Machine Learning for Source Code Classification

Dr Timothy Chappell
Queensland University of Technology

Classification

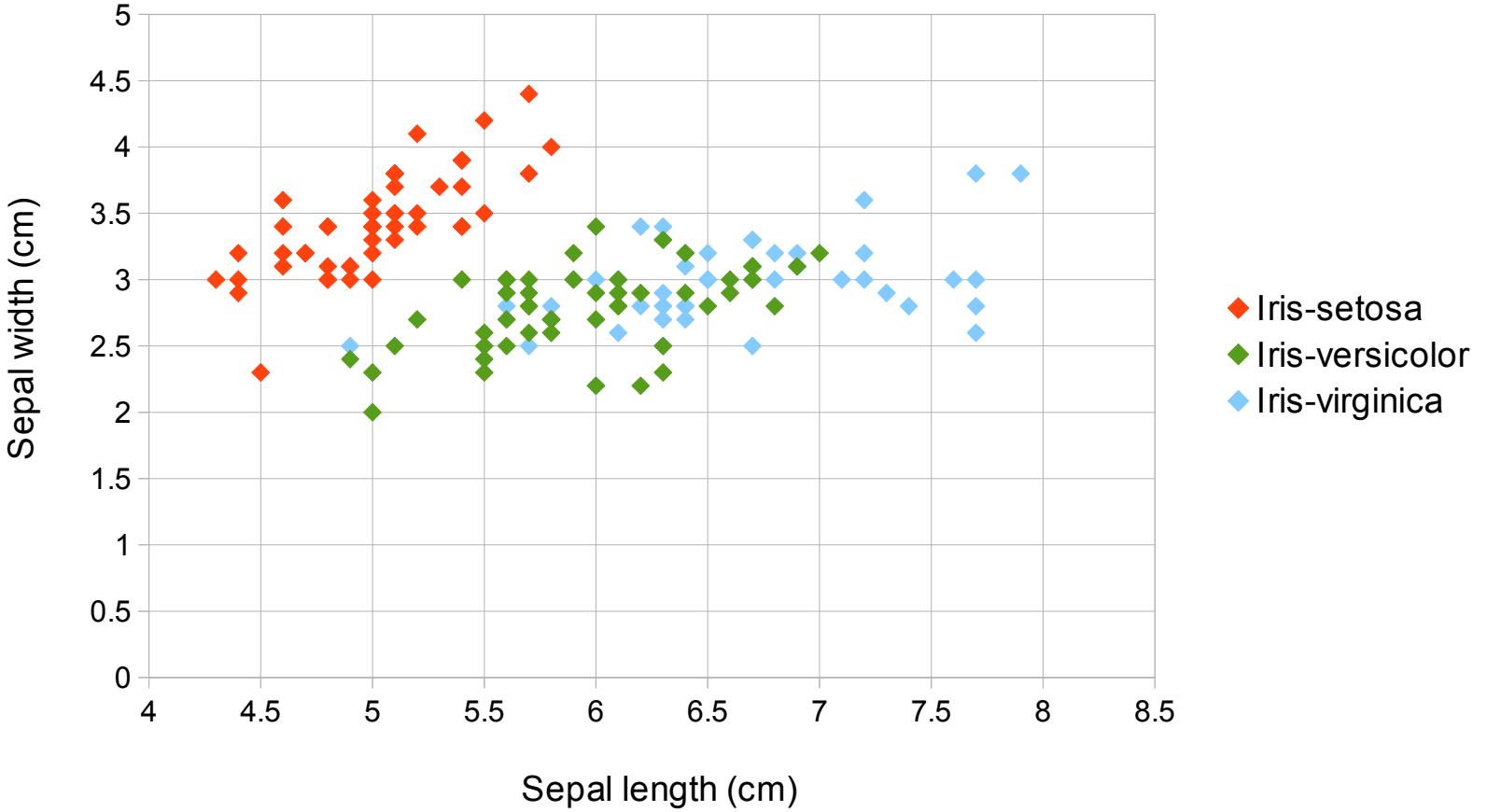
- Classification is the task of assigning *classes* to items based on the *features* that characterise those items
- One famous example in machine learning is the Iris data set, for classifying Iris plants according to the physical features of the individual specimens

Iris plant data set

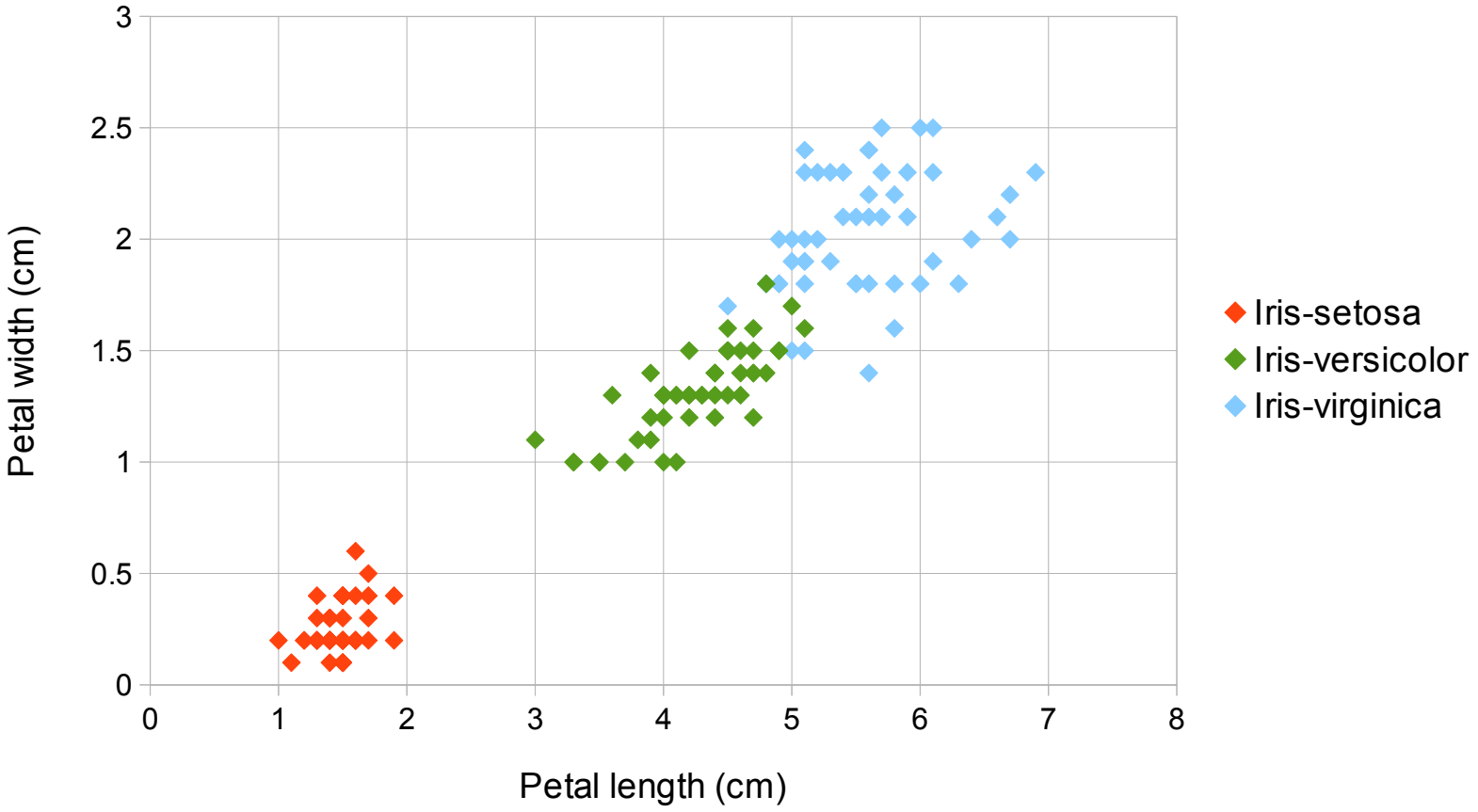
Sepal length (cm)	Sepal width (cm)	Petal length (cm)	Petal width (cm)	Class
5.1	3.5	1.4	0.2	Iris-setosa
4.9	3	1.4	0.2	Iris-setosa
4.7	3.2	1.3	0.2	Iris-setosa
4.6	3.1	1.5	0.2	Iris-setosa
7	3.2	4.7	1.4	Iris-versicolor
6.4	3.2	4.5	1.5	Iris-versicolor
6.9	3.1	4.9	1.5	Iris-versicolor
5.5	2.3	4	1.3	Iris-versicolor
6.3	3.3	6	2.5	Iris-virginica
5.8	2.7	5.1	1.9	Iris-virginica
7.1	3	5.9	2.1	Iris-virginica
6.3	2.9	5.6	1.8	Iris-virginica

(150 total instances, 50 belonging to each class)

Iris plant data set



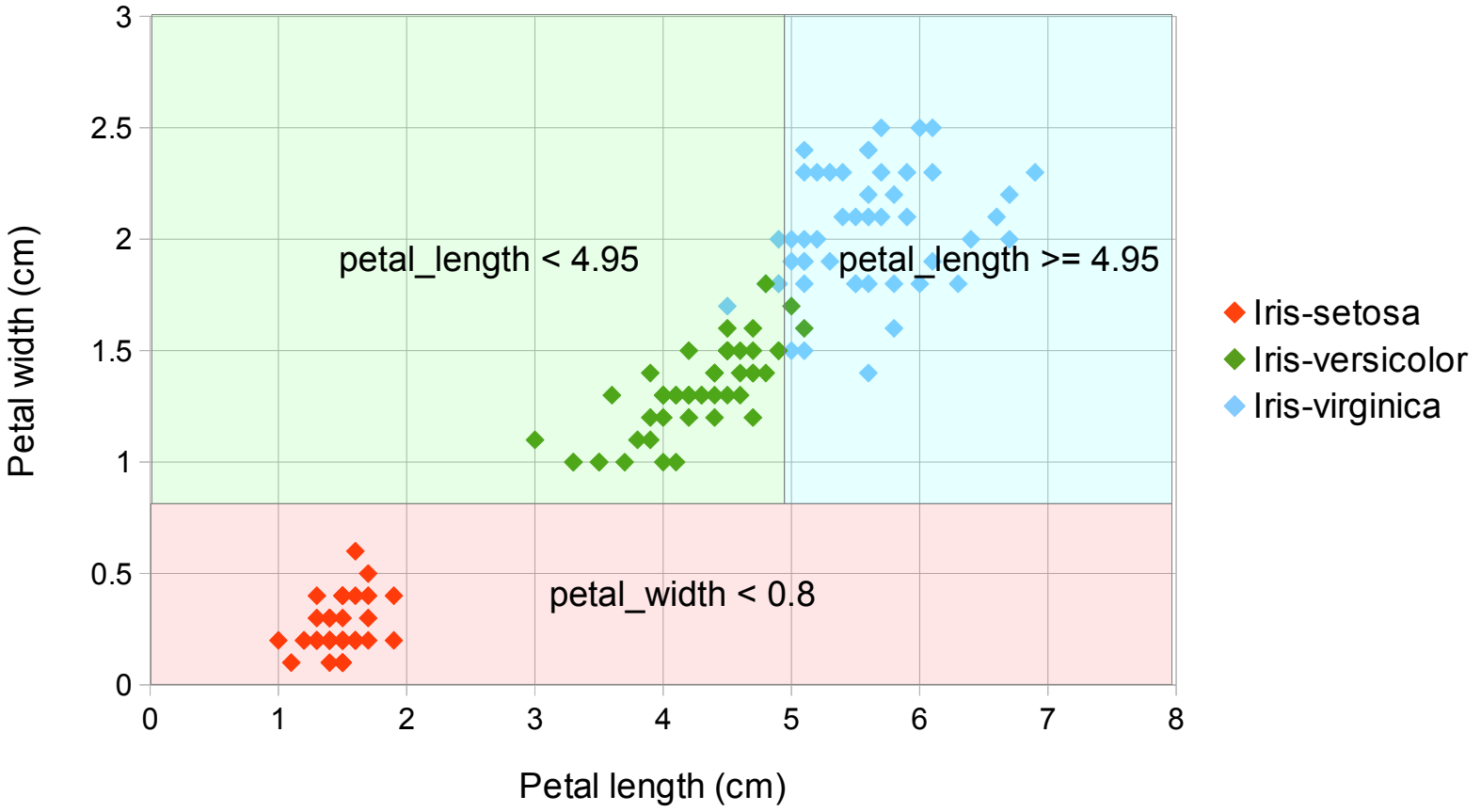
Iris plant data set



Possible classification rules

- If petal_width < 0.8
  - Iris-setosa (100% certainty)
- If petal_width >= 0.8
  - If petal_length < 4.95
    - Iris-versicolor (94% certainty)
  - If petal_length >= 4.95
    - Iris-virginica (96% certainty)

