RICE UNIVERSITY

Linkify: A Web-Based Collaborative Content Tagging System for Machine Learning Algorithms

by

Dante Soares

A THESIS SUBMITTED IN PARTIAL FULFILLMENT OF THE REQUIREMENTS FOR THE DEGREE

Master of Science in Electrical Engineering

APPROVED, THESIS COMMITTEE

Dr. Richard Baraniuk, Chair
Victor E. Cameron Professor of Electrical Engineering

Dr. Joseph Cavallaro
Professor, Department of Electrical Engineering

Dr. Sidney Burrus
Maxfield-Oshman Professor Emeritus of Electrical and Computer Engineering

HOUSTON, TEXAS
October 2014
ABSTRACT

**Linkify: A Web-Based Collaborative Content Tagging System for Machine Learning Algorithms**

by

**Dante Soares**

Automated tutoring systems that use machine learning algorithms are a relatively new development which promises to revolutionize education by providing students on a large scale with an experience that closely resembles one-on-one tutoring. Machine learning algorithms are essential for these systems, as they are able to perform, with fairly good results, certain data processing tasks that have usually been considered difficult for artificial intelligence. However, the high performance of several machine learning algorithms relies on the existence of information about what is being processed in the form of tags, which have to be manually added to the content. Therefore, there is a strong need today for tagged educational resources. Unfortunately, tagging can be a very time-consuming task. Proven strategies for the mass tagging of content already exist: collaborative tagging systems, such as Delicious, StumbleUpon and CiteULike, have been growing in popularity in recent years. These websites allow users to tag content and browse previously tagged content that is relevant to the user’s interests.

However, attempting to apply this particular strategy towards educational resource tagging presents several problems. Tags for educational resources to be
used in tutoring systems need to be highly accurate, as mistakes in recommending or assigning material to students can be very detrimental to their learning, so ideally subject-matter experts would perform the resource tagging. The issue with hiring experts is that they can sometimes be not only scarce but also expensive, therefore limiting the number of resources that could potentially be tagged. Even if non-experts are used, another issue arises from the fact that a large user base would be required to tag large amounts of resources, and acquiring large numbers of users can be a challenge in itself.

To solve these problems, we present Linkify, a system that allows the more accurate tagging of large amounts of educational resources by combining the efforts of users with certain existing machine learning algorithms that are also capable of tagging resources. This thesis will discuss Linkify in detail, presenting its database structure and components, and discussing the design choices made during its development. We will also discuss a novel model for tagging errors based on a binary asymmetric channel. From this model, we derive an EM algorithm which can be used to combine tags entered into the Linkify system by multiple users and machine learning algorithms, producing the most likely set of relevant tags for each given educational resource. Our goal is to enable automated tutoring systems to use this tagging information in the future in order to improve their capability of assessing student knowledge and predicting student performance. At the same time, Linkify’s standardized structure for data input and output will facilitate the development and testing of new machine learning algorithms.
I would like to thank, first of all, the members of my thesis committee, Dr. Richard Baraniuk, Dr. Sidney Burrus and Dr. Joseph Cavallaro, for the time devoted to reading the thesis, hearing my defense and debating the merit of this thesis. Dr. Baraniuk, in particular, gets special thanks for being my advisor throughout my graduate career. Dr. Andrew Waters has earned my most sincere thanks for helping brainstorm ideas about the algorithm presented in this thesis and for reviewing several different versions of the thesis draft. J. P. Slavinsky also deserves recognition for being the one to first imagine the foundations of the Linkify system and its role in the OpenStax family of websites, and for helping me review the thesis outline. Last but not least, I would like to thank my parents for their continued support during all of these years.
Contents

Acknowledgments ........................................................................................................ iv

Contents ......................................................................................................................... v

List of Figures .............................................................................................................. vii

List of Tables ............................................................................................................... viii

List of Equations ......................................................................................................... 1

Nomenclature ............................................................................................................... 2

Introduction .................................................................................................................. 3

1.1. One-on-One Tutoring .......................................................................................... 3

1.2. Automated Tutors .............................................................................................. 4

1.2.1. Effectiveness ................................................................................................. 4

1.2.2. OpenStax Tutor ............................................................................................ 5

1.3. Machine Learning Algorithms .......................................................................... 6

1.4. Tagging Educational Resources ....................................................................... 6

1.5. Linkify ................................................................................................................ 8

1.6. Ensemble Methods ............................................................................................. 9

Literature Review ....................................................................................................... 10

2.1. Cognitive Tutors ............................................................................................... 10

2.2. Community Tagging Systems .......................................................................... 11

2.3. Ensemble Learning ........................................................................................... 13

2.3.1. Ensemble Learning Classifier Generation ................................................ 13

2.3.2. Combining Discrete Outputs ...................................................................... 15

2.3.3. Applications of Ensemble Learning ........................................................... 16

Research Question ................................................................................................... 19

The Linkify System .................................................................................................... 20

4.1. Language and Framework .............................................................................. 21

4.2. Data Model ....................................................................................................... 21

4.3. API .................................................................................................................. 26

4.4. Linkify Summary ............................................................................................. 27
Combining Tagging Algorithms via Expectation Maximization .......................... 28
5.1. Tagging Model .................................................................................................................. 28
5.2. EM Combination Rule .......................................................................................................... 30
5.3. Algorithm Analysis ............................................................................................................... 31
  5.3.1. Space Complexity .......................................................................................................... 31
  5.3.2. Time Complexity ............................................................................................................. 32
  5.3.3. Rate of Convergence ...................................................................................................... 33
5.4. Limitations ............................................................................................................................ 36
5.5. Candidate Algorithms for Combination ............................................................................... 36

Testing with Synthetic Data ..................................................................................................... 37
  6.1. Combining 2 Classifiers ...................................................................................................... 38
  6.2. Combining 5 Classifiers ...................................................................................................... 40
  6.3. Comparison with Majority Voting Combination Rules ..................................................... 42

Testing with Real World Data .................................................................................................. 44
  7.1. Predicting a Single Tag ....................................................................................................... 45
  7.2. Predicting Multiple Tags ..................................................................................................... 46

Conclusion and Future Work ..................................................................................................... 48
  8.1. Conclusion ......................................................................................................................... 48
  8.2. Future Work ....................................................................................................................... 49

References .................................................................................................................................... 50
Appendix A ................................................................................................................................. 57
List of Figures

Figure 1 – A simplified view of Linkify’s class diagram. ................................................23

Figure 2 – Example Linkify representation of a hypothetical book with 2 chapters with 1 concept and 2 problems each. .................................................................24

Figure 3 – Asymmetric binary channel with parameters $q$ and $r$. ..........................30

Figure 4 – Number of iterations until convergence for the EM algorithm versus the average error rate of its 3 classifiers.................................................................34

Figure 5 – Number of iterations until convergence for the EM algorithm versus the average error rate of its 5 classifiers.................................................................35

Figure 6 – Ensemble error rate versus average error rate of its 3 classifiers..38

Figure 7 – Ensemble average error rate with variance versus maximum allowed classifier error rate, with 3 classifiers.................................................................39

Figure 8 – Ensemble error rate versus average error rate of its 5 classifiers..40

Figure 9 – Ensemble average error rate with variance versus maximum allowed classifier error rate, with 5 classifiers.................................................................41

Figure 10 – Performance comparison of the EM combination rule with the majority voting rule on synthetic data. .................................................................42
List of Tables

Table 1 - Precision and Recall of individual machine learning algorithms and EM ensemble when tagging the NUS-WIDE-LITE dataset........................................47
List of Equations

Equation 1 – Bernoulli distribution with parameter $p_k$. ..............................................29
## Nomenclature

<table>
<thead>
<tr>
<th>Abbreviation</th>
<th>Full Form</th>
</tr>
</thead>
<tbody>
<tr>
<td>AdaBoost</td>
<td>Adaptive Boosting</td>
</tr>
<tr>
<td>API</td>
<td>Application Programming Interface</td>
</tr>
<tr>
<td>CRUD</td>
<td>Create, Read, Update, Delete</td>
</tr>
<tr>
<td>EM</td>
<td>Expectation Maximization</td>
</tr>
<tr>
<td>JSON</td>
<td>JavaScript Object Notation</td>
</tr>
<tr>
<td>LDA</td>
<td>Latent Dirichlet Allocation</td>
</tr>
<tr>
<td>MCMC</td>
<td>Markov Chain Monte Carlo</td>
</tr>
<tr>
<td>MLE</td>
<td>Maximum Likelihood Estimate</td>
</tr>
<tr>
<td>OST</td>
<td>OpenStax Tutor</td>
</tr>
<tr>
<td>PDF</td>
<td>Probability Density Function</td>
</tr>
<tr>
<td>RDBMS</td>
<td>Relational DataBase Management System</td>
</tr>
<tr>
<td>SPARFA</td>
<td>SPARse Factor Analysis</td>
</tr>
</tbody>
</table>
Chapter 1

Introduction

For a great part of human history, classes have been traditionally taught in a classroom, by a single teacher, to a large group of students. Under this model, all students receive the same lecture, have to read the same material and are asked the same questions. Until recently, we had little reason to believe that this standard practice in education was not the optimal way to teach.

