CS 6604 Digital Libraries

Spring 2014

Final Project Report

Ensemble Classification

Project Members

Vijayasarathy Kannan
Mohammed Alabdulhadi

Manikandan Soundarapandian
Tania Hamid

Project Client
Yinlin Chen

Project Advisor
Dr. Edward A. Fox

8 May 2014
Virginia Tech, Blacksburg

Executive Summary

Transfer learning[1] unlike traditional machine learning is a technique that allows domains, tasks and distributions used in training and testing to be different. Knowledge gained from one domain can be utilized to learn a completely different domain.

Ensemble computing portal[2]is a digital library that contains resources, communities and technologies to aid in teaching. The major objective of this project is to apply the learning gained from the ACM Computing Classification System and classify educational YouTube videos so that they can be included in the Ensemble computing portal.

Metadata of technical papers published in ACM are indexed in a SOLR server and we issue REST calls to retrieve the required metadata viz. title, abstract and general terms that we use to build the features. We make use of the ACM Computing Classification System 2012's classification hierarchy to train our classifiers. We build classifiers for the level-2 and level-3 categories in the classification tree to help in classifying the educational YouTube videos.

We utilize YouTube data API to search for educational videos in YouTube and retrieve the metadata including title, description and transcripts of the videos. These become the features of our test set. We specifically search for YouTube playlists that contain educational videos as we found out from our experience that neither a regular video search nor a search for videos in channels do retrieve relevant educational videos.

We evaluate our classifiers using 10-fold cross-validation and present their accuracy in this report. With the classifiers built and trained using ACM metadata, we provide them the metadata that we collect from YouTube as the test data and manually evaluate the predictions. The results of our manual evaluation and the accuracy of our classifiers are also discussed.

We identified that the ACM Computing Classification System's hierarchy is sometimes ambiguous and YouTube metadata are not always reliable. These are the major factors that contribute to the reduced accuracy of our classifiers. In the future, we hope sophisticated natural language processing techniques can be applied to refine the features of both training and target data, which would help in improving the performance. We believe that more relevant metadata from YouTube in the form of transcripts and embedded text can be collected using sophisticated voice-to-text conversion and image retrieval algorithms respectively. This idea of transfer learning can also be extended to classify the presentation slides that are available in slideshare (http://www.slideshare.net) and also to classify certain educational blogs.

TABLE OF CONTENTS

List of Figures

List of Tables

I. INTRODUCTION

Transfer learning [1]is a type of machine learning technique that helps the learning process of a new task by transferring the knowledge learned in a previous task. Figure-1 illustrates the difference between the learning processes of traditional and transfer learning techniques. As we can notice, traditional machine learning tries to learn each task from scratch. On the other hand, transfer learning makes use of the knowledge acquired from previous tasks to learn a target task or domain. The target task usually has small amount of good quality training data.

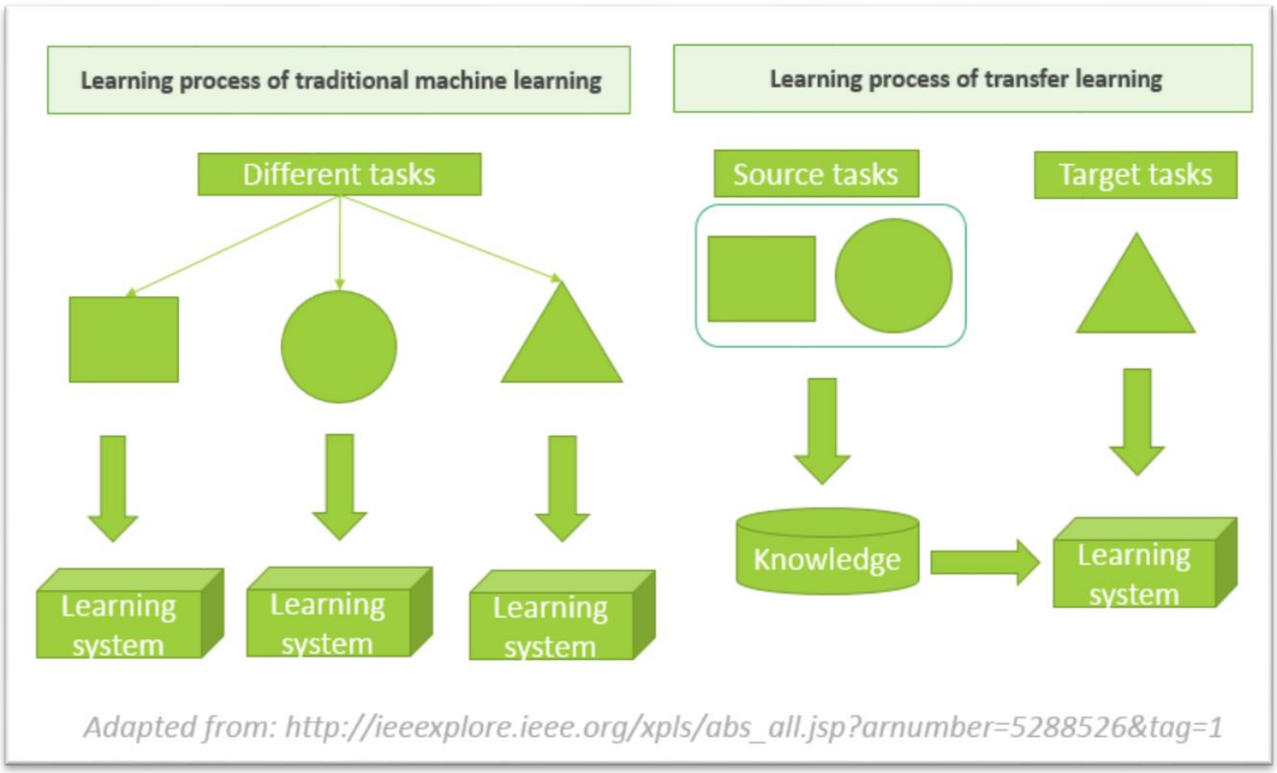
Learning process of traditional machine learning
Learning process of transfer learning
Different tasks
Source tasks
Target tasks
Knowledge
Learning system
Learning system
Learning system
Learning system
Adapted from: http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=5288526&tag=1

Figure-1: Traditional machine learning vs. Transfer learning

The main objective of this project is to utilize the transfer learning concept in order to develop classifiers which will be used to classify educational resources for the Ensemble portal. The main classification technique used in this project is text classification. The training data are pre-classified ACM metadata classified based on the 2012 ACM Computing Classification System (CCS). On the other hand, the target data are educational YouTube videos belonging to several computing fields.

II. PROJECT BIG PICTURE

Figure-2 illustrates the big picture of the project. Initially, ACM metadata is retrieved using REST calls issued to our SOLR server. After formatting the metadata, we place them to Weka (a popular machine learning tool) where we filter the metadata using Weka's "String to Word Vector" filter to get the training features. Next, we use some classification algorithms, such as Naïve Bayes, Naïve Bayes Multinomial, J48 and SMO to build our classifiers. After that, we transfer knowledge from the trained classifier to help in classifying the target data that are, primarily the metadata collected from YouTube educational videos related to computing fields. The final outcome will be classified YouTube videos.

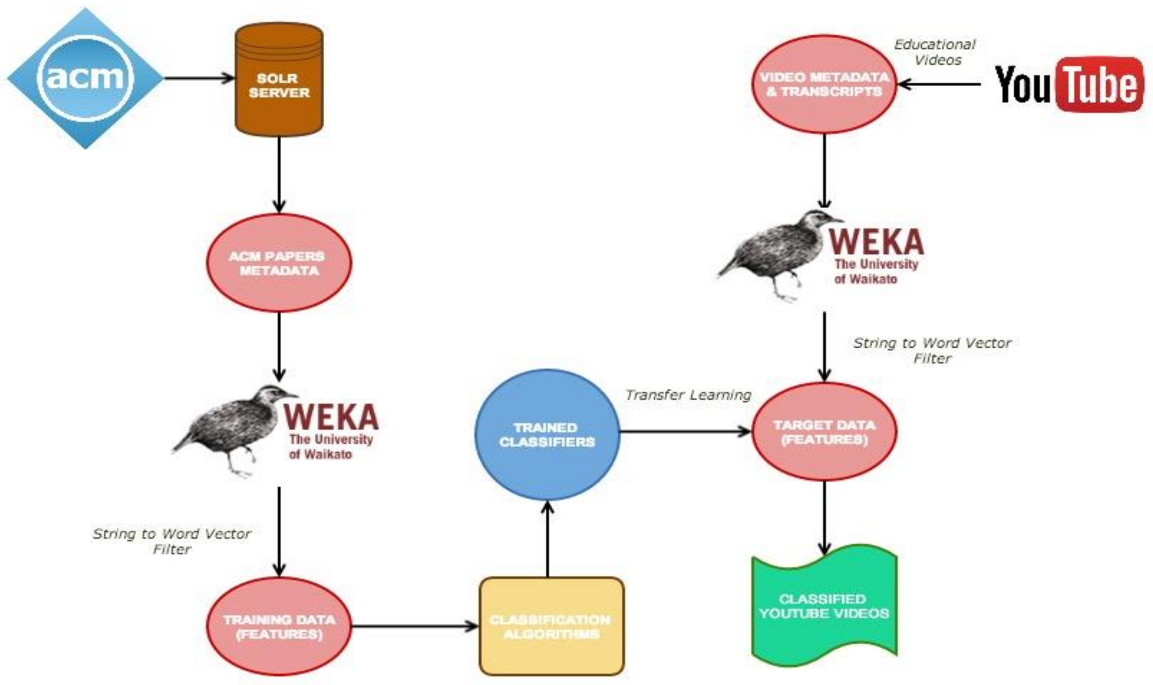
acm
SOLR SERVER
ACM PAPERS METADATA
WEKA The University of Waikato
String to Word Vector Filter
TRAINING DATA (FEATURES)
CLASSIFICATION ALGORITHMS
TRAINED CLASSIFIERS
Transfer Learning
TARGET DATA (FEATURES)
CLASSIFIED YOUTUBE VIDEOS
Educational Videos
YouTube
VIDEO METADATA & TRANSCRIPTS
WEKA The University of Waikato
String to Word Vector Filter

