
                    (v/ /*) |  (0/0/1)  (0/1/0)  (0/1/0)  (0/1/0)  (0/1/0)  (0/1/0)


Key:
(1) rules.ZeroR
(2) bayes.NaiveBayes
(3) functions.Logistic
(4) functions.SMO
(5) lazy.IBk
(6) trees.REPTree
(7) trees.J48
```

Listing 23.2: Results from Algorithm Comparison Experiment With A New Base.

We now see a very different story. Although the results for Logistic look better, the analysis suggests that the difference between these results and the results from all of the other algorithms are not statistically significant. From here we could choose an algorithm based on other criteria, like understandability or complexity. From this perspective Logistic Regression and Naive Bayes are good candidates. We could also seek to further improve the results of one or more of these algorithms and see if we can achieve a significant improvement. If we change the *Significance* to less constraining values of 0.50, we can see that the tree and KNN algorithms start to drop away. This suggests we could spend more time on the remaining methods. Change *significance* back to *0.05*. Let's choose to stick with Logistic Regression. We can collect some numbers we can use to describe the performance of the model on unseen data.

- 13. Check *Show std. deviations* to show standard deviations of accuracy scores.
- 14. Click the *Select* button for *Displayed Columns* and choose *functions.Logistic*, click *Select* to accept the selection. This will only show the results for the Logistic Regression algorithm.
- 15. Click *Perform test* to rerun the analysis.

We now have a final result we can use to describe our model.

```
Dataset                (3) functions.Logist
-------------------------------------------
iris                   (50)  96.33(3.38) |
-------------------------------------------
(v/ /*) |