1.1. One-on-One Tutoring

However, in the 1980’s, studies started comparing this conventional method of instruction to one-on-one tutoring. Results from these studies were surprising: the average tutored student outperformed 98% of the students in conventional classes [1]. Unfortunately, one-on-one tutoring is simply too costly in terms of human resources to be implemented on a large scale. Therefore, since then, education researchers have been looking for ways to improve conventional
teaching, in order to bring the performance of conventionally-taught students closer to that of tutored students, without incurring the extremely high costs of one-on-one tutoring.

1.2. Automated Tutors

In recent years, certain technological advancements have shown great promise towards the goal of improving student learning. Automated tutors, for example, are computer programs that control student exposure to reading materials, problems and other educational resources. The driving force behind the development of those systems is the belief that they will one day provide educational benefits to students very similar to human one-on-one tutoring, but on a large scale.

1.2.1. Effectiveness

While not quite as effective as actual one-on-one tutoring with human tutors, automated tutors have been shown in several studies to improve student performance in exams when compared to conventional learning, particularly in science-related fields, such as mathematics [2], [3] and physics [4]. With these results, it is not surprising that automated tutors are becoming more and more widely used and research into new tutoring systems is growing.
1.2.2. OpenStax Tutor

Efforts to develop an automated tutoring system at Rice University began in 2011 with the development of the first version of our tutoring system, originally named PLS, for Personalized Learning System. Later, PLS would evolve into Rice’s current tutoring system, OpenStax Tutor (OST), developed by Rice University in partnership with Kindling Labs [5]. At present, OST has been successfully used in the classroom setting to validate widely-accepted theories from Cognitive Science, such as the learning benefits of retrieval practice, spaced repetition, timely feedback and requiring students to view grader feedback [6].

OpenStax Tutor relies on several other supporting websites in the OpenStax family to provide the different kinds of educational content necessary for learning. Quadbase [7] (soon to be known as OpenStax Exercises) is OpenStax’s bank of practice, homework and exam problems. OpenStax CNX [8] is a repository of remixable modules that serve as the basis for the free OpenStax College textbooks [9]. Finally, Linkify will eventually allow different users and algorithms to tag educational resources that are used by OST.

Work on the system is ongoing in order to use the lessons learned from the various studies done using OST to improve student learning, and new studies are conducted each semester on the effects that different classroom interventions have on student learning. Another future goal of the system that is currently being implemented is to use the recent advances in machine learning in order to provide a more individualized treatment to each student, on a case by case basis.
1.3. Machine Learning Algorithms

OpenStax Tutor will use machine learning algorithms in order to provide personalization that will suit each individual student’s needs. In recent years, machine learning algorithms have started to enjoy widespread success in solving several problems ordinarily considered hard for artificial intelligence, such as natural language processing, object recognition and others. Based on the success of these, our long-term goal is to implement and test different machine learning algorithms in our tutor system, in order to develop a flexible system capable to adapting to each student’s individual needs. If successful, this accomplishment would move us one step closer to the results obtained by one-on-one tutoring.

In order to produce good results, machine learning algorithms often rely on the existence of metadata about the content being processed. A common way to present this metadata is in the form of tags associated with each resource. Tags provide information about the content that would otherwise be difficult to extract with an algorithm, such as the topic a certain reading material explains, or the difficulty of a given practice problem.

1.4. Tagging Educational Resources

Since our goal is to develop a tutoring system based on machine learning algorithms, tagged educational resources are, therefore, essential for the system’s success. However, tagging large amounts of new educational resources being developed every day is not an easy task. Traditionally, this task has been done by
subject-matter experts. When compared with novices, experts have been shown to be able to produce more relevant sets of tags, using as ground truth the tags supplied by the content authors [10]. Unfortunately, experts can be very expensive and sometimes scarce, which caused us to explore different ways to perform the mass tagging of content.

One such approach consists of using social bookmarking websites, which are capable of tagging large quantities of resources. Some social bookmarking websites have enjoyed a certain degree of popularity in recent years, in particular Delicious [11], StumbleUpon [12] and CiteULike [13]. These sites operate under a very simple premise: first, the user is asked to bookmark and provide tags for some online resources that they find interesting. After doing this enough times, these systems are capable of suggesting new resources that match the user’s interests. While the main purpose of the user-provided tags is most likely to provide information for their internal machine learning algorithms, this tagging process could be leveraged to tag our educational resources.

Unfortunately, attempting to create a social bookmarking site that could be used to tag educational resources presents several problems. First of all, a considerable user base would be required to tag a large number of educational resources. Furthermore, while existing social bookmarking sites reward returning users by showing similar content aligned to their interests, it is not clear if the same kind of reward applied to educational resources would attract enough users. Finally,
the tags used in our tutoring system need to be highly accurate, in order to avoid mistakenly assigning the wrong materials to students.

Usually tags are handled by each content provider website, being stored alongside the content itself. Under this model, different websites can have different rules for tagging, such as allowing only content creators to tag their own content, or allowing anyone to suggest new content tags. This has a few advantages in terms of ease of use and reduced storage cost of the tags, but the lack of a universal agreed-upon list of tags means that those tags are less useful outside of their original websites. By separating the tagging system from the content sites we can not only enforce a common set of tags to all of our content websites, but also allow the tagging system to be designed and scaled up independently of the other systems, as long as they all follow a common Application Programming Interface (API).

1.5. Linkify

We developed our tagging system, Linkify, in order to meet all of our tagging requirements. This system allows not only users, but also machine learning algorithms, to tag content. It aims to utilize user input combined with the outputs of multiple machine learning algorithms in order to tag educational resources to be used with OpenStax Tutor, combining those results using ensemble methods. By using algorithms capable of tagging content alongside human users, we can obtain a large number of tags, even if those tags are not as relevant as tags created by subject-matter experts. These weak tags can be further processed by other
algorithms and recorded back into Linkify, in order to produce new sets of tags that more closely resemble the set that would be generated by experts. These more accurate tags can then be submitted for expert review, if necessary, and then used in our tutoring system. The data stored in Linkify can also potentially aid in the development and testing of novel machine learning algorithms. The first part of this thesis will describe the system’s architecture in detail and discuss the decisions made during its design.

1.6. Ensemble Methods

In order to combine tags generated by multiple different actors, be they human users or algorithms, Linkify will rely on ensemble methods. When trying to find a solution to a hard problem, we often consider several different possible approaches and pick the one that seems the most appropriate for the situation. The same concept can be applied to developing algorithms to solve complicated problems. Instead of developing a single complex algorithm to handle all the possible approaches to solving a difficult problem, we can take several simple algorithms and combine them into a single learner that is capable of handling those complex situations. This is the concept behind ensemble learning [14]. The second part of this thesis will present a simple model for the tagging process and its associated Expectation Maximization (EM) algorithm, which can be considered a novel ensemble learning algorithm, as it can be used to combine tags created by different users and algorithms in order to improve tagging accuracy when compared to the known ground truth.
Chapter 2

Literature Review

2.1. Cognitive Tutors

Limited information is available about tutoring systems in general, as many of them are proprietary. Furthermore, even when information is available about other tutoring systems, the variety of different approaches used by each system can make comparisons somewhat difficult. With that said, there are still a few well known examples of cognitive tutors.

