Chapter 7
Feature Extraction

Abstract In the previous chapters we have examined static and dynamic methods of program analysis. These features must be translated into mathematical representations and birthmarks to be useful. Furthermore, mathematical representations may be embedded in other mathematical types to make birthmarks more amenable to similarity comparisons and for use in classification algorithms. Another approach is to represent features using kernels. This allows for the use of classification algorithms including the support vector machine for complex data types. This chapter examines the mathematical representations that we use to describe program features.

Keywords Program feature processing · Strings · Vectors · Sets · Sets of vectors · Trees · Graphs · Embeddings · Kernels

7.1 Processing Program Features

Program features are the basis of software similarity and classification, but must be transformed or into a meaningful representation that allows for similarity comparisons and indexing. Different representations are possible ranging from highly efficient but least expressive, to highly expressive but least efficient. For example, representing birthmarks as vectors allows for very efficient comparisons, but tends to lose structural information that is present in graph based representations.

Combining features into a unified form may result in the establishment of software metrics. Attribute counting is one approach. Attributes that can be tallied might include the number of specific keywords, the number of conditionals, the number of loops and so forth. The final metric is the set of counted attributes. Processing might be done on these counted attributes to result in other measures.

S. Cesare and Y. Xiang, *Software Similarity and Classification*,
SpringerBriefs in Computer Science, DOI: 10.1007/978-1-4471-2909-7_7,

The Halstead complexity measures [1] are a set of software metrics that uses attribute counting at its core to give a measure on a programs complexity. Its initial use was for the purpose of software maintenance metrics but it has also been applied to software similarity.

Another approach to combine the expressiveness of complex objects, such as graphs, is to transform or embed one representation into another. For example, a graph can be transformed into a vector based representation. Information is lost, but in many cases this is still useful as a birthmark.

7.2 Strings

A string describes a sequence of tokens or characters. An example of a string could be a sequence of instruction opcodes making up a program path.

Definition 7.1 Let $\sum$ be an alphabet of symbols. Let s be a string over the alphabet where $s \in \sum^*$.

7.3 Vectors

Vectors are one of the simplest representations and are efficient to work with. A vector is an ordered list or tuple of a fixed number of elements or dimensions. A feature vector describes the frequency of particular features occurring. If the number of features is very large then dimensionality reduction can be used to filter unimportant features, or combine features together such as when using Principle Component Analysis (PCA).

Examples of using vectors include describing features based on the occurrence of a specific n characters or n-grams.

7.4 Sets

A set is a collection of unique objects. A set of features is sometimes a useful representation. It ignores ordering of those features. An example use of sets is to describe the set of API calls a program makes.

7.5 Sets of Vectors

A set of vectors may sometimes be useful. If we consider that a procedure can be represented as a vector, then the set of procedures can be represented as a set of vectors.

7.6 Trees

Trees capture the structure of data, but are not as general as graphs. A tree is a connected undirected graph without cycles. Abstract syntax trees and parse trees are naturally represented by trees. Structured control flow can also be represented by trees. Trees can have a defined ordering of child nodes or be unordered.

7.7 Graphs

Graphs model structure in the data. Many program features are naturally represented as graphs include control flow graphs, call graphs, and dependency graphs.

Definition 7.2 A graph is $g = (V, E)$ where V is a set of vertices. $E = \{(u, v) \mid u, v \in V\} \subseteq V \times V$

Definition 7.3 A labelled graph $g = (V, \alpha, \beta)$ where V is a set of vertices $\alpha : V \to L$ is the node labelling function, and $\beta : V \times V \to L$ is the edge labelling function.

7.8 Embeddings

Strings may be embedded in vectors. To reduce the string problem into an n-gram vector problem, a string may be divided into n-grams where the specific n-grams represent features.

Definition 7.4 Given a set of strings L, and a set of vectors V there is a function f such that $f : L \to V$

Strings may be embedded in sets. To reduce the string problem into a set problem, a string may be divided into n-grams or shingles where the unique n-grams represent set elements.

Definition 7.5 Given a set of strings L, and a set of sets S there is a function f such that $f : L \to S$

Trees may be embedded in vectors. A tree may be decomposed into fixed sized subtrees. These subtrees can represent features in a feature vector.

Definition 7.6 Given a set of trees T, and a set of vectors V there is a function f such that $f : T \to V$

Trees may be embedded in sets. Similar to a tree to vector problem, decomposing the tree into unique features can be represented by sets.