Key:
(3) functions.Logistic
```

Listing 23.3: Final Algorithm Performance Results.

We can see that the estimated accuracy of the model on unseen data is 96.33% with a standard deviation of 3.38%.

23.5 Finalize Model and Present Results

We can create a final version of our model trained on all of the training data and save it to file.

- 1. Close the *Weka Experiment Environment*.
- 2. Open the *Weka Explorer* and load the `data/iris.arff` dataset.
- 3. Click on the *Classify* tab.
- 4. Select the *functions.Logistic* algorithm.
- 5. Change the *Test options* from *Cross-Validation* to *Use training set*.
- 6. Click the *Start* button to create the final model.

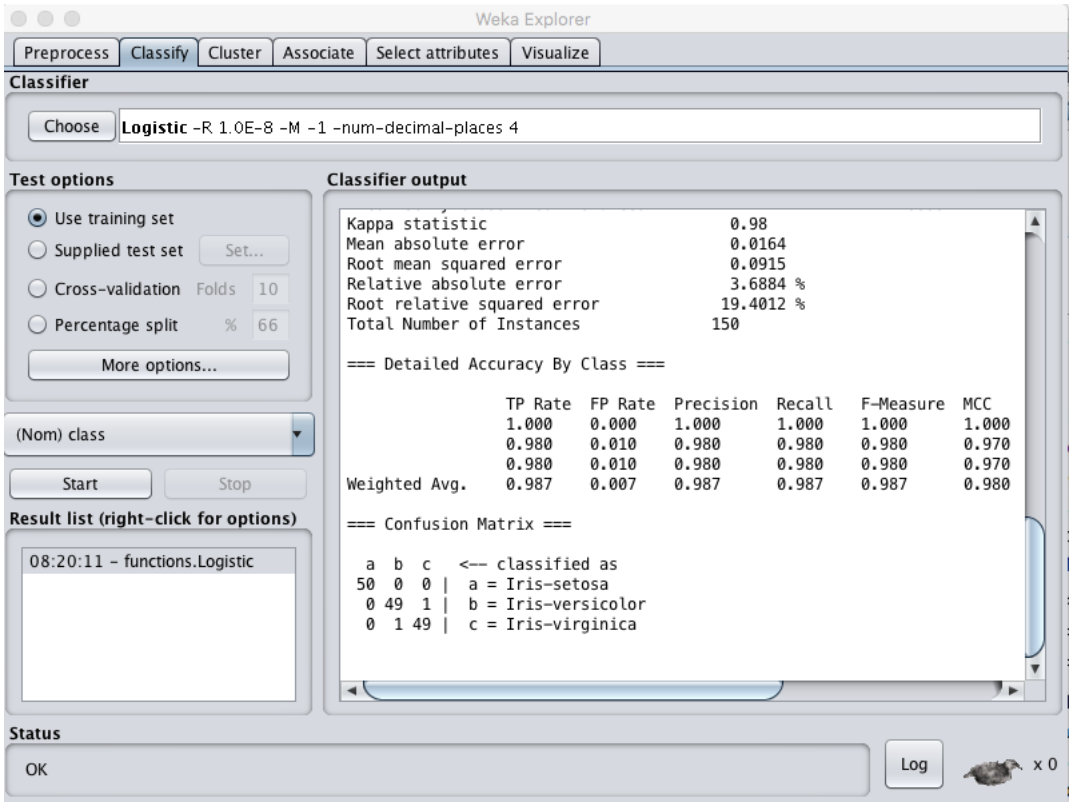


Figure 23.9: Weka Train Finalized Model on Entire Training Dataset.

- 7. Right click on the result item in the *Result list* and select *Save model*. Select a suitable location and type in a suitable name, such as *iris-logistic* for your model.

This model can then be loaded at a later time and used to make predictions on new flower measurements. We can use the mean and standard deviation of the model accuracy collected in the last section to help quantify the expected variability in the estimated accuracy of the model on unseen data. For example, we know that 95% of model accuracies will fall within two standard deviations of the mean model accuracy. Or, restated in a way we can explain to other people, we can generally expect that the performance of the model on unseen data will be 96.33% plus or minus 2×3.38 or 6.76, or between 87.57% and 100% accurate.

23.6 Summary

In this project you completed your first machine learning project end-to-end using the Weka machine learning workbench. Specifically, you learned:

- How to analyze your dataset and suggest at specific data transform and modeling techniques that may be useful.
- How to spot-check a suite of algorithms on the problem and analyze their results.
- How to finalize the model for making predictions on new data and presenting the estimated accuracy of the model on unseen data.

23.6.1 Next

You need to practice applied machine learning to get better. In the next project you will work through a binary classification problem and investigate using multiple views of the dataset in an effort to improve performance.

Chapter 24

How To Work Through a Binary Classification Project

The fastest way to get good at applied machine learning is to practice on end-to-end projects. In this project you will discover how to work through a binary classification problem in Weka, end-to-end. After completing this project you will know:

- How to load a dataset and analyze the loaded data.
- How to create multiple different transformed views of the data and evaluate a suite of algorithms on each.
- How to finalize and present the results of a model for making predictions on new data.

Let's get started.

24.1 Tutorial Overview

This tutorial will walk you through the key steps required to complete a machine learning project. We will work through the following process:

1. Load the dataset.
2. Analyze the dataset.
3. Prepare views of the dataset.
4. Evaluate algorithms.
5. Finalize model and present results.

24.2 Load the Dataset

The problem used in this tutorial is the Pima Indians Onset of Diabetes dataset. You can learn more about this dataset in Section 8.2.1.

- 1. Open the *Weka GUI Chooser*.
- 2. Click the *Explorer* button to open the *Weka Explorer*.
- 3. Click the *Open file...* button, navigate to the `data/` directory and select `diabetes.arff`. Click the *Open button*.

The dataset is now loaded into Weka.

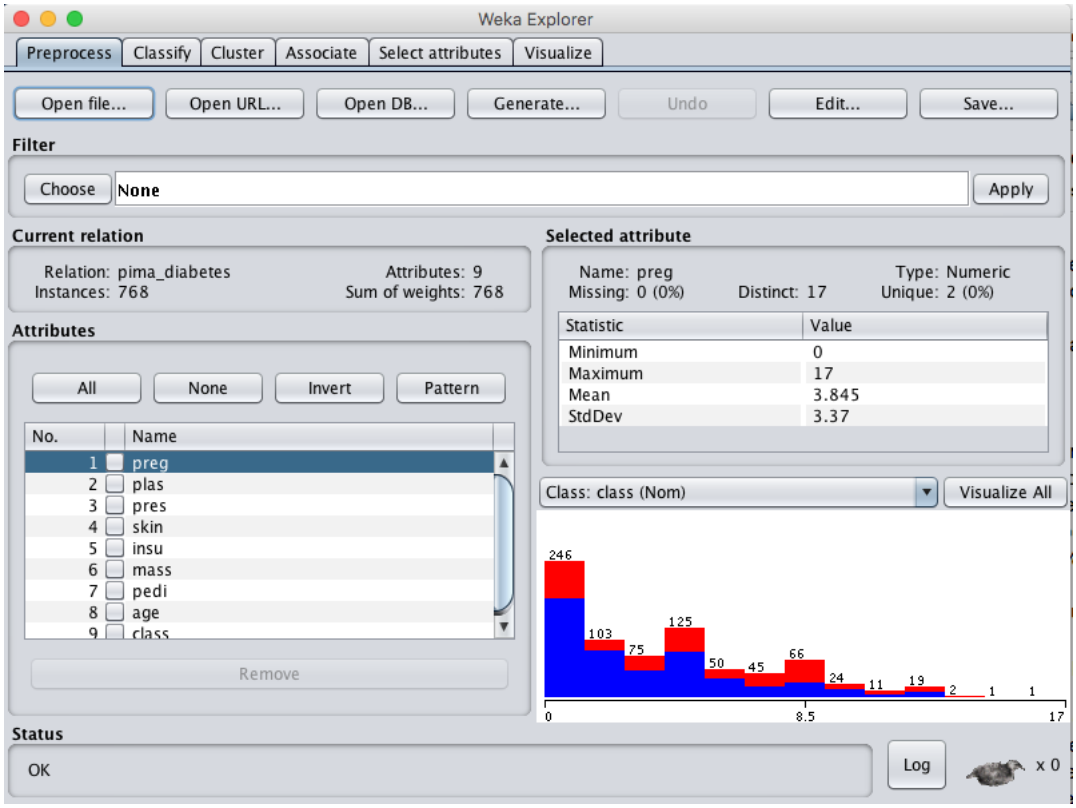


Figure 24.1: Weka Load Pima Indians Onset of Diabetes Dataset.

24.3 Analyze the Dataset

It is important to review your data before you start modeling. Reviewing the distribution of each attribute and the interactions between attributes may shed light on specific data transforms and specific modeling techniques that we could use.

24.3.1 Summary Statistics

Review the details about the dataset in the *Current relation* pane. We can notice a few things:

- The dataset has the name *pima_diabetes*.
- There are 768 instances in the dataset. If we evaluate models using 10-fold cross-validation then each fold will have about 76 instances, which is fine.

- There are 9 attributes, 8 input and one output attributes.
- Click on each attribute in the *Attributes* pane and review the summary statistics in the *Selected attribute* pane.

We can notice a few facts about our data:

- The input attributes are all numerical and have differing scales. We may see some benefit from either normalizing or standardizing the data.
- There are no missing values marked.
- There are values for some attributes that do not seem sensible, specifically: `plas`, `pres`, `skin`, `insu`, and `mass` have values of 0. These are probably missing data that could be marked.
- The class attribute is nominal and has two output values meaning that this is a two-class or binary classification problem.
- The class attribute is unbalanced, 1 `positive` outcome to 1.8 `negative` outcomes, nearly double the number of negative cases. We may benefit from balancing the class values.

24.3.2 Attribute Distributions

Click the *Visualize All* button and let's review the graphical distribution of each attribute.

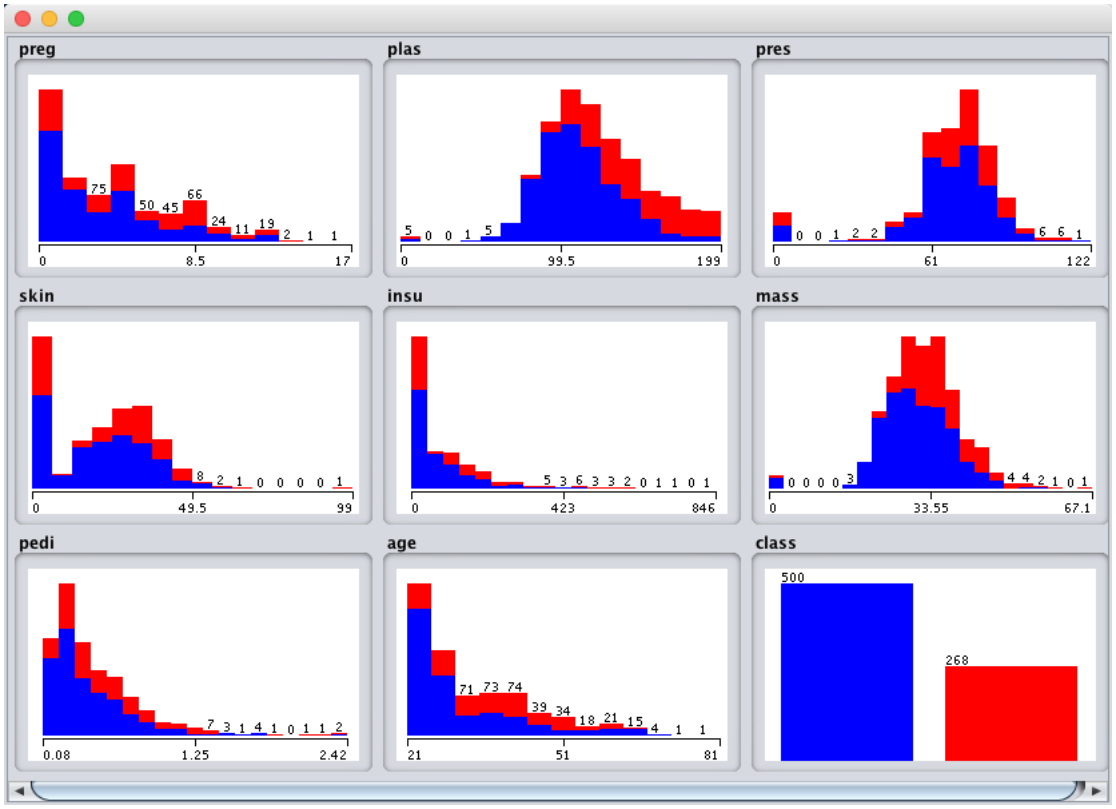


Figure 24.2: Weka Pima Indians Univariate Attribute Distributions.

We can notice a few things about the shape of the data:

- Some attributes have a Gaussian-like distribution such as `plas`, `pres`, `skin` and `mass`, suggesting methods that make this assumption could achieve good results, like Logistic Regression and Naive Bayes.
- We see a lot of overlap between the classes across the attribute values. The classes do not seem easily separable.
- We can clearly see the class imbalance graphically depicted.

24.3.3 Attribute Interactions

- Click the *Visualize* tab and let's review some interactions between the attributes.
- Increase the window size so all plots are visible.
- Increase the *PointSize* to 3 to make the dots easier to see.
- Click the *Update* button to apply the changes.

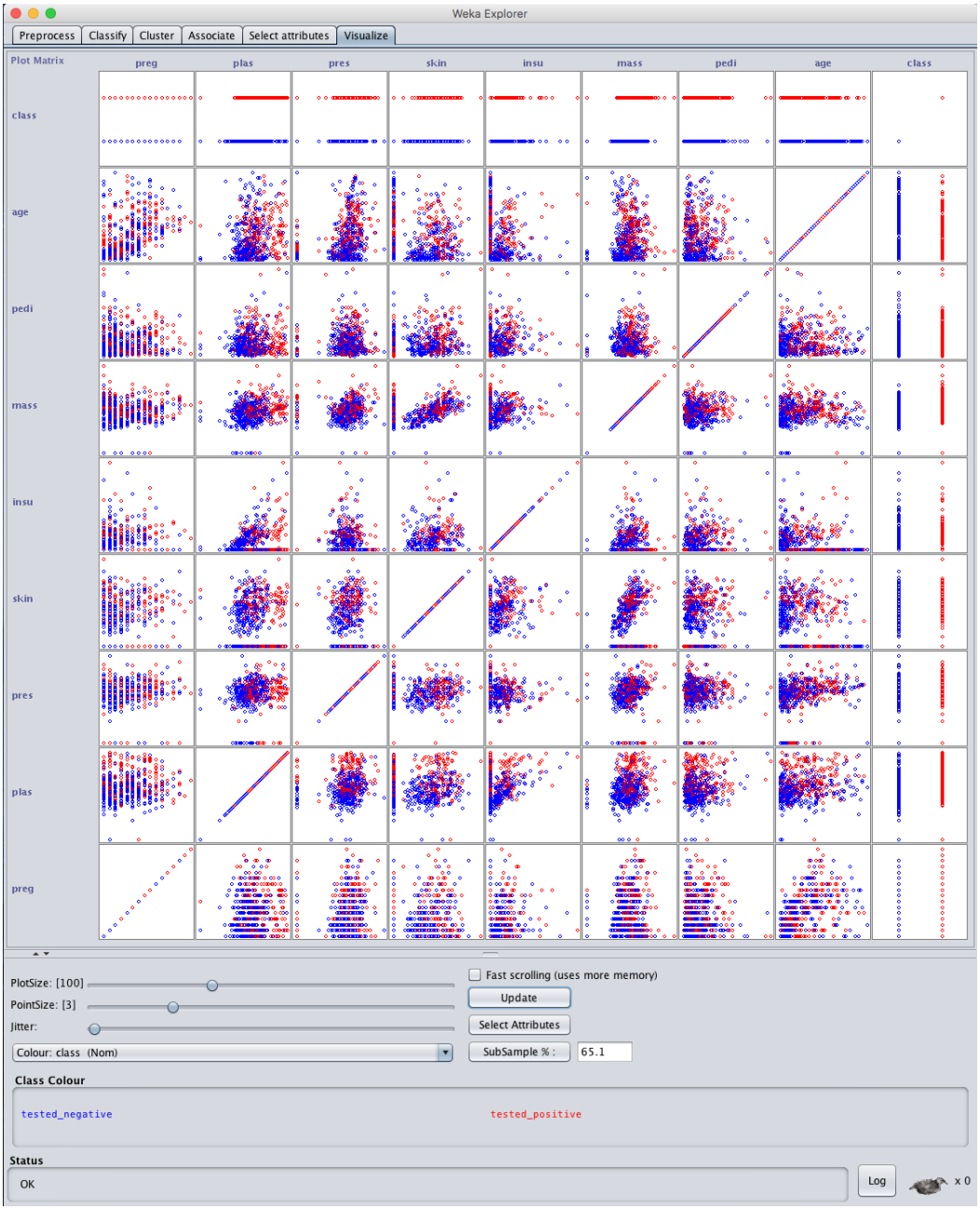


Figure 24.3: Weka Pima Indians Scatter Plot Matrix.

Looking across the graphs for the input variables, we can generally see poor separation between the classes on the scatter plots. This dataset will not be a walk in the park. It suggests that we could benefit from some good data transforms and creating multiple views of the dataset. It also suggests we may get benefits from using ensemble methods.

24.4 Prepare Views of the Dataset

We noted in the previous section that this may be a difficult problem and that we may benefit from multiple views of the data. In this section we will create varied views of the data, so

that when we evaluate algorithms in the next section we can get an idea of the views that are generally better at exposing the structure of the classification problem to the models. We are going to create 3 additional views of the data, so that in addition to the raw data we will have 4 different copies of the dataset in total. We will create each view of the dataset from the original and save it to a new file for later use in our experiments.

24.4.1 Normalized View

The first view we will create is of all the input attributes normalized to the range 0 to 1. This may benefit multiple algorithms that can be influenced by the scale of the attributes, like regression and instance-based methods.

1. In the *Weka Explorer* with the `data/diabetes.arff` file loaded.
2. Click the *Choose* button in the *Filter* pane and choose the *unsupervised.attribute.Normalize* filter.
3. Click the *Apply* button to apply the filter.
4. Click each attribute in the *Attributes* pane and review the min and max values in the *Selected attribute* pane to confirm they are 0 and 1.
5. Click the *Save...* button, navigate to a suitable directory and type in a suitable name for this transformed dataset, such as `diabetes-normalize.arff`.
6. Close the *Weka Explorer* interface to avoid contaminating the other views we want to create.

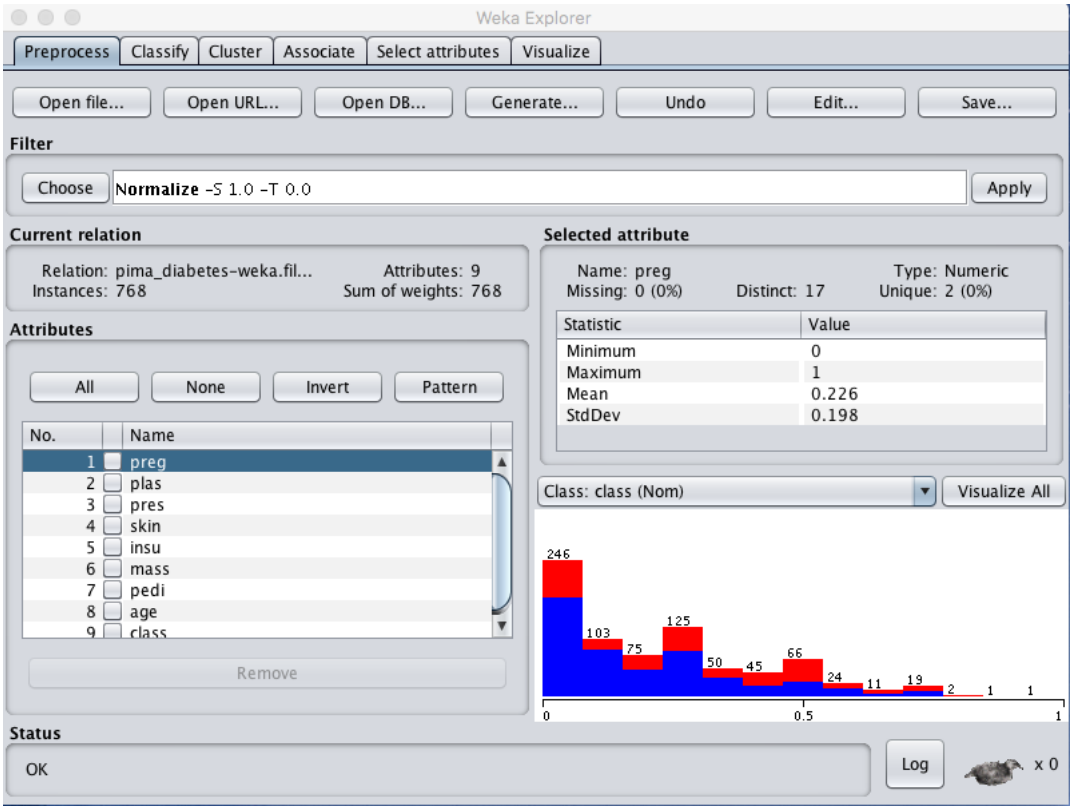


Figure 24.4: Weka Normalize Pima Indian Dataset.

24.4.2 Standardized View

We noted in the previous section that some of the attribute have a Gaussian-like distribution. We can rescale the data and take this distribution into account by using a standardizing filter. This will create a copy of the dataset where each attribute has a mean value of 0 and a standard deviation (mean variance) of 1. This may benefit algorithms in the next section that assume a Gaussian distribution in the input attributes, like Logistic Regression and Naive Bayes.

1. Open the *Weka Explorer*.
2. Load the Pima Indians onset of diabetes dataset `data/diabetes.arff`.
3. Click the *Choose* button in the *Filter* pane and choose the *unsupervised.attribute.Standardize* filter.
4. Click the *Apply* button to apply the filter.
5. Click each attribute in the *Attributes* pane and review the mean and standard deviation values in the *Selected attribute* pane to confirm they are 0 and 1 respectively.
6. Click the *Save...* button, navigate to a suitable directory and type in a suitable name for this transformed dataset, such as `diabetes-standardize.arff`.
7. Close the *Weka Explorer* interface.

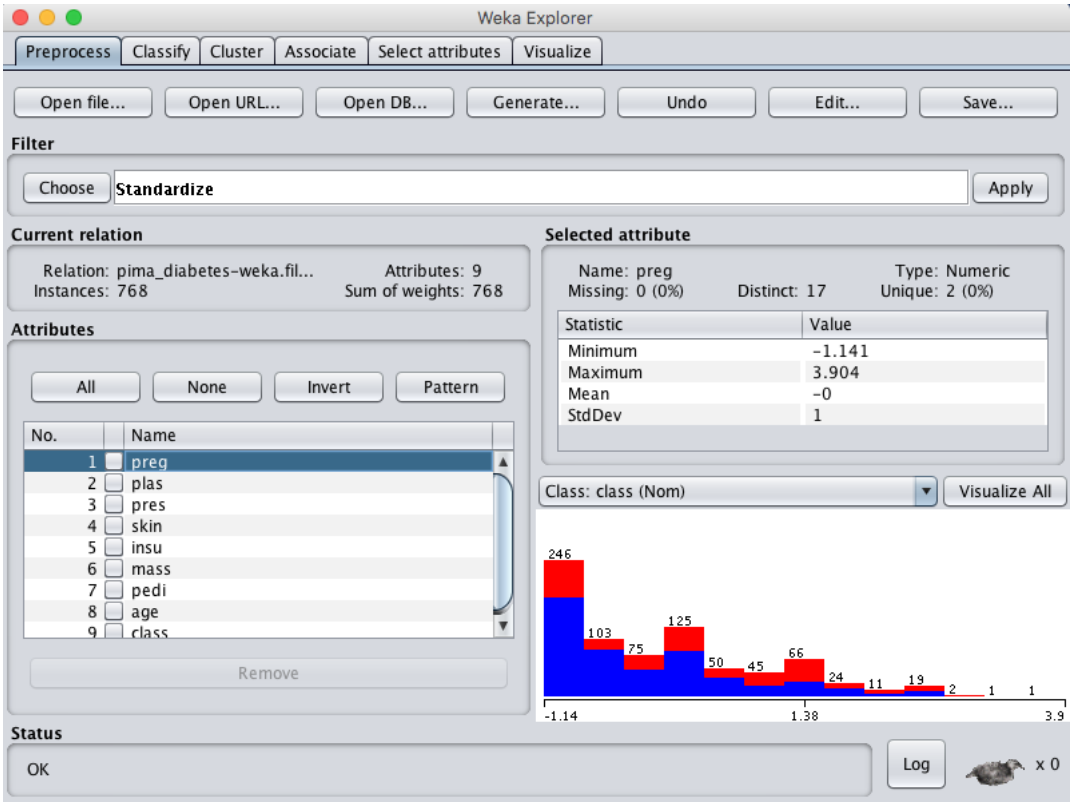


Figure 24.5: Weka Standardize Pima Indians Dataset.

24.4.3 Missing Data

In the previous section we suspected some of the attributes had bad or missing data marked with 0 values. We can create a new copy of the dataset with the missing data marked and then imputed with an average value for each attribute. This may help methods that assume a smooth change in the attribute distributions, such as Logistic Regression and instance-based methods. First let's mark the 0 values for some attributes as missing.

1. Open the *Weka Explorer*.
2. Load the Pima Indians onset of diabetes dataset `data/diabetes.arff`.
3. Click the *Choose* button for the Filter and select the *unsupervised.attribute.NumericalCleaner* filter.
4. Click on the filter to configure it.
5. Set the *attributeIndices* to 2-6
6. Set *minThreshold* to 0.1E-8 (close to zero), which is the minimum value allowed for each attribute.
7. Set *minDefault* to NaN, which is unknown and will replace values below the threshold.
8. Click the *OK* button on the filter configuration.

9. Click the *Apply* button to apply the filter.

10. Click each attribute in the *Attributes* pane and review the number of missing values for each attribute. You should see some non-zero counts for attributes 2 to 6.

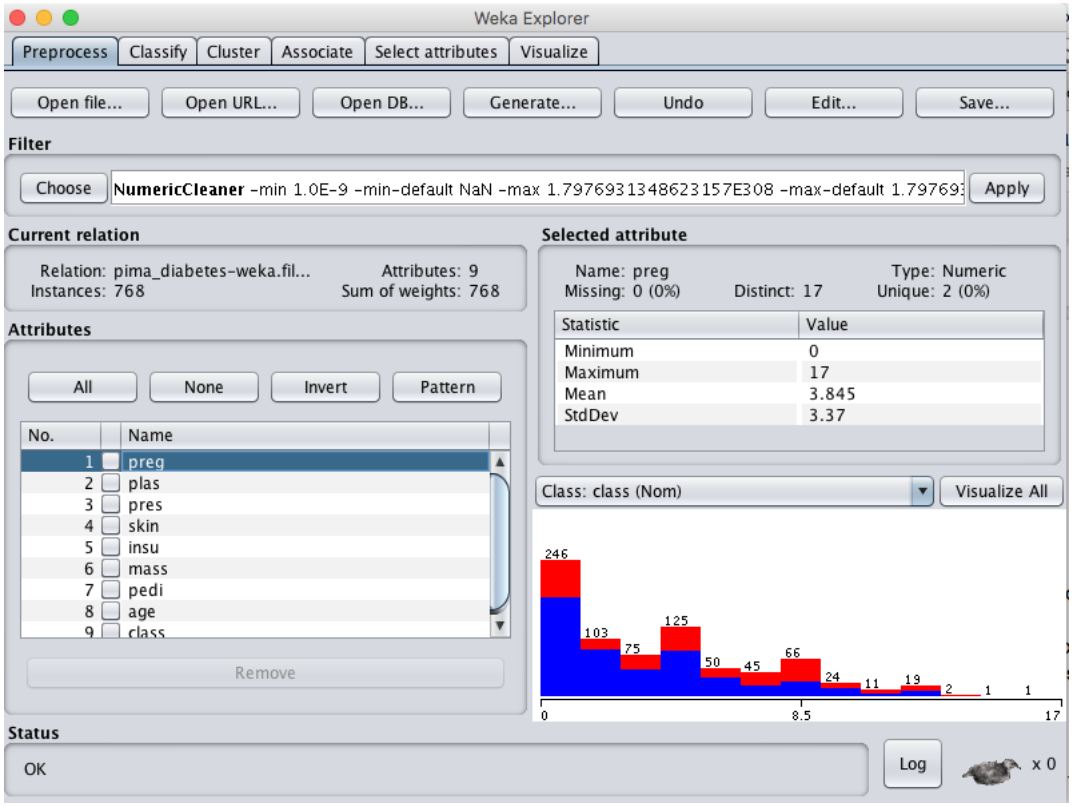


Figure 24.6: Weka Numeric Cleaner Data Filter For Pima Indians Dataset.

Now, let's impute the missing values as the mean.

1. Click the *Choose* button in the *Filter* pane and select *unsupervised.attribute.ReplaceMissingValues* filter.

2. Click the *Apply* button to apply the filter your dataset.

3. Click each attribute in the *Attributes* pane and review the number of missing values for each attribute. You should see all attributes should have no missing values and the distribution of attributes 2 to 6 should have changed slightly.

4. Click the *Save...* button, navigate to a suitable directory and type in a suitable name for this transformed dataset, such as `diabetes-missing.arff`.

5. Close the *Weka Explorer*.

Other views of the data you may want to consider investigating are subsets of features chosen by a feature selection method and a view where the class attribute is rebalanced.

24.5 Evaluate Algorithms

Let's design an experiment to evaluate a suite of standard classification algorithms on the different views of the problem that we created.

- 1. Click the *Experimenter* button on the *Weka GUI Chooser* to launch the *Weka Experiment Environment*.
- 2. Click *New* to start a new experiment.
- 3. In the *Datasets* pane click *Add new...* and select the following 4 datasets:
  - `data/diabetes.arff` (the raw dataset)
  - `diabetes-normalized.arff`
  - `diabetes-standardized.arff`
  - `diabetes-missing.arff`
- 4. In the *Algorithms* pane click *Add new...* and add the following 8 classification algorithms:
  - *rules.ZeroR*
  - *bayes.NaiveBayes*
  - *functions.Logistic*
  - *functions.SMO*
  - *lazy.IBk*
  - *rules.PART*
  - *trees.REPTree*
  - *trees.J48*
- 5. Select *IBk* in the list of algorithms and click the *Edit selected...* button.
- 6. Change *KNN* from *1* to *3* and click the *OK* button to save the settings.

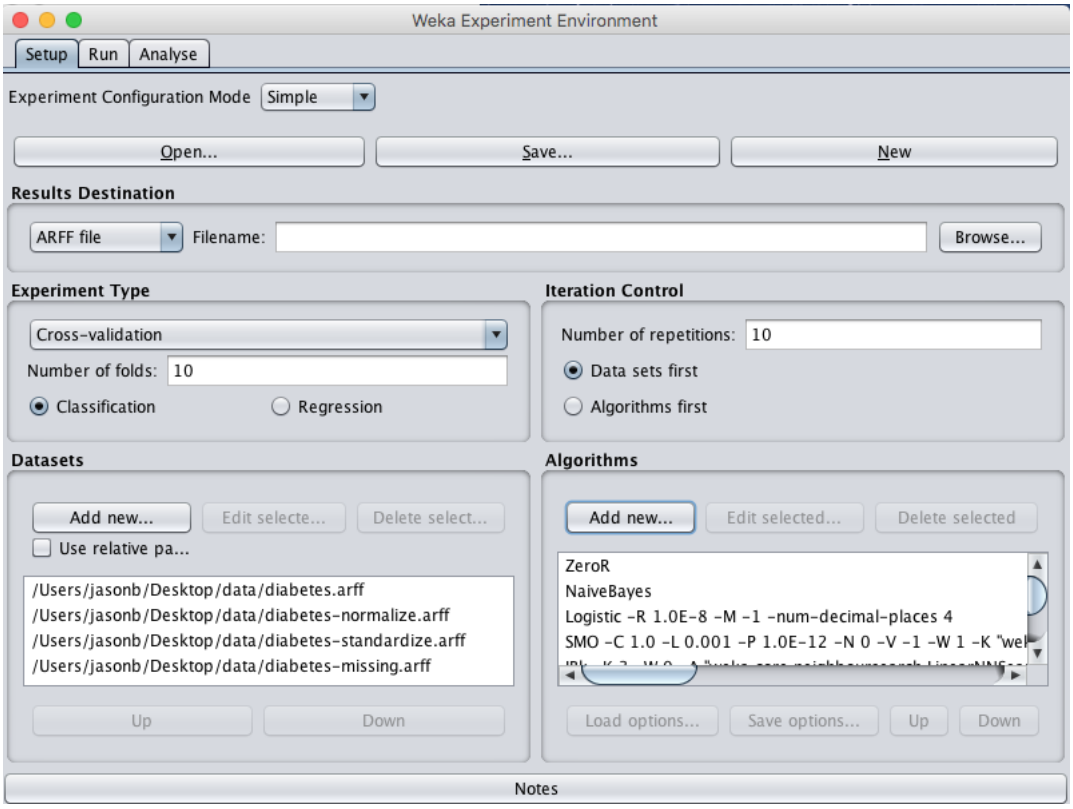


Figure 24.7: Weka Algorithm Comparison Experiment for Pima Indians Dataset.

- 7. Click on *Run* tab and click the *Start* button to run the experiment. The experiment should complete in just a few seconds.
- 8. Click on the *Analyse* tab. Click the *Experiment* button to load the results from the experiment.

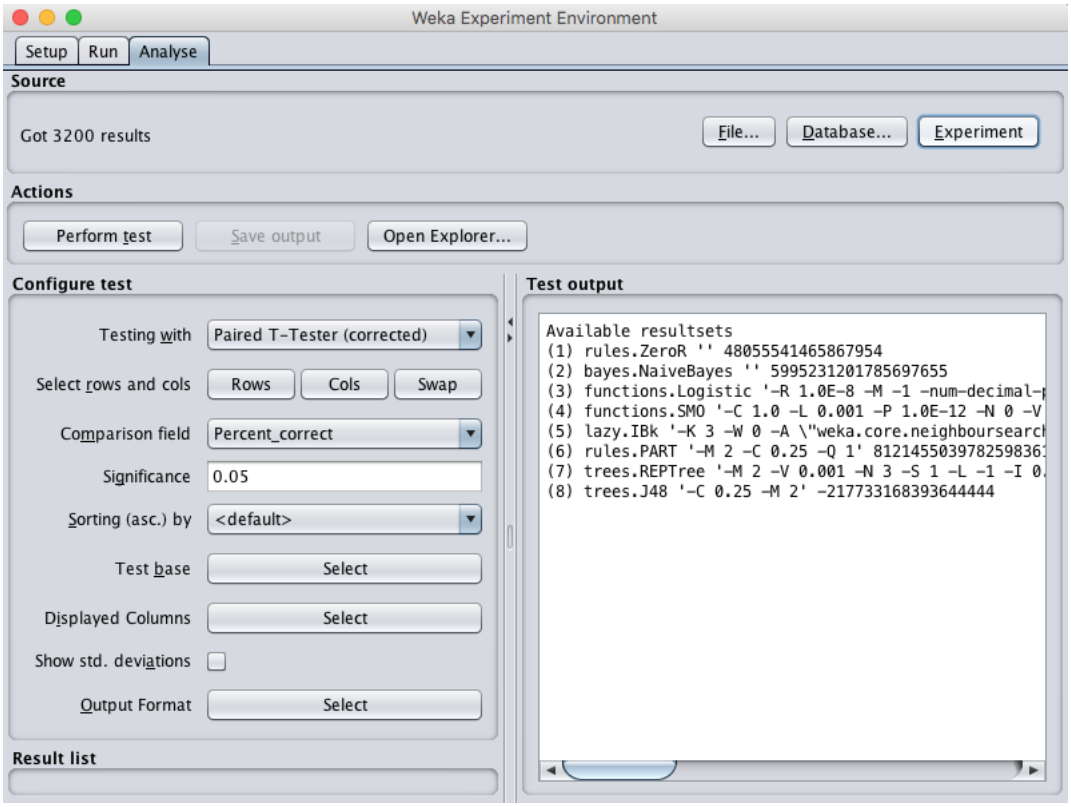


Figure 24.8: Weka Load Algorithm Comparison Experiment Results for Pima Indians Dataset.

- 9. Click the *Perform test* button to perform a pairwise test-test comparing all of the results to the results for *ZeroR*.