One of the earlier attempts at building a cognitive tutor was the Practical Algebra Tutor (PAT). PAT was a high school mathematics tutor that contained a model of the steps required to solve each problem in the system and used knowledge tracing to determine the student’s strengths and weaknesses, based on the student’s performance in each problem solving step [2]. Based on these
strengths and weaknesses, it determined optimal problems to be assigned. However, the problem model used in PAT had to be manually constructed beforehand by experts.

Another example of an early cognitive tutor was Andes, a system built for the purpose of teaching college physics [4]. Its strategy consisted of assisting students in solving physics problems step by step, while keeping a Bayesian network model of the student’s mental state internally. Homework problems were still selected manually by the instructor. The entire system was based on a large set of rules for problem solving that were, once again, defined by experts.

OpenStax Tutor differs from these tutoring systems because it does not attempt to guide students through step-by-step problem solutions, preferring instead, to focus on optimal problem selection and presentation conditions. For this reason, OST facilitates the testing of cognitive science concepts and machine learning algorithms that might improve learning through its highly customizable research interface. Machine learning algorithms that have been successfully tested are currently being integrated into OST.

### 2.2. Community Tagging Systems

Social bookmarking websites such as Delicious [11], StumbleUpon [12] and CiteULike [13] are an example of tagging systems that are separate from content serving systems. These sites allow users to bookmark and tag content on the web,
and share those bookmarks and tags with users with similar interests [15]. However, researchers have found in [16] that allowing users to create any tag and assign it to content with little or no control over the set of allowed tags tends to lead to several problems. On one hand, a small group of tags that seems to be widely used by different users, but is too general and provides too little information to be useful for grouping different pieces of content. On the other hand, there is a very large group of highly specific tags which could be used to distinguish pieces of content, but unfortunately are strongly correlated with individual users, indicating that they are specific to each individual user and not shared with others, making it hard to find an agreed-upon content grouping.

Linkify was built to work like a common social bookmarking site, but it avoids these problem by providing a more constrained, predetermined set of tags that can be used to tag content, Since Linkify handles only educational resources, the set of tags is designed to relate directly to topics that students can learn in the classroom. Furthermore, Linkify differentiates itself from existing systems by allowing and encouraging tags made by non-human users, such as machine learning algorithms. Using ensemble methods, Linkify can combine tags generated by different users, as well as tags created by machine learning algorithms, in order to produce a set of tags that would better approximate the tags that would be created by subject-matter experts.
2.3. Ensemble Learning

Different users and machine learning algorithms can solve different learning problems with varying degrees of accuracy. In the case of tagging, different users or algorithms may potentially generate different sets of relevant tags for the same piece of content, depending on the strategy they use. Therefore, it should be possible to use ensemble learning methods to combine their tags. Ensemble methods attempt to combine learners in order to produce a result that can better approximate some predetermined ground truth. Their performance relies on there being significant differences, or diversity, among the learners being combined [14]. In the case of tagging, the ground truth refers to the tags that would be assigned to the content by subject-matter experts.

Ensemble learning methods can be thought of as consisting of two steps. First, a set of simple classifiers is trained to solve the desired problem. Then, a combination rule is used to produce a final result from the outputs of all the classifiers. We will investigate some common algorithms for classifier generation and, since we are focusing on tagging, some combination rules for discrete outputs.

2.3.1. Ensemble Learning Classifier Generation

As listed by [14], common methods for generating classifiers include bootstrap aggregating (bagging), boosting methods (the most famous of which is AdaBoost), stacked generalization and mixtures of experts.
Bagging algorithms train classifiers by drawing random samples from the dataset. According to [14], a large percentage of the dataset is usually used, in order to ensure that there are enough training samples, and the classifiers used are usually unstable, such as decision trees and neural networks, so that the final set of classifiers is diverse. Since the classifiers in bagging are independently trained, this approach is very well suited for parallelization [17].

Boosting algorithms work similarly to bagging, however, after the first classifier is trained, the subsets of the data used to train subsequent classifiers are carefully chosen so these classifiers can correct errors made by the previous classifiers [14]. The most well known is probably AdaBoost, short for Adaptive Boosting, which is a generalization of boosting that can be applied to several different problems [18]. Many boosting algorithms based on AdaBoost also exist, such as LogitBoost, which applies AdaBoost to logistic regression [19].

Stacked generalization and mixtures of experts are conceptually similar methods, in which multiple levels of classifiers are used in succession [14]. In stacked generalization, the second level classifier receives the outputs of a number of first level classifiers and makes a final decision. In mixtures of experts, instead of using a classifier to combine the outputs of previous classifiers, a gating network is used to pick which of the classifiers will be used for each data point. The outputs from the chosen classifiers can then be combined using one the available combination rules.
In this work, we used a variety of different machine learning algorithms to generate the classifiers to be combined, including some bagging and boosting methods. However, we present a novel combination rule which we developed, based on a simple model of the tagging process and on the EM algorithm, in order to combine their results.

2.3.2. Combining Discrete Outputs

Due to Linkify’s architecture, algorithm outputs are stored as tags, which are basically discrete labels. Therefore, we investigated combination rules capable of merging discrete classifier results. For the purpose of combining discrete outputs, [14] lists techniques such as (weighted or not) majority voting, behavior knowledge space table and borda count.

Majority voting is the simplest and probably the widely known method for combining discrete labels. Depending on the problem, it might be sufficient to pick the label with the most votes out of all the classifiers, while for certain applications where the cost of a false positive is large, we might require more than half the votes or even unanimous voting in order to assign a certain label to a data point. As explained in [14], classifiers can be assigned weights based on how apt they are at solving a particular problem and those weights can then be considered for weighted majority voting.

Behavior knowledge space relies on creating a lookup table with all possible classifier outputs [14]. This table maps each combination of classifier outputs to the
most common ground truth result in the training set. Although in theory more accurate than majority voting, the amount data needed to complete the table and size of the table itself become intractably large with larger numbers of classifiers, due to its exponential growth, requiring special precautions to reduce the amount of space used [20].

Borda count is another famous voting method, where voters rank different labels, with each rank being worth a different number of points. The points earned from all voters are added up and the candidate label or class with the most points wins. Unfortunately, borda count only works when the classifiers being used are able to rank the different possible labels, which is not always the case.

Different classifier training schemes and combination rules are more or less suited for different problems. The work in [14] cites the no free lunch theorem [21] as proof that no single algorithm or combination rule can be considered the best for all different problems. This thesis focuses on a novel combination rule for discrete classifiers based on a simple model of the tagging process and on the EM algorithm, which thereby differs from these other widely used combination rules.

**2.3.3. Applications of Ensemble Learning**

Ensemble methods have already been used successfully in several applications. In 2010, [22] used linear combinations of 6 different methods to drastically improve the performance of computer-aided pulmonary nodule detection from tomography scans. Similarly, [23] also reported great results for the
automatic detection of the optic disc and macula in retinal images, while [24] did the same for the automatic detection of QRS complexes in electrocardiograms.

The most famous example of the success of an ensemble method is probably the algorithm that won the Netflix Prize. The Netflix prize was a competition sponsored by Netflix between 2006 and 2009. The goal was to develop a machine learning algorithm that could predict movie ratings given by users with a 10% smaller Root-Mean-Square Error than the company’s own proprietary algorithm, using the actual ratings given by users as the ground truth. The best performing algorithm each year would win $50,000, while the first algorithm to reach a 10% improvement would be awarded a prize of $1,000,000 [25].

The winning algorithm was developed by researchers at AT&T Labs [26], in combination with a couple of other teams. They merged hundreds of different classifiers to win both the 2007 and 2008 Progress Prizes, as well as the Grand Prize in 2009. Initially, they combined the classifiers using linear regression [27], although near the end they switched to using a boosting-based strategy with decision trees [28].

