View Validation: A Case Study for Wrapper Induction and Text Classification

Ion Muslea, Steven Minton, and Craig A. Knoblock
University of Southern California & Fetch Technologies
4676 Admiralty Way
Marina del Rey, CA 90292, USA
muslea@isi.edu, minton@fetch.com, knoblock@isi.edu

Abstract

Wrapper induction algorithms, which use labeled examples to learn extraction rules, are a crucial component of information agents that integrate semi-structured information sources. *Multi-view* wrapper induction algorithms reduce the amount of training data by exploiting several types of rules (i.e., *views*), each of which being sufficient to extract the relevant data. All multi-view algorithms rely on the assumption that the views are *sufficiently compatible* for multi-view learning (i.e., *most* examples are labeled identically in all views). In practice, it is unclear whether or not two views are sufficiently compatible for solving a new, unseen learning task. In order to cope with this problem, we introduce a *view validation* algorithm: given a learning task, the algorithm predicts whether or not the views are sufficiently compatible for solving that particular task. We use information acquired while solving several exemplar learning tasks to train a classifier that discriminates between the tasks for which the views are *sufficiently* and *insufficiently compatible* for multi-view learning. For both wrapper induction and text classification, view validation requires only a modest amount of training data to make high accuracy predictions.

Introduction

Most mediator agents that integrate data from multiple Web-based information sources use *wrappers* to extract the relevant data from semi-structured documents. As a typical mediator integrates data from dozens or even hundreds of sources, it is crucial to minimize the amount of time and energy that users spend wrapping each source. To address this issue, researchers created *wrapper induction* algorithms such as STALKER (Muslea, Minton, & Knoblock 2001a), which learns extraction rules based on user-provided examples of items to be extracted. In previous work (Muslea, Minton, & Knoblock 2000), we introduced a *multi-view* version of STALKER, Co-Testing, that reaches higher extraction accuracy based on fewer labeled examples. Co-Testing exploits the fact that, for most items of interest, there are *alternative ways* to extract the data of interest (i.e., Co-Testing is a *multi-view* algorithm). Even though Co-Testing outperforms it's single-view counterpart on most extraction tasks,

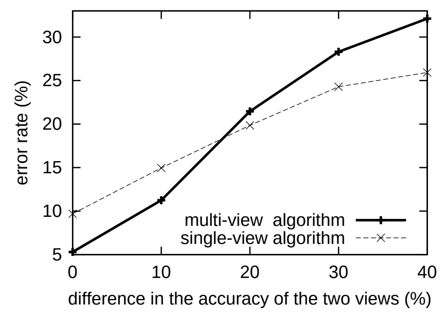

Figure 1: **As the difference in the accuracy of the two views increases, the views become more incompatible, and the single-view algorithm outperforms its multi-view counterpart.**

there are tasks on which (single-view) STALKER converges faster. In this paper we focus on the following problem: for a new, unseen task, we want to predict whether or not a multi-view algorithm will outperform its single-view counterpart. Given the practical importance of this problem, we are interested in a general purpose solution that can be applied to any multi-view problem, including wrapper induction.

In a multi-view problem, one can partition the domain's features into subsets (*views*) each of which are *sufficient* for learning the target concept. For instance, one can classify segments of televised broadcasts based *either* on the video or on the audio information; or one can classify Web pages based on the words that appear *either* in the documents or in the hyperlinks pointing to them. (Blum & Mitchell 1998) proved that by using the views to bootstrap each other, the target concept can be learned from a few labeled and many unlabeled examples. Their proof relies on the assumption that the views are *compatible* and *uncorrelated* (i.e., every example is identically labeled in each view; *and*, given the label of any example, its descriptions in each view are independent).

In real-world problems, both assumptions are often violated for a variety of reasons such as correlated or insufficient features. In a companion paper (Muslea, Minton, & Knoblock 2002), we introduced an active learning algorithm that performs well even when the independence assumption is violated. We focus here on the view incompatibility issue, which is closely related to the accuracy of the hypotheses learned in the two views: the more accurate the views, the fewer examples can be incompatible (i.e., labeled differently in the two views). Figure 1, which is based on our results in (Muslea, Minton, & Knoblock 2001b), illustrates the relationship between the incompatibility of the views and the applicability of the multi-view algorithms: as the difference between the accuracy of the hypotheses learned in the two views increases (i.e., the views become more incompatible), the single-view algorithm outperforms its multi-view counterpart. This observation immediately raises the following question: for a new, unseen learning task, should we use a multi-view or a single-view learning algorithm?