```
Dataset (1) rules.Ze | (2) bayes (3) funct (4) funct (5) lazy. (6) rules (7) trees (8) trees
----------------------------------------------------------------------------------------------
pima_diabetes 65.11 |  75.75 v  77.47 v  76.80 v  73.86 v  73.45 v  74.46 v  74.49 v
normalized    65.11 |  75.77 v  77.47 v  76.80 v  73.86 v  73.45 v  74.42 v  74.49 v
standardized  65.11 |  75.65 v  77.47 v  76.81 v  73.86 v  73.45 v  74.39 v  74.49 v
missing       65.11 |  74.81 v  76.86 v  76.30 v  73.54 v  73.03 v  73.70 v  74.69 v
----------------------------------------------------------------------------------------------
            (v/ /*) |  (4/0/0)  (4/0/0)  (4/0/0)  (4/0/0)  (4/0/0)  (4/0/0)  (4/0/0)


Key:
(1) rules.ZeroR
(2) bayes.NaiveBayes
(3) functions.Logistic
(4) functions.SMO
(5) lazy.IBk
(6) rules.PART
(7) trees.REPTree
(8) trees.J48
```

Listing 24.1: Results from the Algorithm Comparison Experiment.

We can see that all of the algorithms are skillful on all of the views of the dataset compared to *ZeroR*. We can also see that our baseline for skill is 65.11% accuracy. Just looking at the raw

classification accuracies, we can see that the view of the dataset with missing values imputed looks to have resulted in lower model accuracy in general. It also looks like there is little difference between the standardized and normalized results as compared to the raw results other than a few fractions of percent. It suggests we can probably stick with the raw dataset. Finally, it looks like Logistic regression may have achieved higher accuracy results than the other algorithms, let's check if the difference is significant.

- 10. Click the *Select* button for *Test base* and choose *functions.Logistic*. Click the *Perform test* button to rerun the analysis.

```
Dataset (3) function | (1) rules (2) bayes (4) funct (5) lazy. (6) rules (7) trees (8) trees
-------------------------------------------------------------------------------------------
pima_diabetes 77.47 |  65.11 *  75.75    76.80    73.86 *  73.45 *  74.46 *  74.49
normalized    77.47 |  65.11 *  75.77    76.80    73.86 *  73.45 *  74.42 *  74.49
standardized  77.47 |  65.11 *  75.65    76.81    73.86 *  73.45 *  74.39 *  74.49
missing       76.86 |  65.11 *  74.81    76.30    73.54 *  73.03 *  73.70 *  74.69
-------------------------------------------------------------------------------------------
            (v/ /*) |  (0/0/4)  (0/4/0)  (0/4/0)  (0/0/4)  (0/0/4)  (0/0/4)  (0/4/0)


Key:
(1) rules.ZeroR
(2) bayes.NaiveBayes
(3) functions.Logistic
(4) functions.SMO
(5) lazy.IBk
(6) rules.PART
(7) trees.REPTree
(8) trees.J48
```

Listing 24.2: Results from the Algorithm Comparison Experiment With a New Test Base.

It does look like the logistic regression results are better than some of the other results, such as *IBk*, *PART*, *REPTree* and *ZeroR*, but not statistically significantly different from *NaiveBayes*, *SMO* or *J48*.

- 11. Check *Show std. deviations* to show standard deviations.
- 12. Click the *Select* button for *Displayed Columns* and choose *functions.Logistic*, click *Select* to accept the selection. This will only show results for the logistic regression algorithm.
- 13. Click *Perform test* to rerun the analysis.

We now have a final result we can use to describe our model.