By combining multiple different approaches, ensemble learning seems to currently be the most robust way to reduce errors when looking for answers to hard detection or prediction problems. Therefore, applying a similar method to machine learning problems in the educational setting through Linkify seemed to be the correct move in our quest to improve student learning.
An example of an algorithm which we hope to be able to use in the future in Linkify is SPARFA, short for SPARse Factor Analysis. Rice University’s proprietary SPARFA algorithm can be used both to calculate how different questions relate to different concepts and to estimate each student’s mastery of each of the different concepts [29]. It uses as input only the information about which questions each student got right or wrong. From that, it produces a set of concepts present in the questions (W matrix) and a value for each student’s mastery of each concept (C matrix). While it was designed to work alone, SPARFA (or one of its variations) could potentially benefit from being combined with tagging algorithms. Since Linkify focuses on educational resources and tags, SPARFA’s W matrix could be entered into Linkify and combined with other tagging algorithms. If the literature on ensemble methods is any indication, the final result should perform better than any of the individual algorithms it is made of.
Chapter 3

Research Question

This thesis is divided into two parts. Each part will answer one of the following questions:

- Can we design a system that allows multiple users and algorithms to tag educational resources and then applies ensemble methods to combine these tags in order to better approximate the tags that would be created by subject-matter experts?
- Can we develop a combination rule for content tags that outperforms other rules such as majority voting in situations similar to ours?
Chapter 4

The Linkify System

Linkify is a web-based collaborative tagging system being developed at Rice University. It will eventually be part of the OpenStax family of websites. Linkify is structured just like any other web application, designed to be used by both human users and by machine learning algorithms or other web applications directly. It has both basic HTML views (web pages) for human users to interact with the system, as well as a JavaScript Object Notation (JSON) API for use by web applications and algorithms. Linkify aims to provide machine learning algorithm developers with a common interface for recording their results, in order to facilitate the use and combination of the data produced by those algorithms.
4.1. Language and Framework

Linkify is being developed with the Ruby on Rails framework, which uses the Ruby programming language. This tested and proven framework is used for fast web development. Linkify will have to eventually interface with other OpenStax websites, so writing them in a common language and framework is a bonus from the point of view of reusing common code libraries among the different sites. Although competitors have surfaced in recent years, Ruby on Rails is still one of the frameworks most widely used to develop complex web applications.

Scalability concerns have been raised about Rails in the past, particularly when Twitter switched from their Ruby-on-Rails frontend to their own Java-based solution and claimed to have reduced their search latency to $1/3$ of the original value [30]. Although scalability may be a concern when extremely large numbers of users are involved, such as in Twitter’s case, there seem to be effective ways around these limitations, as shown in [31]. Furthermore, we don’t expect Linkify to ever grow to the same size as Twitter.

4.2. Data Model

There are three main object types in Linkify that represent the system’s model of the tagging process: resources, concepts and links. Linkify focuses on storing educational materials and their relations between each other and with concepts. Student data is handled separately by other OpenStax systems.
A resource refers to anything that can help a student learn and that can be leveraged by our tutoring system. Resources include online books, articles, problems, videos, etc. Since resources need to be accessible to students, we require all resources to have valid URI’s and encourage only the use of resources intended to be permanently available (often called permalinkable). Resources will initially be added to Linkify by either administrators or directly by partner websites, although in the future we plan to have mechanisms to allow users to add specific third party content, such as YouTube videos.

Concepts represent certain areas of knowledge. Any topic that could be taught in an academic course is considered a concept in Linkify. They can also be thought of as “tags” that can be added to resources and are directly related to the outputs of our machine learning algorithms. New concepts can be created by administrators and by machine learning algorithms. It is critical for the success of Linkify that concepts be defined in specific enough terms so that all users and algorithms can agree on their meaning, and that we avoid duplicated concepts, otherwise, applying ensemble methods to the system could be problematic. Concepts in Linkify are assigned a unique internal URL and from then on treated like any other resource.

Links represent a semantic relation between two resources, between two concepts or between a resource and a concept that Linkify knows about. They can be unidirectional or bidirectional. Links know not only which resources they are linking, but also the nature of the relation between the two resources. The available
types of links (relation types) will be managed by system administrators; however, any user or algorithm will be able to create a link of any existing type between resources or concepts in Linkify.

Creating a link (not to be confused with a simple hyperlink) in Linkify is similar to tagging a resource, although it allows for more expressiveness. A link between a resource and a concept is equivalent to a tag; however, a link between two resources or two concepts indicates the presence of some relation between them. For example, two resources can be linked to indicate that one is easier than the other, a relationship that is not so easily explained by tags. Links represent beliefs held by their creator; therefore they must keep track of who created them, as well as which users or algorithms agree with their existence.

Figure 1 – A simplified view of Linkify’s class diagram.
As an example of how tagging would be represented in Linkify, consider two chapters of a hypothetical book. Each chapter teaches a single concept and contains two different problems to be solved. Furthermore, assume that the concept taught in chapter 1 needs to be understood before it is possible to understand the concept in chapter 2. A representation of this data in Linkify might look like the following:

Figure 2 – Example Linkify representation of a hypothetical book with 2 chapters with 1 concept and 2 problems each.

In the figure above, the arrows represent unidirectional link objects. The text next to each arrow indicates the type of relation described by the Link. Machine learning algorithms could use this data to infer, for example, that students should master chapter 1 before beginning chapter 2, or that problems 1 and 2 should be
done before attempting problems 3 and 4. Using this representation, Linkify can store complex relationships between educational resources.

Linkify’s model of the world means that the structure of its data will be a graph, with resources being nodes and links being the edges between nodes. We have given thought to the idea of using a graph database for Linkify. Relational databases are widely regarded as the go-to database when parts of the data have to be queried randomly (for example, when searching), while graph databases can make graph-specific operations such as graph traversal much faster. Although a graph database might allow us to perform graph traversal on Linkify’s data at a lower cost, so far it seems like graph traversal will not be a very common operation. The most compelling case for graph traversal would be to generate a customized learning plan for a student with the goal of learning a particular concept in the future. Even this kind of query seems like it would be infrequent enough that a Relational DataBase Management System (RDBMS) would be able to perform it just fine. On the other hand, being able to query the data in multiple ways is one of the major strengths of the RDBMS and the reason why we are continuing to use a SQL-based database.

Linkify’s data model allows us to describe complex relations between educational resources. Tags can be entered into the system as links by both users using its HTML web interface and machine learning algorithms using its JSON API.
4.3. API

Linkify is an API-centered service, providing a stable, versioned JSON API to store results from machine learning algorithms. The system’s API follows the Create, Read, Update, Delete (CRUD) pattern whenever possible, which is standard in the development of web-facing API’s. In this pattern, four API endpoints usually exist for each kind of object in the system. Each of those endpoints performs one of the four commonly needed operations on that object (create, read, update or delete).

Without getting too technical, an object is first created by using the create endpoint and specifying the necessary parameters, which can vary for different objects. The system then creates the object in the database and returns a unique ID for that object, which can then be used to read, update or delete it.

For example, a resource object in Linkify could be created by specifying the URL of an educational resource, such as an online video. The API would then return a response containing the new object’s ID. This ID could then be used to create a link object between this resource and some existing concept, for example the concept of multiplication. At the end of the process, the system is now aware of this new resource, which is associated with the concept of multiplication.

In order for data to be useful, users and algorithms need to be able to query it in different ways. For this purpose, Linkify will also provide search APIs. There are two main ways to query the objects in Linkify. The first way is the related object search, to be accessed by other web applications. When given a resource or concept,
Linkify is able to return all the links that point to that object, as well as all the resources on the other side of each link. This type of search will allow us to quickly find and suggest other activities related to what a student is currently doing, for example. The results can be filtered based on who created each link, allowing each web application to query its own set of trusted users or machine learning algorithms. The second type of search returns all the results recorded by a particular user or algorithm in Linkify. This search will most often used by machine learning algorithms, specifically those designed to aggregate results from multiple other algorithms. Incremental updates may also be implemented for this API endpoint in the future.