Figure-2: Project big picture

III. USER’S MANUAL

A user of the Ensemble classification system can customize various configurations by altering the parameters in the ***properties.json*** file. User can customize the following parameters:

- SOLR server URL
- Directories for storing training and target instances (To be loaded into ARFF files later)
- ARFF file names for training and target data
- Stop words file
- Bootstrap file
- Number of training instances
- Keywords used to search for YouTube playlists
- Classes of ACM taxonomy tree

Pre-requisite:

- JDK 1.5 or above and JRE. We recommend using Eclipse IDE.
- .jar files (attached along with the source code).

To run the classifier, follow the steps below:

1. Download and extract source code from VTechWorks.
2. Change the default parameters in the *properties.json* file, if necessary.
3. Open the project in Eclipse IDE, include all the required JAR files and run the file YouTubeClassifier3.java
4. Videos with the respective class will be displayed as output.

IV. DEVELOPER'S MANUAL

1. Concept map

Figure-3 illustrates the concept map of the project. It includes the main concepts, resources, processes and techniques used in the project.

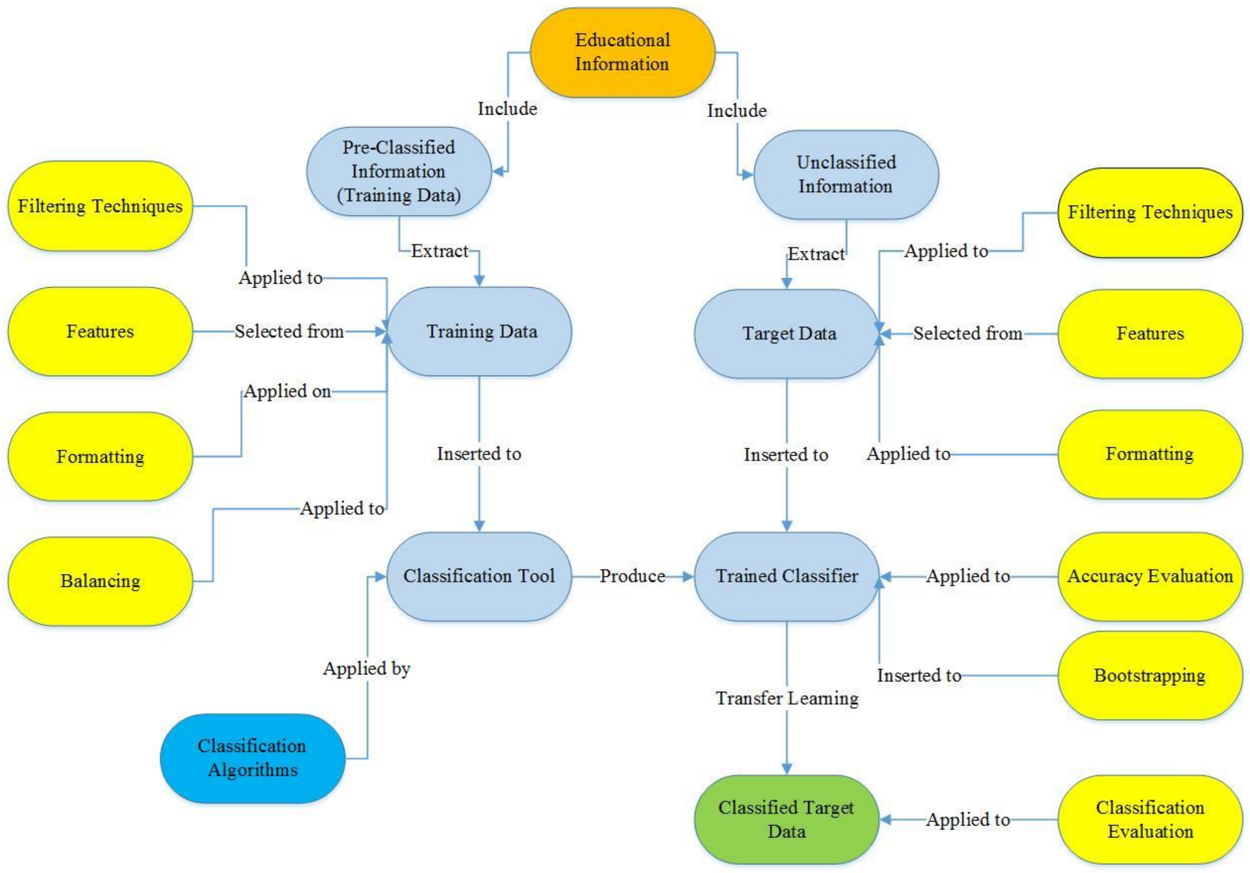

Figure-3: Project concept map

2. Inventory of Data and Program Files

Inventory of data files

The following is a list of data files (either supplied as input to the program files or files/directories created during program execution).

Table-1: List of data files

Data file name	Type of file	Description
properties.json	JSON	A JSON file with the parameters for program files
Level_2_Topics	Text	List of level-2 topics in ACM taxonomy

SearchTopics	Text	List of topics spanning the computing domain (may be used for target data collection)
Stopwords	Text	List of stop-words (used in filtering features phase)
Bootstrap	Text	List of instances (with features) of target data to be used for bootstrapping
TrainingTextFiles	Directory	A directory with sub-directories for each topic. Sub-directory for a topic *X* will be loaded into ARFF file during training on topic *X.* Each sub-directory has a separate directory for positive and negative instances.
TargetTextFiles	Directory	A directory with sub-directories for each topic. Sub-directory for a topic *X* will be loaded into ARFF file during testing on topic *X.* Each sub-directory has a separate directory for positive and negative instances.
TrainingSingleClassDataSet	ARFF	ARFF file created from *TrainingtextFiles* directory for *training* (loaded through code by Weka's TextToDirectoryLoader feature)
TargetSingleClassDataSet	ARFF	ARFF file created from text files for *TargettextFiles* directory (loaded through code by Weka's TextToDirectoryLoader feature)

Inventory of program files

This section describes the organization of program files. Please refer to the user manual for details on dependencies.

Class***YouTubeClassifier3***

Table-2: Field Summary for YouTubeClassifier3

Modifier and Type	Field and Description
String	*solrUrl* Solr URL of the collection of ACM metadata.

String	*trainingArffFile* Name of the ARFF file created from the directory of text files for training.
String	*targetArffFile* Name of the ARFF file created from the directory of text files for testing.
String	*topicFile* Name of the topic file (Level_2_Topics data file)
String	*trainingTextDir* Name of the directory containing text files of training instances.
String	*targetTextDir* Name of the directory containing text files of target instances.
String	*stopWordsFile* Name of the file containing the list of stop-words.
String	*boostrapFile* Name of the file containing bootstrapping instances.
Integer	*NUMBER_OF_INSTANCES* Number of training instances to be retrieved for a topic.

Table-3: Method Summary for YouTubeClassifier3

Modifier and Type	**Method and Description**
Void	*initializeEnv()* Reads configuration parameters from JSON file and sets appropriate fields. Also creates directories for storing intermediate text files.
Void	*getTrainingData()* Queries SOLR for ACM metadata and creates text files for training instances.
Void	*loadTrainingDataIntoArff (***String** dirName) Loads the text files in directory *dirName,* into an ARFF file.
Void	*identifyClasses()* Identifies classes for classification (the list of topics in *topicFile* file).
Void	*getLabeledTargetData()* Collects labeled target data and creates text files for target instances.