```
Dataset                 (3) functions.Logist
----------------------------------------------
pima_diabetes                 77.47(4.39) |
normalized                    77.47(4.39) |
standardized                  77.47(4.39) |
missing                       76.86(4.90) |
----------------------------------------------
```

```
(v/ /*) |


Key:
(3) functions.Logistic
```

Listing 24.3: Final Results for the Logistic Regression Algorithm.

We can see that the estimated accuracy of the model on unseen data is 77.47% with a standard deviation of 4.39%.

- 14. Close the *Weka Experiment Environment*.

24.6 Finalize Model and Present Results

We can create a final version of our model trained on all of the training data and save it to file.

1. Open the *Weka Explorer* and load the `data/diabetes.arff` dataset.
2. Click on the *Classify*.
3. Select the *functions.Logistic* algorithm.
4. Change the *Test options* from *Cross-Validation* to *Use training set*.
5. Click the *Start* button to create the final model.
6. Right click on the result item in the *Result list* and select *Save model*. Select a suitable location and type in a suitable name, such as *diabetes-logistic* for your model.

This model can then be loaded at a later time and used to make predictions on new data. We can use the mean and standard deviation of the model accuracy collected in the last section to help quantify the expected variability in the estimated accuracy of the model on unseen data. We can generally expect that the performance of the model on unseen data will be 77.47% plus or minus $(2 \times 4.39)\%$ or 8.78%. We can restate this as between 68.96% and 86.25% accurate. What is surprising about this final statement of model accuracy is that at the lower end, the model is only a shade better than the *ZeroR* model that achieved an accuracy if 65.11% by predicting a negative outcome for all predictions.

24.7 Summary

In this project you completed a binary classification machine learning project end-to-end using the Weka machine learning workbench. Specifically, you learned:

- How to analyze your dataset and suggest at specific data transform and modeling techniques that may be useful.
- How to design and save multiple views of your data and spot-check multiple algorithms on these views.
- How to finalize the model for making predictions on new data and presenting the estimated accuracy of the model on unseen data.

24.7.1 Next

Next in the final project, you will work through a regression machine learning problem. This is the largest project and involves also using algorithm tuning and ensemble methods in an effort to achieve improved model performance.

Chapter 25

How to Work Through a Regression Machine Learning Project

The more time that you spend working through projects, the faster you will develop your skills in applied machine learning. As such, it is important to work on a suite of different problem types. In this project you will discover how to work through a regression problem in Weka, end-to-end. After completing this project you will know:

- How to load and analyze a regression dataset in Weka.
- How to create multiple different transformed views of the data and evaluate a suite of algorithms on each.
- How to finalize and present the results of a model for making predictions on new data.

Let's get started.

25.1 Tutorial Overview

This tutorial will walk you through the key steps required to complete a machine learning project in Weka. We will work through the following steps:

1. Load the dataset.
2. Analyze the dataset.
3. Prepare views of the dataset.
4. Evaluate algorithms.
5. Tune algorithm performance.
6. Evaluate ensemble algorithms.
7. Present results.

25.2 Load the Dataset

In this tutorial we will work on the Boston House Price dataset. In this dataset, each instance describes the properties of a Boston suburb and the task is to predict the house prices in thousands of dollars. You can learn more about this dataset in Section 8.4.2.

1. Open the *Weka GUI Chooser*.
2. Click the *Explorer* button to open the *Weka Explorer*.
3. Click the *Open file...* button, navigate to the `numeric/` directory and select `housing.arff`. Click the *Open button*.

The dataset is now loaded into Weka.

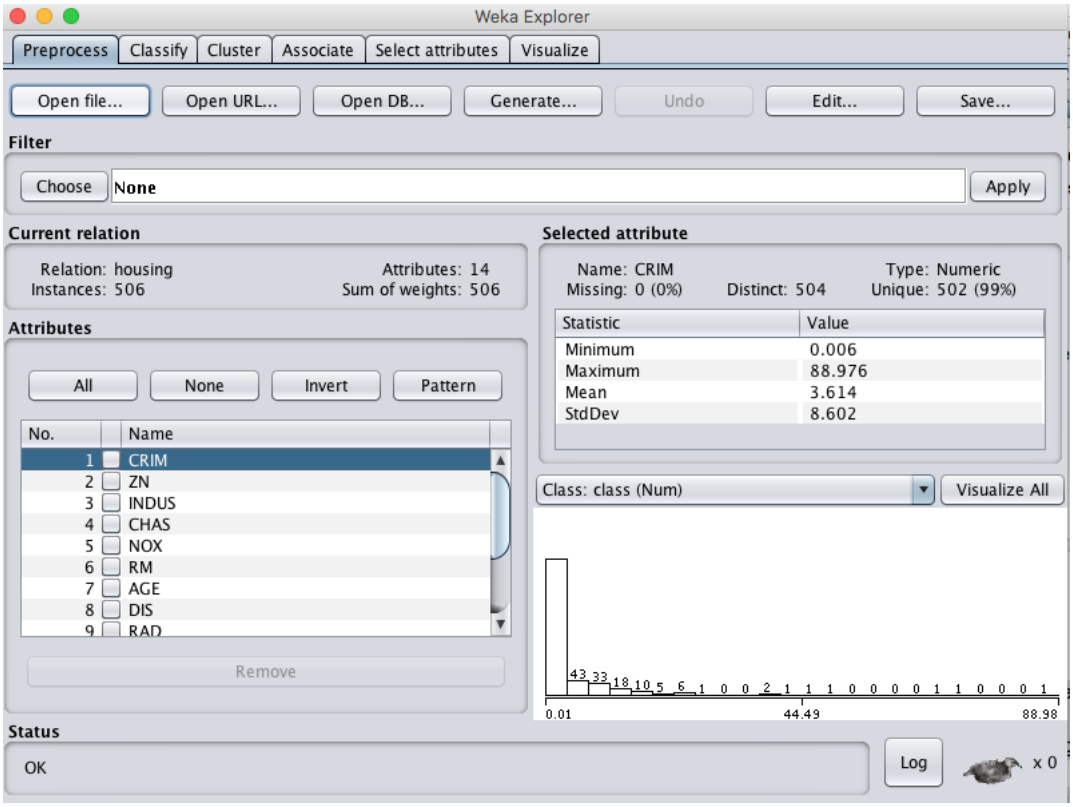


Figure 25.1: Weka Load the Boston House Price Dataset.

25.3 Analyze the Dataset

It is important to review your data before you start modeling. Reviewing the distribution of each attribute and the interactions between attributes may shed light on specific data transforms and specific modeling techniques that we could use.

25.3.1 Summary Statistics

Review the details about the dataset in the *Current relation* pane. We can notice a few things:

- The dataset is called housing.
- There are 506 instances. If we use 10-fold cross-validation later to evaluate the algorithms, then each fold will be comprised of about 50 instances, which is fine.
- There are 14 attributes, 13 inputs and 1 output variable.
- Click on each attribute in the *Attributes* pane and review the summary statistics in the *Selected attribute* pane.

We can notice a few facts about our data:

- There are no missing values for any of the attributes.
- All inputs are numeric except one binary attribute, and have values in differing ranges.
- The last attribute is the output variable called class, it is numeric.

We may see some benefit from either normalizing or standardizing the data.

25.3.2 Attribute Distributions

- Click the *Visualize All* button and let's review the graphical distribution of each attribute.

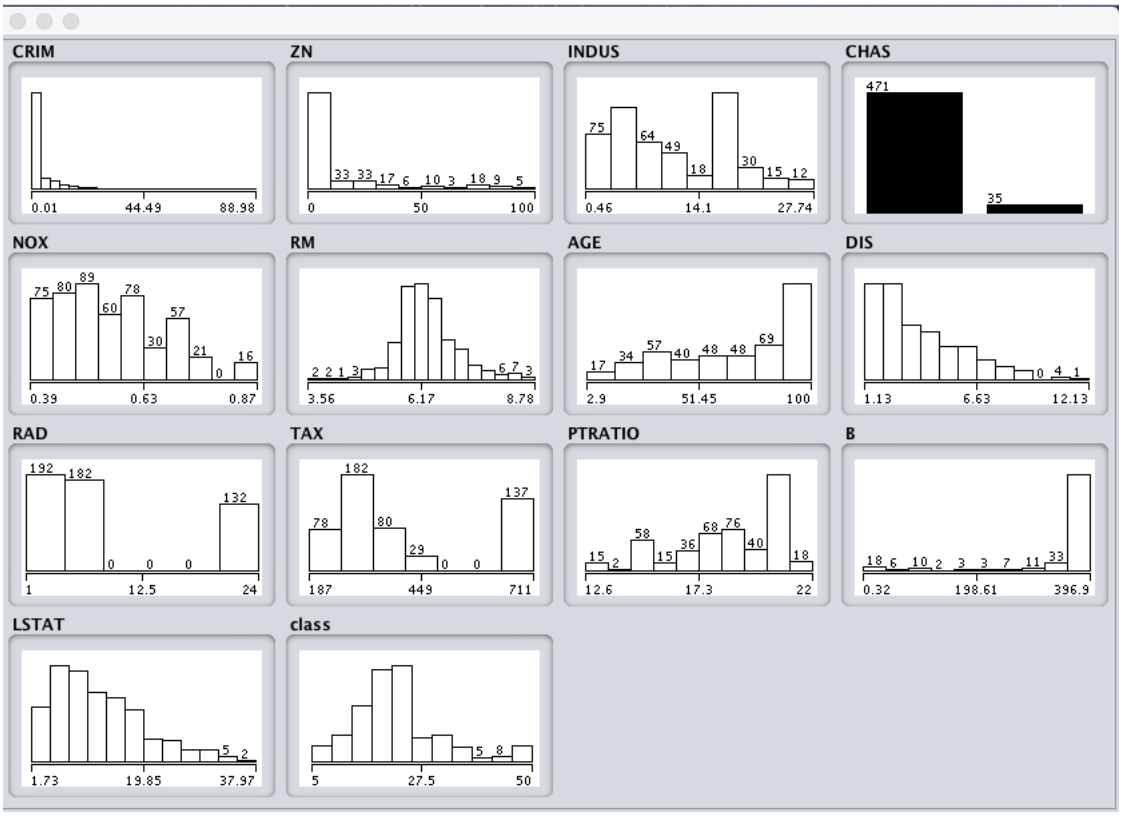


Figure 25.2: Weka Boston House Price Univariate Attribute Distributions.

We can notice a few things about the shape of the data:

- We can see that the attributes have a range of differing distributions.
- The `CHAS` attribute looks like a binary distribution (two values).
- The `RM` attribute looks like it has a Gaussian distribution.

We may see more benefit in using nonlinear regression methods like decision trees and such, than using linear regression methods like linear regression.

25.3.3 Attribute Interactions

- Click the *Visualize* tab and let's review some interactions between the attributes.

1. Decrease the *PlotSize* to 50 and adjust the window size so all plots are visible.
2. Increase the *PointSize* to 3 to make the dots easier to see.
3. Click the *Update* button to apply the changes.

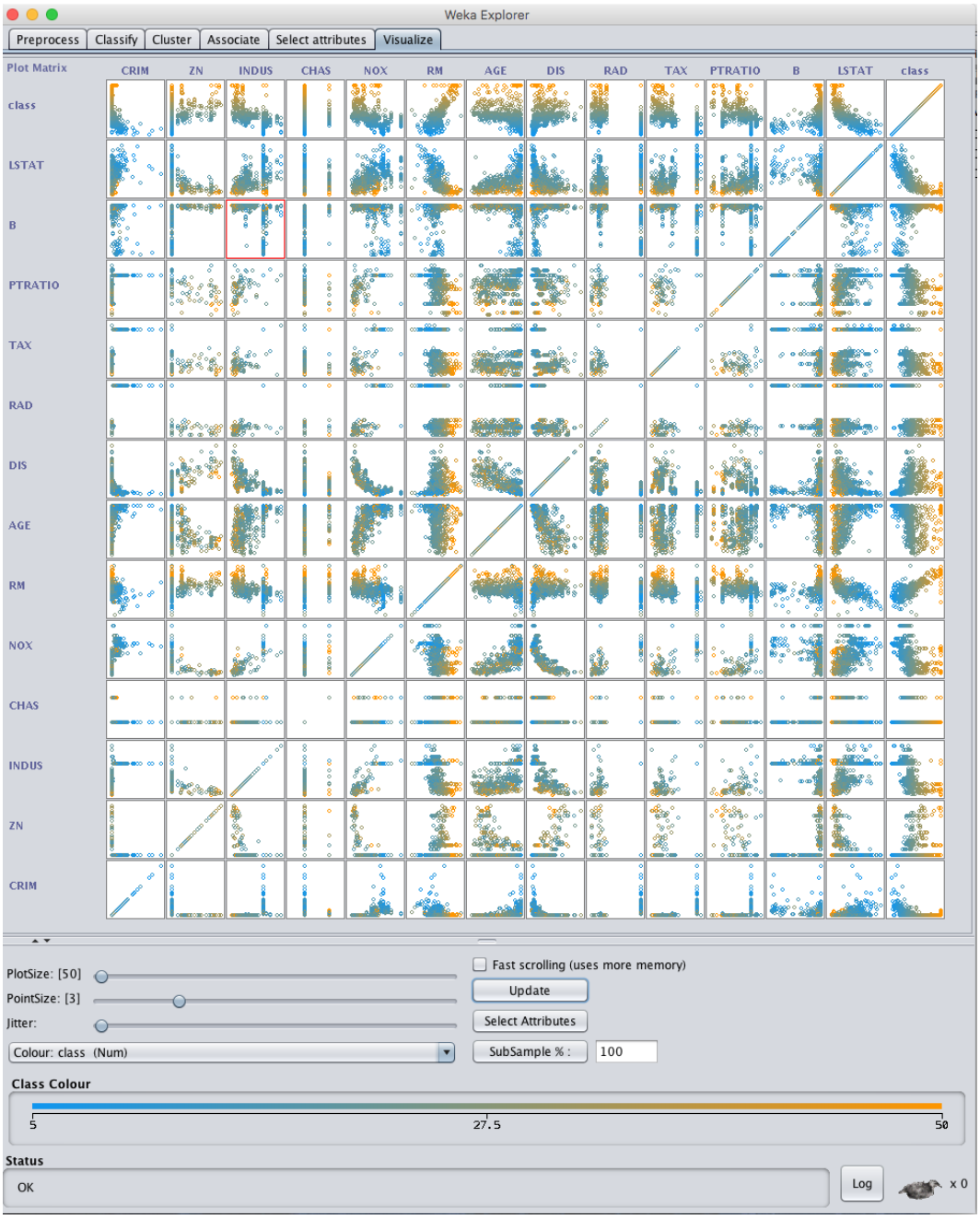


Figure 25.3: Weka Boston House Price Scatter Plot Matrix.

Looking across the graphs we can see some structured relationships that may aid in modeling such as `DIS` vs `NOX` and `AGE` vs `NOX`. We can also see some structure relationships between input attributes and the output attribute such as `LSTAT` and `CLASS` and `RM` and `CLASS`.

25.4 Prepare Views of the Dataset

In this section we will create some different views of the data, so that when we evaluate algorithms in the next section we can get an idea of the views that are generally better at exposing the structure of the regression problem to the models. We are first going to create a

modified copy of the original `housing.arff` data file, then make 3 additional transforms of the data. We will create each view of the dataset from our modified copy of the original and save it to a new file for later use in our experiments.

25.4.1 Modified Copy

The `CHAS` attribute is nominal (binary) with the values `0` and `1`. We want to make a copy of the original `housing.arff` data file and change `CHAS` to a numeric attribute so that all input attributes are numeric. This will help with transforming and modeling the dataset.

- Locate the `housing.arff` dataset and create a copy of it in the same directory called `housing-numeric.arff`.
- Open this modified file `housing-numeric.arff` in a text editor and scroll down to where the attributes are defined, specifically the `CHAS` attribute on line 56.

