

# Using Active Learning with Text Filtering to Generate a Support Vector Machine Training Set

Joseph M. Geyer

23 March 2009

## Abstract

The need to understand software systems is an important part of their update and maintenance. If one does not understand a software system, he/she will have difficulty modifying, maintaining, or updating it. This can be costly in terms of both time and money. Reverse software engineering alleviates this by creating models of a system to aid system comprehension. A well known problem in this domain is the concept assignment problem [2]. This is the task of assigning human level concepts or meaning to the code that actuates it. This problem can be extended to the class level where the goal is to assign concepts to classes. Carey and Gannod [6] have automated the classification of the concept classes in a software system by using support vector machines. Support vector machines require a training set to train the learner, however, manually labeling this training set is inefficient. The goal of the proposed research is to present a method and tool to semi-automate the creation of a training set for support vector machine learning in the context of reverse software engineering.

## 1 Introduction

The documentation of the design of legacy software systems is often neglected as the system ages. It is easy for updates and modifications to be made to a software system without those changes reflected in the design models. Knowing the architecture of a software system is critical to its continued efficacy. The process of regaining this knowledge is termed design recovery. Reverse software engineering is important in the maintenance, upkeep, and use of software systems. Maintaining software systems is difficult for many reasons. It is often the case that the designers of a software system are not the same people who will maintain the system [7]. This can cause a gap in the knowledge about the system. The same principle is true for upgrading and using a software system. It is important to fully understand the system in order to best maintain, upgrade, and use it. Chikofsky and Cross [7, page 15] define reverse software engineering as “the process of analyzing a subject system to identify the system’s components and their interrelationships and create representations of the system in another form or at a higher level of abstraction.” There exists an increasing demand for the ability to create models and blueprints of the design of software systems.

A well known problem in the domain of reverse software engineering is the concept assignment problem [2]. This is the task of assigning human level concepts to the code that actuates it. Carey and Gannod [6] have developed a tool to automate the classification of the concept classes in a software system by using support vector machines. However, this requires a great amount of work to manually label a training set of classes

as either concept or non-concept to train the learner. The goal of the research presented in this paper is to present a method to reduce the load on this aspect of training the support vector machine

Dietterich [8] defines machine learning as “the study of methods for programming computers to learn”. There are many applications for machine learning. Data mining uses machine learning algorithms to find knowledge from data. For example, data mining medical records can lead to medical knowledge [12]. Some other examples are speech recognition, spam classification, autonomous driving, and book recommendation programs. In each of these examples, the actions of the computer are not explicitly programmed. Learning can be thought of as the process of training a function to correctly give an output based on the previous examples it was given. A binary classification learning problem is where each data example is in one of two distinct classes. Thus, the learning function is being trained to take as input the features of the example and to give as output the classification to which the input features is mapped. In our problem, the training examples are classes in a software system where the input features are object oriented metrics about the class. The output is whether that example should be classified as a concept class or a non-concept class.

An ontology is a way to better understand a domain. Ontologies consist of concepts in a domain and their interrelationships and hierarchical structure. Algorithms and tools have been developed to reason and draw conclusions in an ontology. Thus, new knowledge is often created via ontologies. Another important use for ontologies is the sharing of information. Since ontologies are built with strict logic rules with hierarchy, users who are interested in a certain domain can often use previously created ontologies related to that topic. The creation of ontologies can either be done manually, semi-automatically, or automatically. Consider the task of creating an ontology from text. Ontology creation from text has its roots in text mining. Often, clustering is used [5, 19, 20, 11, 10] to group words into groups. Hierarchy can be added to these clusters by parsing the text and looking for hierarchical phrasing. For example, the phrase “...*skeleton*, *bobsleigh*, and other *winter olympic sports*...” indicates that *skeleton* and *bobsleigh* are types of *sports*. It also indicates that *olympic sport* is a type of *sport* and that *winter olympic sport* is a type of *winter sport*. [5] Key concepts from the text can then be readily accessed from the ontology that was created.

## 2 Background and Related Work

### 2.1 Background

In the domain of reverse software engineering, the concept assignment problem is a classic task for recovering design rationale. Another important technique is that of active learning where the user observes the results of the software tool and indicates whether it was correct or not. Finally, in machine learning supervised learning is used successfully in many cases. We are interested in classification area of machine learning where we can classify an item in one of two different classes.