Void	*getTargetData()* Collects target data and creates text files for target instances. Communicates with YouTube API to get metadata.
Void	*loadTargetDataIntoArff (***String** targetDataDir, **boolean** isLabeled) Loads the text files in directory *targetDataDir*, into an ARFF file for testing. Setting *isLabeled* to *True* will preserve the labels on the instances.
Void	*buildAndEvaluateClassifier* (**int** topicId) Prepares the classifier (builds and trains the classifier). Also classifies target instances and evaluates the classification.
Void	*loadBootstrappingInstances()* Loads bootstrapping instances.
Void	*createInstancesForTraining* (**String** topic, **int** topicId) Creates the instances for training from the ARFF file for a given topic ***topic***
Void	*printPredictions()* Prints the predictions (collated) of the classifiers. Basically a list of video ID followed by the classes the video has been classified into.

Class **Channels**

Table-4: Field Summary for Channels

Modifier and Type	**Field and Description**
String	*videoTopicFile* Name of the file with the list of topics to be used as search terms for retrieving videos.

Table-5: Method Summary for Channels

Modifier and Type	**Method and Description**
List<Map<String, String>>	*getPlaylists(**String** queryTerm, **long** NUMBER_OF_PLAYLISTS)* Uses YouTube Data API to search for playlists with *queryTerm* as search term and returns a list of videos with their metadata. Fetches playlists from at most *NUMBER_OF_PLAYLISTS* pages in the result set.
List<Map<String, String>>	*getVideoMetadata()* Collates the videos for all topics in *videoTopicFile* and returns as list of videos along with their metadata.

String[]	*getVideMetadataGivenId(***String** videoId) Returns the metadata (title and description) of the video with Id *videoId*.

Class **YTChannel**

Table-6: Field Summary for YTChannel

Modifier and Type	**Field and Description**
Boolean	*onlyVideoWithCaption* When set to *True,* fetches only videos which have transcripts.

Table 7: Method Summary for YTChannel

Modifier and Type	**Method and Description**
ArrayList<String>	*getVideos(**String** playlistId)* Uses YouTube Data API to retrieve all videos associated with playlist whose Id is *playlistId*. Returns a list of Ids of the retrieved videos.
ArrayList<String>	*readJsonStream(**InputStream** in)* A helper function which parses a JSON structure and extracts video IDs.
String	*captionData(***String** videoId) Returns the transcripts associated with the video with Id as *videoId*.

Sample "properties.json" file

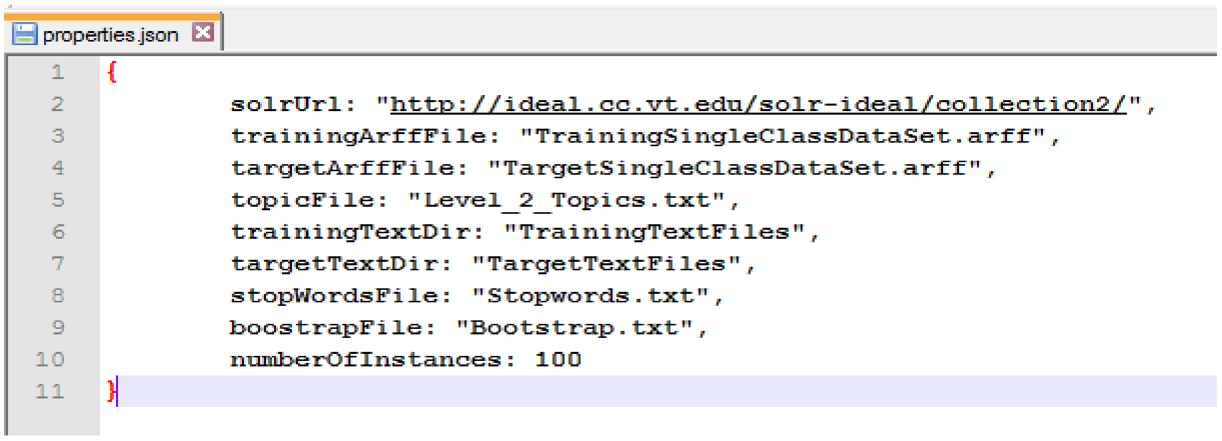
```
{
	solrUrl: "http://ideal.cc.vt.edu/solr-ideal/collection2/",
	trainingArffFile: "TrainingSingleClassDataSet.arff",
	targetArffFile: "TargetSingleClassDataSet.arff",
	topicFile: "Level_2_Topics.txt",
	trainingTextDir: "TrainingTextFiles",
	targetTextDir: "TargetTextFiles",
	stopWordsFile: "Stopwords.txt",
	boostrapFile: "Bootstrap.txt",
	numberOfInstances: 100
}
```

Figure-4: Sample JSON file

3. Tools Used

- *Weka*: Weka[8] is a collection of machine learning algorithms developed at the University of Waikato, New Zealand. Weka provides tools that can aid in pre-processing, classification, regression, clustering, association rules, and visualization. We used the various classification algorithms built into Weka to train and build our classifiers.
- *YouTube Data API*: The YouTube Data API[4] lets you incorporate YouTube functionality into your own application. You can use the API to fetch search results and to retrieve, insert, update, and delete resources like videos or playlists [4]. We used the search API extensively to retrieve our target data.
- *Apache SOLR*: SOLR[9] is a stand-alone enterprise search server with a REST-like API. You put documents in it (called "indexing") via XML, JSON, CSV or binary over HTTP. You query it via HTTP GET and receive XML, JSON, CSV or binary results [9]. The metadata from ACM is indexed in a SOLR server to which we issued REST calls to retrieve the metadata (title, abstract, keywords) that we used for training.
- *Eclipse IDE*: We used Eclipse, an open source IDE as our main development environment.

V. ENSEMBLE CLASSIFICATION

1. Training Data Preparation

Following are the techniques used to prepare and tune the training data before using it as an input to various classifier algorithms in Weka:

- **Feature Extraction**
  Feature extraction is the process of extracting relevant features from the training data. We use “title”, “abstract” and “general terms” attributes of the ACM Metadata to obtain suitable features for training classifiers. We also included the category name from the ACM taxonomy tree as a feature. The values for these attributes are obtained by querying the SOLR server using concept description as the query parameter. Level 2 topic names from the ACM taxonomy tree are used to query and retrieve the metadata. Separate text files are generated for each metadata record obtained from SOLR. Each text file contains title, abstract and general terms attributes of the corresponding record. These attributes are written to the text files after removing all non-alpha numeric characters.

- **Formatting**
  Formatting is done using TextDirectoryLoader of Weka. TextDirectoryLoader is used to convert the text files into an ARFF file. ARFF file contains two attributes: a string that is a concatenation of title, abstract and general terms and a class attribute denoting the category of ACM CCS that the corresponding string belongs to.

- **Filtering**
  String to Word Vector Filter is applied to the ARFF file, with following filter attributes:
  - setWordsToKeep = 1000
  - setDoNotOperateOnPerClassBasis =true;
  - filter.setMinTermFreq = 2;
  - filter.setTokenizer =AlphabeticTokenizer
  - setStemmer =IteratedLovinsStemmer;
  - useStoplistTipText = true

  In the above filter, we use a customized stop-word list.

- **Balancing training data set**
  We balanced the training data for a particular classifier by using equal number of positive and negative training data samples. For a given class C, positive samples are all those documents that belong to class C and negative samples are those that do not belong to class C. We choose 100 positive samples and 100 negative samples while training for a particular class. The 100 negative samples are uniformly distributed over all classes other than the one for which the given classifier is being trained.

2. ACM Computing Classification System

ACM Computing Classification System (ACM CCS) is a conceptual four level classification of the Computer Science subject area. Categories in the ACM CCS reflect various concepts of the computing discipline. Figure-5 shows the 13 categories that form the second level of the ACM CCS taxonomy. Each of the second level topics has subtopics, which constitute the third level of the tree. There are 84 topics in the third level. Henceforth, categories at the second and third level of the ACM CCS will be referred as L2 topics and L3 topics respectively, in this report.