```
@relation 'housing'
@attribute CRIM real
@attribute ZN real
@attribute INDUS real
@attribute CHAS { 0, 1}
@attribute NOX real
@attribute RM real
@attribute AGE real
@attribute DIS real
@attribute RAD real
@attribute TAX real
@attribute PTRATIO real
@attribute B real
@attribute LSTAT real
@attribute class real
@data
```

Figure 25.4: Weka Boston House Price Attribute Data Types.

Change the definition of the `CHAS` attribute from:

```
@attribute CHAS { 0, 1}
```

to:

```
@attribute CHAS real
```

The `CHAS` attribute is now numeric rather than nominal. This modified copy of the dataset `housing-numeric.arff` will now be used as the baseline dataset.

```
@relation 'housing'
@attribute CRIM real
@attribute ZN real
@attribute INDUS real
@attribute CHAS real
@attribute NOX real
@attribute RM real
@attribute AGE real
@attribute DIS real
@attribute RAD real
@attribute TAX real
@attribute PTRATIO real
@attribute B real
@attribute LSTAT real
@attribute class real
@data
```

Figure 25.5: Weka Boston House Price Dataset With Numeric Data Types.

25.4.2 Normalized Dataset

The first view we will create is of all the input attributes normalized to the range 0 to 1. This may benefit multiple algorithms that can be influenced by the scale of the attributes, like regression and instance-based methods.

1. Open the *Weka Explorer*.
2. Open the modified numeric dataset `housing-numeric.arff`.
3. Click the *Choose* button in the *Filter* pane and choose the *unsupervised.attribute.Normalize* filter.
4. Click the *Apply* button to apply the filter.
5. Click each attribute in the *Attributes* pane and review the min and max values in the *Selected attribute* pane to confirm they are 0 and 1.
6. Click the *Save...* button, navigate to a suitable directory and type in a suitable name for this transformed dataset, such as `housing-normalize.arff`.
7. Close the *Weka Explorer* interface.

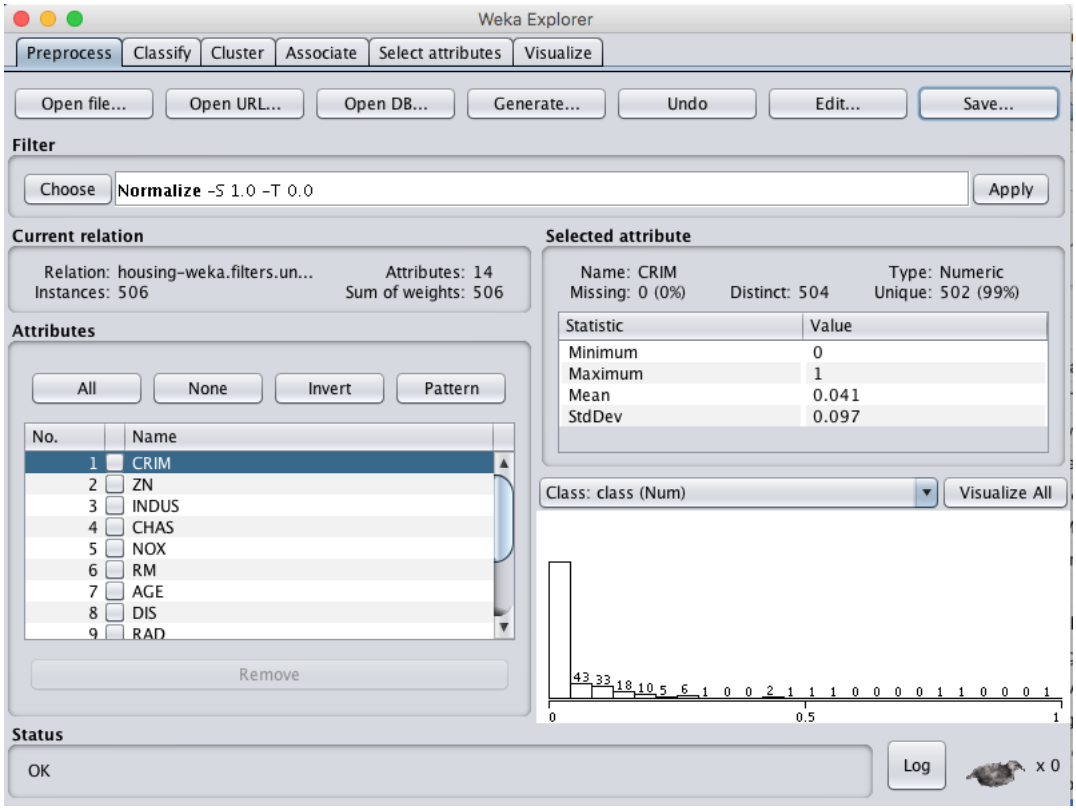


Figure 25.6: Weka Boston House Price Dataset Normalize Data Filter.

25.4.3 Standardized Dataset

We noted in the previous section that some of the attribute have a Gaussian-like distribution. We can rescale the data and take this distribution into account by using a standardizing filter. This will create a copy of the dataset where each attribute has a mean value of 0 and a standard deviation (mean variance) of 1. This may benefit algorithms in the next section that assume a Gaussian distribution in the input attributes, like Logistic Regression and Naive Bayes.

1. Open the *Weka Explorer*.
2. Open the modified numeric dataset `housing-numeric.arff`.
3. Click the *Choose* button in the *Filter* pane and choose the *unsupervised.attribute.Standardize* filter.
4. Click the *Apply* button to apply the filter.
5. Click each attribute in the *Attributes* pane and review the mean and standard deviation values in the *Selected attribute* pane to confirm they are 0 and 1 respectively.
6. Click the *Save...* button, navigate to a suitable directory and type in a suitable name for this transformed dataset, such as `housing-standardize.arff`.
7. Close the *Weka Explorer* interface.

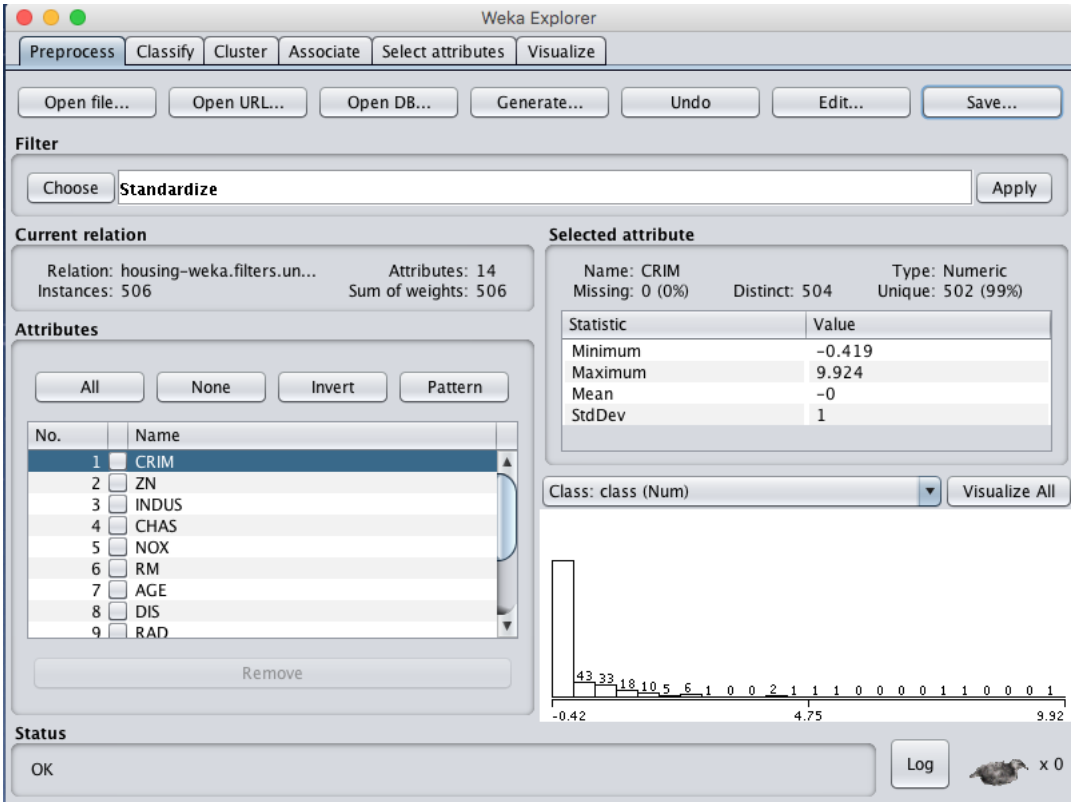


Figure 25.7: Weka Boston House Price Dataset Standardize Data Filter.

25.4.4 Feature Selection

We are unsure whether all of the attributes are really needed in order to make predictions. Here, we can use automatic feature selection to select only those most relevant attributes in the dataset.

1. Open the *Weka Explorer*.
2. Open the modified numeric dataset `housing-numeric.arff`.
3. Click the *Choose* button in the *Filter* pane and choose the *supervised.attribute.AttributeSelection* filter.
4. Click the *Apply* button to apply the filter.
5. Click each attribute in the *Attributes* pane and review the 5 chosen attributes.
6. Click the *Save...* button, navigate to a suitable directory and type in a suitable name for this transformed dataset, such as `housing-feature-selection.arff`.
7. Close the *Weka Explorer* interface.

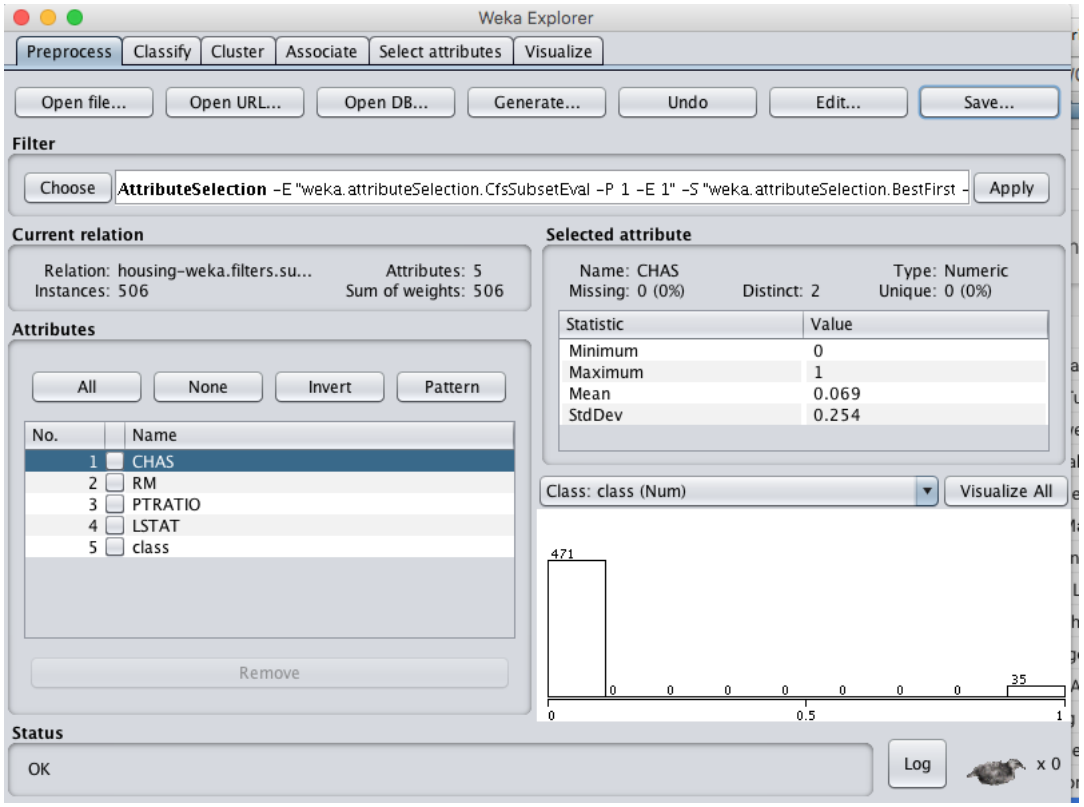


Figure 25.8: Weka Boston House Price Dataset Feature Selection Data Filter.

25.5 Evaluate Algorithms

Let's design an experiment to evaluate a suite of standard classification algorithms on the different views of the problem that we created.

- 1. Click the *Experimenter* button on the *Weka GUI Chooser* to launch the *Weka Experiment Environment*.
- 2. Click *New* to start a new experiment.
- 3. In the *Experiment Type* pane change the problem type from *Classification* to *Regression*.
- 4. In the *Datasets* pane click *Add new...* and select the following 4 datasets:
  - `housing-numeric.arff`
  - `housing-normalized.arff`
  - `housing-standardized.arff`
  - `housing-feature-selection.arff`
- 5. In the *Algorithms* pane click *Add new...* and add the following 6 regression algorithms:
  - *rules.ZeroR*
  - *bayes.SimpleLinearRegression*

  - *functions.SMOreg*
  - *lazy.IBk*
  - *trees.REPTree*

- 6. Select *IBk* in the list of algorithms and click the *Edit selected...* button.
- 7. Change *KNN* from *1* to *3* and click the *OK* button to save the settings.

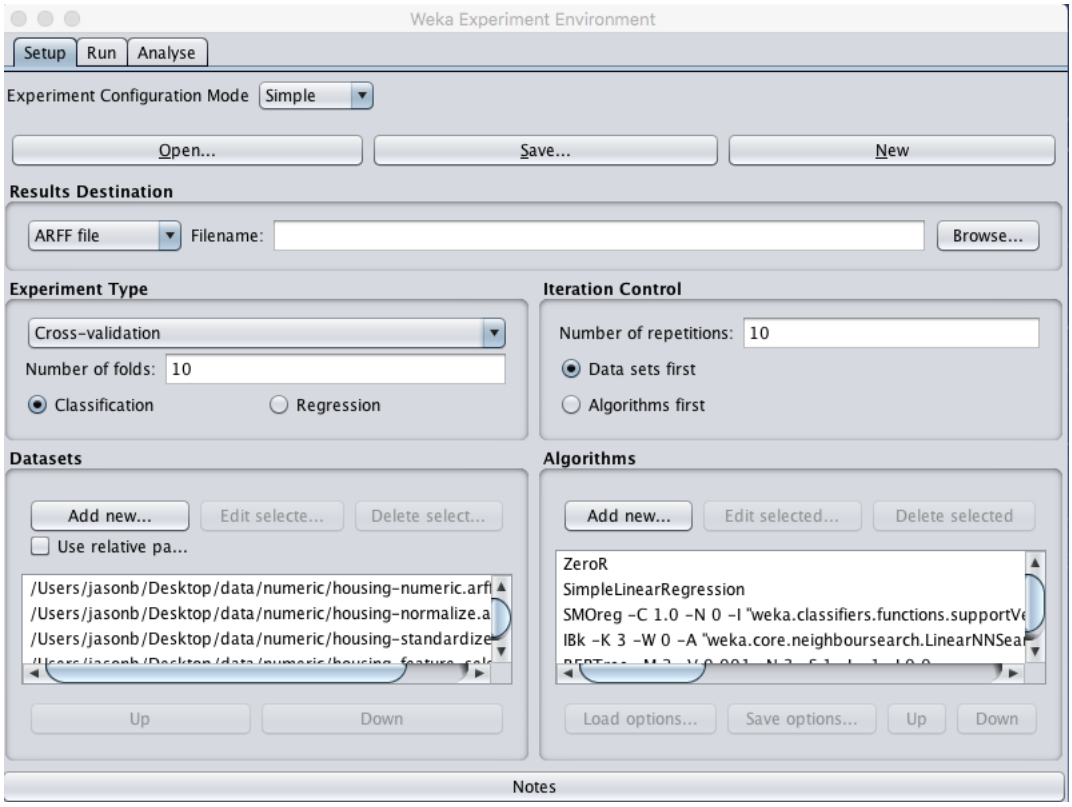


Figure 25.9: Weka Boston House Price Algorithm Comparison Experiment Design.

- 8. Click on *Run* tab and click the *Start* button to run the experiment. The experiment should complete in just a few seconds.
- 9. Click on the *Analyse* to open the *Analyse* tab. Click the *Experiment* button to load the results from the experiment.

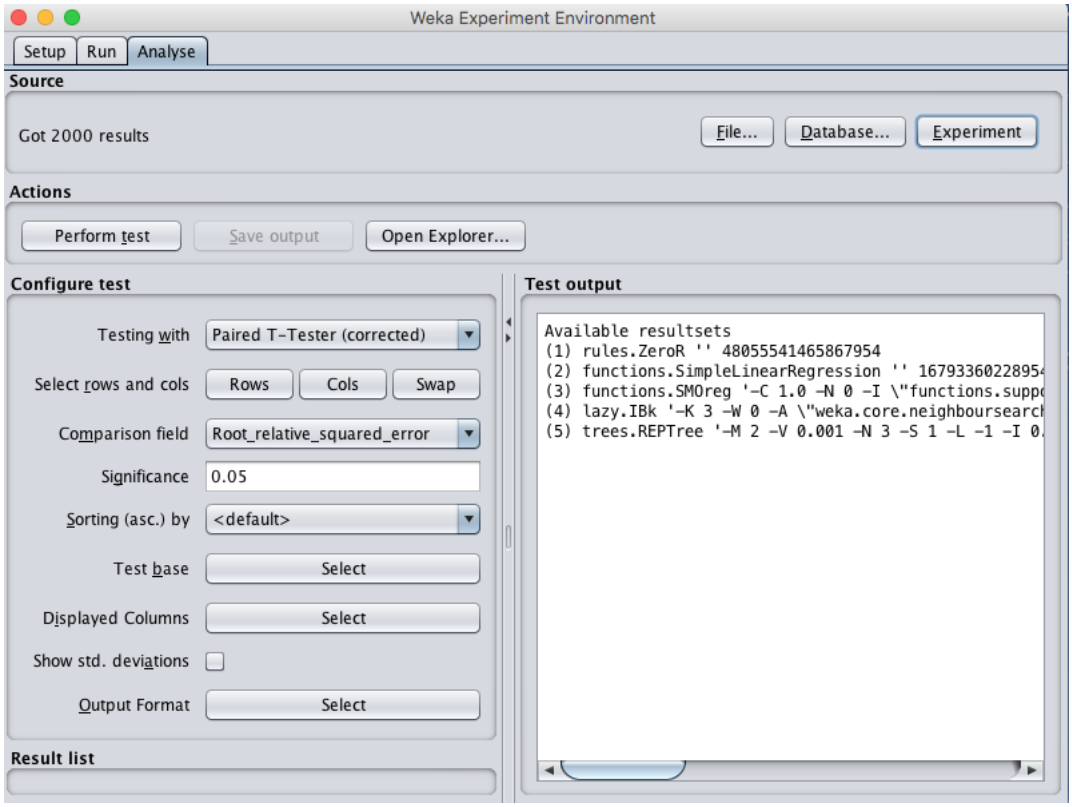


Figure 25.10: Weka Boston House Price Dataset Load Algorithm Comparison Experiment Results.

- 10. Change the *Comparison field* to *Root_mean_squared_error*.
- 11. Click the *Perform test* button to perform a pairwise test comparing all of the results to the results for *ZeroR*.