#### 2.1.1 Machine Learning

Machine learning uses algorithms so that computers can solve problems without being explicitly programmed. The machine changes its behavior to gain improved performance for subsequent iterations of the problem. [12]. Machine learning is a specialized field of artificial intelligence and can be broken up into further categories. Supervised machine learning is where training samples are labeled with the appropriate output. If the output

is continuous, it is known as a regression problem. If the output is in the range of a small number of discrete variables, it is a classification problem. For our problem, we will focus on this - classification supervised learning. Unsupervised learning is where no knowledge is given about the output of the training samples. A common solution for this type of problem is to use clustering. Unsupervised learning problems can also be regression or classification. Reinforcement learning is where a sequence of decisions are made with a reward function. An example of this is training a robot to drive a car. There is not a single classification being done for this task, or a single target value to achieve. The goal is a series of acceptable actions for a larger goal.

For our research, we are interested in supervised learning for a binary classification problem. This means that we have a training set with labeled outputs that can be used to train the learner. In a general sense, the problem can be defined in the following way. The variable  $x$  is a vector of input variables and  $y$  is the output variable and can either be a value from  $\{-1, 1\}$ . The vector  $x$  consists of any number of features depending on the problem. Thus, the feature vector or input vector with  $n$  features is

$$x = \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix}$$

The ordered pair  $(x^{(i)}, y^{(i)})$  is called a training examples. The training set is composed of  $m$  training examples so that  $i$  has a range from 1 to  $m$ .

$$trainingset = \begin{bmatrix} (x^{(1)}, y^{(1)}) \\ (x^{(2)}, y^{(2)}) \\ \vdots \\ (x^{(m)}, y^{(m)}) \end{bmatrix}$$

We want to use the training set to learn a function  $h : X \rightarrow Y$ , called the hypothesis, where  $X$  is the space of input variables and  $Y$  is the space of output variables. This mapping should accurately classify a sample based on the value of its feature vector. The goal of learning this function is to find the parameter vector  $\theta$  that can be used to get accurate performance.

One way to understand support vector machines is to visualize the problem geometrically. Each training example in the training set can be placed in a hyperplane of degree  $n$  where  $n$  is the number of features in the feature vector  $x$ . Each example point has output of either +1 or -1 and can be identified with some binary identification scheme, like using a circle for +1 or a triangle for -1. If the two sets of points are linearly separable, a hyperplane can be used to separate the samples into the two classifications. But, there are infinitely many separating hyperplanes. The goal then is to find the hyperplane that gives the maximal distance between the closest vectors on either side. These vectors that are closed are called the support vectors. This problem becomes more complicated when the training examples are not linearly separable. However, by projecting these samples into a higher dimensional hyperplane, the points can be linearly separable.

### 2.1.2 Reverse Engineering and the Concept Assignment Problem

In order to maintain or update a software systems, it is important to understand it. Reverse software engineering takes a software systems and extracts knowledge about it's design to further the understanding of the system. Chikofsky and Cross [7, page 15] define reverse software engineering as the the ability to “identify the system’s components and their interrelationships and create representations of the system in another form or at a higher level of abstraction.” With the number and complexity of legacy software systems, there is a real demand for reverse software engineering.

Reverse software engineering is basically a two step process [4].

- extracting information
- abstracting information

These abstractions are models to help the user better understand the system. Two of the subcategories under reverse software engineering are redocumentation and design recovery [4, 7]. Redocumentation is where new models or abstractions are created that just give a different view of the system. No new meaning is given, usually just a more useful perspective. An example of this is printing out the source code in more readable text or creating diagrams directly from source code [7]. Design recovery is more invasive and results in new knowledge about the system. The ultimate goal for design recovery is that the user can have full knowledge of the system.

The famous concept assignment problem presented by Biggerstaff [2] is an example of design recover that involves two parts. The first is to identify the real human concepts that are in a software system. For example, a concept might be to “calculate revenue”. Then the user assigns that concept (calculate revenue) to the code that accomplishes that concept. This was first described on a line-by-line micro level of the code. However the principle still applies to a larger scale when we consider the class level. In Carey and Gannod’s [6] work, they consider classes as either concept or non-concept classes. They assigned each class a vector of object oriented metrics and used machine learning to classify the concept and non-concept classes. They used supervised learning with cross validation to assess the validity of their classifier.

### 2.1.3 Active Learning

Active learning is where the user can play a part in the acceptance or rejection of the classifications of the learner. This was successfully used in the the work of Bowring et al. [3]

## 2.2 Related Work

Sartipi did some research on reverse engineering with machine learning. Others also...