The question above can be restated as follows: given two views and a set of learning tasks, how can one identify the tasks for which these two views are *sufficiently compatible* for multi-view learning? In order to answer this question, we introduce a *view validation algorithm* that, for a given pair of views, discriminates between the tasks for which the views are *sufficiently* and *insufficiently compatible* for multi-view learning. In other words, view validation judges the usefulness of the views for a particular learning task (i.e., *it validates the views for a task of interest*).

View validation is suitable for applications such as wrapper induction (Muslea, Minton, & Knoblock 2000) and Web page classification (Blum & Mitchell 1998), where the same views are repeatedly used to solve a variety of unrelated learning tasks. Consider, for instance, the Web page classification problem, in which the two views consist of "words that appear in Web pages" and "words in hyperlinks pointing to them". Note that, in principle, we can use these two views in learning tasks as diverse as distinguishing between homepages of *professors* and *students* or distinguishing between articles on *economics* and *terrorism*. However, for any of these learning tasks, it may happen that the text in the hyperlinks is so short and uninformative that one is better off using just the words in the Web pages. To cope with this problem, one can use view validation to predict whether or not multi-view learning is appropriate for a task of interest.

This paper presents a general, meta-learning approach to view validation. In our framework, the user provides several exemplar learning tasks that were solved using the *same views*. For each solved learning task, our algorithm generates a *view validation example* by analyzing the hypotheses learned in each view. Then it uses the `C4.5` algorithm to identify common patterns that discriminate between the learning tasks for which the views are *sufficiently* and *insufficiently compatible* for multi-view learning. An illustrative example of such a pattern is the following: "**IF** for a task T the difference in the training errors in the two views is larger than 20% *and* the views agree on less than 45% of the unlabeled examples **THEN** the views are *insufficiently compatible* for applying multi-view learning to T". We consider two application domains: wrapper induction and text classification. On both domains, the view validation algorithm makes high accuracy predictions based on a modest amount of training data.

View validation represents a first step towards our long-term goal of automatic *view detection*, which would dramatically widen the practical applicability of multi-view algorithms. Instead of having to rely on user-provided views, one can use view detection to search for adequate views among the possible partitions of the domain's features. In this context, a view validation algorithm becomes a key component that verifies whether or not the views that are generated during view detection are sufficiently compatible for applying multi-view learning to a learning task.

`Given`:
- a learning task with two views V_1 and V_2
- a learning algorithm $\mathcal{L}$
- the sets T and U of labeled and unlabeled examples

`LOOP` for k iterations
- use $\mathcal{L}$, $V_1(T)$, and $V_2(T)$ to learn classifiers h_1 and h_2
- FOR EACH class C_i DO
  - let E_1 and E_2 be the e unlabeled examples on which h_1 and h_2 make the most confident predictions for C_i
  - remove E_1 and E_2 from U, label them according to h_1 and h_2,respectively, and add them to T
- combine the prediction of h_1 and h_2

Figure 2: **The Co-Training algorithm.**

Background

The *multi-view setting* (Blum & Mitchell 1998) applies to learning tasks that have a natural way to partition their features into subsets (*views*) each of which are *sufficient* to learn the target concept. In such tasks, an example x is described by a different set of features in each view. For example, in a domain with two views V_1 and V_2, any example x can be seen as a triple $[x_1, x_2, l]$, where x_1 and x_2 are its descriptions in the two views, and l is its label.

Blum and Mitchell (1998) proved that by using two views to bootstrap each other, a target concept can be learned from a few labeled and many unlabeled examples, provided that the views are *compatible* and *uncorrelated*. The former requires that all examples are labeled identically by the target concepts in each view. The latter means that for any example $[x_1, x_2, l]$, x_1 and x_2 are independent given l.

The proof above is based on the following argument: one can learn a weak hypothesis h_1 in V_1 based on the few labeled examples and then apply h_1 to all unlabeled examples. If the views are uncorrelated, these newly labeled examples are seen in V_2 as a random training set with classification noise, based on which one can learn the target concept in V_2. An updated version of (Blum & Mitchell 1998) shows that the proof also hold for some (low) level of view incompatibility, provided that the views are uncorrelated.

However, as shown in (Muslea, Minton, & Knoblock 2001b), in practice one cannot ignore view incompatibility because one rarely, if ever, encounters real world domains with uncorrelated views. Intuitively, view incompatibility affects multi-view learning in a straightforward manner: if the views are incompatible, the target concepts in the two views *label differently* a large number of examples. Consequently, from V_2's perspective, h_1 may "misslabel" so many examples that learning the target concept in V_2 becomes impossible.

