CS299 Detailed Plan

Shawn Tice

February 5, 2013

Overview

The high-level steps for classifying web pages in Yioop are as follows:

1. Create a new classifier for a unique label.
2. Train it on a labelled training set containing both positive and negative examples of the class. If desired, repeat these first two steps to create multiple classifiers.
3. Add a page field extraction rule to save the class of a page into a meta variable.
4. Start a new crawl.

As suggested above, there may be many classifiers, and they are independent of any particular crawl. That is, classifiers are trained one at a time, and once trained, may be used many times. It is possible to train a classifier entirely on a hand-selected collection of labelled documents, but in the case that no such collection exists for a class of interest, one can take advantage of previous crawls to build one interactively. The idea is to use a classifier trained on a minimal training set to find documents from a previous crawl which are likely to be good positive and negative examples of the class, then to request labels for these documents from the person training the classifier, so that they may be added to the training set. The classifier is trained on the augmented set, and the process iterated until there are no more documents or the desired accuracy is achieved.

There are several algorithmic choices to be made in order to realize this system:

• We must choose a classification method to assign a likelihood that a particular document belongs to a particular class.

• To reduce the time it takes to train a classifier, we must adopt a strategy for selecting a limited but informative subset of document features (i.e., words).
• We need a way to combine the likelihoods computed by several classifiers in order to choose a single best class.

• When iterating through a collection of unlabelled candidate training documents, we must decide which ones are most in need of human verification, and which (if any) can be labelled automatically.

The rest of this plan will briefly outline the decisions we’ve made for each of these choices, provide a set of detailed use cases for each of the classification steps, and conclude with a rough plan for the changes that will need to be made to the Yioop source.

**Algorithms**

Here we describe and explain our decisions for the choices listed above. This section is intended to serve as a reference for the algorithms used rather than as a complete description of each.

**Classification** We restrict ourselves to binary classification, and to methods which others have demonstrated to be amenable to text classification: Naive Bayes, logistic regression, and support vector machines (SVMs). Of these, we reject SVMs because in previous experiments they have only provided modest improvements over logistic regression, but are an order of magnitude less efficient to train. For the remaining methods, we choose a standard Naive Bayes implementation [1] and a Bayesian logistic regression approach with a Laplace prior to avoid overfitting [2]. The latter can achieve much higher accuracy with a smaller training set, but is much slower to train.

**Feature selection** Because the logistic regression algorithm we’ve chosen to implement is relatively slow to train—even on small training sets of one to two hundred documents—it’s important to reduce the feature set as much as possible without significantly degrading accuracy. We use \( \chi^2 \) feature selection [5] to choose the features which appear to be most correlated with a positive label. Others have investigated measures for selecting what appear to be the most informative terms, but for simplicity we take a fixed percentage of the top terms.

**Multi-way classification** There may be many classifiers; they all make a binary decision, and a one-vs-rest strategy [4, p. 306] is used to pick the best match for a particular document. If no classifier can classify a document with greater than some threshold likelihood, then the document isn’t assigned to any class.

**Active training** When building a training set we want to minimize the number of documents that an administrator must label by hand. We propose to accomplish this using a Query-by-Committee (QBC) approach
with density-based pool sampling and Expectation Maximization (EM) [3]. Documents which result in greater “disagreement” between a panel of classifiers sampled from the distribution of naive Bayes model parameters are ordered first. EM is used to estimate the best labels for those documents in the corpus which the administrator hasn’t labeled by hand.

Use Cases

The following use cases cover the creation, use, and deletion of classifiers. They all describe operations at the level of the Yioop web-based configuration interface, omitting discussion of the underlying data structures and algorithms.

01 Create a new classifier

*Tom, an administrator, wants to create a new classifier so that, during the next crawl, web pages which conceptually belong to class $x$ will automatically have a ‘u:class:x’ tag added to their metadata.*

**Preconditions**

1. Tom is logged in to the Yioop administrator interface.

**On success**

1. A classifier $C$ is created with label $x$; no training corpus has been selected.
2. Tom is taken to a new interface where he can train $C$.

**On failure**

1. No classifier is created; the system is unchanged.

**Scenario**

1. Tom clicks on the ‘Classifiers’ activity tab.
2. Tom is presented with a list of existing classifiers and a text box into which he can type the name of a new one.
3. Tom types the classifier name $x$ into the text box and clicks a ‘Create’ button to create it.
4. The classifier is created, and Tom is taken to the training interface.

**Extensions**

4. A classifier named $x$ already exists. In this case, Tom would be able to select it from the list in step 3.
02 Edit an existing classifier

Tom, an administrator, wants to edit the existing classifier \( C \) in order to, for example, add documents to the training set or change the label name.

Preconditions

1. Tom is logged in to the Yioop administrator interface.
2. The classifier \( C \) exists.

On success

1. Tom is presented with the relevant details for the classifier \( C \), as well as an interface for adding examples.

Scenario

1. Tom clicks on the ‘Classifiers’ activity tab.
2. Tom is presented with a list of existing classifiers, each associated with several action links such as ‘Edit’ and ‘Delete’.
3. Tom clicks on the ‘Edit’ link associated with the classifier \( C \).

03 Start training an existing classifier

Tom, an administrator, wants to begin training a classifier \( C \) for the class \( x \) by selecting an initial set of example pages which do and do not belong to the class.

Preconditions

1. Tom has opened the classifier for editing (see use case 02).
2. There is no active crawl. [?]

On success

1. The classifier \( C \) has an initial training set containing both negative and positive examples, and could be used to classify web pages.