### 2.2.1 Reverse Engineering through Data Mining

Sartipi et al. [14, 13] have used data mining with reverse software engineering. The view of the structure of a legacy software system is important in understanding it. The goal of their research is to create an improved view of the structure of the software system and a tool to actually restructure the system using an architectural query language. The queries are matched based on clustering using a score function. They

define a module in a software system as the set of functions, the set of datatypes, and the set of variables. These sets are all defined in a similar way. Consider the set of functions in a module. The definition of this set is that they are either contained in the module or they are imported from another module. They can also be exported to other modules. Using clustering, a frequent itemset is created based on how often a function calls another function, uses a datatype, or uses a variable. These clusterings are candidates for a module. Finally, these modules create the structural view of the software system.

This research is different from ours for several reasons. The ultimate goal of their research is to create a structural view or possibly restructure a software system where we are ultimately trying to identify classes as either concept or not. But more specifically, the research we are proposing involves the semi-automatic creation of a machine learning training set, where they are using data mining to look at clustering of module based on the source code.

Shirabad et al. [16] also conducted research in reverse software engineering with a data mining approach. They considered the problem of identifying the interdependence of files or routines on a given block of code. In other words, when a user encounters some code, what are the other files or routines that would be relevant or helpful in understanding and maintaining that piece of code? Thus, two files can either be relevant or not relevant to each other. They also considered the classification that they could be potentially relevant. Data was obtained from logging user's normal interactions with the files in a system. If a user interacts with two files in the same session, they are considered to be more related to each other. This and other attributes make up the features in a data sample. Then, the data is classified and compared to the actual classifications.

### 2.2.2 Automatic Training Set Creation

TODO: Tang [17] developed a method for automatically creating a training set for video annotation.

### 2.2.3 Ontology creation through Clustering

Clustering is a method used for ontology building from text [5, 19, 20, 11, 10]. Early work involved simpler similarity metrics and algorithms while current work is much more sophisticated. The general idea is to build the ontology bottom up. After identifying concept terms, a similarity metric is used to measure the closeness of terms. Depending on the closeness, the concept terms will be clustered together. These clusters can represent a new concept. The process continues with these new clusters, building a hierarchy.

Caraballo [5] developed an algorithm for automatically creating a hypernym-labeled noun hierarchy from text corpus. Using the Wall Street Journal, she identified 50,000 nouns. When two or more nouns are used together in a conjunction or an appositive, the assumption is that they are semantically related. Each noun is given a vector where the values are the number of times each other noun is found in either a conjunction or an appositive in the text. Similarity between nouns is then calculated by comparing the angle between each pair of vectors. The most similar nouns are clustered into a new parent node, and the process continues. Hierarchy is then extracted by looking for the use of the word *other* with nouns in a conjunction clause. For example, *birds, squirrels, and other small mammals* indicates that mammal is a hypernym of birds and squirrels. This algorithm correctly assigned hyponyms to randomly selected hypernyms with a 33% accuracy according to human judges participating in the experiment. One of the top three hypernyms that the algorithm produced for a noun matched the judge's hypernym with a 60% accuracy.

Instead of a bottom up approach, Yang and Callan [18] proposed an incremental approach where each term is considered and placed in the correct spot in the hierarchy.

### 3 Proposed Research

This section discusses the research to be conducted. It starts by defining the research problem and the proposed solution. Next it examines the feasibility and validity measures of the proposed solution. This section also includes some preliminary results conducted on a UML Modeling software system. Finally, a timetable for completing the research is presented.

#### 3.1 Specific Research Problem

Classification problems with supervised learning algorithms allow the machine to create a learner from the training set that will be applied to the rest of the data set. The user must manually classify a subset of the data so that the learner can automatically classify the rest of the instances in the data set. As an example, if a researcher wants to create a training set that is 10% of a software system with 5000 classes, he/she would have to manually classify 500 classes. This creates a bottleneck in the productivity of the reverse software engineering process. Through this research, we want to reduce the load of manually classifying the training set.

#### 3.2 Proposed Solution and Methodology

The proposed solution will be implemented in an eclipse plugin that utilizes filtering and active learning techniques. This plugin will be an extension on the plugin by Carey et al. [6] The flowchart in figure 1 shows a model of the proposed solution. As shown in figure 1, there are two points of input for our system - the *software system domain document* and the *class names*. The *software system domain document* is a text document that is about the domain of the software system and the *class names* are the actual names of the classes of the software system. From here, our system is automated until the active learning stage, where the user can accept or reject examples for the training set.