Iris data set with classification rules



Pitfalls when designing classification rules

- Classification rules should not be fit too tightly around the training data
- The purpose of classification rules is to classify new instances where the class is not known and overfitting the classification rules can reduce the accuracy of classification in the long run

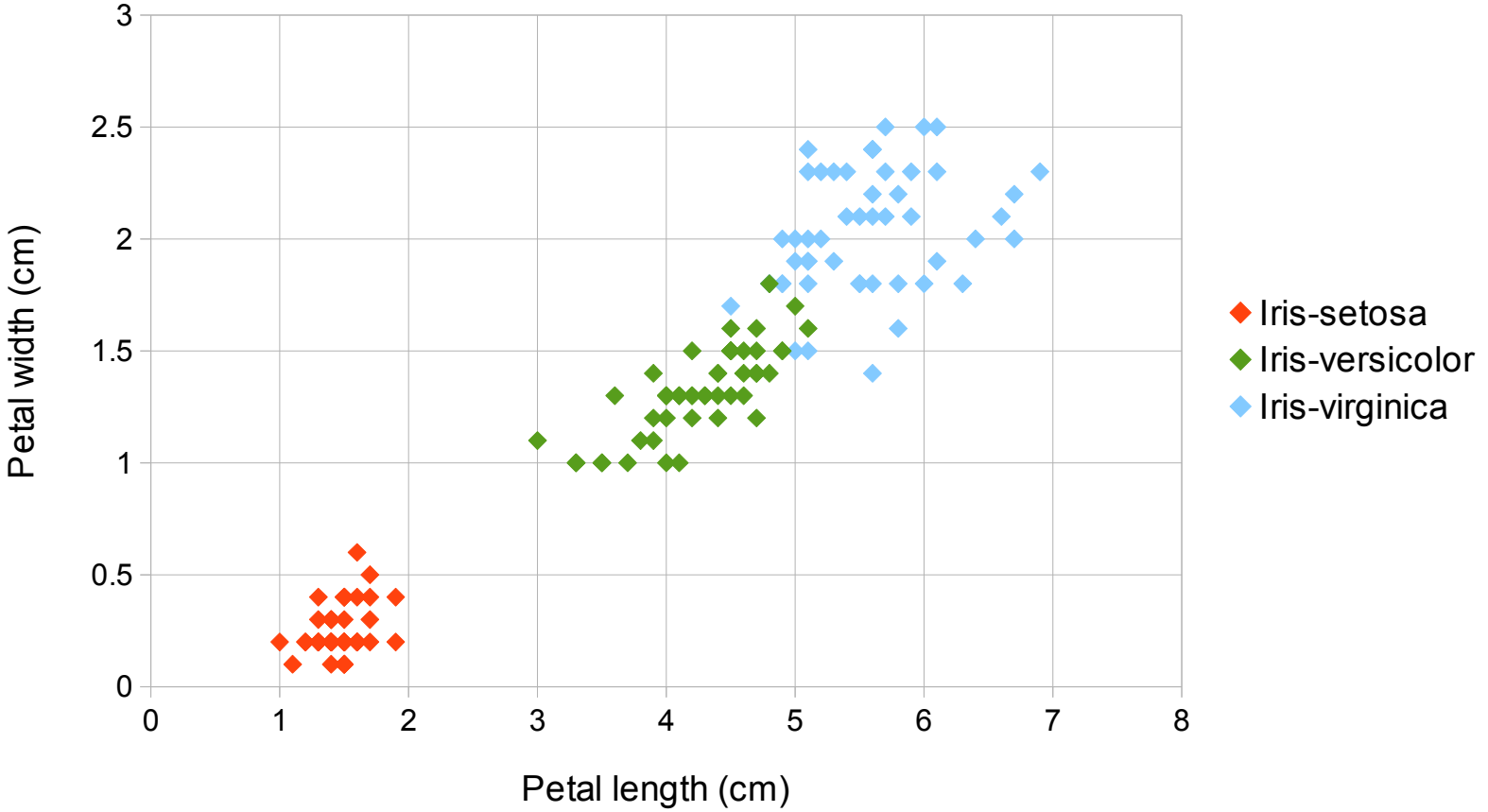
Machine learning

- When there are few features and when the relationship between features and classes is simple, classification rules can often be created manually
- In more complex cases, machine learning can be used to infer classification rules from the training data

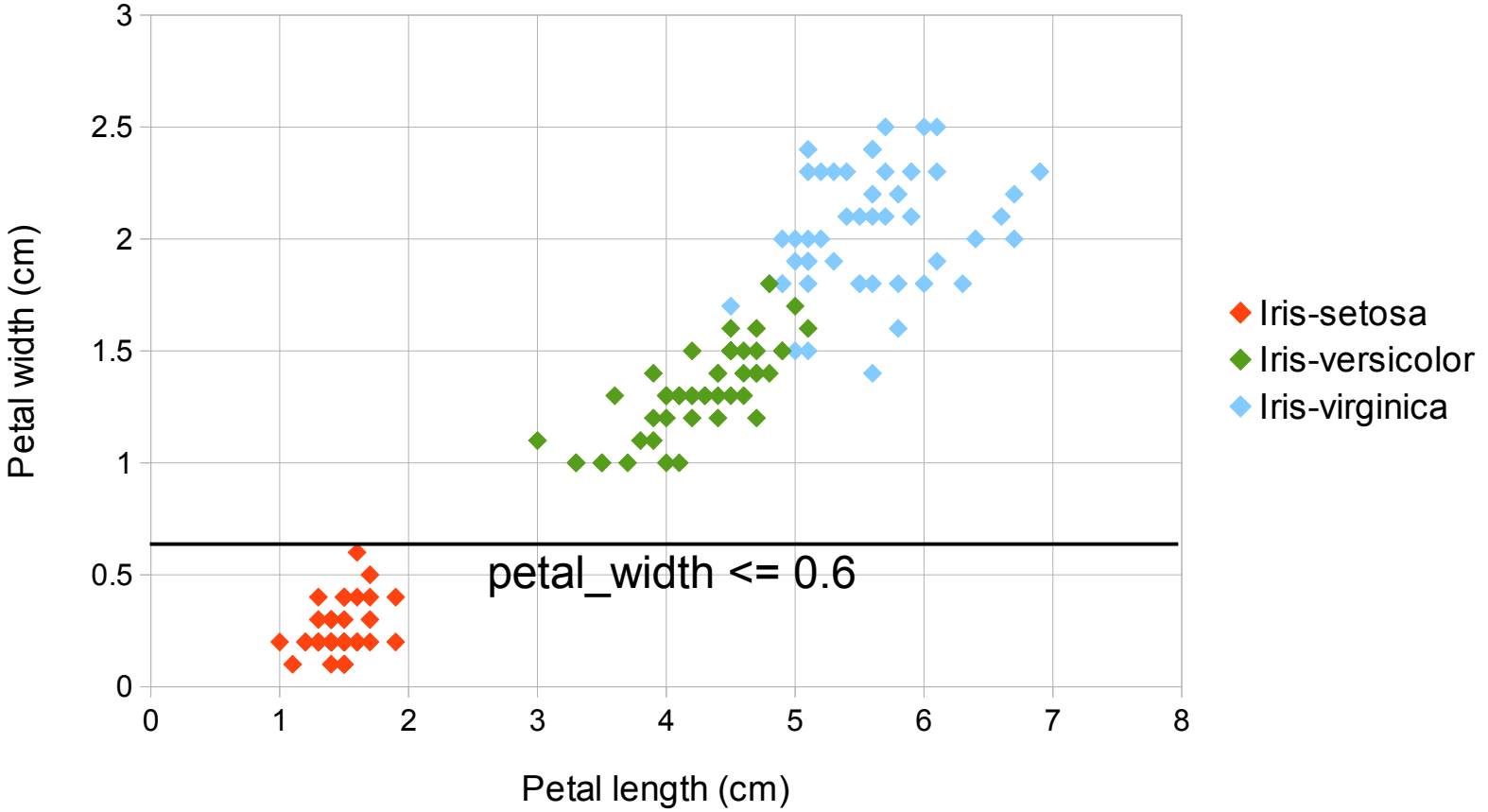
Decision trees

- One of the simplest algorithms for learning classification rules
- Generates a human-readable classifier model in the form of a tree of if/else tests
- Recursive implementation:
  - A rule is created by choosing a feature and a threshold that best partition the data set
  - This creates two subsets, which are recursively partitioned in the same way