```
Dataset                 (1) rules.Z | (2) func (3) func (4) lazy (5) tree
-------------------------------------------------------------------------
housing                       9.11 |  6.22 *  4.95 *  4.41 *  4.64 *
normalized                    9.11 |  6.22 *  4.94 *  4.41 *  4.63 *
standardized                  9.11 |  6.22 *  4.95 *  4.41 *  4.64 *
feature-selection             9.11 |  6.22 *  5.19 *  4.27 *  4.64 *
-------------------------------------------------------------------------
                            (v/ /*) | (0/0/4) (0/0/4) (0/0/4) (0/0/4)


Key:
(1) rules.ZeroR
(2) functions.SimpleLinearRegression
(3) functions.SMOreg
(4) lazy.IBk
(5) trees.REPTree
```

Listing 25.1: Results From Algorithm Comparison Experiment.

Remember that the lower the RMSE the better. These results are telling. Firstly, we can see that all of the algorithms are better than the baseline skill of *ZeroR* and that the difference is

significant (a little * next to each score). We can also see that there does not appear to be much benefit across the evaluated algorithms from standardizing or normalizing the data. It does look like we may see a small improvement from the selecting less features view of the dataset, at least for *IBk*. Finally, it looks like the *IBk* (KNN) may have the lowest error. Let's investigate further.

- 12. Click the *Select* button for the *Test base* and choose the *IBk* algorithm as the new test base.
- 13. Click the *Perform test* button to rerun the analysis.

```
Dataset                 (4) lazy.IB | (1) rule (2) func (3) func (5) tree
----------------------------------------------------------------------------
housing                        4.41 |  9.11 v  6.22 v  4.95    4.64
normalized                     4.41 |  9.11 v  6.22 v  4.94    4.63
standardized                   4.41 |  9.11 v  6.22 v  4.95    4.64
feature-selection              4.27 |  9.11 v  6.22 v  5.19 v  4.64
----------------------------------------------------------------------------
                            (v/ /*) | (4/0/0) (4/0/0) (1/3/0) (0/4/0)


Key:
(1) rules.ZeroR
(2) functions.SimpleLinearRegression
(3) functions.SMOreg
(4) lazy.IBk
(5) trees.REPTree
```

Listing 25.2: Results From Algorithm Comparison Experiment With New Test Base.

We can see that it does look there is a difference between *IBk* and the other algorithms is significant, except when comparing to the *REPTree* algorithm and *SMOreg*. Both the *IBk* and *SMOreg* algorithms are nonlinear regression algorithms that can be further tuned, something we can look at in the next section.

25.6 Tune Algorithm Performance

Two algorithms were identified in the previous section as performing well on the problem and good candidates for further tuning: k-Nearest Neighbors (*IBk*) and Support Vector Regression (*SMOreg*). In this section we will design experiments to tune both of these algorithms and see if we can further decrease the root mean squared error. We will use the baseline `housing-numeric.arff` dataset for these experiments as there did not appear to be a large performance difference between using this variation of the dataset and the other views.

25.6.1 Tune k-Nearest Neighbors

In this section we will tune the *IBk* algorithm. Specifically, we will investigate using different values for the k parameter.

- 1. Open the *Weka Experiment Environment* interface.

- 2. Click *New* to start a new experiment.
- 3. In the *Experiment Type* pane change the problem type from *Classification* to *Regression*.
- 4. In the *Datasets* pane add the `housing-numeric.arff` dataset.
- 5. In the *Algorithms* pane the *lazy.IBk* algorithm and set the value of the *K* parameter to 1 (the default). Repeat this process and add the following additional configurations for the *IBk* algorithms:
  - *lazy.IBk* with *K=3*
  - *lazy.IBk* with *K=5*
  - *lazy.IBk* with *K=7*
  - *lazy.IBk* with *K=9*

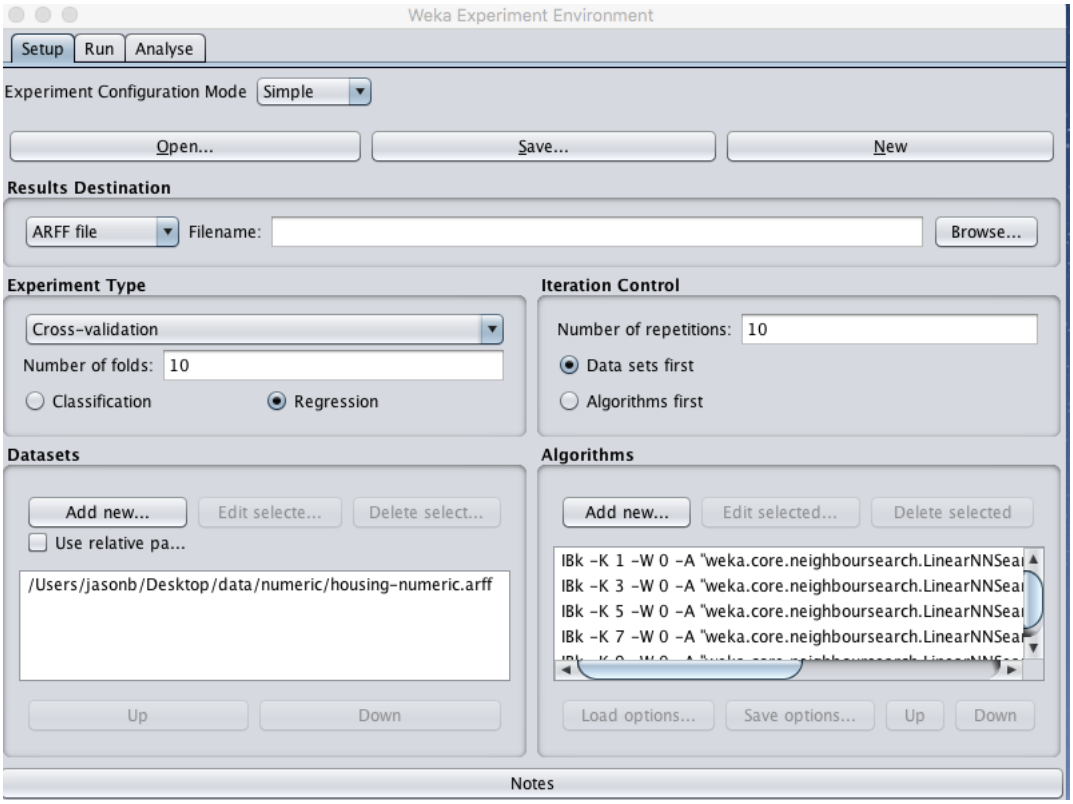


Figure 25.11: Weka Boston House Price Dataset Tune *k*-Nearest Neighbors Algorithm.

- 6. Click on *Run* tab and click the *Start* button to run the experiment. The experiment should complete in just a few seconds.
- 7. Click on the *Analyse* ttab. Click the *Experiment* button to load the results from the experiment.
- 8. Change the *Comparison field* to *Root_mean_squared_error*.
- 9. Click the *Perform test* button to perform a pairwise test.