##### 3.2.1 Text to Ontology

Software systems have a concept domain associated with its high level functioning. Some domain examples are given below.

- Database
- IDE
- Data Mining
- UML Modeling
- Chat Client
- Text Editor

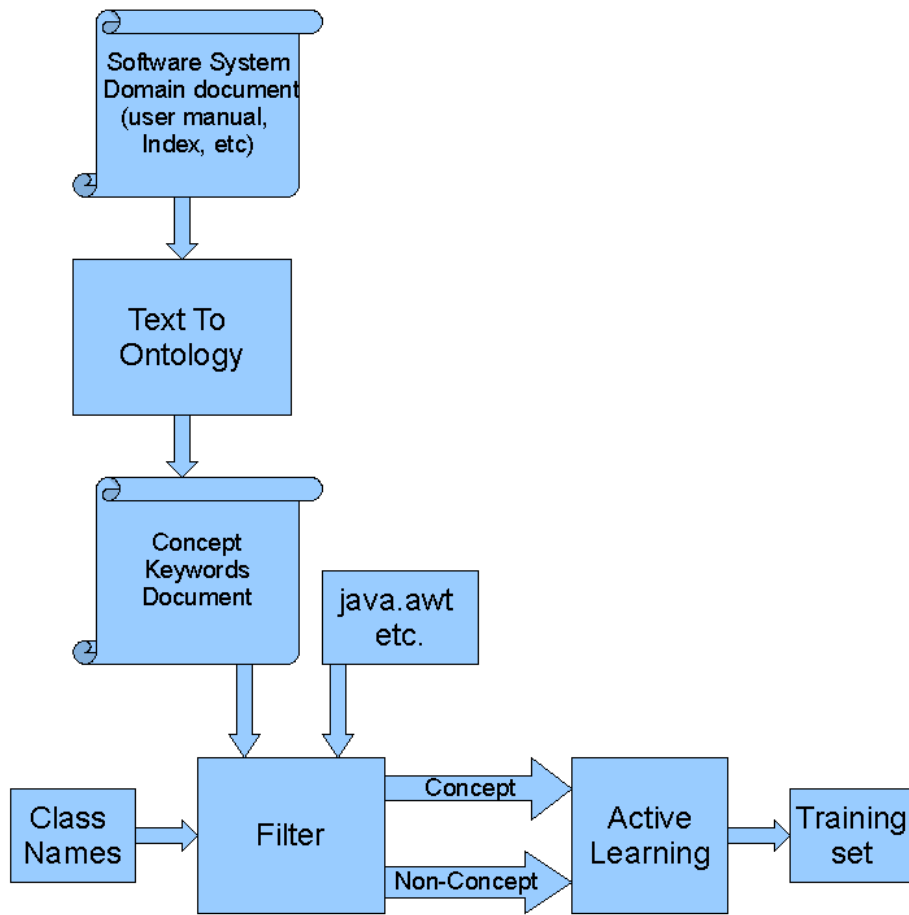


Figure 1: Flowchart Model of Proposed Research

- Online Store
- Chat Client
- Project Manager

Consider a software system used for UML development. In UML, some examples of conceptual terms are *inheritance*, *use case*, and *association*. We would expect that some of the classes in this system would use these words as part of the class name or as the whole class names. These terms can be taken directly out of a textbook or other document on UML. The same is true of other domains as well. We can extract these terms from any document (user's guides, manuals, textbooks, tutorials, etc.) associated with the concept domain of the system. We will use this document as input into the text-to-ontology builder (see figure 1. This will create an ontology of all the words in the document. We are especially interested in the clustering and hierarchical structure of this ontology. The ontology will group concepts into clusters and hierarchies that we can use to build our concept filter. Specifically, the set of these keywords will make up the Concept Keywords Document as shown in figure 2.

### 3.2.2 Training Set Selection via Filtering

The filtering stage consists of a concept filter and a non-concept filter as shown in figure 2. In our problem, every class is either a *concept class* or *other*. The SVM learner created by Carey [6] requires the user to manually classify at least one positive example and at least one negative example. The goal of this step is to automatically filter out probable concept classes and those classes that are most likely not a concept class.

These keywords together make up the *concept filter*. That is, these are the terms to which the class names will be compared. Likewise there are other terms that are usually associated with the inner workings of a software system. We compiled a list of 98 terms that would filter out classes that are probably not concept classes. These terms are from the java.awt package. Some examples are *JButtonItem*, *Scrollbar*, and *JTextPane*. These non-concept keywords are used in the *non-concept filter*.