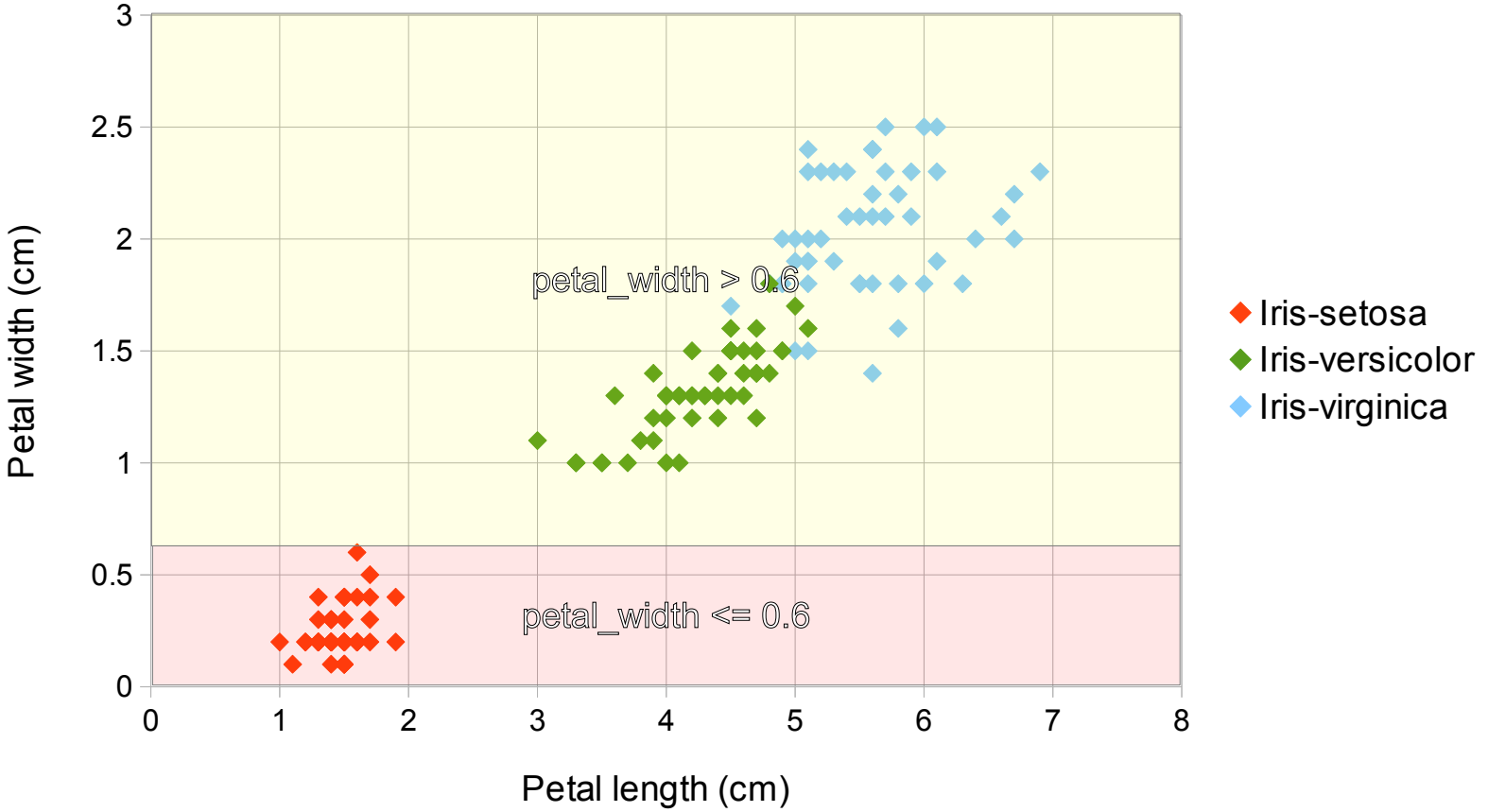
Example decision tree construction



Example decision tree construction

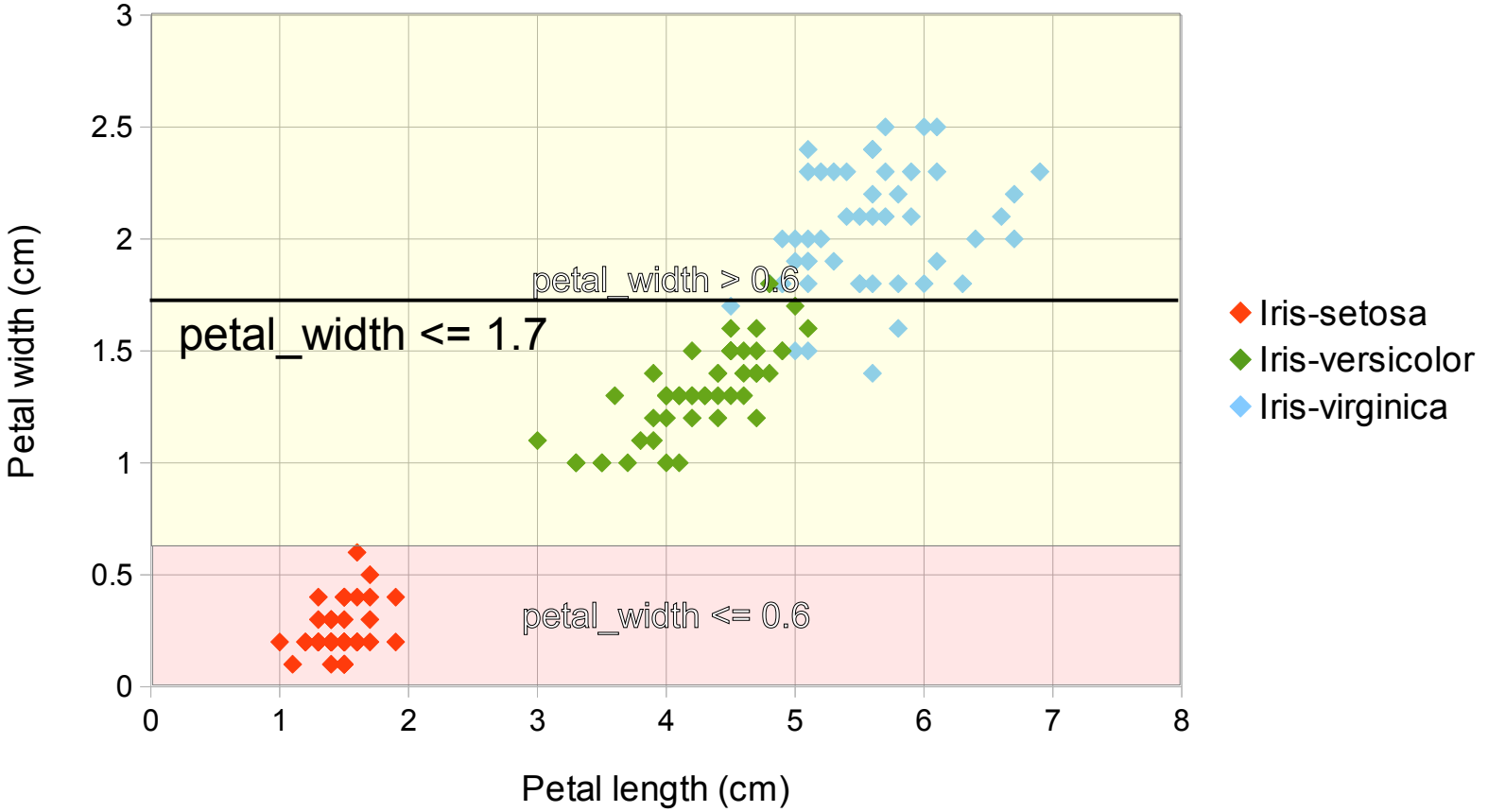


Example decision tree construction



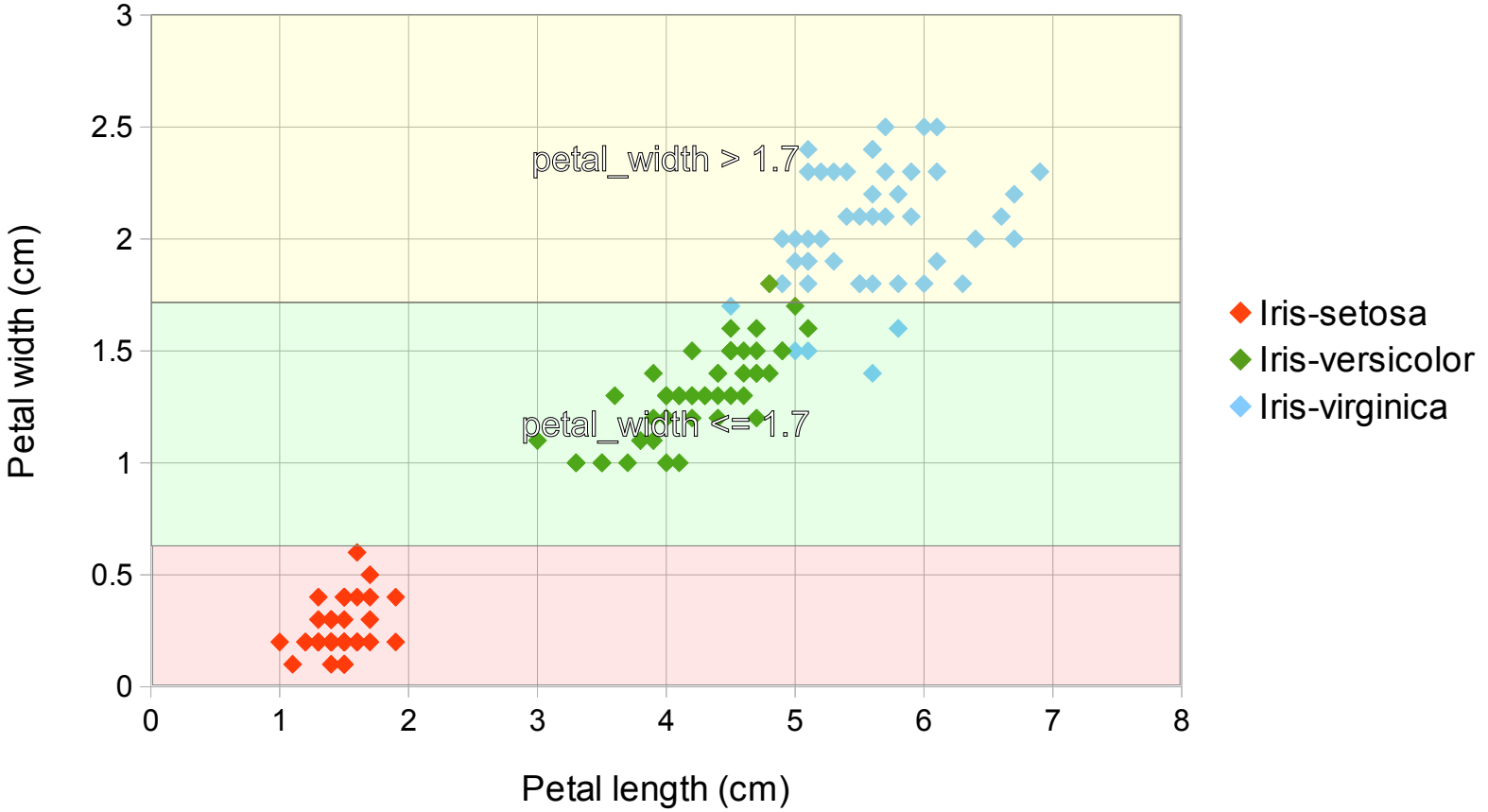

```
petalwidth <= 0.6: Iris-setosa (50/50)
petalwidth > 0.6: Iris-versicolor (50/100)
```

Example decision tree construction



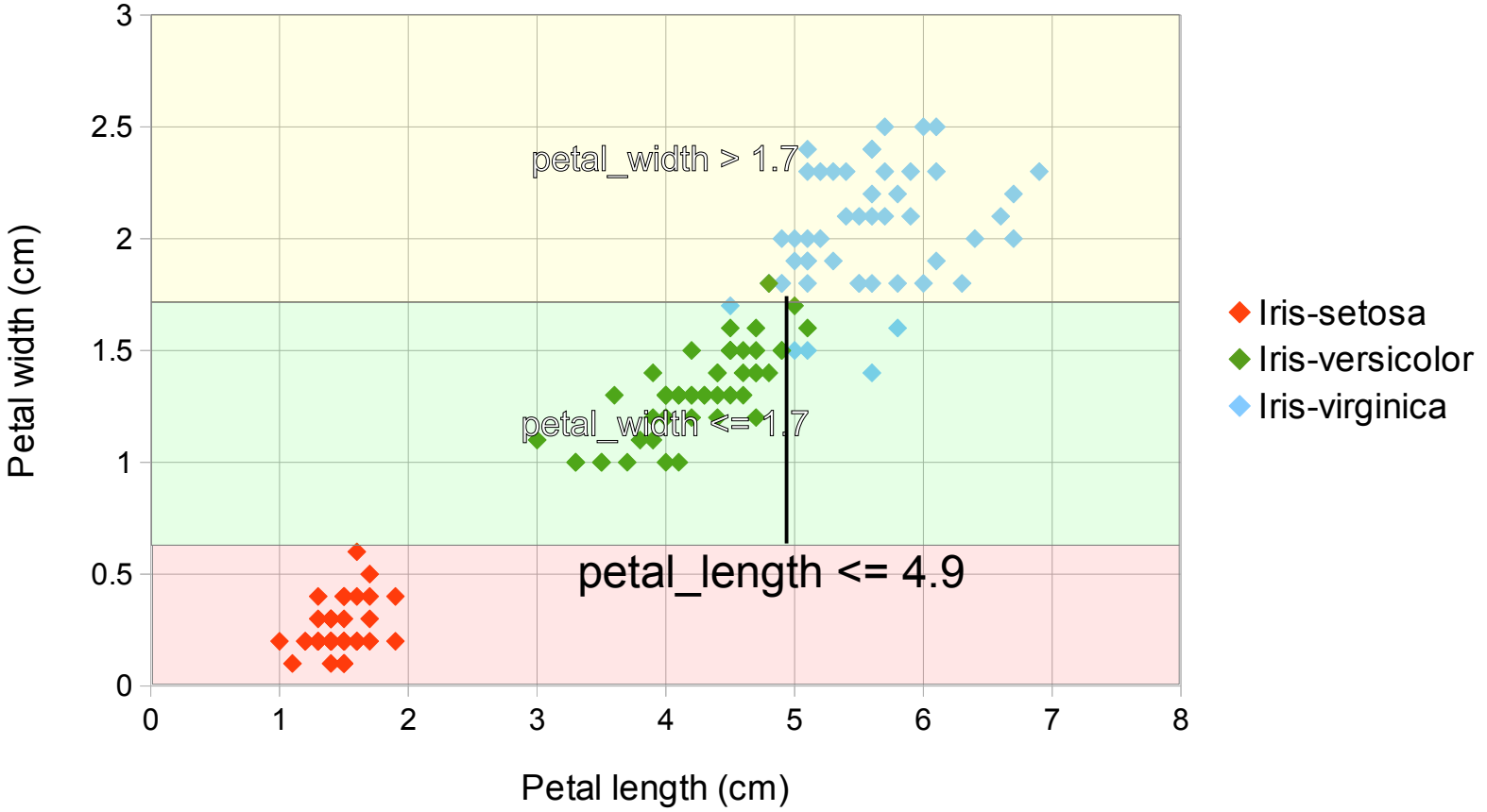

```
petalwidth <= 0.6: Iris-setosa (50/50)
petalwidth > 0.6: Iris-versicolor (50/100)
```