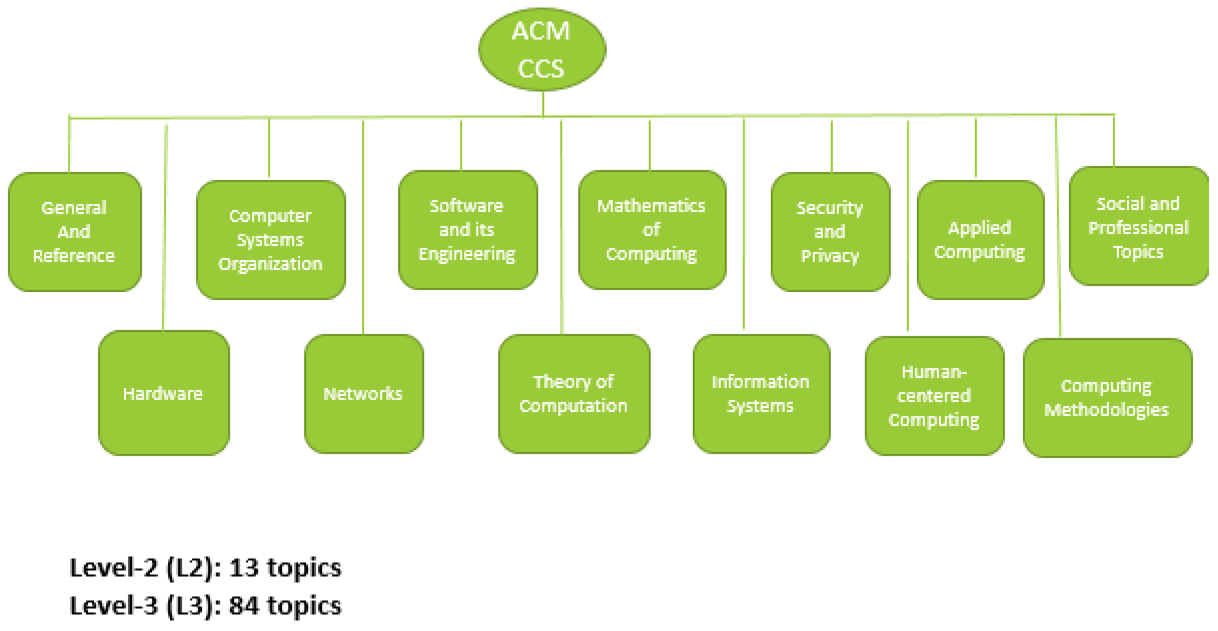

Figure-5: ACM CCS with second level categories

3. Classification Strategies

Single-class classification

In single-class classification, each training instance either belongs to or does not belong to the class for which the classifier is being built. The classifier, hence, predicts whether a test instance belongs to a given class or not. To classify documents into N classes, single-class classification requires N different binary classifiers. For the classifier on class X, positive examples are all instances that belong to class X and negative examples are all points that do not belong to class X.

Multi-class classification

In multi-class classification, each training instance belongs to one of the N different classes. The classifier predicts the classes to which a test instance belongs. To classify documents into N classes, multi-class classification requires only 1 classifier.

We compared performance and accuracy of single-class classification and multi-class classification to classify documents into L3 topics. Single-class classification yielded better results and hence we preferred using single-class classification to build our classifiers.

4. Pruning the ACM taxonomy

We have built classifiers for classifying documents into

- L2 topics
- L3 topics

Through experiments (elaborated in the next section of the report) carried out on the target data using the above two types of classifiers, we have identified that classification of documents into a subset of L2 topics yields most accurate results. The selection of a subset of L2 topics was done after careful examination of the ACM taxonomy tree. "General and Reference", "Applied Computing" and "Social and Professional Topics" topics being too generic, we did not find relevant target data to classify into them. Hence these topics are not being considered for our classification purpose. Also we have merged "Hardware" topic with "Computer Systems and Organization" since we found them to be related to each other. The pruning process is summarized in Figure-6 and the pruned ACM CCS is shown in Figure-7.

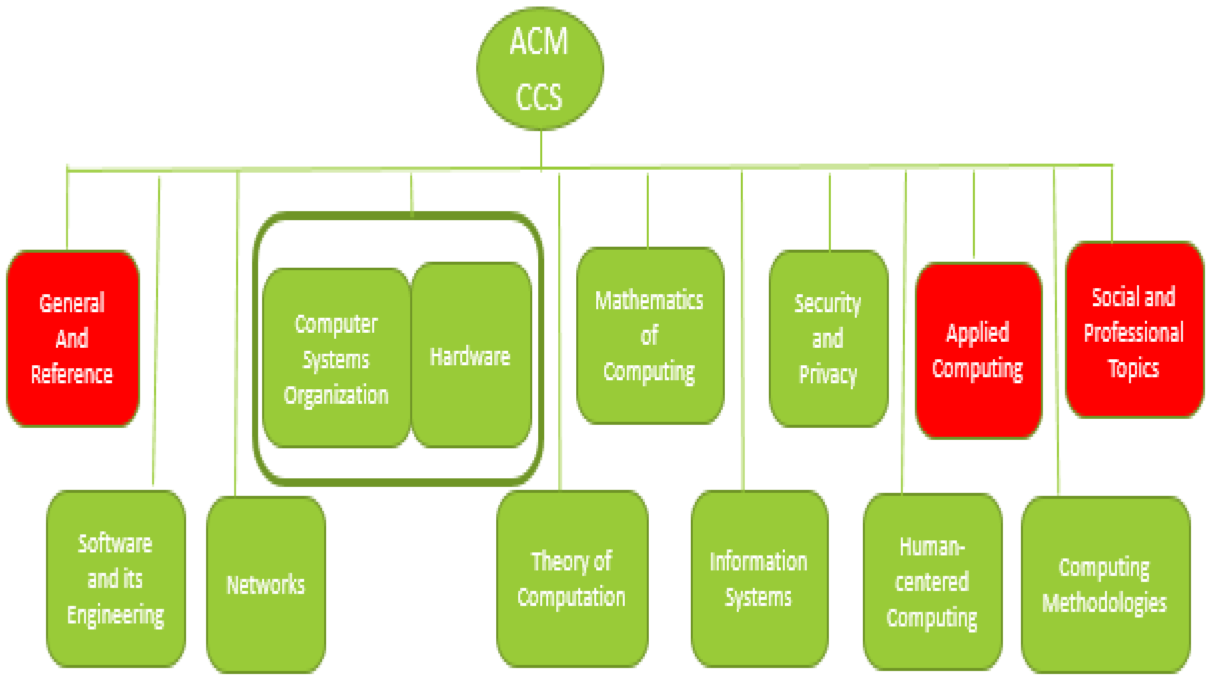

Figure-6: Pruning the ACM CCS

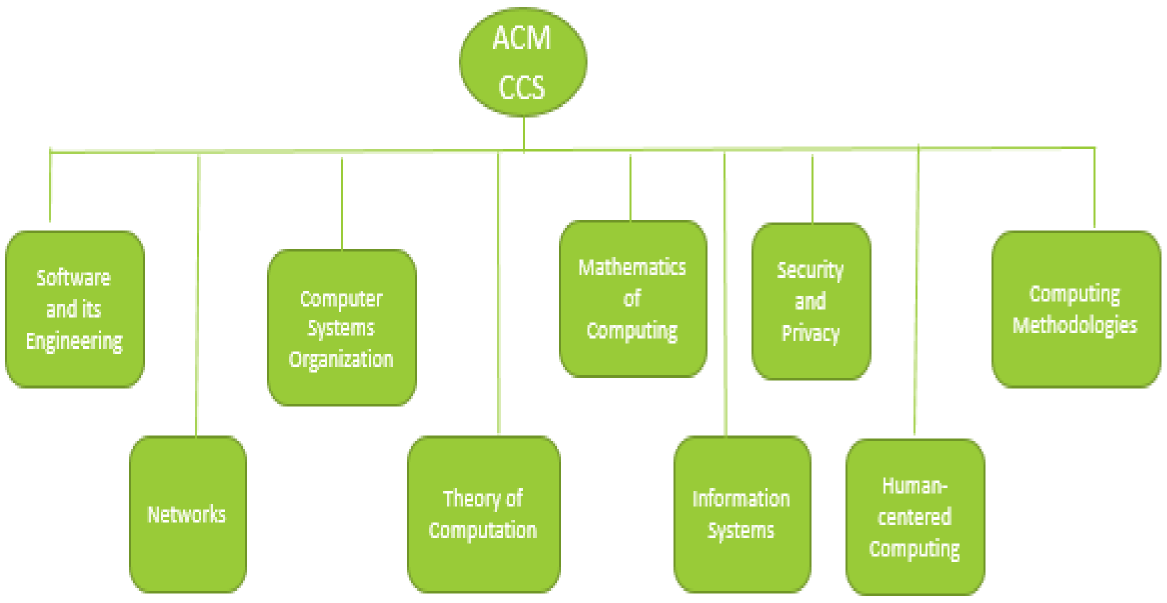

Figure-7: Pruned ACM CCS

5. Approaches to target data collection

In this section we describe the approaches we considered for collecting target data. The target data for this project is a collection of videos gathered from YouTube. We adapted both manual and automatic methods depending on how we query YouTube, how we use the collected data and the accuracy of the collections in terms of how relevant they are with respect to the search queries we used. We experimented with several different approaches and analyzed the accuracy of the collections.

We list here few terms we use in this section and their definition before describing the approaches in detail.

- **YouTube API:** Version-3 of YouTube Data API.
- ***Search Query:*** The set of words we use to search YouTube for collecting videos. This forms the query terms for the search. We use words that belong to two categories as part of the search terms.
    - ACM taxonomy: As described in earlier sections, ACM's taxonomy has a hierarchy of topics. For our search, we focused on the topics at level-2 and level-3 of the taxonomy tree.

    - Computing domains: A list of computing topics spanning across various domains in computer science. These topics do not necessarily belong to the ACM taxonomy.