```
Dataset                 (1) lazy.IB | (2) lazy (3) lazy (4) lazy (5) lazy
----------------------------------------------------------------------------
housing                (100)  4.61 |  4.41    4.71    5.00    5.16
----------------------------------------------------------------------------
                            (v/ /*) | (0/1/0) (0/1/0) (0/1/0) (0/1/0)


Key:
(1) lazy.IBk '-K 1 -W 0 -A \"weka.core.neighboursearch.LinearNNSearch -A
    \\\"weka.core.EuclideanDistance -R first-last\\\"\"' -3080186098777067172
(2) lazy.IBk '-K 3 -W 0 -A \"weka.core.neighboursearch.LinearNNSearch -A
    \\\"weka.core.EuclideanDistance -R first-last\\\"\"' -3080186098777067172
(3) lazy.IBk '-K 5 -W 0 -A \"weka.core.neighboursearch.LinearNNSearch -A
    \\\"weka.core.EuclideanDistance -R first-last\\\"\"' -3080186098777067172
(4) lazy.IBk '-K 7 -W 0 -A \"weka.core.neighboursearch.LinearNNSearch -A
    \\\"weka.core.EuclideanDistance -R first-last\\\"\"' -3080186098777067172
(5) lazy.IBk '-K 9 -W 0 -A \"weka.core.neighboursearch.LinearNNSearch -A
    \\\"weka.core.EuclideanDistance -R first-last\\\"\"' -3080186098777067172
```

Listing 25.3: Results From Tuning KNN Experiment.

We see that *K=3* achieved the lowest error.

- 10. Click the *Select* button for the *Test base* and choose the *lazy.IBk* algorithm with *K=3* as the new test base.
- 11. Click the *Perform test* button to rerun the analysis.

```
Dataset                 (2) lazy.IB | (1) lazy (3) lazy (4) lazy (5) lazy
----------------------------------------------------------------------------
housing                (100)  4.41 |  4.61    4.71 v  5.00 v  5.16 v
----------------------------------------------------------------------------
                            (v/ /*) | (0/1/0) (1/0/0) (1/0/0) (1/0/0)


Key:
(1) lazy.IBk '-K 1 -W 0 -A \"weka.core.neighboursearch.LinearNNSearch -A
    \\\"weka.core.EuclideanDistance -R first-last\\\"\"' -3080186098777067172
(2) lazy.IBk '-K 3 -W 0 -A \"weka.core.neighboursearch.LinearNNSearch -A
    \\\"weka.core.EuclideanDistance -R first-last\\\"\"' -3080186098777067172
(3) lazy.IBk '-K 5 -W 0 -A \"weka.core.neighboursearch.LinearNNSearch -A
    \\\"weka.core.EuclideanDistance -R first-last\\\"\"' -3080186098777067172
(4) lazy.IBk '-K 7 -W 0 -A \"weka.core.neighboursearch.LinearNNSearch -A
    \\\"weka.core.EuclideanDistance -R first-last\\\"\"' -3080186098777067172
(5) lazy.IBk '-K 9 -W 0 -A \"weka.core.neighboursearch.LinearNNSearch -A
    \\\"weka.core.EuclideanDistance -R first-last\\\"\"' -3080186098777067172
```

Listing 25.4: Results From Tuning KNN Experiment With New Test Base.

We can see that *K=3* is significantly different and better than all the other configurations except *K=1*. We learned that we cannot trivially get a significant lift in performance by tuning k of *IBk*. Further tuning may look at using different distance measures or tuning *IBk* parameters using different views of the dataset, such as the view with selected features.

25.6.2 Tune Support Vector Machines

In this section we will tune the *SMOreg* algorithm. Specifically, we will investigate using different values for the *exponent* parameter for the *Polynomial* kernel.

- 1. Open the *Weka Experiment Environment* interface.
- 2. Click *New* to start a new experiment.
- 3. In the *Experiment Type* pane change the problem type from *Classification* to *Regression*.
- 4. In the *Datasets* pane add the `housing-numeric.arff` dataset.
- 5. In the *Algorithms* pane the *functions.SMOreg* algorithm and set the value of the *exponent* parameter of the *Polynomial* kernel to 1 (the default). Repeat this process and add the following additional configurations for the *SMOreg* algorithms:
  - *functions.SMOreg*, *kernel=Polynomial*, *exponent=2*
  - *functions.SMOreg*, *kernel=Polynomial*, *exponent=3*

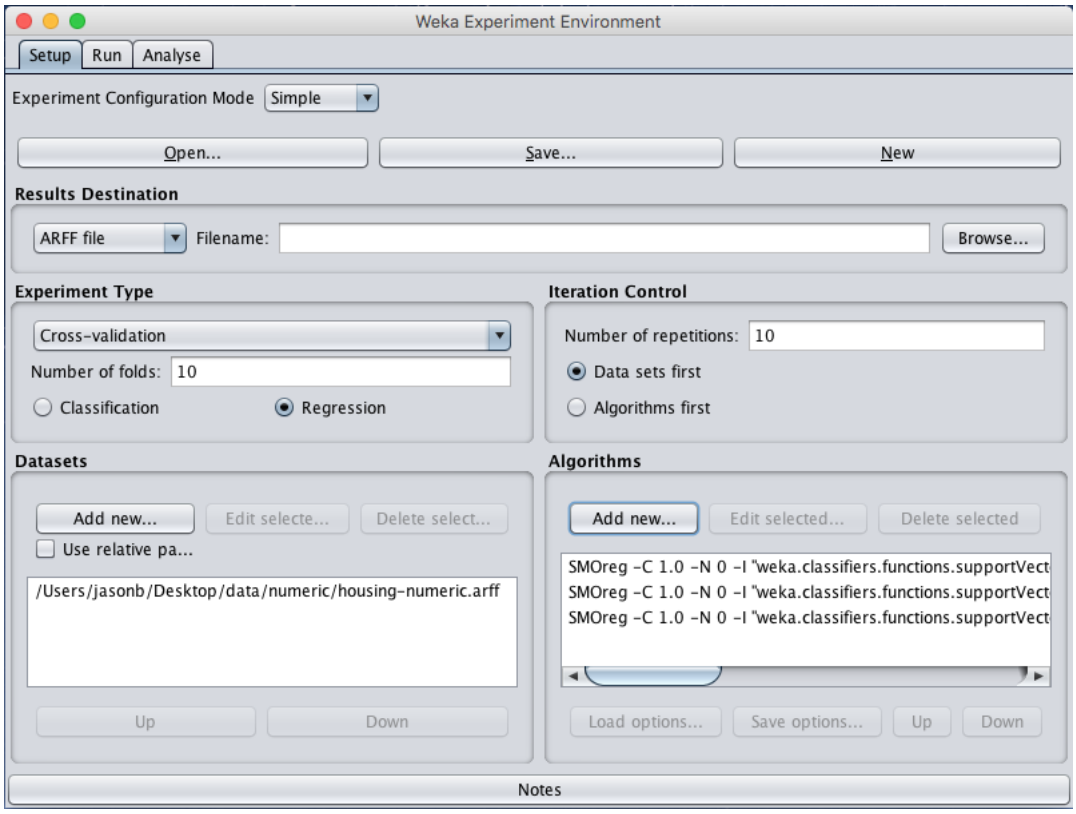


Figure 25.12: Weka Boston House Price Dataset Tune the Support Vector Regression Algorithm.

- 6. Click on *Run* tab and click the *Start* button to run the experiment. The experiment should complete in a about 10 minutes, depending on the speed of your system.
- 7. Click on the *Analyse* tab. Click the *Experiment* button to load the results from the experiment.

- 8. Change the *Comparison field* to *Root_mean_squared_error*.
- 9. Click the *Perform test* button to perform a pairwise test.

```
Dataset                 (1) functio | (2) func (3) func
----------------------------------------------------------
housing                (100)  4.95 |  3.57 *  3.41 *
----------------------------------------------------------
                            (v/ /*) | (0/0/1) (0/0/1)


Key:
(1) functions.SMOreg '-C 1.0 -N 0 -I \"functions.supportVector.RegSMOImproved -T 0.001 -V
    -P 1.0E-12 -L 0.001 -W 1\" -K \"functions.supportVector.PolyKernel -E 1.0 -C 250007\"'
    -7149606251113102827
(2) functions.SMOreg '-C 1.0 -N 0 -I \"functions.supportVector.RegSMOImproved -T 0.001 -V
    -P 1.0E-12 -L 0.001 -W 1\" -K \"functions.supportVector.PolyKernel -E 2.0 -C 250007\"'
    -7149606251113102827
(3) functions.SMOreg '-C 1.0 -N 0 -I \"functions.supportVector.RegSMOImproved -T 0.001 -V
    -P 1.0E-12 -L 0.001 -W 1\" -K \"functions.supportVector.PolyKernel -E 3.0 -C 250007\"'
    -7149606251113102827
```

Listing 25.5: Results From Tuning SVR Experiment.

It looks like the kernel with an *exponent=3* achieved the best result. Set this as the *Test base* and rerun the analysis.

```
Dataset                 (3) functio | (1) func (2) func
----------------------------------------------------------
housing                (100)  3.41 |  4.95 v  3.57
----------------------------------------------------------
                            (v/ /*) | (1/0/0) (0/1/0)


Key:
(1) functions.SMOreg '-C 1.0 -N 0 -I \"functions.supportVector.RegSMOImproved -T 0.001 -V
    -P 1.0E-12 -L 0.001 -W 1\" -K \"functions.supportVector.PolyKernel -E 1.0 -C 250007\"'
    -7149606251113102827
(2) functions.SMOreg '-C 1.0 -N 0 -I \"functions.supportVector.RegSMOImproved -T 0.001 -V
    -P 1.0E-12 -L 0.001 -W 1\" -K \"functions.supportVector.PolyKernel -E 2.0 -C 250007\"'
    -7149606251113102827
(3) functions.SMOreg '-C 1.0 -N 0 -I \"functions.supportVector.RegSMOImproved -T 0.001 -V
    -P 1.0E-12 -L 0.001 -W 1\" -K \"functions.supportVector.PolyKernel -E 3.0 -C 250007\"'
    -7149606251113102827
```

Listing 25.6: Results From Tuning SVR Experiment With New Test Base.

The results with the *exponent=3* are statistically significantly better than *exponent=1*, but not *exponent=2*. Either could be chosen although the lower complexity exponent=2 may be faster and less fragile.

25.7 Evaluate Ensemble Algorithms

In the section on evaluating algorithms, we noticed that the *REPtree* also achieved good results, not statistically significantly different from *IBk* or *SMOreg*. In this section we consider ensemble

varieties of regression trees using bagging. As with the previous section on algorithm tuning, we will use the numeric copy of the housing dataset.

- 1. Open the *Weka Experiment Environment* interface.
- 2. Click *New* to start a new experiment.
- 3. In the *Experiment Type* pane change the problem type from *Classification* to *Regression*.
- 4. In the *Datasets* pane add the `housing-numeric.arff` dataset.
- 5. In the *Algorithms* pane add the following algorithms:
  - *trees.REPTree*
  - *trees.RandomForest*
  - *meta.Bagging*

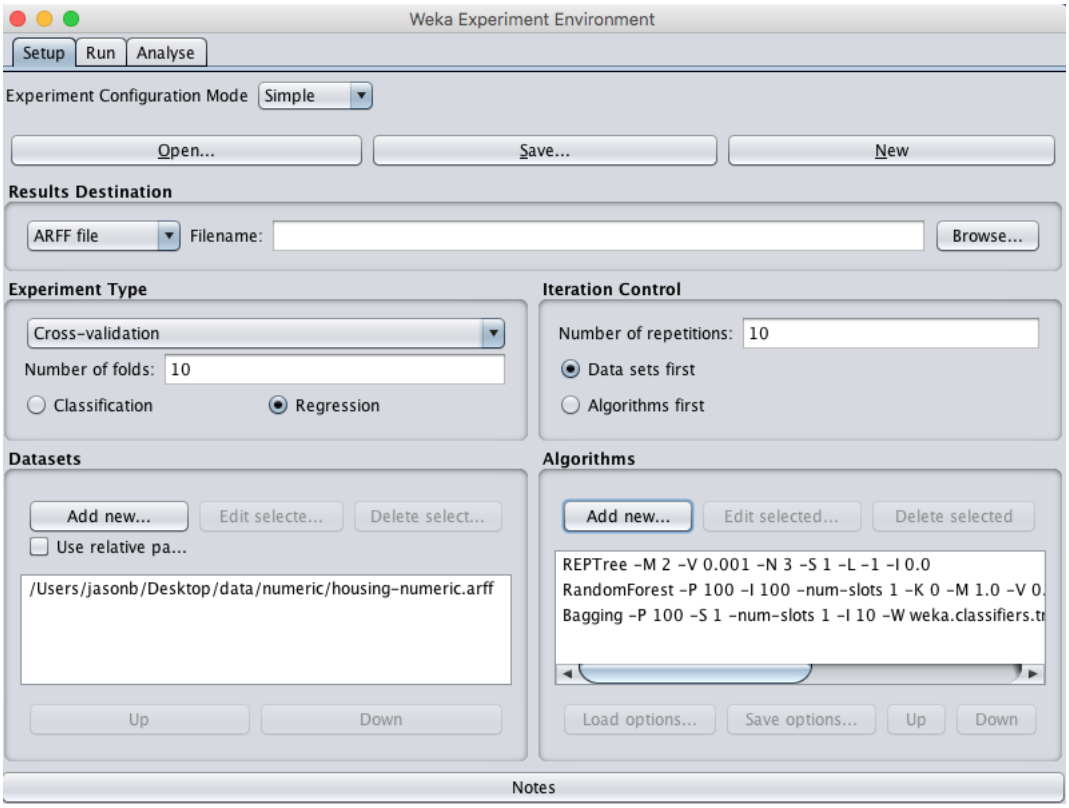


Figure 25.13: Weka Boston House Price Dataset Ensemble Experiment Design.

- 6. Click on *Run* tab and click the *Start* button to run the experiment. The experiment should complete in just a few seconds.
- 7. Click on the *Analyse* tab. Click the *Experiment* button to load the results from the experiment.
- 8. Change the *Comparison field* to *Root_mean_squared_error*.

- 9. Click the *Perform test* button to perform a pairwise test.