4.4. Linkify Summary

We have shown that Linkify is capable of keeping track of content tags created by users and algorithms alike and associated to educational resources. By using a pre-determined set of tags, we enable the comparison of sets of tags generated by different users and algorithms. Through our querying APIs, we allow ensemble methods to be applied to the sets in order to produce new tags that are closer to those which would be assigned by experts.
Chapter 5

Combining Tagging Algorithms via Expectation Maximization

In order to combine the resource tags added to Linkify by different users and algorithms, we developed a simple model of the tagging process, based on an asymmetric binary channel. From this model, we also derived an algorithm based on Expectation Maximization that can be used to find the most likely set of tags.

5.1. Tagging Model

In our model, the association of tags with resources is represented by a tagging matrix. Matrix dimensions are indexed as follows: \( i \) indexes the first dimension, containing tagging algorithms, while \( j \) indexes the second dimension, containing resources to be tagged and \( k \) indexes the third dimension, containing the
tags in the set of possible tags. The tagging matrix contains a 1 at position \((i, j, k)\) if
algorithm \(i\) assigned tag \(k\) to resource \(j\) and 0 otherwise.

We also keep track of a ground truth tagging matrix, representing the ground
truth set of tags. We model the ground truth tags as being drawn from a Bernoulli
distribution with parameter \(p_k\). That is, our model assumes that the values in the
ground truth tagging matrix, which determine whether or not each resource
receives a particular tag, are drawn from the following distribution:

\[
P(z_{jk}) = \begin{cases} 1 - p_k, & z_{jk} = 0 \\ p_k, & z_{jk} = 1 \end{cases} = \chi(z_{jk} = 0)(1 - p_k) + \chi(z_{jk} = 1)p_k
\]

**Equation 1 – Bernoulli distribution with parameter \(p_k\).**

The tagging matrix is sparse, since for any given resource-tag combination
there is a very small chance of that resource actually having that tag. Therefore, the
value of \(p_k\) is small (close to 0).

We also modeled each algorithm \(i\) as being an asymmetric binary channel. In
our model, each tagging algorithm or user is allowed to observe the ground truth
tagging matrix only through this noisy binary channel. The noisy observations that
come out of each channel are the outputs of each algorithm:
Therefore, each algorithm $i$ has the following probabilities of making mistakes:

$$0 \to 1 \text{ with probability } q_i$$

$$1 \to 0 \text{ with probability } r_i$$

The model parameters are thus $p_k$, $q_i$ and $r_i$.

### 5.2. EM Combination Rule

Based on the tagging model, we developed a simple algorithm based on EM that can combine the observations from each individual algorithm in order to produce the Maximum Likelihood Estimate (MLE) of the true data. Calculating the MLE for our model parameters directly is intractable for large matrices, as the number of combinations grows exponentially. So it is necessary to use an algorithm like Expectation Maximization [32] or Markov Chain Monte Carlo (MCMC) [33].
MCMC methods are known to be slow for large numbers of variables [33], thus we opted to implement the EM algorithm, which is generally known to be fast.

The algorithm developed assumes the tagging matrices are binary. If real values are present in the tagging matrix, they can be handled by applying a threshold. The full step-by-step derivation of the EM algorithm is presented in Appendix A.

5.3. Algorithm Analysis

Let \( I \) be the number of classifiers being combined, \( J \) the number of items being tagged, \( K \) is the number of tags. We will analyze the space and time complexity of the EM algorithm, independent of the other algorithms being used in the ensemble. Since the EM algorithm only uses the final outputs of the other algorithms, the space and time complexities of the whole ensemble are simply the largest space and time complexities among all algorithms, including the EM. That is, the space complexity of the whole ensemble is the same as the space complexity of the algorithm that uses the most space, and the time complexity of the ensemble is the same as the time complexity of the slowest algorithm.

5.3.1. Space Complexity

The EM algorithm needs to store an \( I \times J \times K \) matrix, but all space used during each iteration can be reused in the next. Therefore, its space complexity is simply
$O(IJK)$, that is, on the order of the number of classifiers used times the number of pieces of content in the training set times the number of allowed tags.

### 5.3.2. Time Complexity

The main stopping criterion for the EM algorithm is based on the change in the likelihood per iteration, which depends on the overall shape of the likelihood function. Since the likelihood function is hard to estimate before the EM algorithm is applied, there is no closed form solution for the time complexity of the EM algorithm according to this particular stopping criterion.

However, as is customary with EM algorithms, there is a secondary stopping criterion in the form of a maximum number of iterations allowed, which we represent here by $M$. This secondary criterion is not ideally reached, but it still provides us with a way to derive a loose upper bound on the number of operations performed by the EM algorithm. With this secondary stopping criterion, the EM algorithm's time complexity is $O(IJKM)$, as operations done in each iteration involve the sum and element-wise multiplication and division of matrices of up to size $IxJxK$. That is, the time complexity is on the order of the number of classifiers being combined times the number of pieces of content in the training set times the number of tags in the set of allowed tags times the maximum allowed number of iterations.
5.3.3. Rate of Convergence

The EM algorithm is guaranteed to converge to a local maximum of the likelihood function, since the expectation of the log likelihood monotonically increases with each iteration [32]. However, there is no guarantee that we will converge to the global maximum of the log likelihood. In fact, it seems to frequently get stuck on local maxima. Thus, we use random restarts to provide a better chance of reaching the global maximum and to produce a better estimate of the model parameters. The rate of convergence of the EM algorithm depends, according to [32], on the amount of information lost in the model. This corresponds to the average probability that the algorithms being combined will make mistakes.

A closed form solution for the EM algorithm's rate of convergence for this particular model does not currently exist, therefore we resorted to using simulated data to calculate the number of iterations necessary for the algorithm's convergence for different values of the error rate of the classifiers used. The results are shown in the following plots, when combining 3 classifiers:
Figure 4 – Number of iterations until convergence for the EM algorithm versus the average error rate of its 3 classifiers.

The number of iterations required for convergence of the EM algorithm shows a strong dependency on the average number of errors made by the algorithms being combined. For this plot, a total of 20 different sets of 3 classifiers with different average error rates were generated (the x-axis) and each set of classifiers was applied to 10 different generated datasets (the y-axis). For the purposes of this plot, convergence was said to be achieved when the log likelihood changed by less than 0.0001 in some iteration.
The following plot depicts the exact same scenario as the previous one, except that 5 classifiers were combined instead of 3:

Figure 5 – Number of iterations until convergence for the EM algorithm versus the average error rate of its 5 classifiers.

As we can see, the EM algorithm converges much faster when more classifiers are being combined.
5.4. Limitations

The EM combination rule described possesses a few limitations: it can only be used if all the algorithms being combined agree on the set of tags to be used and on the individual meaning of each tag. That is, all classifiers must agree on the labels being used. Additionally, it does not model the fact that some tagging algorithms might do remarkably better or worse than others at observing certain tags. The first limitation is solved by enforcing a set of allowed tags common to all algorithms. The second limitation could lead to less than ideal results in some situations, if the accuracy of some algorithm depends heavily on the tag being considered. Because of this, classifiers capable of using only a very limited set of tags, such as face detectors, should not be used with this combination rule.

5.5. Candidate Algorithms for Combination

For the purpose of tagging educational content, few algorithms stand out. SPARFA could be used to tag practice problems, as previously mentioned. Of all the variations of SPARFA, Tag-Aware SPARFA [34] seems to be the best suited for use with the EM algorithm shown in this chapter, since it can be constrained to use only tags from the set of available tags. Similarly, Labeled LDA [35] constrains Latent Dirichlet Allocation to also use tags from a predefined set. This characteristic makes those two algorithms the primary candidates for use with our model and EM algorithm and for future investigation.
Chapter 6

Testing with Synthetic Data

Our EM combination rule was first tested using synthetic data generated according to our tagging model. For each test, we assumed the presence of 1,000 resources to be tagged with 30 different tags. Each tag was present on less than 10% of the resources, with the actual rate for each tag being uniformly distributed on the interval between 0 and 0.1. Based on this, 10 ground truth tagging matrices were generated for each test, each with 30,000 binary entries.

The maximum classifier error rate for all simulated classifiers was stepped from 0 to 0.5. For each value of the maximum error rate, 10 classifier error rates were drawn from a uniform distribution between 0 and the maximum error rate. Each of the classifiers then received as input the 10 ground truth tagging matrices and produced as output 10 matrices of observed tags, with each observed tag being different from the ground truth tag with a probability equal to the classifier's error
rate. Finally, the EM combination rule was applied to these tagging matrices in order to generate the ensemble tagging results.