Example decision tree construction



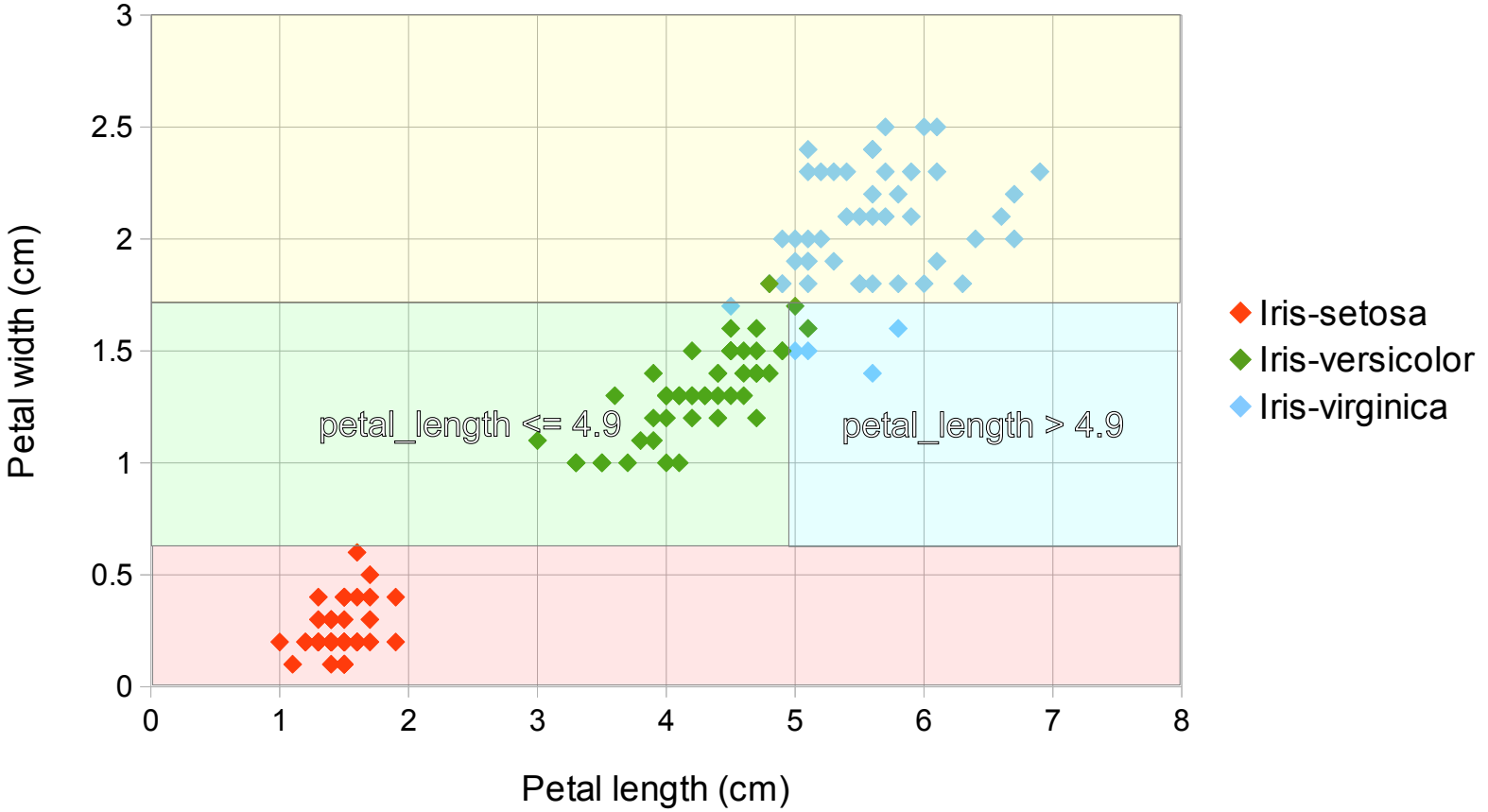

```
petalwidth <= 0.6: Iris-setosa (50/50)
petalwidth > 0.6
|   petalwidth <= 1.7: Iris-versicolor (49/54)
|   petalwidth > 1.7: Iris-virginica (45/46)
```

Example decision tree construction



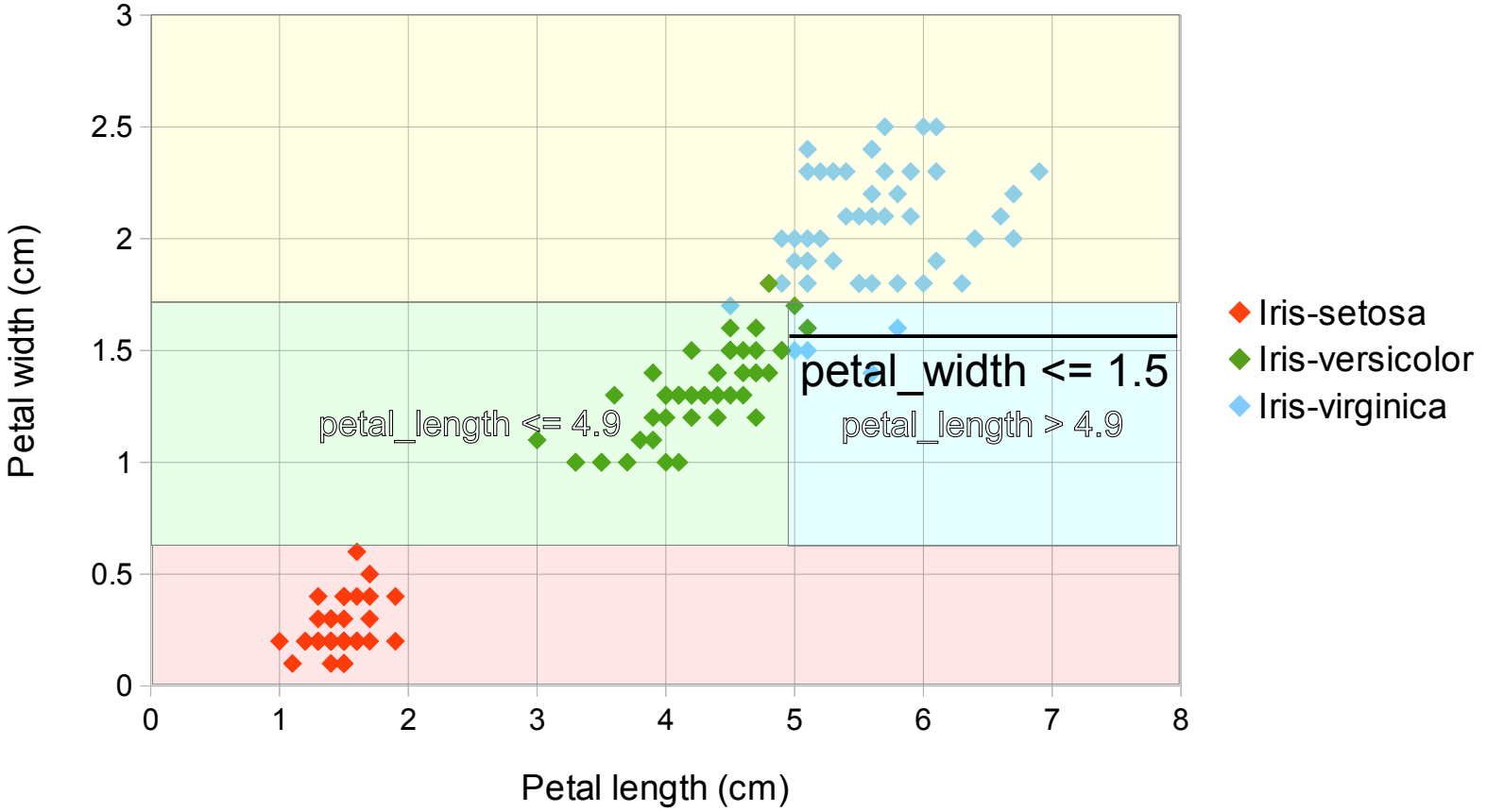

```
petalwidth <= 0.6: Iris-setosa (50/50)
petalwidth > 0.6
|   petalwidth <= 1.7: Iris-versicolor (49/54)
|   petalwidth > 1.7: Iris-virginica (45/46)
```

Example decision tree construction



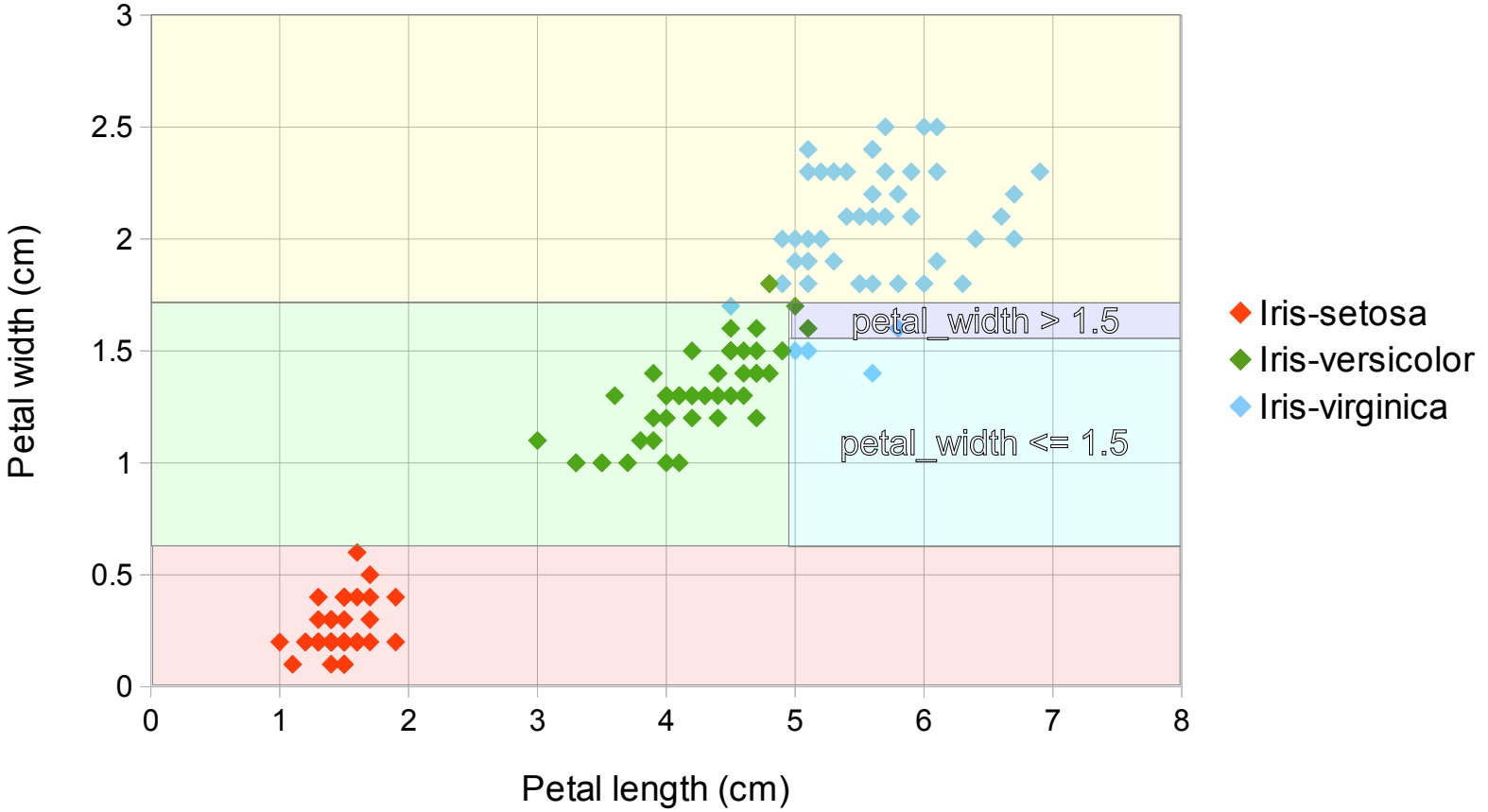

```
petalwidth <= 0.6: Iris-setosa (50/50)
petalwidth > 0.6
|   petalwidth <= 1.7
|   |   petallength <= 4.9: Iris-versicolor (47/48)
|   |   petallength > 4.9: Iris-virginica (4/6)
|   petalwidth > 1.7: Iris-virginica (45/46)
```