The filtering of the classes takes place by examining matches of substrings of the software systems class names with substrings elements of the *concept filter* and the *non-concept filter*. If a match occurs on the *concept filter*, then the class name is placed in a list of probable concept classes. If a match occurs on the *non-concept filter*, then the class name is placed in a list of probable non-concept classes. If a class does not get matched by either filter, then it is not placed in either list. In this way, we have reduced the number of classes to choose from for manual classification.

### 3.2.3 Training Set Selection via Active Learning

Next we allow the user to accept or reject the automatic filtering. Figure 1 shows the concept and non-concept classes as input into the active learning module. Here the user will be presented with the list of probable concept classes generated by the *concept filter*. The user accepts individual classes as concept by using a checkbox. In accepting a class as a concept class, the user is finalizing the positive examples for the training set for the learner. Similarly, the user will be presented with the list of probable non-concept classes generated by the *non-concept filter*. Again, the user accepts individual classes as non-concept classes by clicking a checkbox. After this process is complete, the training set has been finalized, complete with

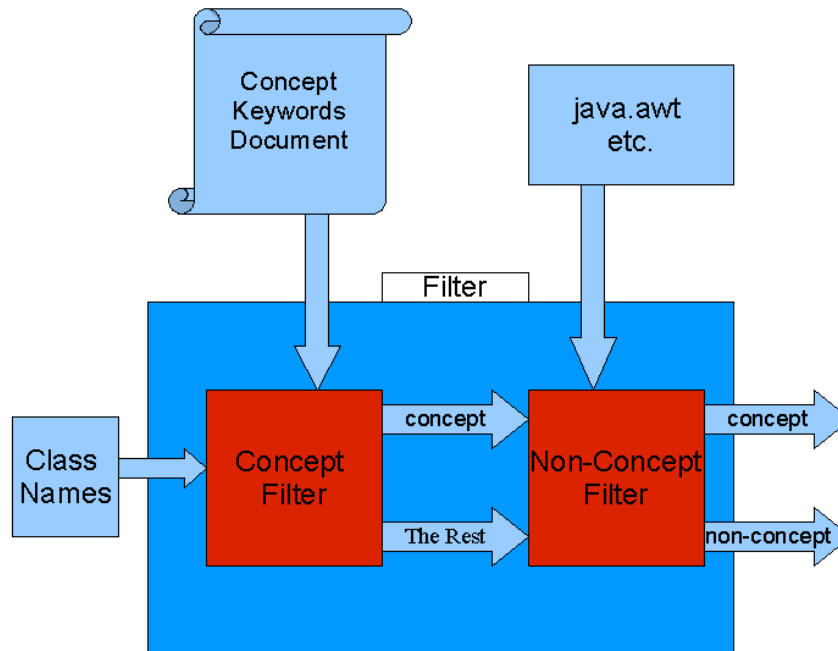


Figure 2: Model of Filter

positive and negative examples. This training set can then be used with the support vector machine to classify all of the classes in the system as either concept or other.

### 3.3 Feasibility

We will expand on the support vector machine annotation classification software developed by Carey [6] and expect this to be available. Our tool will be written as a plugin using Eclipse. Data will be acquired from open source software systems like ArgoUML. We will manually label the classes in our experiment and use this as the oracle for our research.

### 3.4 Validation Approach and Measures

Gueheneuc et al. [9] have developed a framework for evaluating design recovery tools. This framework consists of fifty-three criteria in eight different categories. These categories include the “the *Context* in which it is applied, its *Intent*, its *Users*, its *Input* and *Output*, the *Technique* which it implements, its actual *Implementation*, and the *Tool* itself.” We will use this framework, and the fifty-three criteria to help guide and evaluate the design and goals of our design recover tools. This will help ensure that our tool meets a specific need in the reverse software engineering community.

TODO: For this section, review some of the statistics used by Carey. Also consider exploring the quality of the training set as a result. For example, maybe the technique for generating a training set falls within the acceptable categories, but is it complete insofar as it describes or defines what a concept class should

and should not be?

### 3.5 Preliminary Results

TODO: After cleaning up some of the statistics, this section will contain some of the initial results from the ArgoUML experiments.

Argo UML is a UML modeling software systems. Since the domain of the systems is UML, I used a glossary of UML terms found online. TODO: Find this source. Also put the document as an appendix to this proposal? I parsed through this document to extract all of the terms.