6.1. Combining 3 Classifiers

The following plots show the EM algorithm’s performance for combining 3 tagging algorithms. The x-axis shows the average classifier error rate for that trial, while the y-axis represents the EM ensemble’s error rate:

Figure 6 – Ensemble error rate versus average error rate of its 3 classifiers.
As shown by the plot, the number of errors made by the ensemble increases as the number of error made by each individual algorithm increases, but in a smaller proportion.

For the following plot, the x-axis shows the maximum classifier error rate, as described in the beginning of this chapter. The y-axis once again contains the error rate of the EM ensemble. The line represents the average number of errors made by the ensemble, while the error bars indicate the variance:

![Graph showing EM vs max classifier error rate](image)

Figure 7 – Ensemble average error rate with variance versus maximum allowed classifier error rate, with 3 classifiers.
As we can see, both the average ensemble error rate and the variance increase as the error rate of the individual classifiers increases.

6.2. Combining 5 Classifiers

The following plots show the EM algorithm combining 5 classifiers:

Figure 8 – Ensemble error rate versus average error rate of its 5 classifiers.
The performance shown in the previous plot is clearly better than the one for the 3 algorithm case, showing how the ensemble with the EM combination rule performs better with more classifiers.

![EM vs max classifier error rate](image)

**Figure 9 - Ensemble average error rate with variance versus maximum allowed classifier error rate, with 5 classifiers.**

Here we see that both the ensemble error rate and its variance are smaller for the 5 classifier case when compared with the 3 classifier plot.
6.3. Comparison with Majority Voting Combination Rules

This next plot compares the performance of the EM combination rule with several flavors of the majority voting combination rule. For the majority voting rules, a tag is thought to be present if the number of classifiers assigning that tag exceeds a threshold. Since the tagging matrix is known to be sparse, supermajority rules were also tested. These tests were done with 15 different tagging algorithms, in order to allow some differentiation between the different majority voting schemes. Here are the results:

![Graph showing comparison between EM and majority voting rules]

Figure 10 – Performance comparison of the EM combination rule with the majority voting rule on synthetic data.
The synthetic data used for this plot was generated according to the model upon which the EM algorithm is based, so it is expected to outperform the majority voting rules in this dataset. As expected, the ensemble using the EM combination rule shows a lower average error rate when compared to the majority voting algorithms, especially when the error rate of the individual classifiers increases.
Chapter 7

Testing with Real World Data

While tests with synthetic data serve as a sanity check, tests with datasets that come from the real world are necessary to determine if the new combination rule really works as expected. In order to test with real world data and machine learning algorithms, we several needed implementations of different algorithms. Since implementing multiple tagging algorithms was outside of the scope of this work, we made use of the Weka toolbox [36], an open source machine learning toolbox that provides several popular classifiers. The Weka toolbox can be downloaded from [37]. Although we did not use educational data directly, due to the difficulty in preparing those datasets for the tests, we tackled similar tagging problems using two readily available datasets.
7.1. Predicting a Single Tag

The first test we performed with real data involved predicting a single tag from other variables in the dataset. We used the Weka diabetes example dataset, which is provided with the Weka framework in a convenient format. In this dataset, patients are assigned a label that denotes whether or not they tested positive for diabetes according to WHO criteria. Several biometric measurements for each patient are also provided. The goal of this test is to try to predict the test label (whether they would test positive for diabetes or not) from the rest of the biometric data. All patients in this dataset were females 21 years or older of Pima Indian heritage, which probably contributed towards the high accuracy rate of most of the classifiers used.

Several machine learning algorithms were applied to the dataset to produce a simple yes or no classification result for each patient. The EM combination rule was then applied to different mixes of results in order to produce ensemble sets of tags. When applied to most of the mixes of results, the EM algorithm did just slightly worse than the best result in the mix, when compared against the ground truth set.

However, for one particular combination of algorithms, the ensemble with the EM combination rule actually slightly surpassed the results of all other algorithms in the mix. In this trial, the best single algorithm was the sequential minimal optimization (SMO), which had an accuracy rate of 77.34%, while the EM ensemble obtained an accuracy rate of 77.86% in predicting the test results. The mix
of classifiers contained the default Weka implementations of the following algorithms: logistic regression, multilayer perceptron, j48 (decision tree), decision forests, sequential minimal optimization, Bayesian networks, naïve Bayesian classifiers (simple linear and multinomial), LogitBoost with simple logistic functions and AdaBoost with decision stumps.

### 7.2. Predicting Multiple Tags

For the second test, we used the NUS-WIDE-LITE dataset [38]. Although previously available online at the National University of Singapore’s website, this dataset seems to be unavailable for download at the moment. This dataset is comprised of over 50,000 images from Flickr, a photo sharing website [39], split half and half for the purposes of training and testing machine learning algorithms. It contains extracted features as well as 2 preexisting sets of tags for each image. The first set of tags was assigned by Flickr users, while the second one was manually assigned by the researchers and their students and is considered the ground truth set.

Due to the size of this dataset, we could not apply all of the algorithms used with the previous dataset. Instead, we chose the 5 fastest algorithms from the previous test and applied them to the image features contained in the NUS-WIDE-LITE dataset, generating 5 different sets of tags. These tags were then combined with the tags assigned by Flickr users using our EM combination rule. The results are shown in the table below:
<table>
<thead>
<tr>
<th>Classifier</th>
<th>Precision</th>
<th>Recall</th>
</tr>
</thead>
<tbody>
<tr>
<td>Flickr Users</td>
<td>76.27%</td>
<td>30.72%</td>
</tr>
<tr>
<td>AdaBoost</td>
<td>70.13%</td>
<td>37.90%</td>
</tr>
<tr>
<td>LogitBoost</td>
<td>72.29%</td>
<td>39.03%</td>
</tr>
<tr>
<td>Voted Perceptron</td>
<td>74.74%</td>
<td>46.42%</td>
</tr>
<tr>
<td>j48 Decision Tree</td>
<td>47.92%</td>
<td>46.29%</td>
</tr>
<tr>
<td>Random Forests</td>
<td>76.76%</td>
<td>37.21%</td>
</tr>
<tr>
<td>EM Ensemble</td>
<td>67.40%</td>
<td>51.84%</td>
</tr>
</tbody>
</table>

Table 1 – Precision and Recall of individual machine learning algorithms and EM ensemble when tagging the NUS-WIDE-LITE dataset.

The high precision indicates that a high percentage of the tags returned by the algorithms were actually present in the ground truth set, which corresponds to a low rate of false positives. The low recall rate indicates that the algorithms missed a high percentage of tags that were actually in the ground truth set, revealing a high rate of false negatives.

As we can see, all algorithms had very low recall rates for this dataset, meaning that they missed a lot of tags that should have been assigned. The ensemble with the EM combination rule did not do too well in terms of precision in this case, although it had a higher recall rate than all the other algorithms being combined. It was middle of the pack in terms of total number of errors, but could still be useful if the cost of false negatives was higher than the cost of false positives for some applications of this image tagging scheme.
8.1. Conclusion

We have described the details of the Linkify system and have shown that it can be used to represent and store tagging data coming from both users and machine learning algorithms. We have also discussed how we can use ensemble algorithms with Linkify in order to improve the accuracy of tagging when compared to some ground truth, such as tags generated by subject-matter experts. This allows us to obtain a better estimate of which learning resources related to which concepts and thus improve our estimates of student knowledge and our predictions of student performance.

We have also derived a novel combination rule for classifiers based on a simple model of the tagging process and on the EM algorithm. So far, testing results
of this EM combination rule both with synthetic and with real data have shown great promise. This combination rule can be applied to the ensemble algorithms used in the Linkify system to further improve the accuracy of its tags.