Example decision tree construction



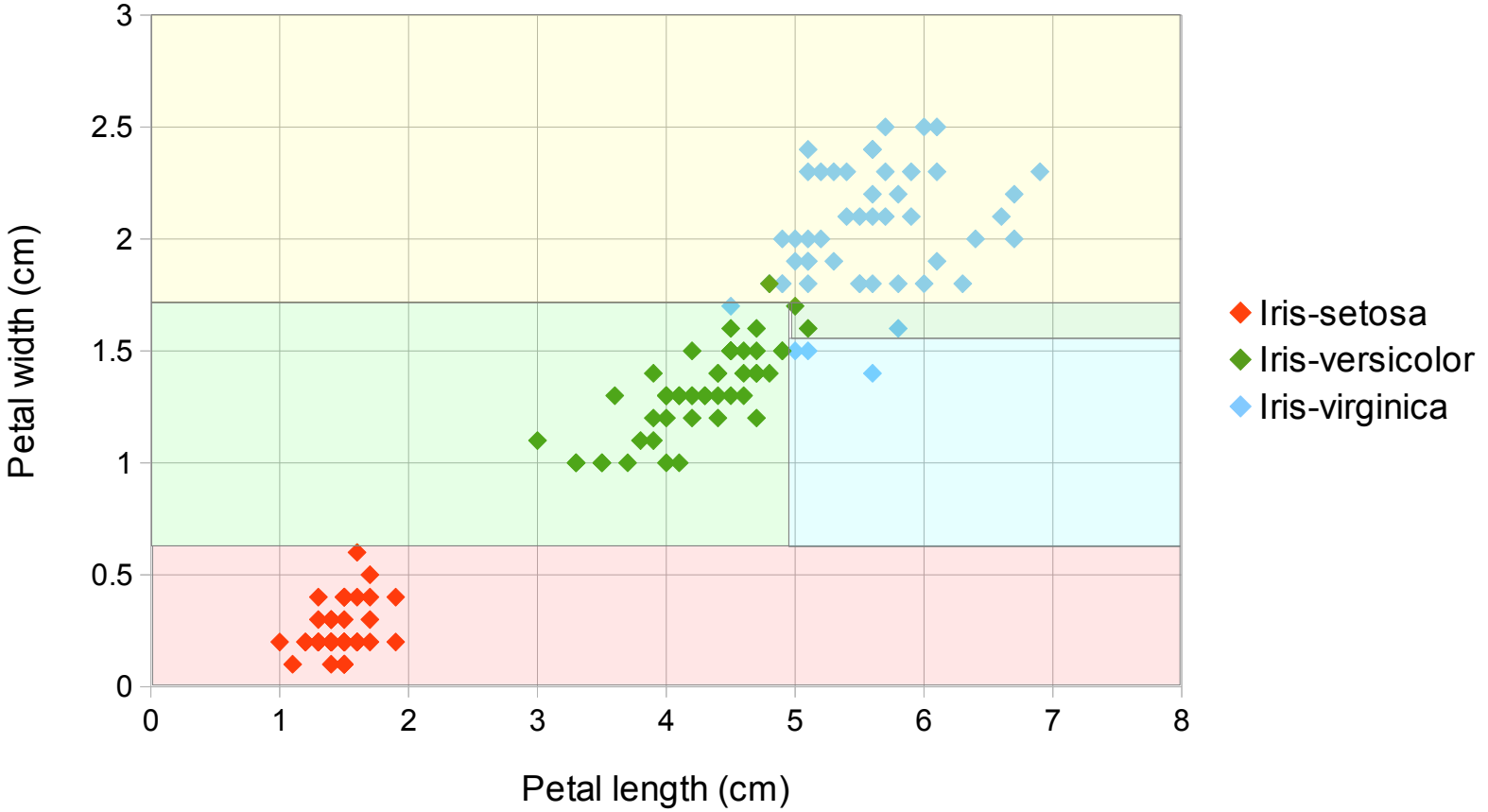

```
petalwidth <= 0.6: Iris-setosa (50/50)
petalwidth > 0.6
|   petalwidth <= 1.7
|   |   petallength <= 4.9: Iris-versicolor (47/48)
|   |   petallength > 4.9: Iris-virginica (4/6)
|   petalwidth > 1.7: Iris-virginica (45/46)
```

Example decision tree construction




```
petalwidth <= 0.6: Iris-setosa (50/50)
petalwidth > 0.6
|   petalwidth <= 1.7
|   |   petallength <= 4.9: Iris-versicolor (47/48)
|   |   petallength > 4.9
|   |   |   petalwidth <= 1.5: Iris-virginica (3/3)
|   |   |   petalwidth > 1.5: Iris-versicolor (2/3)
|   petalwidth > 1.7: Iris-virginica (45/46)
```

Example decision tree construction




```
petalwidth <= 0.6: Iris-setosa (50/50)
petalwidth > 0.6
|   petalwidth <= 1.7
|   |   petallength <= 4.9: Iris-versicolor (47/48)
|   |   petallength > 4.9
|   |   |   petalwidth <= 1.5: Iris-virginica (3/3)
|   |   |   petalwidth > 1.5: Iris-versicolor (2/3)
|   petalwidth > 1.7: Iris-virginica (45/46)
```

147/150 instances correctly classified

Decision trees

- Providing each instance can be uniquely identified by its features, the recursive construction of the decision tree can be run until all the classes are perfectly separated
- However, most algorithms stop earlier when certain criteria are met to avoid overfitting
- The leaves of the tree carry both a prediction and a confidence score, but the confidence score is only useful when the tree cuts off early enough to avoid overfitting

Evaluation

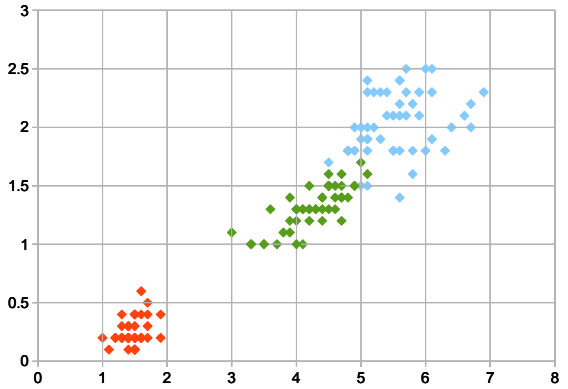
- Once you have created a classifier model, you want to know how well it will fare on future data
- Unfortunately, as you don't have this future data yet you can't use it to evaluate your classifier
- The classifier's accuracy on the training data is not a good indicator of how well the classifier will work on future data, as the model may overfit the training data
- Standard approach is to hold back some data for evaluation and not use it in training

Cross-validation

- The quality of the classifier depends on the amount of data used to train it
- As a result, when the amount of training data available is small, holding parts of it back for testing can be undesirable
- One solution to this is *n*-fold cross-validation
- The training data is randomly divided into *n* groups and each group is independently tested using the remaining groups as training data

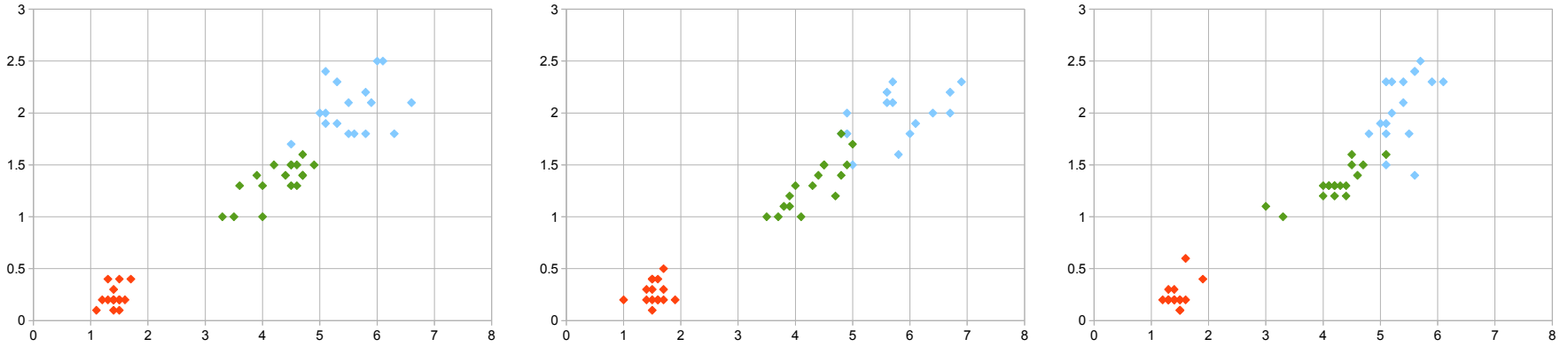
3-fold cross-validation

- For 3-fold cross-validation, you take your training data...



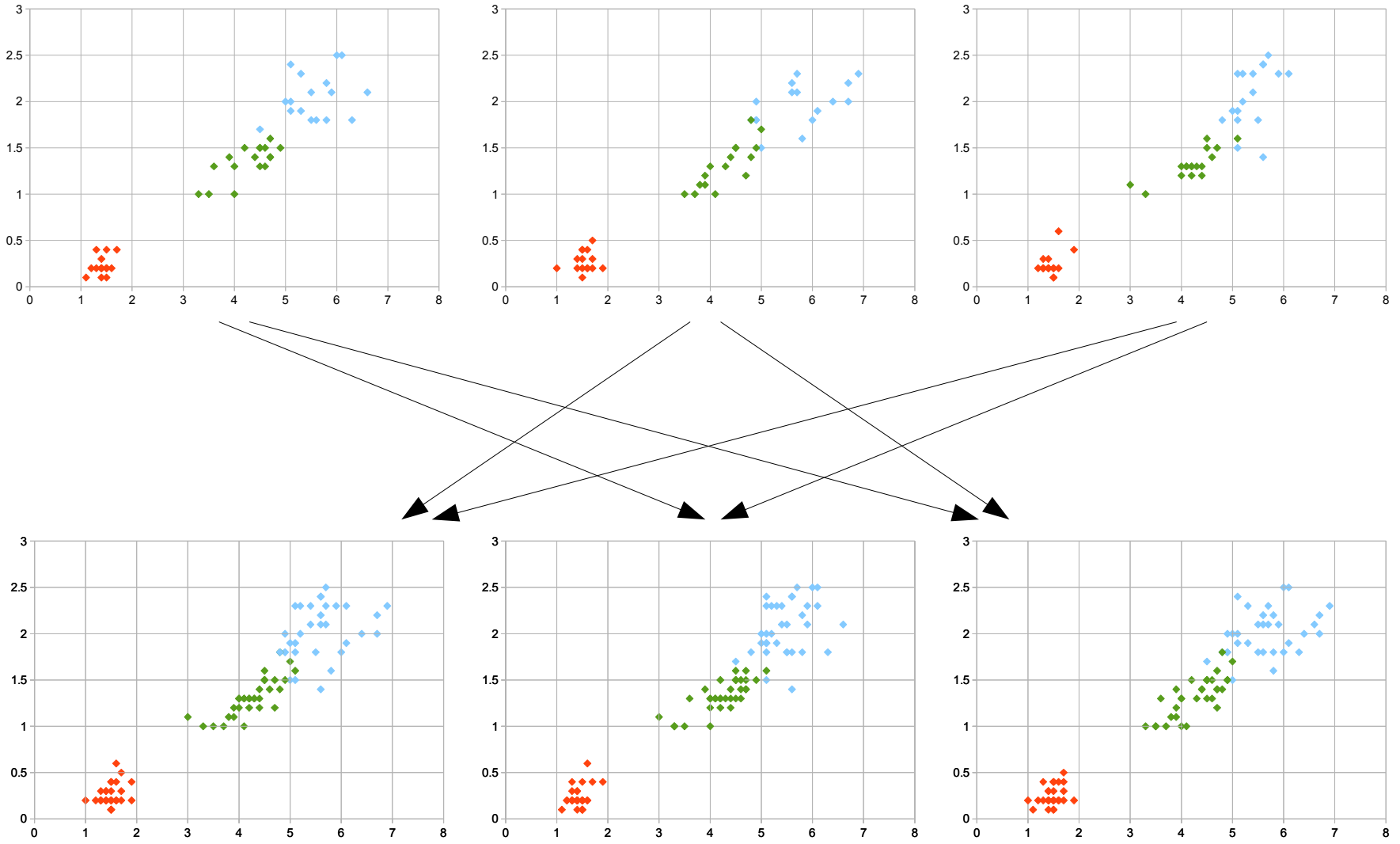
3-fold cross-validation

- ...and divide it into three subsets.