To illustrate how view incompatibility affects an actual multi-view algorithm, let us consider Co-Training (Blum &

Given:
- a multi-view problem P with views V_1 and V_2
- a learning algorithm $\mathcal{L}$
- a set of pairs { $\langle I_1, L_1 \rangle$, $\langle I_2, L_2 \rangle$, $\ldots$, $\langle I_n, L_n \rangle$ }, where I_k are instances of P, and L_k labels I_k as having or not views that are *sufficiently compatible* for multi-view learning

FOR each instance I_k DO
- let T_k and U_k be labeled and unlabeled examples in I_k
- use $\mathcal{L}$, $V_1(T_k)$, and $V_2(T_k)$ to learn classifiers h_1 and h_2
- $CreateViewValidationExample(h_1, h_2, T_k, U_k, L_k)$

- train `C4.5` on the view validation examples
- use the learned classifier to discriminates between problem instances for which the views are *sufficiently* and *insufficiently compatible* for multi-view learning

Figure 3: **The View Validation Algorithm.**

Mitchell 1998), which is a semi-supervised, multi-view algorithm.[1] Co-Training uses a small set of labeled examples to learn a (weak) classifier in the two views. Then each classifier is applied to all unlabeled examples, and Co-Training detects the examples on which each classifier makes the most confident predictions. These high-confidence examples are labeled with the estimated class labels and added to the training set (see Figure 2). Based on the updated training set, a new classifier is learned in each view, and the process is repeated for several iterations.

When Co-Training is applied to learning tasks with compatible views, the information exchanged between the views (i.e., the high-confidence examples) is beneficial for both views because most of the examples have the same label in each view. Consequently, after each iteration, one can expect an increase in the accuracy of the hypotheses learned in each view. In contrast, Co-Training has a poor performance on domains with incompatible views: as the difference between the accuracy of the two views increases, the low-accuracy view feeds the other view with a larger amount of mislabeled training data.

The View Validation Algorithm

Before introducing view validation, let us briefly present the terminology used in this paper. By definition, a *multi-view problem* is a collection of learning tasks that use the same views; each such learning task is called an *instance of the multi-view problem* or a *problem instance*. For example, consider a multi-view problem P_1 that consists of all Web page classification tasks in which the views are "words in Web pages" and "words in hyperlinks pointing to the pages". We can use these two views to learn a classifier that distinguishes between homepages of *professors* and *students*, another classifier that distinguishes between articles on *gun control* and *terrorism*, and so forth. Consequently, all these learning tasks represent instances of the same problem P_1.

Note that, in practice, we cannot expect a pair of views to be sufficiently compatible for all learning tasks. For instance, in the problem P_1 above, one may encounter tasks in which the text in the hyperlinks is too short and uninformative for the text classification task. More generally, because of corrupted or insufficient features, it is unrealistic to expect the views to be *sufficiently compatible* for applying multi-view learning to all problem instances. In order to cope with this problem, we introduce a *view validation* algorithm: for any problem instance, our algorithm predicts whether or not the views are sufficiently compatible for using multi-view learning for that particular task.

In practice, the level of "acceptable" view incompatibility depends on both the domain features and the algorithm $\mathcal{L}$ that is used to learn the hypotheses in each view. Consequently, in our approach, we apply view validation to a given multi-view problem (i.e., pair of views) and learning algorithm $\mathcal{L}$. Note that this is a natural scenario for multi-view problems such as wrapper induction and text classification, in which the same views are used for a wide variety of learning tasks.

Our view validation algorithm (see Figure 3) implements a three-step process. First, the user provides several pairs $\langle I_k, L_k \rangle$, where I_k is a problem instance, and L_k is a label that specifies whether or not the views are sufficiently compatible for using multi-view learning to solve I_k. The label L_k is generated automatically by comparing the accuracy of a single- and multi-view algorithm on a test set. Second, for each instance I_k, we generate a *view validation example* (i.e., a feature-vector) that describes the properties of the hypotheses learned in the two views. Finally, we apply `C4.5` to the view validation examples; we use the learned decision tree to discriminate between learning tasks for which the views are sufficiently or insufficiently compatible for multi-view learning,

In keeping with the multi-view setting, we assume that for each instance I_k the user provides a (small) set T_k of labeled examples and a (large) set U_k of unlabeled examples. For each instance I_k, we use the labeled examples in T_k to learn a hypothesis in each view (i.e., h_1 and h_2). Then we generate a *view validation example* that is labeled L_k and consists of a feature-vector that describes the hypotheses h_1 and h_2. In the next section, we present the actual features used for view validation.