- ***Labeled/Unlabeled target data:*** Labeled target data denote that the collected videos have an associated topic name that is used by the classifiers for evaluating their predictions. The topic name is typically one of the nodes in ACM taxonomy (either level-2 or level-3). On the other hand, unlabeled target data do not carry any labels and are directly fed to the classifiers.
- ***Channels and playlists:*** Same definition as channels and playlists in YouTube.
- ***Bootstrapping:*** A strategy to expose classifiers to the vocabulary of the target domain. We add a user-defined set of target instances to the training set.
- ***Final test set:*** The collection of videos we use as the final target data. These form the basis for our evaluations and analyses.

We now outline the strategies and the usage of collected data.

- **Manual extraction:**
  In this approach, we manually search YouTube to collect videos and associated metadata. We use topics from the computing domain as search terms. We also manually analyze the features of each video and label them with one of the topics from ACM taxonomy. Since done manually, this approach has high reliability but is limited in the number of videos. We use this collection for *bootstrapping* purposes. This is depicted by the flow labeled “1” in Figure-8.

- **YouTube API (automatic extraction):**
  This approach uses *YouTube Data API* for searching and collecting videos. We considered several approaches using the API. Each flow in Figure-8 corresponds to a transfer learning approach which is shown in Figure-9.

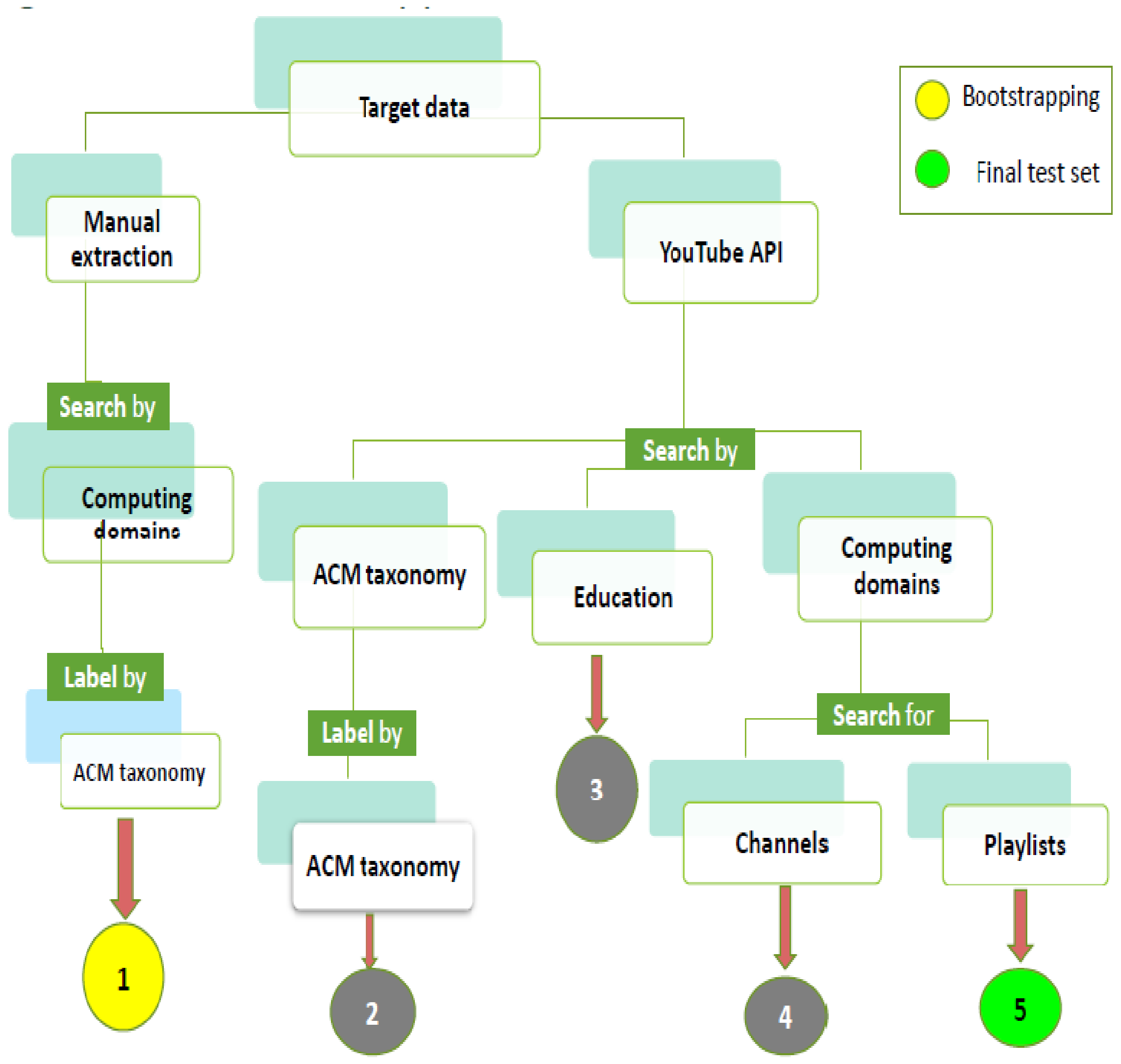

Figure-8: Target data collection approaches

- We use topics from level-3 of the ACM taxonomy as search terms. We also label each video with the name of the topic for which it was part of the search results. Flow labeled "2" in the Figure-8corresponds to this approach. This collection is also labeled. We use this collection for classifying videos into topics in level-3 in ACM taxonomy. The classifiers are trained on data at the same level.
- Flow labeled "3" shows our next approach. This is our first experiment with *unlabeled* collections. This approach uses a client-provided collection of videos. The objective behind this approach is to collect videos belonging to a broader domain such as "education". We feed this collection to classifiers which are trained on level-3 of the ACM taxonomy. The classifier predictions also belong to level-3.

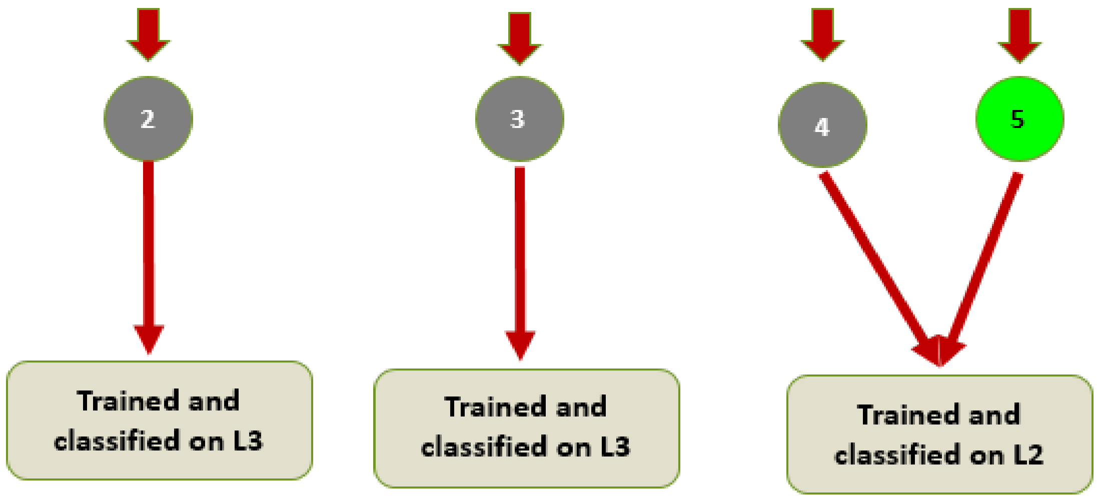
2
3
4
5
Trained and classified on L3
Trained and classified on L3
Trained and classified on L2

Figure-9: Transfer learning approaches

- The previous two approaches suffered issues with reliability (relevancy of videos to topics). This led us to our next approach (flow labeled "4" in Figure-8). We search for *channels* in YouTube with topics from the computing domains as search queries. This collection is also unlabeled. The result sets included a reasonable number of relevant videos. We use this collection on classifiers trained on level-2 in ACM taxonomy and classify videos into the same level.
- From our experience we identified that videos within a playlist are more likely to be related than videos within a channel. So, we search for *playlists* in YouTube with topics from the computing domains as search queries. This approach improved the reliability of the search and provided a collection with a very high number of relevant videos. We collected a set of unlabeled videos, which we later fed to our classifiers trained on level-2 in ACM taxonomy and classify them into the same level. Our evaluations of classifiers are based on this collection of videos.

6. Bootstrapping

We used bootstrapping as a methodology for exposing the classifiers to the vocabulary of target domains, YouTube videos in our case. Given a training set T_{train}, we add to it a set of instances $T_{bootstrap}$ which is a collection of labeled target instances. For each topic in level-2 in ACM taxonomy, we collect a set of videos using the approaches described earlier. For a single-class classifier on topic *X*, we use a label '1' for videos that belong to *X* and a label '0' for those that do not belong to *X*. We include these labeled instances as part of the training set for classifier on *X*. All these instances have associated metadata (title and description). We followed two different methodologies for selecting bootstrapping instances. Figure-10 shows the two approaches we considered.