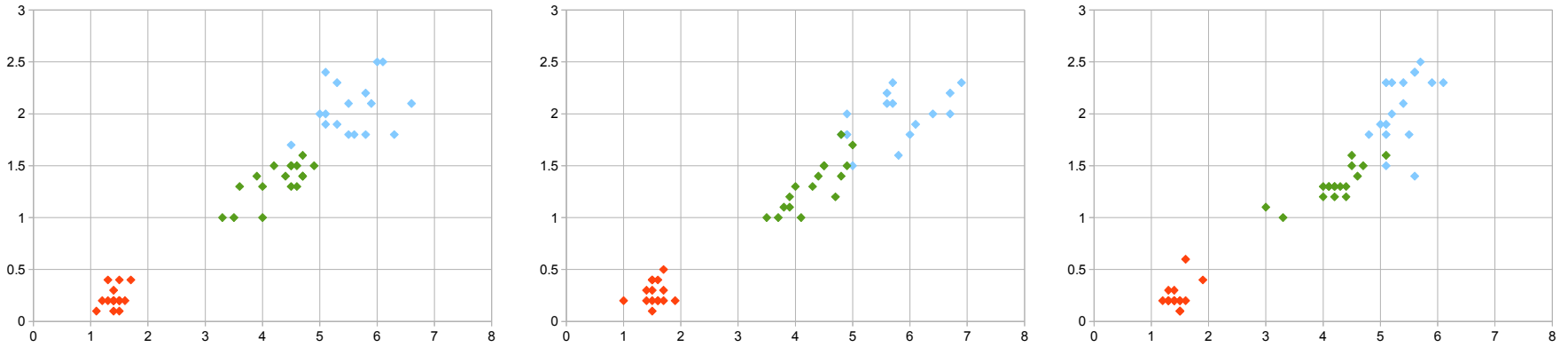
- The subdivision is commonly *stratified*, meaning each subset has approximately equal numbers of each class

3-fold cross-validation

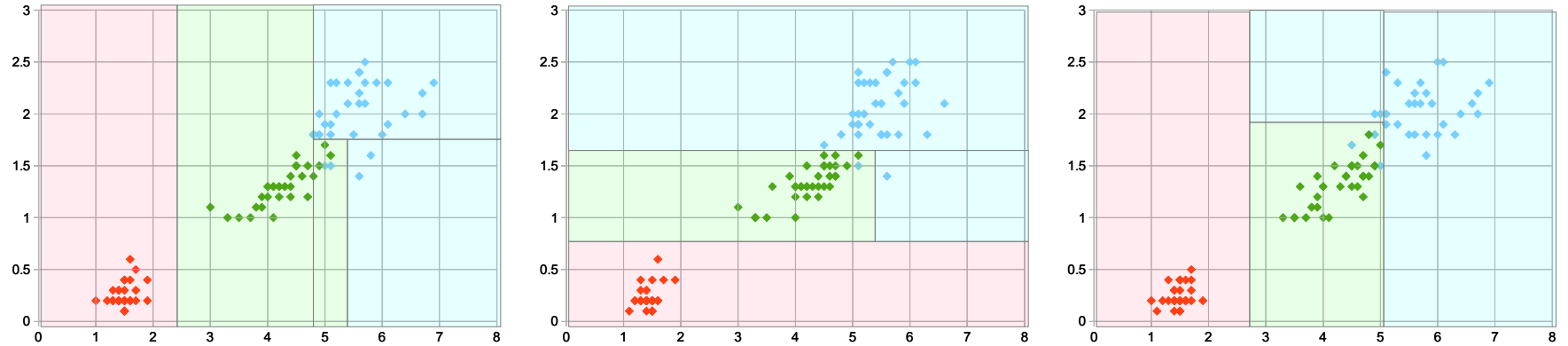


The subset pairs are combined to create training data for the remaining subset

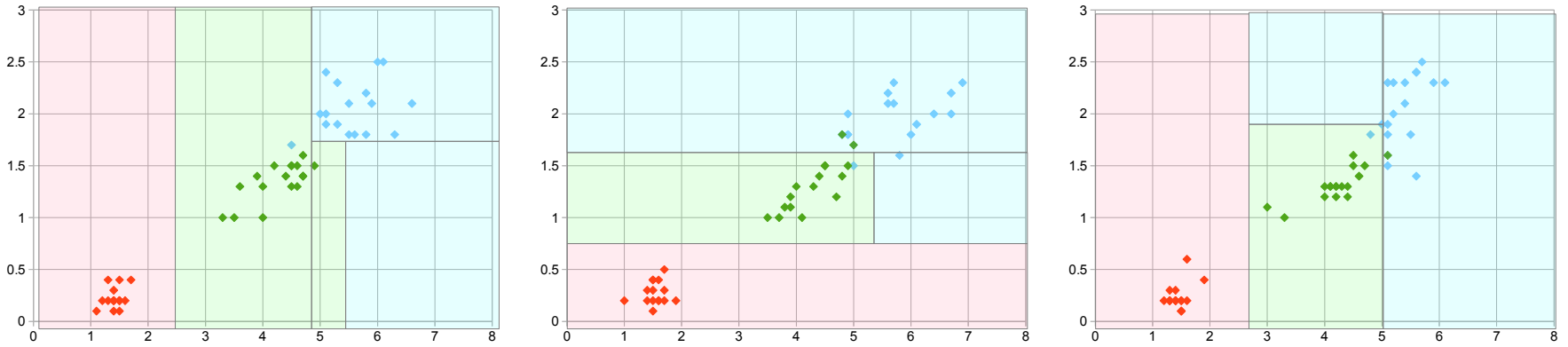
3-fold cross-validation



- Classification models are trained on each training set...



3-fold cross-validation



- ...and then applied on their respective test sets to produce an evaluation
- Every instance has now been classified, and the accuracy of the algorithm on this data set can be reasonably expected to apply to unseen data, providing the training set sufficiently represents the unseen data

Machine learning for source code classification

- Primary data set used for training and testing is the 'begbunch' accuracy test suites, used for testing Parfait
- After some files were removed, either for data quality reasons or because they couldn't be compiled, the data set consists of 5434 functions across 3627 C and C++ source files

Machine learning for source code classification

- The data set consists of source files that have been marked up with tags identifying bugs and potential vulnerabilities:

```
int main(int argc, char *argv[])
{
  int inc_value;
  int loop_counter;
  char buf[10];

  inc_value = 4105 - (4105 - 1);

  for(loop_counter = 0; ; loop_counter += inc_value)
  {
    if (loop_counter > 4105) break;
/* <bug buffer-overflow> */    buf[loop_counter] = 'A'/* </bug> */;
  }

  return 0;
}
```

Machine learning for source code classification

- The problem is treated as a classification task with each bug type considered to be an independent class variable
- Example bug types:
  - `buffer-overflow`
  - `double-free`
  - `memory-leak`
  - `race-condition`
  - `read-outside-array-bounds`

Machine learning for source code classification

- Classification is performed at the function level:
  - A single source line is not likely to have enough information in it to meaningfully identify whether it has a bug or not
  - An entire source file may be too coarse for classifications to be meaningful
- The class labels (bug types) are provided, but the features are not necessarily obvious
  - Unlike problems where you have an existing data set, like the Iris plants set

Features for classifying source code

- There are two main kinds of features that we hope to find:
  - Features that indicate the presence of or absence of a particular kind of bug
    - Risky programming styles that happen to correlate strongly with certain bugs or vulnerabilities
  - Features that allow us to find duplicates or near-duplicates of functions or blocks of code
    - The logic here is that code gets copied around a lot and this can cause certain bugs or vulnerabilities to propagate

Features for classifying source code

- Outputs from the Parfait “Complexity” tool:
  - Cyclomatic
  - Dataflow
  - DefUseChains
  - Edges
  - Effort
  - FuncEndLine
  - FuncStartLine
  - Knots
  - Length
  - Level
  - LineCount
  - Nesting
  - Operators
  - Vertices
  - Vocabulary
  - Volume

Features for classifying source code

- Text features:

- !
- (
-)
- ,
- 00
- 1
- 10
- 101
- 1024
- 12
- 1900
- 2
- 20
- 25
- 320x200
- 4G
- 55
- 553
- 8000
- ==
- APLOG_ERR
- APMG_CLK_REG_VAL_BSM_CLK_RQT
- BAD
- CV_CHARPTR
- CV_FLOAT
- CV_INT
- Con_DrawRuler2
- Con_GetAlias
- Con_GetByte
- Con_GetCommand
- Con_GetFloat
- Con_GetVariable
- DDKEY_ENTER
- DEFAULT
- DEFAULT_LOG_FORMAT
- DHparams_dup
- DIAGstr
- EVP_PKEY_EC
- EXITCODE_OK
- Expired
- FILE
- FONT
- FSM_COMPATIBILITY
- FSM_IDADD
- FSM_IDDEL
- FSM_MACRO
- HITN
- HSP_TYPE_LOCAL
- IEEE80211_CHAN_ANYC
- ImageWidthsOrHeightsDiffer
- In
- Input
- Logged
- MagickEpsilon
- MagickExport
- Mel
- Only
- PATH_SEP
- PKCS7_TEXT
- PlayMusic
- REPLY_LEDS_CMD
- RESERVED
- RXON_FILTER_ACCEPT_GRP_MSK
- RX_QUEUE_SIZE
- Record
- Rectify
- SCD_DRAM_BASE_ADDR
- SCD_INTERRUPT_MASK
- SCD_QUEUE_RDPTR
- SDSSC_OKAY
- SGE_INTR_MAXBUCKETS
- SGE_PL_INTR_MASK
- SGE_RX_COPY_THRESHOLD
- SLC_IP
- SSL3_MT_CERTIFICATE_VERIFY

...etc...

Features for classifying source code

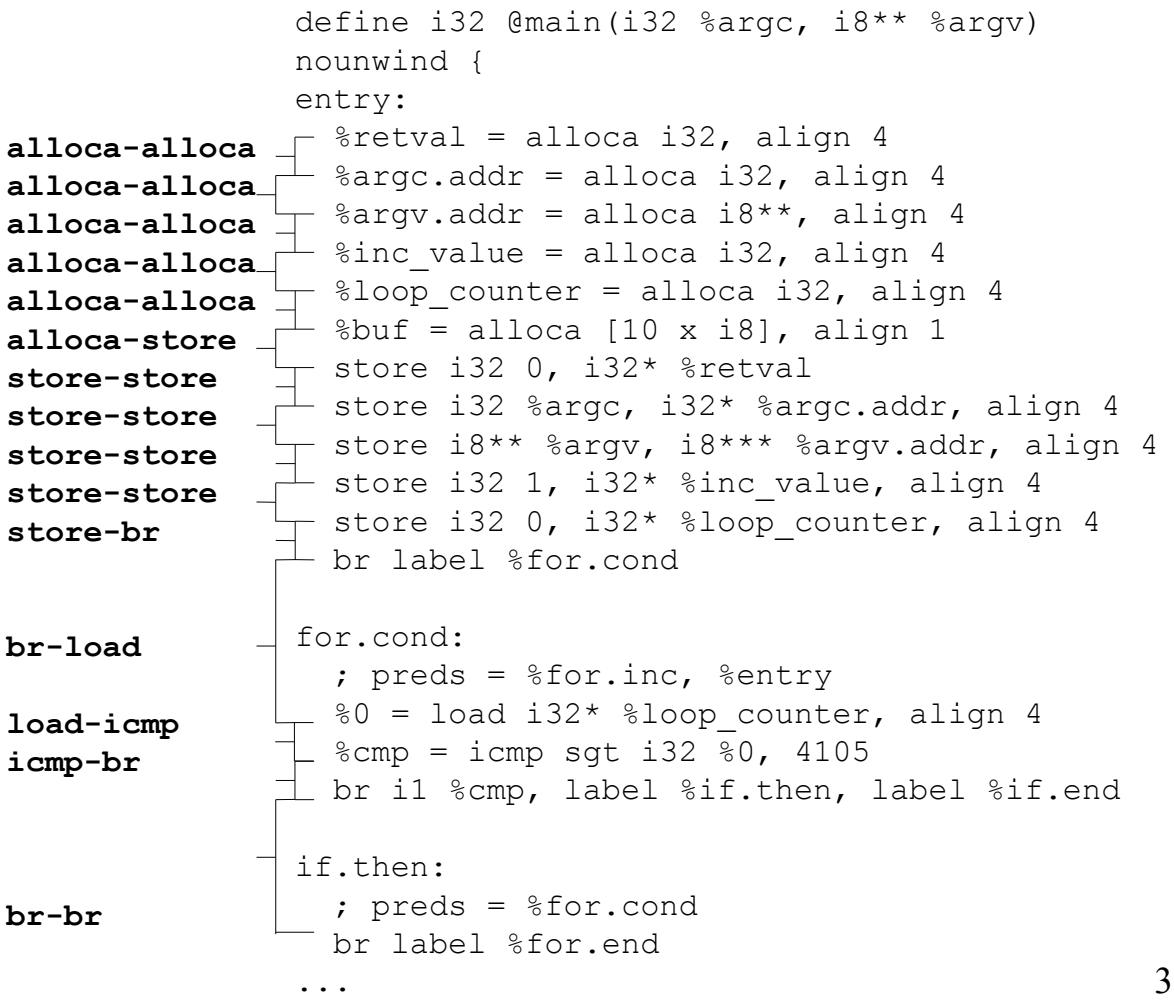
- LLVM disassembly instruction frequencies:
  - `add`
  - `alloca`
  - `and`
  - `ashr`
  - `bitcast`
  - `br`
  - `call`
  - `extractvalue`
  - `fadd`
  - `fcmp`
  - `fdiv`
  - `fmul`
  - `fpext`
  - `fptosi`
  - `fptoui`
  - `fptrunc`
  - `fsub`
  - `getelementptr`
  - `icmp`
  - `insertvalue`
  - `inttoptr`
  - `invoke`
  - `landingpad`
  - `load`
  - `lshr`
  - `mul`
  - `or`
  - `phi`
  - `ptrtoint`
  - `resume`
  - `ret`
  - `sdiv`
  - `select`
  - `sext`
  - `shl`
  - `sitofp`
  - `srem`
  - `store`
  - `sub`
  - `switch`
  - `trunc`
  - `udiv`
  - `uitofp`
  - `unreachable`
  - `urem`
  - `xor`
  - `zext`

Features for classifying source code

- LLVM disassembly instruction *n*-grams:

- add-add
- add-and
- add-br
- add-call
- add-getelementptr
- add-icmp
- add-invoke
- add-load
- add-lshr
- add-mul
- add-sdiv
- add-sext
- add-shl
- add-sitofp
- add-srem
- add-store
- add-sub
- add-switch
- add-trunc
- add-udiv
- add-urem
- add-zext
- alloca-alloca
- alloca-br
- alloca-call

...etc...