Features Used for View Validation

Ideally, besides the label L_k, a *view validation example* would consist of a single feature: the percentage of examples that are labeled differently in the two views. Based on this unique feature, one could learn a *threshold value* that discriminates between the problem instances for which the views are sufficiently/insufficiently compatible for multi-view learning. In Figure 1, this threshold corresponds to the point in which the two learning curves intersect. In practice, using this unique feature requires knowing the labels of *all* examples in a domain. As this is an unrealistic scenario, we have chosen instead to use several features that are potential indicators of the how incompatible the views are.

In this paper, each view validation example is described by the following seven features:

[1] View incompatibility affects in a similar manner other semi-supervised, multi-view algorithms such as Co-EM (Nigam & Ghani 2000) or Co-Boost(Collins & Singer 1999).

- f_1: the percentage of unlabeled examples in U_k that are classified identically by h_1 and h_2;
- f_2: $min(TrainingErrors(h_1), TrainingErrors(h_2))$;
- f_3: $max(TrainingErrors(h_1), TrainingErrors(h_2))$;
- f_4: $f_3 - f_2$;
- f_5: $min(Complexity(h_1), Complexity(h_2))$;
- f_6: $max(Complexity(h_1), Complexity(h_2))$;
- f_7: $f_6 - f_5$.

Note that features f_1-f_4 are measured in a straightforward manner, regardless of the algorithm $\mathcal{L}$ used to learn h_1 and h_2. By contrast, features f_5-f_7 dependent on the representation used to describe these two hypotheses. For instance, the complexity of a boolean formula may be expressed in terms of the number of disjuncts and literals in the disjunctive or conjunctive normal form; or, for a decision tree, the complexity measure may take into account the depth and the breadth (i.e., number of leaves) of the tree.

The intuition behind the seven view validation features is the following:

- the fewer unlabeled examples from U_k are labeled identically by h_1 and h_2, the larger the number of potentially incompatible examples;
- the larger the difference in the training error of h_1 and h_2, the less likely it is that the views are equally accurate;
- the larger the difference in the complexity of h_1 and h_2, the likelier it is that the most complex of the two hypotheses overfits the (small) training set T_k. In turn, this may indicate that the corresponding view is significantly less accurate than the other one.

In practice, features f_1-f_4 are measured in a straightforward manner; consequently, they can be always used in the view validation process. In contrast, measuring the complexity of a hypothesis may not be always possible or meaningful (consider, for instance, the case of a Naive Bayes or a k nearest-neighbor classifier, respectively). In such situations, one can simply ignore features f_5-f_7 and rely on the remaining features.

The Test Problems for View Validation

We describe now the two problems that we use as case studies for view validation. First we present the *wrapper induction* problem, which consists of a collection 33 information extraction tasks that originally motivated this work. Then we describe a family of 60 parameterized text classification tasks (for short, PTCT) that we used in (Muslea, Minton, & Knoblock 2001b) to study the influence of view incompatibility and correlation on multi-view learning algorithms.

Multi-View Wrapper Induction

To introduce our approach to wrapper induction (Muslea, Minton, & Knoblock 2000), let us consider the illustrative task of extracting phone numbers from documents similar to the Web-page fragment in Figure 4. In our framework, an *extraction rule* consists of a *start rule* and an *end rule* that identify the beginning and the end of the item, respectively; given that start and end rules are extremely similar, we describe here only the former. For instance, in order to find the beginning of phone number, we can use the start rule

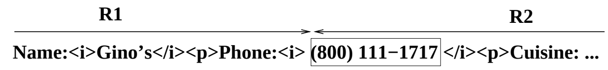

Figure 4: **Extracting the phone number.**

R1 = $SkipTo$**(** `Phone:<i>` **)**.

This rule is applied *forward*, from the beginning of the page, and it ignores everything until it finds the string `Phone:<i>`. For a slightly more complicated extraction task, in which only the toll-free numbers appear in italic, one can use a disjunctive start rule such as

R1′ = EITHER $SkipTo$**(** `Phone:<i>` **)**
OR $SkipTo$**(** `Phone:` **)**

An alternative way to detect the beginning of the phone number is to use the start rule

R2 = $BackTo$**(** `Cuisine` **)** $BackTo$**(** ($Number$) **)**

which is applied *backward*, from the end of the document. **R2** ignores everything until it finds "`Cuisine`" and then, again, skips to the first number between parentheses.