8.2. Future Work

In the future, the performance of the EM combination rule should be compared against other combination rules such as the behavior knowledge space on synthetic and real world data and the majority voting rules on real world data. Two possible extensions also come to mind: the first would be to include a new model and an algorithm that take into account the fact that tagging algorithms can have different performances on certain sets of tags; the second would be to incorporate classifiers with continuous outputs, perhaps taking the continuous value of the output to mean how certain the classifier is about the existence of the tag. Finally, Linkify needs to be updated to use the latest OpenStax libraries before it is ready for deployment.
References


Appendix A

Simple Tagging Model and EM Combination Rule Derivation

Matrix dimension indexes:

\[ i = \text{index for the first dimension (classifiers)} \]

\[ j = \text{index for the second dimension (content)} \]

\[ k = \text{index for the third dimension (tags)} \]

Constants:

\[ I = \text{total number of classifiers (tagging algorithms)} \]

\[ J = \text{total number of pieces of content being tagged} \]

\[ K = \text{total number of tags in the set of allowed tags} \]

Functions:

\[ P = \text{probability density function (PDF)} \]

\[ \chi = \text{indicator function} \]
Random variables:

\( Y \) = algorithm outputs (observed random variable)

\( Z \) = ground truth taggings (latent random variable)

Random variable notation:

\( Y \) = random variable

\( Y \) = vector of random variables

\( y \) = sampled value for \( Y \)

\( y \) = vector of sampled values for \( Y \)

Therefore:

\( y_{ijk} \) = whether algorithm \( i \) thinks content \( j \) is associated with tag \( k \)

\( z_{jk} \) = whether content \( j \) is actually associated with tag \( k \) (ground truth)
In this derivation, $P(z_{jk})$ should be taken to mean $P(Z_{jk} = z_{jk})$.

Simple Tagging Model:

Ground truth tags are drawn from a Bernoulli distribution:

$$P(z_{jk}|\theta) = \begin{cases} 1 - p_k, z_{jk} = 0 \\ p_k, z_{jk} = 1 \end{cases} = \chi(z_{jk} = 0)(1 - p_k) + \chi(z_{jk} = 1)p_k$$

$$P(z|\theta) = \prod_j \prod_k P(z_{jk}|\theta)$$

Where $\theta$ is our current estimation of the set of parameters.

According to the asymmetric binary channel model, each algorithm $i$ will make a mistake with the following probabilities:

$$0 \rightarrow 1 \text{ with probability } q_i$$

$$1 \rightarrow 0 \text{ with probability } r_i$$

The model parameters are then $p_k, q_i \text{ and } r_i$. 
EM Algorithm Derivation:

Calculating the Maximum Likelihood Estimate for our model parameters directly is intractable for large matrices, as the number of possible values of $Z$ grows exponentially with the dimensions of the tagging matrix. So we rely on the EM algorithm.

The elements of $Y$ are correlated. However, they are conditionally independent, given the model parameters and the elements of $Z$:

$$
P(y_{ijk} | z_{jk}, \theta) = \begin{cases} 
(1 - q_i), z_{jk} = 0, y_{ijk} = 0 \\
r_i, z_{jk} = 1, y_{ijk} = 0 \\
q_i, z_{jk} = 0, y_{ijk} = 1 \\
(1 - r_i), z_{jk} = 1, y_{ijk} = 1 
\end{cases}
$$

$$
= \chi(z_{jk} = 0)(\chi(y_{ijk} = 0)(1 - q_i) + \chi(y_{ijk} = 1)q_i) \\
+ \chi(z_{jk} = 1)(\chi(y_{ijk} = 0)r_i + \chi(y_{ijk} = 1)(1 - r_i))
$$

$$
P(y_{jk} | z, \theta) = P(y_{jk} | z_{jk}, \theta) = \prod_i P(y_{ijk} | z_{jk}, \theta)
$$

$$
= \prod_i \left( \chi(z_{jk} = 0)(\chi(y_{ijk} = 0)(1 - q_i) + \chi(y_{ijk} = 1)q_i) \\
+ \chi(z_{jk} = 1)(\chi(y_{ijk} = 0)r_i + \chi(y_{ijk} = 1)(1 - r_i)) \right)
$$

$$
P(y | z, \theta) = \prod_j \prod_k P(y_{jk} | z, \theta) = \prod_j \prod_k P(y_{jk} | z_{jk}, \theta)
$$
Therefore, the likelihood function is:

\[ L(\theta; z, y) = P(z, y|\theta) = P(y|z, \theta)P(z|\theta) \]

\[ = \prod_j \prod_k P(y_{jk}|z_{jk}, \theta) \prod_j \prod_k P(z_{jk}|\theta) \]

\[ = \prod_j \prod_k P(z_{jk}|\theta) \prod_i P(y_{ijk}|z_{jk}, \theta) \]

This gives us the following log-likelihood:

\[ \log L(\theta; z, y) = \sum_j \sum_k \left( \log P(z_{jk}|\theta) + \sum_i \log P(y_{ijk}|z_{jk}, \theta) \right) \]

\[ = \sum_j \sum_k \left( \log(\chi(z_{jk} = 0)(1 - p_k) + \chi(z_{jk} = 1)p_k) \right. \]

\[ + \sum_i \log \left( \chi(y_{ijk} = 0)(\chi(y_{ijk} = 0)(1 - q_i) + \chi(y_{ijk} = 1)q_i) \right) \]

\[ + \left. \chi(z_{jk} = 1) \left( \chi(y_{ijk} = 0)r_i + \chi(y_{ijk} = 1)(1 - r_i) \right) \right) \]

\[ = \sum_j \sum_k \left( \chi(z_{jk} = 0) \left( \log(1 - p_k) + \sum_i \log(\chi(y_{ijk} = 0)(1 - q_i) + \chi(y_{ijk} = 1)q_i) \right) \right) \]

\[ + \chi(z_{jk} = 1) \left( \log p_k + \sum_i \log \left( \chi(y_{ijk} = 0)r_i + \chi(y_{ijk} = 1)(1 - r_i) \right) \right) \]
E-step: Here we derive the $Q$ and $R$ terms that will be reused in the update rules. Elements of $z$ depend only on the outputs for the same $j$ and $k$ ($y_{jk}$), so:

$$P(z_{jk} \mid y, \theta) = P(z_{jk} \mid y_{jk}, \theta) = \frac{P(y_{jk} \mid z_{jk}, \theta)P(z_{jk} \mid \theta)}{P(y_{jk} \mid \theta)}$$

$$= \frac{P(y_{jk} \mid z_{jk}, \theta)P(z_{jk} \mid \theta)}{\sum_{z_{jk}} P(y_{jk} \mid z_{jk}, \theta)P(z_{jk} \mid \theta)}$$

$$P(y_{jk} \mid z_{jk}, \theta)P(z_{jk} \mid \theta) = P(z_{jk} \mid \theta) \prod_i P(y_{ijk} \mid z_{jk}, \theta)$$

$$= (\chi(z_{jk} = 0)(1 - p_k) + \chi(z_{jk} = 1)p_k)$$

$$* \prod_i \left( \chi(z_{jk} = 0)\left(\chi(y_{ijk} = 0)(1 - q_i) + \chi(y_{ijk} = 1)q_i \right) + \chi(z_{jk} = 1)\left(\chi(y_{ijk} = 0)r_i + \chi(y_{ijk} = 1)(1 - r_i) \right) \right)$$

$$= \chi(z_{jk} = 0)(1 - p_k) \prod_i (\chi(y_{ijk} = 0)(1 - q_i) + \chi(y_{ijk} = 1)q_i)$$

$$+ \chi(z_{jk} = 1)p_k \prod_i (\chi(y_{ijk} = 0)r_i + \chi(y_{ijk} = 1)(1 - r_i))$$

$$\rightarrow P(z_{jk} \mid y, \theta)$$

$$\chi(z_{jk} = 0)(1 - p_k) \prod_i (\chi(y_{ijk} = 0)(1 - q_i) + \chi(y_{ijk} = 1)q_i)$$

$$+ \chi(z_{jk} = 1)p_k \prod_i (\chi(y_{ijk} = 0)r_i + \chi(y_{ijk} = 1)(1 - r_i))$$

$$= \frac{(1 - p_k) \prod_i (\chi(y_{ijk} = 0)(1 - q_i) + \chi(y_{ijk} = 1)q_i)}{(1 - p_k) \prod_i (\chi(y_{ijk} = 0)(1 - q_i) + \chi(y_{ijk} = 1)q_i) + p_k \prod_i (\chi(y_{ijk} = 0)r_i + \chi(y_{ijk} = 1)(1 - r_i))}$$
Let $Q_{jk} = (1 - p_k) \prod_i (\chi(y_{ijk} = 0)(1 - q_i) + \chi(y_{ijk} = 1)q_i)$ and

$$R_{jk} = p_k \prod_i (\chi(y_{ijk} = 0)r_i + \chi(y_{ijk} = 1)(1 - r_i))$$

Then the previous equation becomes:

$$P(z_{jk} | \mathbf{y}, \theta) = \frac{\chi(z_{jk} = 0)Q_{jk} + \chi(z_{jk} = 1)R_{jk}}{Q_{jk} + R_{jk}}$$