Features for classifying source code

- Clang –analyze output
  - Array-subscript-is-undefined
  - Assigned-value-is-garbage-or-undefined
  - Bad-free
  - Branch-condition-evaluates-to-a-garbage-value
  - Dead-assignment
  - Dead-increment
  - Dead-initialization
  - Dereference-of-null-pointer
  - Dereference-of-undefined-pointer-value
  - Double-free
  - Function-call-argument-is-an-uninitialized-value
  - Garbage-return-value
  - Memory-leak
  - Out-of-bound-array-access
  - Potential-buffer-overflow-in-call-to-'gets'
  - Result-of-operation-is-garbage-or-undefined
  - Return-value-is-not-checked-in-call-to-'seteuid'
  - Undefined-allocation-of-0-bytes-(CERT-MEM04-C;-CWE-131)
  - Use-after-free

Features for classifying source code

- Output from other static analysis engines:
  - Parfait
  - Splint
  - Uno

Feature selection

- When there are a very large number of features, machine learning algorithms can take a long time to generate models and can perform suboptimally
- This can be solved through manual feature engineerin, but it is also possible to use machine learning to reduce the subset of features

Feature selection

- For example, after mining the data set for text features, 8190 unique text features were identified
- This was reduced to a more manageable 500 through dimensionality reduction using the leave-one-out-nearest-neighbour-error (LOONNE) algorithm

LOONNE

- Leave-one-out-nearest-neighbour-error calculates a global error value for a particular data set by finding each instance's nearest neighbour in a Euclidean space formed out of the data set's features
- The differences between the class of each point and its nearest neighbour are then summed to produce this global error

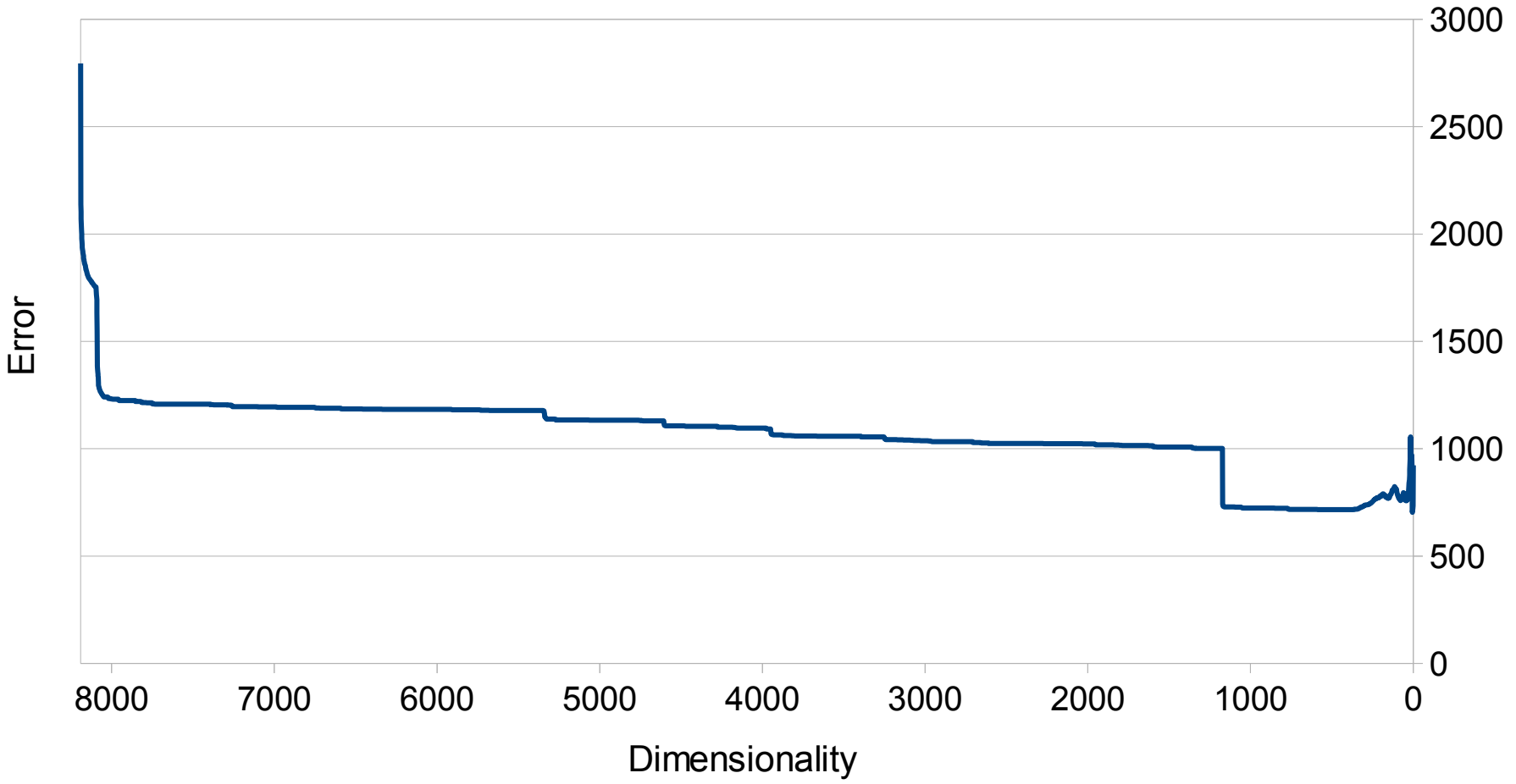
LOONNE

- Each dimension is then experimentally removed and the global error of each new subspace is calculated
- Finally, the subspace with the lowest global error becomes the new feature space
- This process then repeats until all features have been removed
- The result is a ranked list of features in order of representational capacity

LOONNE

- This approach involves backward subspace selection rather than forward subspace selection to preserve groups of features that have greater combined representational capacity than they have individually
- Forward subspace selection would incorrectly omit these groups

LOONNE


Error
3000
2500
2000
1500
1000
500
0
8000
7000
6000
5000
4000
3000
2000
1000
0
Dimensionality

Artificial data vs real data

- Some benchmarks were comprised of real data, others of artificial data designed to test for specific bugs:
  - Cigital (artificial) 50 functions
  - Iowa (artificial) 1686 functions
  - Parfait (real) 547 functions
  - Parfait-2 (real) 417 functions
  - Parfait-examples (artificial) 25 functions
  - Samate (artificial) 2366 functions
  - unportable (real) 162 functions
  - Memory-leak (artificial) 181 functions

Artificial data vs real data

- While classification with the features identified was found to be very effective on the artificial data, in many cases it achieved this effectiveness by selecting for features that only correlated with bugs in the same artificial data
- However, artificial data could not be excluded entirely as the majority of the available data was artificial and many categories of bugs were only represented well in the artificial data sets

Artificial data vs real data

- The compromise used here was to perform cross-validation on the individual bug-testing suites:
  - e.g. when testing on the 'Cigital' suite, the data from the other suites would be used for training
- This ensured that maximum utility was extracted from the data without using the artificial data to “cheat”.
- The 'memory-leak' suite was removed as it was found to contain source files also used in other testing suites

Comparisons

	Biscotti	Parfait	Splint
buffer-overflow	1054/1305 (81%), 20 FP	1073/1305 (82%), 21 FP	821/1305 (63%), 356 FP
double-free	0/23 (0%), 0 FP	16/23 (70%), 3 FP	0/23 (0%), 0 FP
format-string	0/17 (0%), 0 FP	0/17 (0%), 0 FP	4/17 (24%), 17 FP
integer-overflow	0/24 (0%), 0 FP	1/24 (4%), 9 FP	0/24 (0%), 0 FP
memory-leak	29/189 (15%), 3 FP	54/189 (29%), 24 FP	0/189 (0%), 0 FP
null-pointer-deref	0/14 (0%), 0 FP	8/14 (57%), 3 FP	5/14 (36%), 69 FP
read-outside-array-bounds	176/267 (66%), 5 FP	179/267 (67%), 4 FP	232/267 (87%), 890 FP
uninitialised-var	37/125 (30%), 5 FP	81/125 (65%), 83 FP	79/125 (63%), 134 FP
use-after-free	0/31 (0%), 0 FP	17/31 (55%), 3 FP	18/31 (58%), 7 FP
total	1296/2233 (58%), 37 FP	1429/2233 (64%), 150 FP	1159/2233 (52%), 1473 FP

- Bug types no systems scored any correct results on were omitted (they are included in the total, however)
- FP = false positives

Biscotti

- Performance can be tuned to achieve greater accuracy with more false positives or fewer false positives at the cost of lower accuracy
- Presented results balance accuracy and false positives by maximising the score:
  - score = correct results – false positives
- Can be used as a 'filter' for more expensive static analysis, like Parfait, by omitting Parfait and other time-consuming features, then tuning the results to maximise accuracy

Top 10 features

- **[Parfait]buffer-overflow**
- **[Parfait]read-outside-array-bounds**
- **[Splint]xx-fresh-storage-not-released-before-return**
- **[Text],**
- **[Complexity]FuncEndLine**
- **[Parfait]uninitialised-var**
- **[Splint]xx-function-exported-but-not-used-outside**
- **[Splint]xx-for-body-not-block**
- **[Splint]xx-return-value-ignored**
- **[Text]contents**

Algorithms used by Biscotti

- LOONNE
  - Used to reduce the dimensionality of the feature space before passing the results to the next algorithm
- RandomForests
  - Ensemble classifier consisting of 100 randomly-seeded decision trees using different random subsets of the feature set
  - The outcomes of the decision trees are then combined to produce a single outcome for each result

Biscotti architecture

- Developed as a series of individual programs and scripts that interact through shared file formats
- Can be used standalone or in conjunction with Weka and other machine learning tools, with supplied conversion tools to arff and C5 formats