```
Dataset                 (1) trees.R | (2) tree (3) meta
----------------------------------------------------------
housing                (100)  4.64 |  3.14 *  3.78 *
----------------------------------------------------------
                            (v/ /*) | (0/0/1) (0/0/1)


Key:
(1) trees.REPTree '-M 2 -V 0.001 -N 3 -S 1 -L -1 -I 0.0' -9216785998198681299
(2) trees.RandomForest '-P 100 -I 100 -num-slots 1 -K 0 -M 1.0 -V 0.001 -S 1'
    1116839470751428698
(3) meta.Bagging '-P 100 -S 1 -num-slots 1 -I 10 -W trees.REPTree -- -M 2 -V 0.001 -N 3 -S
    1 -L -1 -I 0.0' -115879962237199703
```

Listing 25.7: Results From Ensemble Algorithm Comparison Experiment.

- 10. The results suggest suggest that random forest may have the best performance. Select *trees.RandomForest* as the *Test base* and rerun the analysis.

```
Dataset                 (2) trees.R | (1) tree (3) meta
----------------------------------------------------------
housing                (100)  3.14 |  4.64 v  3.78 v
----------------------------------------------------------
                            (v/ /*) | (1/0/0) (1/0/0)


Key:
(1) trees.REPTree '-M 2 -V 0.001 -N 3 -S 1 -L -1 -I 0.0' -9216785998198681299
(2) trees.RandomForest '-P 100 -I 100 -num-slots 1 -K 0 -M 1.0 -V 0.001 -S 1'
    1116839470751428698
(3) meta.Bagging '-P 100 -S 1 -num-slots 1 -I 10 -W trees.REPTree -- -M 2 -V 0.001 -N 3 -S
    1 -L -1 -I 0.0' -115879962237199703
```

Listing 25.8: Results From Ensemble Algorithm Comparison Experiment With a New Test Base.

This is very encouraging, the result for *RandomForest* is the best we have seen on this problem so far and the difference is statistically significant when compared to *Bagging* and *REPtree*. To wrap this up, let's choose *RandomForest* as the preferred model for this problem. We could perform model selection and evaluate whether the difference in performance of RandomForest is statistically significant when compared to *IBk* with *K=1* and *SMOreg* with *exponent=3*. This is left as an exercise for the reader.

- 11. Check *Show std. deviations* to show standard deviations of the results.

- 12. Click the *Select* button for *Displayed Columns* and choose *trees.RandomForest*, click *Select* to accept the selection. This will just show the results for the Random Forest algorithm.

- 13. Click *Perform test* to re-run the analysis.

We now have a final result we can use to describe our model.

```
Dataset                  (2) trees.RandomFor
-----------------------------------------------
housing                 (100)  3.14(0.65) |
-----------------------------------------------
(v/ /*) |


Key:
(2) trees.RandomForest '-P 100 -I 100 -num-slots 1 -K 0 -M 1.0 -V 0.001 -S 1'
    1116839470751428698
```

Listing 25.9: Results For Final Random Forest Performance.

We can see that the estimated error of the model on unseen data is 3.14 (thousands of dollars) with a standard deviation of 0.64.

25.8 Finalize Model and Present Results

We can create a final version of our model trained on all of the training data and save it to file.

1. Open the *Weka Explorer* and load the `housing-numeric.arff` dataset.
2. Click on the *Classify* tab.
3. Select the *trees.RandomForest* algorithm.
4. Change the *Test options* from *Cross-Validation* to *Use training set*.
5. Click the *Start* button to create the final model.
6. Right click on the result item in the *Result list* and select *Save model*. Select a suitable location and type in a suitable name, such as *housing-randomforest* for your model.

This model can then be loaded at a later time and used to make predictions on new data. We can use the mean and standard deviation of the model accuracy collected in the last section to help quantify the expected variability in the estimated accuracy of the model on unseen data. We can generally expect that the performance of the model on unseen data will be 3.14 plus or minus (2×0.64) or 1.28. We can restate this as the model will have an error between 1.86 and 4.42 in thousands of dollars.

25.9 Summary

In this project you discovered how to work through a regression machine learning problem using the Weka machine learning workbench. Specifically, you learned.

- How to load, analysis and prepare views of your dataset in Weka.
- How to evaluate a suite of regression machine learning algorithms using the *Weka Experiment Environment*.
- How to tune well performing models and investigate related ensemble methods in order to lift performance.

25.9.1 Next

This was your final project and concludes Part III of the book. Next, in Part IV we conclude this book and start off by taking a look at how far you have come.

Part IV

Conclusions

Chapter 26

How Far You Have Come

You made it. Well done. Take a moment and look back at how far you have come.

- You started off with an interest in applied machine learning learning and a strong desire to be able to work through your own projects end-to-end.
- You downloaded, installed and started using Weka, perhaps for the first time, learning how to load standard machine learning datasets and best prepare the data for modeling.
- Slowly and steadily over the course of a number of lessons you learned how to use the various different features of the Weka platform for developing models and improving their performance.
- Building upon the recipes for common machine learning tasks you worked through your first machine learning problems with using Weka.
- Using standard templates, the recipes and experience you have gathered, you are now capable of working through new and different predictive modeling machine learning problems on your own.

Don't make light of this. You have come a long way in a short amount of time. You have developed the important and valuable skill of being able to work through applied machine learning problems end-to-end using Weka. This is the best and most powerful platform for applied machine learning for beginners. The sky's the limit. I want to take a moment and sincerely thank you for letting me help you start your machine learning journey with in applied machine learning. I hope you keep learning and have fun as you continue to master machine learning.

Chapter 27

Getting More Help

The Weka machine learning workbench is an easy to use and powerful platform for applied machine learning. Even though it is easy to use, you may still require some help or advice when using it on your own problems. In this chapter you will discover resources that you can use to get more help with Weka. After reading this chapter you will know:

- About the documentation that is installed with Weka on your workstation.
- Resources online that you can consult to get help with your technical concerns with Weka.
- Where you can go to ask your own unanswered questions about Weka.

Let's get started.

27.1 Weka Offline Documentation

Your installation of Weka provides documentation that you can consult for simpler questions around how to use the platform. Your installation directory contains an HTML file called documentation.html that links to all of the documentation provided with your installation including:

- The Weka User Manual.
- Weka API documentation.

Figure 27.1: Weka Documentation

27.1.1 Weka Manual

The Weka manual provides information on how to use the Weka software. This includes:

- How to use the Weka command line interface.
- How to use the Weka graphical user interface.
- About the Weka data file formats.
- And additional technical documentation.

The documentation is very good and it is the first place you should check if you have a question about the usage of the Weka software.

WEKA Manual
for Version 3-8-0

Remco R. Bouckaert
Eibe Frank
Mark Hall
Richard Kirkby
Peter Reutemann
Alex Seewald
David Scuse

April 14, 2016

Figure 27.2: Weka Manual

27.1.2 Weka Java API Documentation

This is the documentation for the Java Application Programming Interface (API). It is for developers that want to write programs that make use of the Weka Java interface. The structure and style of the documentation will be familiar to Java programmers as it uses the standard Javadoc format.

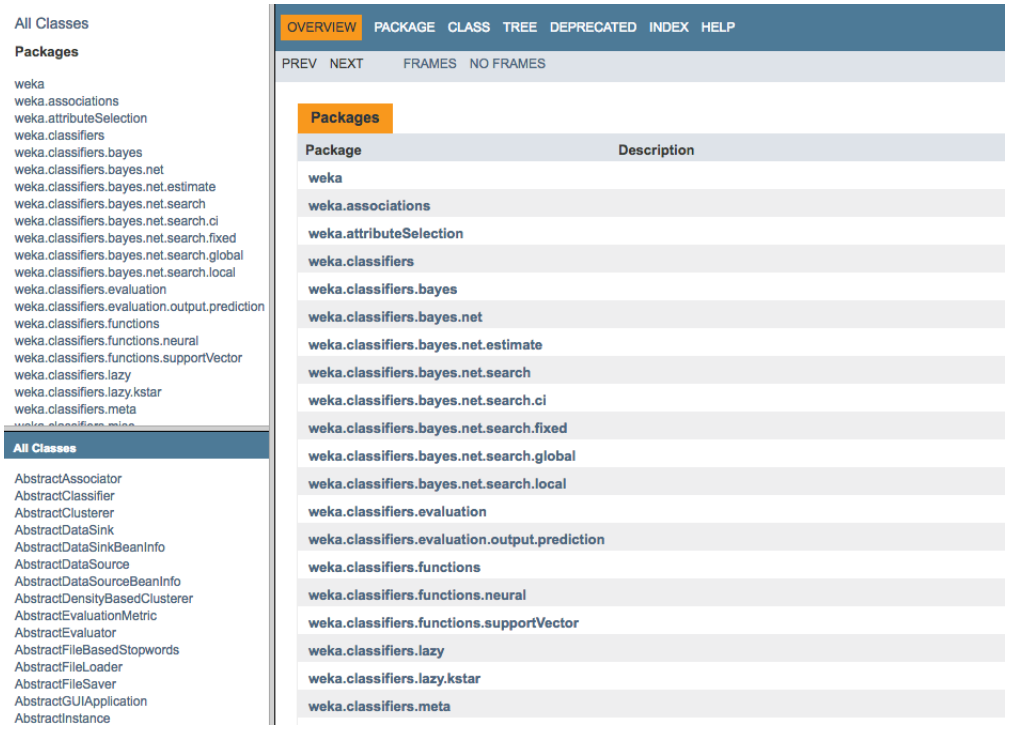


Figure 27.3: Weka Java API Documentation

27.2 Weka Online Documentation

Weka provides a few online sources of documentation including:

- Weka Wiki.
- Weka Mailing List.
- Other Official Resources.

27.2.1 Weka Wiki

The Weka Wiki provides helpful how-to articles on a range of Weka topics[1]. Below are some wiki page that you may find valuable if you are looking to resolve a specific problem:

- Weka Frequently Asked Questions:
  http://weka.wikispaces.com/Frequently+Asked+Questions
- Weka Troubleshooting:
  http://weka.wikispaces.com/Troubleshooting
- Weka Related Projects:
  http://weka.wikispaces.com/Related+Projects

There are a lot of wiki pages, use the search feature to locate pages on a specific topic.

[1] http://weka.wikispaces.com

27.2.2 Weka Mailing List

The Weka mailing list is an email list that you can join to interact with the core Weka community[2]. This includes both answering technical questions about Weka and getting answers to your own technical questions. There is an etiquette to email lists in general and especially on this email list. A big part of this etiquette is checking if your question has been asked and answered before. You can do this by reviewing the Weka email list archives using the search feature. The archives are rich and I recommend spending some time reading through the history.

27.2.3 Other Official Resources

Other online resources that you may find useful include:

- Weka homepage:
  http://www.cs.waikato.ac.nz/~ml/weka/
- Pentaho Data Mining Community Documentation:
  http://goo.gl/YVnxbi
- Data Mining: Practical Machine Learning Tools and Techniques (book):
  http://amzn.to/25YuPcW

27.3 Stack Overflow

The Stack Overflow website is a technical question and answer community for developers. Like the Weka email list, it is good etiquette to check if your question has already been asked and answered before. Use the search feature and pay close attention to the *Related* chapters on the right hand side of the page when reading a given question and answer page. Questions about Weka are tagged with the *Weka* keyword. Filtering questions by this tag allows you to review all Weka related questions[3].

27.4 Summary

In this chapter you discovered resources that you can use to get help with Weka. You learned:

- Your Weka installation includes a Manual with more information about how to use the Weka interface.
- The Weka Wiki is the go-to place for additional technical questions about how to use Weka.
- The mailing list and Stack Overflow are great places to ask unanswered technical questions on Weka.

I am always here to help if you have any questions. You can email me directly via `jason@MachineLearningMastery.com` and put this book title in the subject of your email.

[2] https://list.waikato.ac.nz/mailman/listinfo/wekalist
[3] http://stackoverflow.com/questions/tagged/weka