### 3.6 Timetable

Date	event
May 12	Thesis Proposal written
August 30	Thesis Proposal
TBA	Implement tool
TBA	Evaluate tool
TBA	Experiment and collect results
TBA	Analyze results
TBA	Thesis Draft
TBA	Final Thesis Draft written
TBA	Defend Thesis

## 4 Conclusion

## References

- [1] Kristin P. Bennett and Colin Campbell. Support vector machines: hype or hallelujah? *SIGKDD Explor. Newsl.*, 2(2):1–13, 2000.
- [2] Ted J. Biggerstaff, Bharat G. Mitbander, and Dallas E. Webster. Program understanding and the concept assignment problem. *Commun. ACM*, 37(5):72–82, 1994.
- [3] James F. Bowring, James M. Rehg, and Mary Jean Harrold. Active learning for automatic classification of software behavior. In *ISSTA '04: Proceedings of the 2004 ACM SIGSOFT international symposium on Software testing and analysis*, pages 195–205, New York, NY, USA, 2004. ACM.
- [4] Gerardo CanforaHarman and Massimiliano Di Penta. New frontiers of reverse engineering. In *FOSE '07: 2007 Future of Software Engineering*, pages 326–341, Washington, DC, USA, 2007. IEEE Computer Society.
- [5] Sharon A. Caraballo. Automatic construction of a hypernym-labeled noun hierarchy from text. In *Proceedings of the 37th annual meeting of the Association for Computational Linguistics on Computational Linguistics*, pages 120–126, Morristown, NJ, USA, 1999. Association for Computational Linguistics.

- [6] M.M. Carey and G.C. Gannod. Recovering concepts from source code with automated concept identification. pages 27–36, June 2007.
- [7] E.J. Chikofsky and II Cross, J.H. Reverse engineering and design recovery: a taxonomy. *Software, IEEE*, 7(1):13–17, Jan 1990.
- [8] T. G. Dietterich. Machine learning. In *Nature Encyclopedia of Cognitive Science*. Macmillan, 2003.
- [9] Y.-G. Gueheneuc, K. Mens, and R. Wuyts. A comparative framework for design recovery tools. pages 10 pp.–134, March 2006.
- [10] He Hu and Da-You Liu. Learning owl ontologies from free texts. volume 2, pages 1233–1237 vol.2, Aug. 2004.
- [11] Hyunjang Kong, Myunggwon Hwang, and Pankoo Kim. Design of the automatic ontology building system about the specific domain knowledge. volume 2, pages 4 pp.–1408, Feb. 2006.
- [12] Mitchell. *Machine Learning*. McGraw-Hill Education (ISE Editions), October 1997.
- [13] K. Sartipi. Software architecture recovery based on pattern matching. pages 293–296, Sept. 2003.
- [14] K. Sartipi, K. Kontogiannis, and F. Mavaddat. Architectural design recovery using data mining techniques. pages 129–139, Feb 2000.
- [15] Jochen Seemann and Jürgen Wolff von Gudenberg. Pattern-based design recovery of java software. In *SIGSOFT '98/FSE-6: Proceedings of the 6th ACM SIGSOFT international symposium on Foundations of software engineering*, pages 10–16, New York, NY, USA, 1998. ACM.
- [16] Jelber Sayyad Shirabad, Timothy C. Lethbridge, and Stan Matwin. Supporting maintenance of legacy software with data mining techniques. In *CASCON '00: Proceedings of the 2000 conference of the Centre for Advanced Studies on Collaborative research*, page 11. IBM Press, 2000.
- [17] Jinhui Tang, Yan Song, Xian-Sheng Hua, Tao Mei, and Xiuqing Wu. To construct optimal training set for video annotation. In *MULTIMEDIA '06: Proceedings of the 14th annual ACM international conference on Multimedia*, pages 89–92, New York, NY, USA, 2006. ACM.
- [18] Hui Yang and Jamie Callan. Metric-based ontology learning. In *ONISW '08: Proceeding of the 2nd international workshop on Ontologies and nformation systems for the semantic web*, pages 1–8, New York, NY, USA, 2008. ACM.
- [19] Hui Yang and Jamie Callan. Ontology generation for large email collections. In *dg.o '08: Proceedings of the 2008 international conference on Digital government research*, pages 254–261. Digital Government Society of North America, 2008.
- [20] Jingtao Zhou, Mingwei Wang, Han Zhao, Shusheng Zhang, and Chao Zhang. Concept capture based on column matching and clustering. pages 71–71, Nov. 2005.