FunctionList format

- FunctionID:SourceFile:FunctionName:FirstLine:LastLine

```
0:Samate/Krat-basic-00227-large/src/basic-00227-large.c:main:74:90
1:Samate/Samate-memory_leak_basic1/src/memory_leak_basic.cpp:_Z4funcv:34:35
2:Samate/Samate-memory_leak_basic1/src/memory_leak_basic.cpp:main:39:44
3:Samate/Krat-basic-00017-med/src/basic-00017-med.c:main:74:83
4:Samate/Samate-Using_freed_memory/src/Using_freed_memory.c:main:21:35
5:Iowa/Iowa-C99-c_K_3_2_b/src/../../shared/TEST_PARAM.H:ret:9:24
6:Iowa/Iowa-C99-c_K_3_2_b/src/c_K_3_2_b_s.c:func:34:46
7:Iowa/Iowa-C99-c_K_3_2_b/src/c_K_3_2_b.c:main:45:51
8:Samate/Samate-resource_injection_basic_good/src/resource_injection_basic_good.c:allowed:41:49
9:Samate/Samate-
resource_injection_basic_good/src/resource_injection_basic_good.c:printLine:52:64
10:Samate/Samate-resource_injection_basic_good/src/resource_injection_basic_good.c:main:66:74
11:Memory-leak/Iowa/Iowa-Alloc-c_G_1_3_a/src/../../shared/TEST_PARAM.H:ret:9:24
12:Memory-leak/Iowa/Iowa-Alloc-c_G_1_3_a/src/c_G_1_3_a_s.c:func:34:51
13:Memory-leak/Iowa/Iowa-Alloc-c_G_1_3_a/src/c_G_1_3_a.c:main:35:38
14:Iowa/Iowa-Pointer-c_F_1_8_e/src/../../shared/TEST_PARAM.H:ret:9:24
15:Iowa/Iowa-Pointer-c_F_1_8_e/src/c_F_1_8_e_s.c:func:35:56
16:Iowa/Iowa-Pointer-c_F_1_8_e/src/c_F_1_8_e.c:main:42:49
17:Iowa/Iowa-C99-c_K_4_5_c/src/../../shared/TEST_PARAM.H:ret:9:24
18:Iowa/Iowa-C99-c_K_4_5_c/src/c_K_4_5_c.c:func:50:80
19:Iowa/Iowa-C99-c_K_4_5_c/src/c_K_4_5_c.c:main:84:86
```

BugList format

- FunctionID:BugType:Interproc:Exploitable:Security:Osl:Callsite:LineNo

```
0:buffer-overflow:0:0:0:0:0:85
2:memory-leak:0:0:0:0:0:43
3:buffer-overflow:0:0:0:0:0:79
4:use-after-free:0:0:0:0:0:31
6:buffer-overflow:0:0:0:0:0:42
12:free-of-unallocated:0:0:0:0:0:50
12:memory-leak:0:0:0:0:0:51
15:buffer-overflow:0:0:0:0:0:48
18:read-outside-array-
bounds:0:0:0:0:0:71
21:buffer-overflow:0:0:0:0:0:81
23:buffer-overflow:0:0:1:0:0:50
25:read-outside-array-
bounds:0:0:0:0:0:58
36:buffer-overflow:1:0:0:0:0:842
49:buffer-overflow:0:0:0:0:0:87
51:read-outside-array-
bounds:0:0:0:0:0:49
53:buffer-overflow:0:0:0:0:0:85
54:buffer-overflow:0:0:1:0:0:121
62:buffer-overflow:0:0:0:0:0:85
67:memory-leak:0:0:0:0:0:52
69:memory-leak:0:0:0:0:0:75
```

FeatureFile format

- FunctionID:Features....

```
1:0:0:0:0:0:0:1:0:0:0:0:0:0:0:0:0:0:0:0:0:0:0:0:0:0:0:0:0:0:0:0:0:0:0:0:0:0:0:1:0:0:0:0:0:0:1:0:0:0:0:0:0:0:0:0
2:0:0:0:0:1:1:3:0:0:0:0:0:0:0:0:0:0:0:0:0:0:0:0:0:0:0:0:0:2:0:0:0:0:0:0:2:0:0:0:0:0:0:3:0:0:0:0:0:0:0:0:0:0
3:0:0:0:0:0:0:1:3:0:0:0:0:0:0:0:0:0:0:0:0:2:0:0:0:0:0:0:2:0:0:0:0:0:0:2:0:0:0:0:0:0:3:0:0:0:0:0:0:0:0:0:0:0
4:0:0:0:0:0:0:0:15:0:0:0:0:0:0:0:0:0:0:0:0:1:0:0:0:0:0:0:7:0:0:0:0:0:0:0:1:0:0:0:0:0:0:5:0:0:0:0:0:0:0:0:0:0
5:1:0:0:0:0:0:1:7:0:0:0:0:0:0:0:0:0:0:0:0:0:1:0:0:0:0:5:0:0:0:0:0:0:0:1:0:0:0:0:0:0:3:1:0:0:0:0:0:1:0:0:0:0
6:3:1:0:0:0:0:14:9:0:0:0:0:0:0:0:0:0:0:0:0:4:4:0:0:0:0:22:0:0:0:0:0:0:0:1:0:0:0:0:0:0:0:11:2:0:0:0:0:0:0:0:0:0:0
7:0:0:0:0:0:0:1:2:0:0:0:0:0:0:0:0:0:0:0:0:0:0:0:0:0:0:0:0:3:0:0:0:0:0:0:2:0:0:0:0:0:0:2:0:0:0:0:0:0:0:0:0:0:0:0
8:1:0:0:0:0:0:7:3:0:0:0:0:0:0:0:0:0:0:0:0:0:1:2:0:0:0:0:6:0:0:0:0:0:0:0:3:0:0:0:0:0:0:2:0:0:0:0:0:0:0:0:0:0:0:0
9:0:0:0:0:0:2:4:7:0:0:0:0:0:0:0:0:0:0:0:0:0:2:0:0:0:0:3:0:0:0:0:0:0:0:1:0:0:0:0:0:0:2:0:0:0:0:0:0:0:0:0:0:0:0:0
10:0:0:0:0:0:0:5:4:0:0:0:0:0:0:0:0:0:0:0:0:2:2:0:0:0:0:7:0:0:0:0:0:0:0:2:0:0:0:0:0:0:2:0:0:0:0:0:0:0:0:0:0:0:0:0
11:1:0:0:0:0:0:1:7:0:0:0:0:0:0:0:0:0:0:0:0:0:0:1:0:0:0:0:5:0:0:0:0:0:0:0:1:0:0:0:0:0:0:3:1:0:0:0:0:1:0:0:0:0
12:3:0:0:0:2:11:9:0:0:0:0:0:0:0:0:0:0:0:3:4:0:0:0:0:16:0:0:0:0:0:0:0:1:0:0:0:0:0:0:9:1:0:0:0:0:1:0:0:0:0
13:0:0:0:0:0:0:1:1:0:0:0:0:0:0:0:0:0:0:0:0:0:0:0:0:0:0:0:0:2:0:0:0:0:0:0:2:0:0:0:0:0:0:1:0:0:0:0:0:0:0:0:0:0:0:0
14:1:0:0:0:0:0:1:7:0:0:0:0:0:0:0:0:0:0:0:0:0:0:1:0:0:0:0:5:0:0:0:0:0:0:0:1:0:0:0:0:0:0:3:1:0:0:0:0:1:0:0:0:0
15:3:0:0:0:1:11:7:0:0:0:0:0:0:0:0:0:0:0:6:3:0:0:0:0:17:0:0:0:0:0:0:2:0:0:0:0:0:0:11:0:0:0:0:0:0:0:0:0:0:0:0
16:0:0:0:0:0:0:3:3:0:0:0:0:0:0:0:0:0:0:0:0:0:0:1:0:0:0:0:4:0:0:0:0:0:0:2:0:0:0:0:0:0:2:0:0:0:0:0:0:0:0:0:0:0
17:1:0:0:0:0:0:1:7:0:0:0:0:0:0:0:0:0:0:0:0:0:0:1:0:0:0:0:5:0:0:0:0:0:0:0:1:0:0:0:0:0:0:3:1:0:0:0:0:1:0:0:0:0
18:6:0:0:0:4:26:15:0:0:0:0:0:0:0:0:0:0:0:0:0:9:9:0:0:0:0:38:0:0:0:0:0:0:1:0:0:0:0:0:0:14:0:0:0:0:0:2:0:0:0:0
19:0:0:0:0:0:0:1:1:0:0:0:0:0:0:0:0:0:0:0:0:0:0:0:0:0:0:0:0:2:0:0:0:0:0:0:2:0:0:0:0:0:0:1:0:0:0:0:0:0:0:0:0:0:0:0
```

- The number and identity of the features in the file do not matter. Multiple feature files can be combined together for input into the machine learning tools

ResultsFile format

- FunctionID:BugType:LineNo

```
0:buffer-overflow:85
3:buffer-overflow:79
4:use-after-free:31
15:buffer-overflow:48
21:buffer-overflow:81
23:buffer-overflow:50
25:read-outside-array-bounds:58
42:null-pointer-deref:1305
42:null-pointer-deref:1335
49:buffer-overflow:87
51:read-outside-array-bounds:49
53:buffer-overflow:85
62:buffer-overflow:85
75:buffer-overflow:83
76:buffer-overflow:79
77:buffer-overflow:36
79:buffer-overflow:80
81:buffer-overflow:45
83:buffer-overflow:85
84:buffer-overflow:85
```

- Results files can be converted to feature files, with each bug type making up a column in the feature file

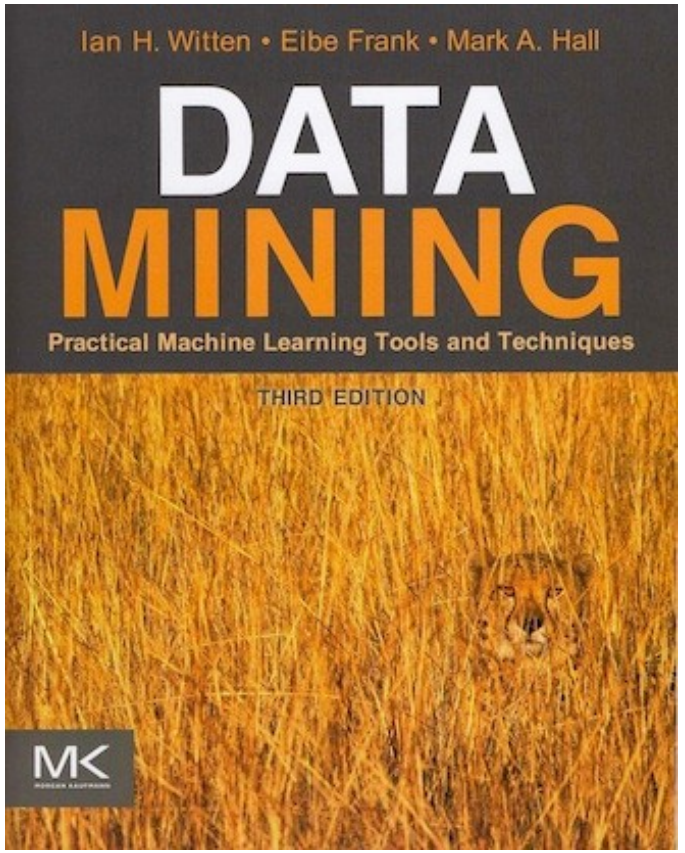
Future research

- Given the problems associated with working with small amounts of training data, future research will look into overcoming this:
  - Semi-supervised learning
  - Using one of the static analysis tools to generate ground truth so that any source code can be used for training data

Future research

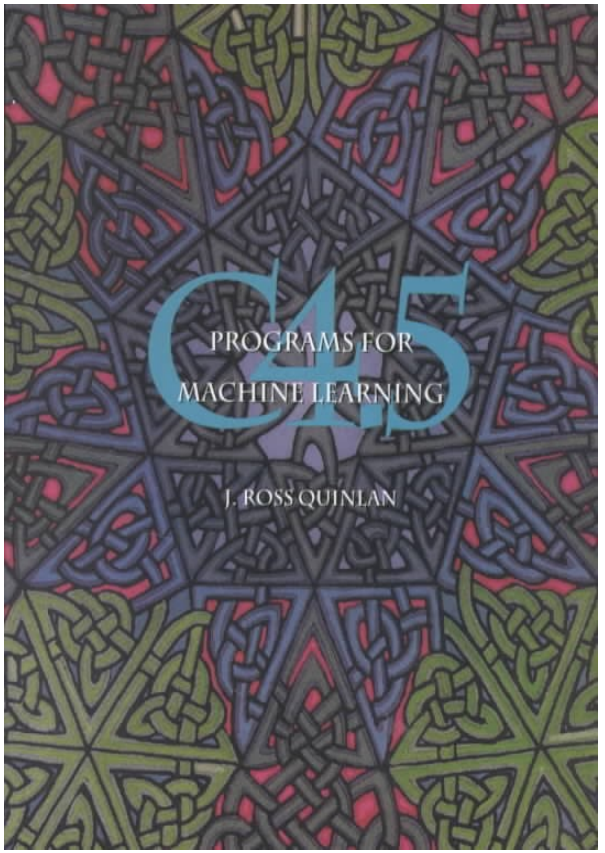
- Overcoming the limits associated with manual mining for features:
  - Convolutional neural networks

Reading material



Data Mining: Practical Machine Learning Tools and Techniques

Ian H. Witten



C4.5: Programs for Machine Learning

J. Ross Quinlan