Definition 7.7 Given a set of trees T, and a set of sets S there is a function f such that $f : T \to S$

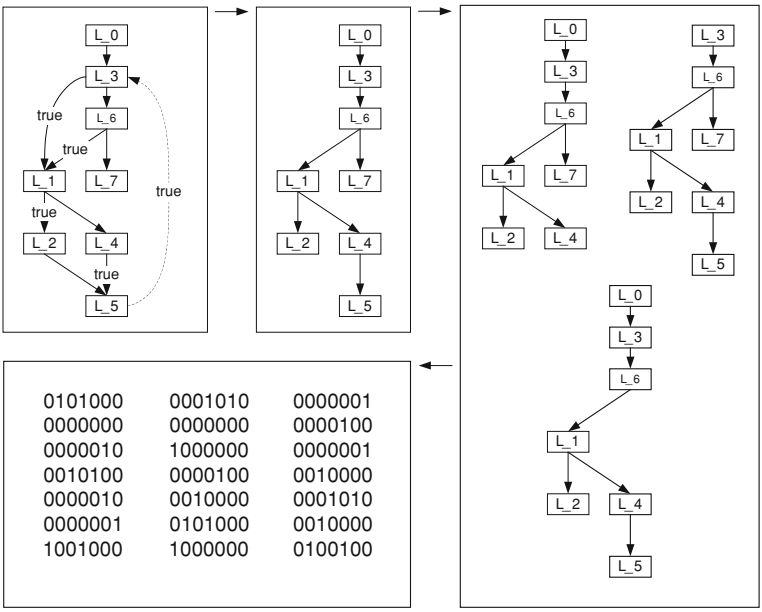

Fig. 7.1 The k-subgraph feature for a graph embedding in a vector

A graph may be embedded in a vector. A graph can be decomposed into fixed sized k-subgraphs. One approach is to construct a spanning tree and then extract the subgraphs. These subgraphs can be canonized into strings and used to represent features in a feature vector. Another approach to embedding a set of control flow graphs into a vector is by embedding the graphs into strings using decompilation and then embedding the strings into vectors using k-grams [2] (Fig. 7.1).

Definition 7.8 Given a set of graphs G, and a set of vectors V there is a function f such that $f : G \rightarrow V$

A graph may be embedded in a set. Transforming a graph into a set is analogous to a graph to vector problem.

Definition 7.9 Given a set of graphs G, and a set of sets S there is a function f such that $f : G \rightarrow S$

A graph may be embedded in a tree. A graph can be represented by tree by constructing a spanning tree.

Definition 7.10 Given a set of graphs G, and a set of trees T there is a function f such that $f : G \rightarrow T$

7.9 Kernels

Kernels are most used in kernel based statistical machine learning classifiers. A kernel function operates in feature space which is typically of much higher dimensionality. A string kernel based on the subsequences in the string known as a subsequence kernels was proposed in [3]. A kernel for sets of features was proposed in [4]. A kernel for vector sets was proposed in [5]. A kernel for trees was proposed in [6]. A kernel based on random walks in a graph was proposed in [7]. Subtree kernels have been proposed. A kernel based the set of all paths in a graph has also been proposed. A kernel based on the shortest paths in a graph was proposed in [8].

7.10 Research Opportunities

Embeddings and kernels present a significant opportunity for researchers. Embeddings have been investigated somewhat, but a comprehensive treatment of different embeddings for different structures has not been performed in the context of software similarity. Kernel methods are effectively unused in software similarity and this presents many opportunities for researchers to apply kernel methods to so the software similarity and classification problem. Graph kernels could be used to perform software classification in applications such as malware classification.

References

1. Halstead MH (1977) Elements of software science (operating and programming systems series). Elsevier Science Inc, NY
2. Cesare S, Xiang Y (2011) Malware variant detection using similarity search over sets of control flow graphs. In: IEEE Trustcom
3. Lodhi H, Saunders C, Shawe-Taylor J, Cristianini N, Watkins C (2002) Text classification using string kernels. J Mach Learn Res 2:419–444
4. Grauman K, Darrell T (2007) The pyramid match kernel: efficient learning with sets of features. J Mach Learn Res 8:725–760
5. Kondor R, Jebara T (2003) A kernel between sets of vectors. In: Proceedings of ICML'2003, vol 1, p 361
6. Collins M, Duffy N (2002) Convolution kernels for natural language. Adv Neural Inf Process Syst 1:625–632
7. Kashima H, Inokuchi A (2002) Kernels for graph classification, p 25
8. Borgwardt KM, Kriegel HP (2005) Shortest-path kernels on graphs. In: Data mining