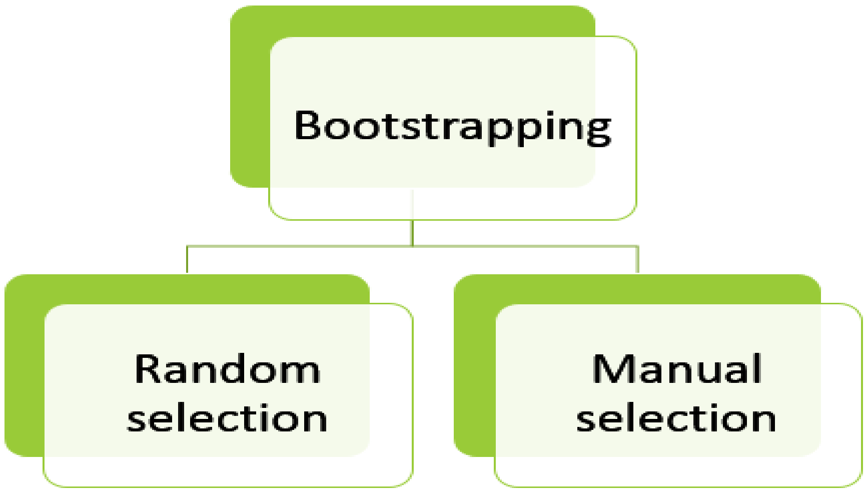

Figure-10: Bootstrapping approaches

- **Random selection:** From a collection of videos searched and labeled by topics in ACM taxonomy, we randomly select a fixed number (user-defined) of videos and include them in the training set of a classifier. This selection method is non-deterministic in that it does not guarantee that only relevant videos be included in $T_{bootstrap}$. This non-determinism can affect the performance of the classifiers.
- **Manual selection:** We adopted a manual approach for bootstrapping to eliminate non-determinism and ensure that only videos with reasonable features and high relevancy to the topic be added to $T_{bootstrap}$. For this, we used the collection of videos obtained by the manual extraction described in an earlier section. As mentioned earlier, this approach is limited in the size of the collection but is more reliable and serves well the purpose of bootstrapping.

7. Evaluation and Analysis

Evaluation and analysis of the classifier are presented below in the following two sections.

Training:

Three machine learning algorithms were evaluated to judge the accuracy of the classifier and Figure-11 depicts the comparison of those algorithms in terms of classifier accuracy. The test was run on 100 instances each for positive and negative data for every class that was considered for evaluation with 10 fold cross-validation.

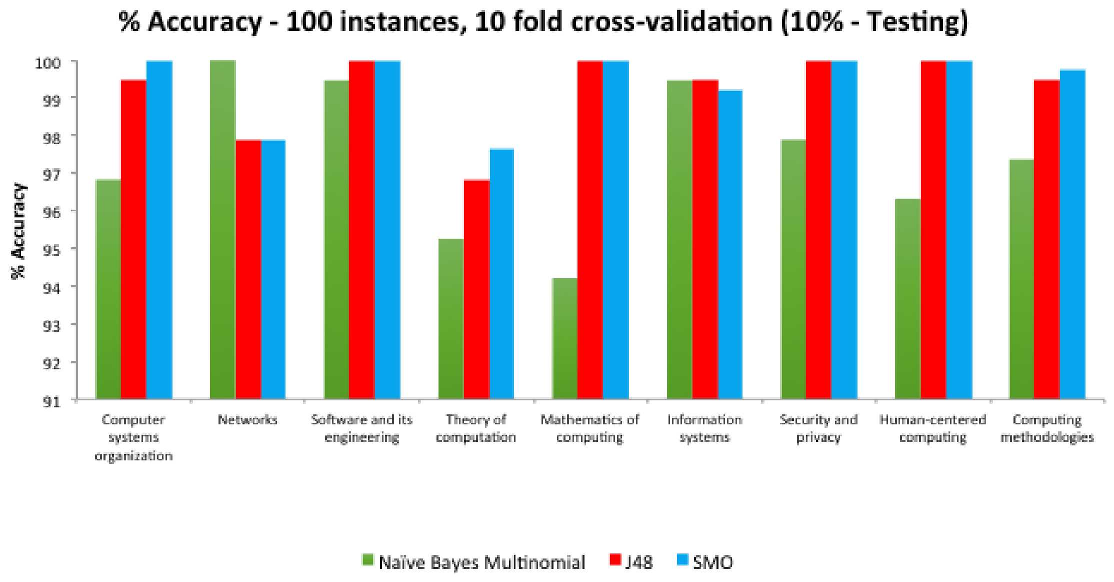

Figure-11: Comparison of accuracy of Naïve Bayes Multinomial, J48, SMO

The reason for the irregularity in the accuracy of algorithms (Naïve Bayes Multinomial performing better for the classes Networks and Information Systems but not for the others) for various classes hasn't been analyzed and further investigation is required. We preferred using Naïve Bayes Multinomial for our target evaluation for the following reasons.

- Other algorithms seem to have considerably higher accuracy and might have high variance (over-fitting).
- Naïve Bayes Multinomial took less time when running tests on larger data sets.

Figure-12 shows the comparison of F-Measure for the same set of algorithms.

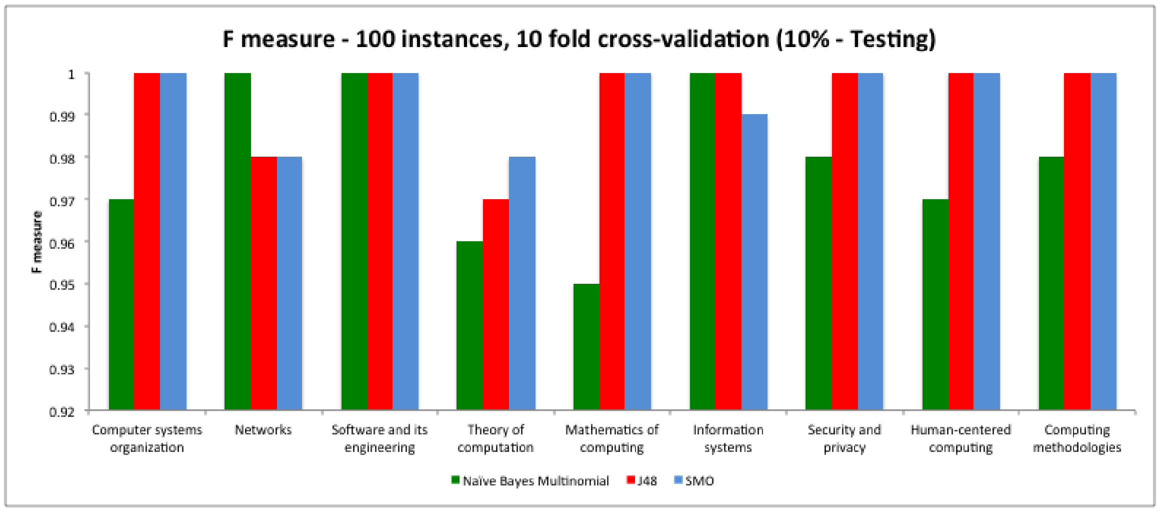

Figure-12: Comparison of F-measure for Naïve Bayes Multinomial, J48, SMO

Figure-13 depicts the comparison of Naïve Bayes Multinomial algorithm for 100 and 500 instances.

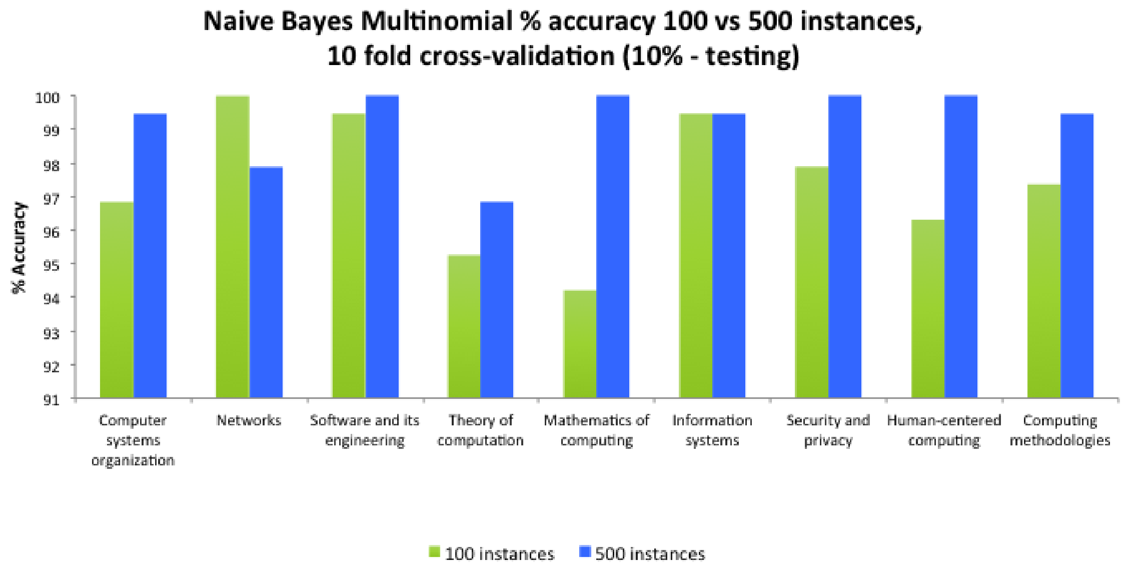

Figure-13: Comparison of Naïve Bayes Multinomial algorithm for 100 and 500 instances

Figure-14 and Figure-15 present the comparison for J48 and SMO with 100 and 500 instances.