Scenario

1. Tom is presented with a drop-down menu from which he may select a previous crawl mix. He also sees some summary statistics for the classifier, such as the number of documents it has been trained on so far, how many positive examples there are, how many negative examples, and so on.
2. Tom selects a previous crawl mix from the drop-down menu. He clicks a check box specifying that all documents in this crawl mix are positive examples, and clicks the ‘Select’ button to load the crawl mix.
3. The classifier $C$ is updated with the positive examples, and Tom is presented with updated statistics.

4. Tom selects a different previous crawl mix, and this time specifies that all documents in the mix are negative examples.

5. The classifier $C$ is updated with the negative examples, and Tom is presented with updated statistics.

Extensions

2. Tom could choose a crawl mix containing both positive and negative examples and not select either check box. In this case nothing is automatically added to the training set, and Tom must manually pick out positive and negative examples (see use case 04).

04 Improve an existing classifier

Tom, an administrator, wants to improve the accuracy of an existing classifier $C$ by adding more labelled examples to its training set.

Preconditions

1. Tom has opened the classifier for editing (see use case 02).

2. The classifier $C$ may or may not already have been trained on some labelled examples.

On success

1. The classifier $C$ is updated with new labelled examples which may be positive, negative, or both.

Scenario

1. Tom is presented with a drop-down menu from which he may select a previous crawl mix.

2. Tom selects an existing crawl mix, but doesn’t check a box indicating that all documents in the mix are either positive or negative. He clicks the ‘Select’ button to load the chosen crawl mix. He can also specify a query which the documents in the crawl mix must satisfy.

3. He is presented with a truncated list of document summaries from the chosen crawl mix, each one classified using the current classifier. Tom can see which class has been assigned and with what confidence; he can also see the title, summary, and a link, much as he would on a normal page of search results. The results are ordered in decreasing order of “usefulness,” so that those which stand to gain the most by a human judgement come first.
4. Tom starts at the top of the list and marks each document as either a positive or negative example, confirming or reversing the classifier’s decision. He can also postpone a decision—removing a document from the list without labelling it. As he makes a choice for each document, that document is removed from the list.

5. After labelling some fixed number of documents $\ell$, the classifier $C$ is re-trained on the extended corpus, and the document list updated according to any changes in the classification of each document.

Extensions

4. Tom doesn’t have to start at the top; he can label any document in the list. He does, however, have to make a judgement or explicitly skip an item in order to see a new document. That is, he cannot seek ahead into the list.

**05 Delete an existing classifier**

*Tom, an administrator, no longer wants a classifier for the class $x$.***

**Preconditions**

1. Tom is logged in to the Yioop administrator interface.
2. A classifier for $x$ exists.
3. There is no active crawl. \[ ? \]

**On success**

1. The classifier $C$ corresponding to $x$ is deleted and no longer appears on the list of classifiers.
2. Tom is taken to the list of remaining classifiers.

**On failure**

1. The classifier is not deleted; the system is unchanged.

**Scenario**

1. Tom clicks on the ‘Classifiers’ activity tab.
2. Tom is presented with a list of existing classifiers, each associated with several action links such as ‘Edit’ and ‘Delete’.
3. Tom clicks on the ‘Delete’ link corresponding to the classifier for $x$.
4. The classifier is deleted, and Tom is presented with the new list of classifiers, now without $C$.

**Extensions**
4. There is an active crawl, so the classifier cannot be deleted. The classifier is not deleted, and Tom is presented with the list of the existing classifiers and an error message informing him that there is an active crawl.

06 Classify web pages

*Tom, an administrator, has trained a classifier $C$ for the label $x$, and now he wants to use this classifier to classify pages retrieved during the next crawl.*

**Preconditions**

1. Tom is logged in to the Yioop administrator interface.
2. Tom has created the classifier $C$ (see use case 01).
3. For the classifier to be useful, Tom should have trained it on a set of labelled positive and negative examples (see use cases 03 and 04).

**On success**

1. When the next crawl takes place, each retrieved page will be classified using the classifier $C$. If the page is determined to belong to the class $x$, then the record for that document will have the meta tag ‘u:class:$x$’ added to it.

**Scenario**

1. Tom clicks on the ‘Page Options’ activity tab.
2. He is presented with several options for how to process downloaded pages, including the number of bytes to download, the types of files to parse, and a field for writing “extraction rules.”
3. Tom scrolls down to the ‘Page Field Extraction Rules’ text area and enters “addMetaWord(class)” on a line by itself, then clicks the ‘Save’ button.
4. During the next crawl, all existing classifiers will be run on each retrieved page, and the one which results in the highest likelihood (above a threshold) will be added to the page’s meta variables (e.g. ‘u:class:$x$’).

**Open Questions**

Creating these use cases has brought to light the following questions, which we haven’t previously considered:

- Can classifiers be edited while there’s an active crawl?
- Should there be an interface for editing previous label assignments?
- As an administrator is labelling documents, when should the current classifier be trained anew on the extended training set? Should it be automatic or manual?
• Should it be possible to disable specific classifiers, so that (without deleting them) they’re not used on the next crawl?

Modifications to Yioop

Implementing the features described by the preceding use cases will require the following high-level changes to Yioop:

• Additions to createdb.php to insert rows for the new ‘Classifiers’ activity method and translation.

• Additions to controllers/admin_controller.php for the functions that will manage the high-level control-flow logic for creating, editing, and deleting classifiers.

• A new element in the views/elements/ directory for the HTML that will create the ‘Classifiers’ interface.

• A new directory in lib/ to hold the classification machinery.

• A new file in lib/indexing_plugins/ implementing the extra processing that will be done for each page to add a new ‘class’ meta word.

• Additions to configs/config.php to activate the new plugin.

References