And the expectation of the log likelihood is then:

$$E_{Z|Y,\theta_t}[\log L(\theta; \mathbf{z}, \mathbf{y})]$$

$$= E_{Z|Y,\theta_t} \left[ \sum_j \sum_k \left( \log P(z_{jk} | \theta) + \sum_i \log P(y_{ijk} | z_{jk}, \theta) \right) \right]$$

$$= \sum_j \sum_k E_{Z|Y,\theta_t} \left[ \log P(z_{jk} | \theta) + \sum_i \log P(y_{ijk} | z_{jk}, \theta) \right]$$

$$= \sum_j \sum_k \sum_{z_{jk}} \left( P(z_{jk} | \mathbf{y}, \theta_t) \left( \log P(z_{jk} | \theta) + \sum_i \log P(y_{ijk} | z_{jk}, \theta) \right) \right)$$
The subscript \( t \) here denotes the previous estimate of each parameter.
M-step: Here we calculate the various update rules.

\( p_k \) update:

\[
\frac{dE_{Z|Y, \theta^{(t)}}[\log L(\theta; Z, Y)]}{dp_k}
\]

\[
= \sum_j \left( \frac{R_{jkt}}{p_k} - \frac{Q_{jkt}}{1 - p_k} \right)
\]

\[
= \sum_j \left( \frac{(1 - p_k)R_{jkt} - p_k Q_{jkt}}{Q_{jkt} + R_{jkt}} \right)
\]

\[
= \sum_j \left( \frac{R_{jkt} - p_k (Q_{jkt} + R_{jkt})}{Q_{jkt} + R_{jkt}} \right) = 0
\]

\[
\rightarrow p_k \sum_j \left( \frac{Q_{jkt} + R_{jkt}}{Q_{jkt} + R_{jkt}} \right) = \sum_j \left( \frac{R_{jkt}}{Q_{jkt} + R_{jkt}} \right)
\]

\[
\rightarrow p_k = \sum_j \frac{R_{jkt}}{\sum_j 1} = \frac{1}{J} \sum_j \left( \frac{R_{jkt}}{Q_{jkt} + R_{jkt}} \right)
\]
$q_i$ update:

$$dE_{Z|Y, \theta^{(t)}}[\log L(\theta; Z, Y)]$$

$$dq_i$$

$$= \sum_j \sum_k \left( \frac{Q_{jkt}}{Q_{jkt} + R_{jkt}} \left( \frac{\chi(y_{ijk} = 1)}{q_i} - \frac{\chi(y_{ijk} = 0)}{1 - q_i} \right) \right)$$

$$= \sum_j \sum_k \left( \frac{Q_{jkt}}{Q_{jkt} + R_{jkt}} \frac{\chi(y_{ijk} = 1)(1 - q_i) - \chi(y_{ijk} = 0)q_i}{q_i(1 - q_i)} \right) = 0$$

$$\rightarrow \sum_j \sum_k \left( \frac{\chi(y_{ijk} = 1)(1 - q_i) - \chi(y_{ijk} = 0)q_i)Q_{jkt}}{Q_{jkt} + R_{jkt}} \right) = 0$$

$$\rightarrow \sum_j \sum_k \left( q_i \left( \frac{\chi(y_{ijk} = 1) + \chi(y_{ijk} = 0)}{Q_{jkt} + R_{jkt}} \right) Q_{jkt} \right)$$

$$= \sum_j \sum_k \left( \frac{\chi(y_{ijk} = 1)}{Q_{jkt} + R_{jkt}} \right)$$

$$q_i = \frac{\Sigma_j \Sigma_k \left( \frac{\chi(y_{ijk} = 1)}{Q_{jkt} + R_{jkt}} \right)}{\Sigma_j \Sigma_k \left( \frac{Q_{jkt}}{Q_{jkt} + R_{jkt}} \right)}$$
\[ r_i \ \text{update:} \]

\[
\frac{dE_{Z,Y,\theta^{(t)}}[\log L(\theta; Z, Y)]}{dr_i} = \sum_j \sum_k \left( R_{jkt} \left( \frac{\chi(Y_{ijk} = 0)}{r_i} - \frac{\chi(Y_{ijk} = 1)}{1 - r_i} \right) \right) \frac{Q_{jkt} + R_{jkt}}{r_i(1 - r_i)}
\]

\[
= \sum_j \sum_k \left( \frac{R_{jkt} \chi(Y_{ijk} = 0)(1 - r_i) - \chi(Y_{ijk} = 1)r_i}{Q_{jkt} + R_{jkt}} \right) = 0
\]

\[
\rightarrow \sum_j \sum_k \left( \frac{\chi(Y_{ijk} = 0)(1 - r_i) - \chi(Y_{ijk} = 1)r_i}{Q_{jkt} + R_{jkt}} R_{jkt} \right) = 0
\]

\[
\rightarrow \sum_j \sum_k \left( \frac{r_i \left( \chi(Y_{ijk} = 0) + \chi(Y_{ijk} = 1) \right) R_{jkt}}{Q_{jkt} + R_{jkt}} \right)
\]

\[
= \sum_j \sum_k \left( \frac{\chi(Y_{ijk} = 0)R_{jkt}}{Q_{jkt} + R_{jkt}} \right)
\]

\[
r_i = \frac{\sum_j \sum_k \left( \frac{\chi(Y_{ijk} = 0)R_{jkt}}{Q_{jkt} + R_{jkt}} \right)}{\sum_j \sum_k \left( \frac{R_{jkt}}{Q_{jkt} + R_{jkt}} \right)}
\]
We can also find our best estimate of the tagging matrix, $z$, using the MLE. $z$ must maximize the log likelihood:

$$\log L(\theta; z, y)$$

$$= \sum_j \sum_k \left( \chi(z_{jk} = 0) \left( \log (1 - p_k) + \sum_l \log (\chi(y_{ijk} = 0)(1 - q_i) + \chi(y_{ijk} = 1)q_i) \right) \\
+ \chi(z_{jk} = 1) \left( \log p_k + \sum_l \log (\chi(y_{ijk} = 0)r_i + \chi(y_{ijk} = 1)(1 - r_i)) \right) \right)$$

So for each $j$ and $k$, we choose the value of $z$ that gives the largest term for the above sum:

Choose $z_{jk} = 1$ if

$$\log (1 - p_k) + \sum_l \log (\chi(y_{ijk} = 0)(1 - q_i) + \chi(y_{ijk} = 1)q_i)$$

$$< \log p_k + \sum_l \log (\chi(y_{ijk} = 0)r_i + \chi(y_{ijk} = 1)(1 - r_i))$$

and $z_{jk} = 0$ otherwise.
Alternatively, we can think of this as choosing $z_{jk} = 1$ if

$$(1 - p_{kt}) \prod_i \left( \chi(y_{ijk} = 0)(1 - q_{it}) + \chi(y_{ijk} = 1)q_{it} \right)$$

$$< p_{kt} \prod_i \left( \chi(y_{ijk} = 0)r_{it} + \chi(y_{ijk} = 1)(1 - r_{it}) \right)$$

and $z_{jk} = 0$ otherwise.

In summary:

$$z_{jk} = \chi(R_{jkt} > Q_{jkt})$$

We have derived update rules for all the model parameters, as well as a way to calculate our best estimate of the tagging matrix. This concludes the derivation of our EM combination rule for ensemble methods applied to tagging algorithms.