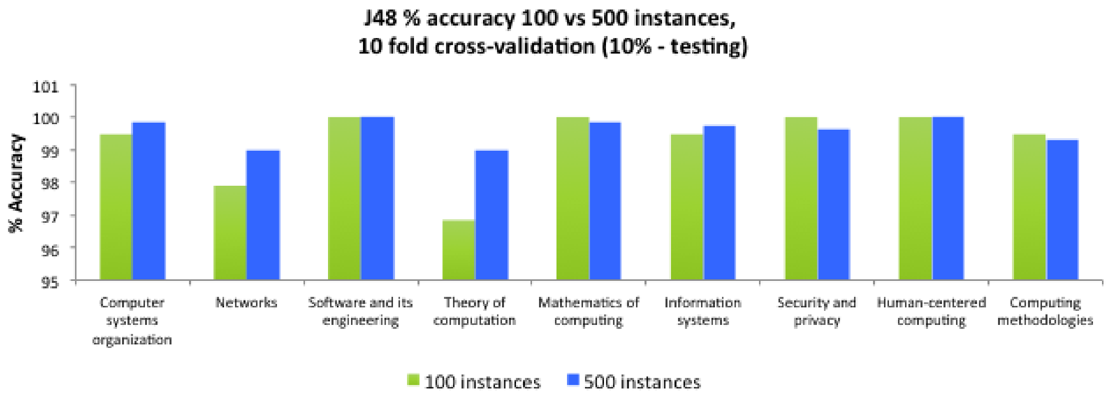

Figure-14: Comparison of J48 algorithm for 100 vs. 500 instances

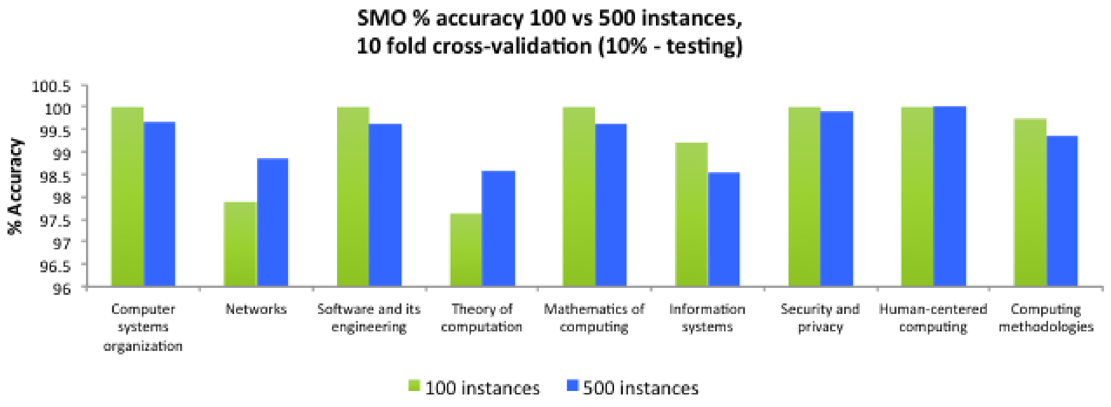

Figure-15: Comparison of SMO algorithm for 100 vs. 500 instances

Target:

After running our classifiers on the target data (YouTube metadata), we verified the results manually and Figures 16-18represent the accuracy of the decision made by the classifiers. We included only videos that were classified into less than 4 classes by our classifiers.

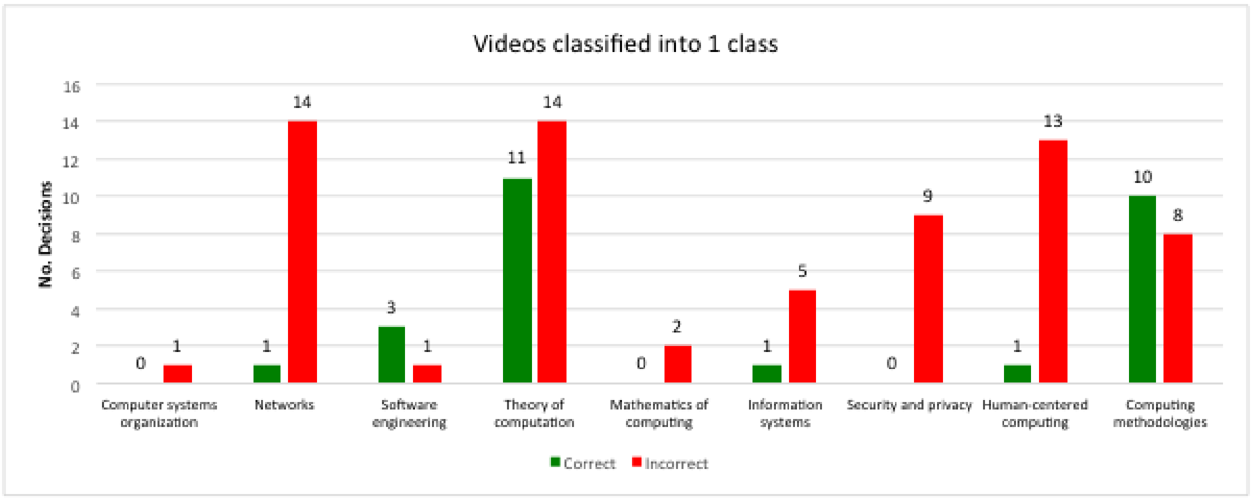

Figure-16: Correct vs. Incorrect decisions – Videos classified into 1 class

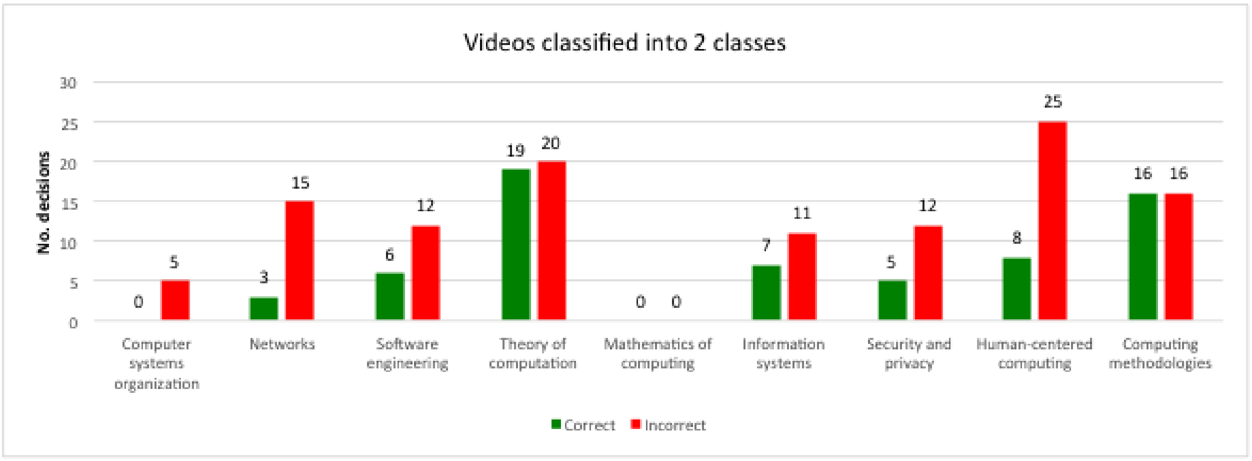

Figure-17: Correct vs. Incorrect decisions – Videos classified into 2 classes

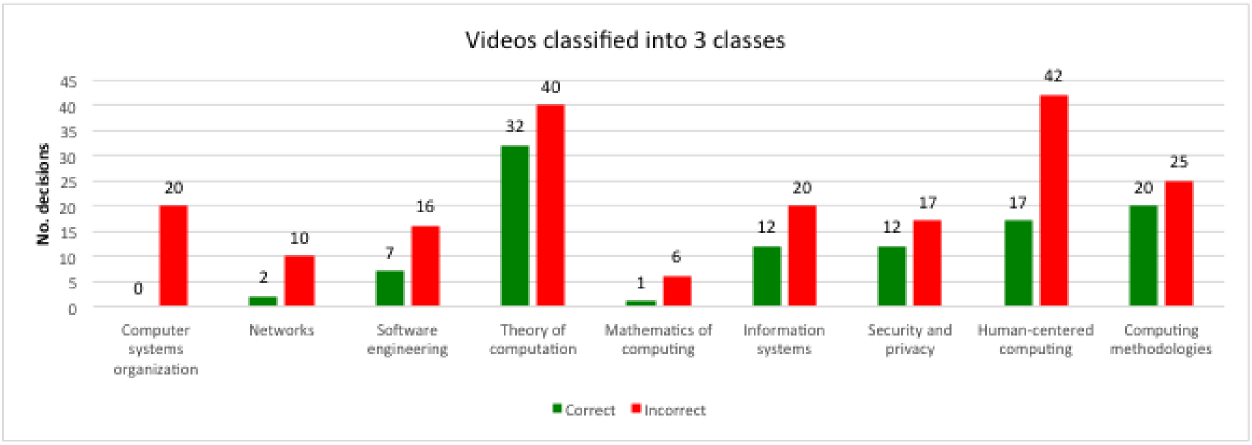

Figure-18: Correct vs. Incorrect decisions – Videos classified into 3 classes

Observations:

- Overlapping of features
  - We observed that the number of videos getting incorrectly classified into the 'Human centered computing' class is always high because it is an applied domain and the features from all other classes can very well appear in this class. Further filtering is required specifically for this class in order to get a better performance.
  - The sub-categories under the level-2 categories mentioned earlier overlap in many cases leading to common features. For example,
    - Parallel architecture is included under 'Computer systems organization' and parallel computing is included under 'Computing methodologies'.
    - Various kinds of algorithms are included under 'Theory of computation', 'Security and privacy' and 'Computing methodologies'.
- Availability of target data
  - The target data that we provided to our classifiers had very less number of videos for certain classes such as 'Mathematics of computing'. More appropriate search queries will be required to get more relevant videos in such cases.
- Commonality of features
  - The word 'node' is used to represent both a network node as well as a node in a linked list or a tree. Because of this, we found many videos that are related to data structures being classified into 'Networks'. Again further filtering of features is required in such cases.

Table-8: Percentage of correct decisions for videos classified into 1, 2 and 3 classes

Number of classes	**% Correct decisions**
1	31
2	35
3	35

We believe that the unavailability of proper metadata for all YouTube videos and ambiguities in the ACM classification system are the major reasons for such a reduced accuracy of predictions. We will elaborate this in section VI.

VI. LESSONS LEARNED

1. Timeline/schedule

Table-9 illustrates the main tasks and timeline of the project.

Table-9: Project Timeline

Task	Completion Date
Identifying the project objectives and requirements	02/10/2014
Conducting a literature review	02/14/2014
Preparing the training data set	02/21/2014
Selecting and extracting features	02/21/2014
Building and training the Multi-class classifiers	02/28/2014
Evaluating the accuracy for Multi-class classifiers	02/31/2014
Presenting midterm project status	03/06/2014
Further Tuning to the target data	03/12/2014
Building and Training Single-class classifiers for the whole ACM tree	03/19/2014
Identifying and collecting the target data set	04/09/2014
Transfer learning (Classifying the target data using different approaches)	04/09/2014
Applying Bootstrapping technique	04/14/2014
Evaluating the Single-class classifiers based on results from the target data	04/16/2014
Filtering and Tuning to get better classifier's accuracy	04/21/2014
Presenting the final project outcomes	05/01/2014
Final review, modifications and failure analysis	05/05/2014
Submitting the final project report along with all project associated files	05/08/2014

2. Challenges and Learning

Following is a brief summary of the challenges we faced and actions we took to solve them.

⇨ **Target Data Collection**

a. **Availability and quality of target data**

   Section V illustrates different approaches that we have used to collect the target data from YouTube. Availability and quality of this target data posed a problem in data collection. Relevant computing-related videos are not available for all ACM CCS topics. Searching for videos related to L2 topics such as "General and Reference" and "Social and Professional Topics", yielded dirty data, which affected the overall accuracy of the classification process. Also all videos do not have complete and accurate title and description associated with it. This causes target data to not have sufficient metadata associated with it.

   Solution:

   We have pruned the ACM CCS taxonomy tree to exclude topics for which relevant videos were not available in YouTube.

b. **Reliability of Search**

   We found YouTube general search to be very unreliable. Using a search query like "computer science lecture" to retrieve video lectures on different topics of computer science, yields a ranked list of videos which contain either of the terms "computer", "science" or "lecture" in their title. Such noisy target data affects the accuracy of the classification process.

   Solution:

   We are searching for relevant videos in YouTube playlists using queries based on common terms in the computing domain. This has resulted in better quality of collected target data.

⇨ **Feature extraction and selection for Target Data (YouTube videos)**

We are using the title and description attributes of YouTube videos to extract features from them. However, these attributes are not comprehensive and do not contain sufficient information for extraction of suitable features from them.

Solution:
We found some YouTube videos do have transcripts associated with them. These contain the text of what is said in the YouTube video. We use transcripts, whenever it is available.

- **Mismatch in training (ACM) data and target (YouTube) data vocabulary**
  The ACM Digital library consists of research papers in the computing discipline. The YouTube Digital Library consists of videos on various topics. These libraries manage objects belonging to different domains, and their vocabularies are different. This leads to a mismatch between features defining the training set and features defining the target set. Such a mismatch caused the classifier, trained on ACM metadata, to imprecisely classify videos.

  Solution:
  The above problem was overcome using bootstrapping, as explained in section-5.

- **Generic nature of ACM CCS**
  Certain categories of the ACM CCS are too generic because of overlapping features and are thus ambiguous in nature. For example, both the categories “Theory of Computation” and “Computing methodologies” have documents with similar features (for example the word “algorithm”). Such non-disjoint features cause confusion and incorrect classifications.

  Solution:
  We have solved this problem to some extent by merging related categories. We have merged the topic “Hardware” with the topic “Computer Systems Organization”, since their subtopics are very similar.

- **Weka Performance and Reliability**
  We found that the time taken to train various classifiers using the GUI of Weka was more than when the same was done using Weka libraries through source code written in JAVA. Weka GUI is also unreliable in the sense that it crashes while training some specific classifiers such as J48, using large training data sets.

  Solution:
  We use different Weka libraries through source code written in Java. This has resulted in reduced running time while training different classifiers.

VII. FUTURE WORK

Our work can be improved and extended in many possible ways and some of them are listed below.

a. Improve the correctness percentage by removing ambiguous classes from the ACM taxonomy that is used for training.
b. Use only videos that have a proper metadata.
c. Use advanced NLP techniques to enhance training and target features.
d. Can adopt a voice-to-text transformation technique to develop transcripts for the YouTube videos and include these transcripts in the feature set.
e. Use image retrieval techniques
    - To retrieve text embedded in the videos like subtitles, presentation slides content shown in the video
    - To distinguish a lecture video from other videos by making use of a sophisticated Content-Based Image Retrieval(CBIR) techniques. For example, CBIR can be used to identify whether a person stands in front of a black or green board in the image.
f. Extend the transfer learning to other possible target domains like *slideshare* (http://www.slideshare.net/), blogs etc.

VIII. ACKNOWLEDGEMENTS

We would like to thank Dr. Edward A. Fox and Mr. Yinlin Chen for their continued support and valuable suggestions throughout the project.

IX. REFERENCES

[1] Sinno Jialin Pan and QiangYang. 2010. A Survey on Transfer Learning. IEEE Transactions on Knowledge and Data Engineering, 22 (October 2010), 1345-1359.
[2] Ensemble: Enriching Communities and Collections to Support Education in Computing.
[3] ACM Digital Library. 2012. Computing Classification System, 2012 Revision. (March 2012). Retrieved February 1, 2014 from http://www.acm.org/about/class/class/2012.

[4] Google, 2013.YouTube Data API (v3). (August 2013). https://developers.google.com/youtube/v3/

[5] Remco R. Bouckaert, Eibe Frank, Mark Hall, Richard Kirkby, Peter Reutemann, Alex Seewald and David Scuse. 2013. WEKA Manual for Version 3-7-10. (July 2013). Retrieved February 1, 2014 from http://www.cs.waikato.ac.nz/ml/weka/documentation.html

[6] Brandon Weinberg. 2013. WEKA Text Classification for First Time & Beginner Users. (28 May 2013). Referred from https://www.youtube.com/watch?v=IY29uC4uem8

[7] Nihil Obstat. 2013. A Simple Text Classifier in Java with WEKA. (April 2013). Retrieved February 1, 2014 from http://jmgomezhidalgo.blogspot.com/2013/04/a-simple-text-classifier-in-java-with.html

[8] Weka. 2014. Weka: Frequently asked questions. Referred from http://weka.wikispaces.com/Frequently+Asked+Questions

[9] SOLR. 2011. Apache Solr. http://lucene.apache.org/solr/

[10] Mkyong. 2011. JSON.Simple Example – Read And Write JSON. (August 2011). Referred from http://www.mkyong.com/java/json-simple-example-read-and-write-json/