As described in (Muslea, Minton, & Knoblock 2001a), rules such as **R1** and **R2** can be learned based on user-provided examples of items to be extracted. Note that **R1** and **R2** represent descriptions of the same concept (i.e., start of phone number) that are learned in two different views. That is, the views V_1 and V_2 consist of the sequences of characters that *precede* and *follow* the beginning of the item, respectively.

For wrapper induction, the view validation features are measured as follows: f_1 represents that percentage of (unlabeled) documents from which the two extraction rules extract the same string; for f_2-f_4, we count the labeled documents from which the extraction rules do not extract the correct string. Finally, to measure f_5-f_7, we define the complexity of an extraction rule as the maximum number of disjuncts that appear in either the start or the end rule.

Multi-View Text Classification

As a second case study, we use the PTCT family of parameterized text categorization tasks described in (Muslea, Minton, & Knoblock 2001b).[2] PTCT contains 60 text classification tasks that are evenly distributed over five levels of view incompatibility: 0%, 10%, 20%, 30%, or 40% of the examples in a problem instance are made incompatible by corrupting the corresponding percentage of labels in one of the views.

PTCT is a text classification domain in which one must predict whether or not various newsgroups postings are of interest for a particular user. In PTCT, a multi-view example's description in each view consists a document from the `20-Newsgroups` dataset (Joachims 1996). Consequently, we use the Naive Bayes algorithm (Nigam & Ghani 2000)

[2] We would have preferred to use a real-world multi-view problem instead of PTCT. Unfortunately, given that multi-view learning represents a relatively new field of study, most multi-view algorithms were applied to just a couple problem instances.

to learn the hypotheses in the two views. As there is no obvious way to measure the complexity of a Naive Bayes classifier, for PTCT we do not use the features f_5-f_7. The other features are measured in a straightforward manner: f_1 represents the percentage of unlabeled examples on which the two Naive Bayes classifiers agree, while f_2-f_4 are obtained by counting the training errors in the two views.

Empirical Results

Generating the WI and PTCT Datasets

To label the 33 problem instances for wrapper induction (WI), we compare the single-view STALKER algorithm (Muslea, Minton, & Knoblock 2001a) with its multi-view version described in (Muslea, Minton, & Knoblock 2000). On the six extraction tasks in which the difference in the accuracy of the rules learned in the two views is larger than 10%, single-view STALKER does at least as well as its multi-view counterpart. We label these six problem instances as having views that are insufficiently compatible for multi-view learning.

In order to label the 60 instances in PTCT, we compare single-view, semi-supervised EM with Co-Training, which is the most widely used semi-supervised multi-view algorithm (Collins & Singer 1999) (Pierce & Cardie 2001) (Sarkar 2001). We use the empirical results from (Muslea, Minton, & Knoblock 2001b) to identify the instances on which semi-supervised EM performs at least as well as Co-Training. We label the 40 such instances as having views that are insufficiently compatible for multi-view learning.

For both WI and PTCT, we have chosen the number of examples in T_k (i.e., $Size(T_k)$) according to the experimental setups described in (Muslea, Minton, & Knoblock 2001a) and (Muslea, Minton, & Knoblock 2001b), in which WI and PTCT were introduced. For WI, in which an instance I_k may have between 91 and 690 examples, $Size(T_k)$=6 and U_k consists of the remaining examples. For PTCT, where each instance consists of 800 examples, the size of T_k and U_k is 70 and 730, respectively.

The Setup

In contrast to the approach described in Figure 3, where a *single* view validation example is generated per problem instance, in our experiments we create *several* view validation examples per instance. That is, for each instance I_k, we generate $ExsPerInst = 20$ view validation examples by repeatedly partitioning the examples in I_k into randomly chosen sets T_k and U_k of the appropriate sizes. The motivation for this decision is two-fold. First, the empirical results should not reflect a particularly (un)fortunate choice of the sets T_k and U_k. Second, if we generate a single view validation example per instance, for both WI and PTCT we obtain a number of view validation examples that is too small for a rigorous empirical evaluation (i.e., 33 and 60, respectively). To conclude, by generating $ExsPerInst = 20$ view validation examples per problem instance, we obtain larger number of view validation examples (660 and 1200, respectively) that, for each problem instance I_k, are representative for a wide variety of possible sets T_k and U_k.

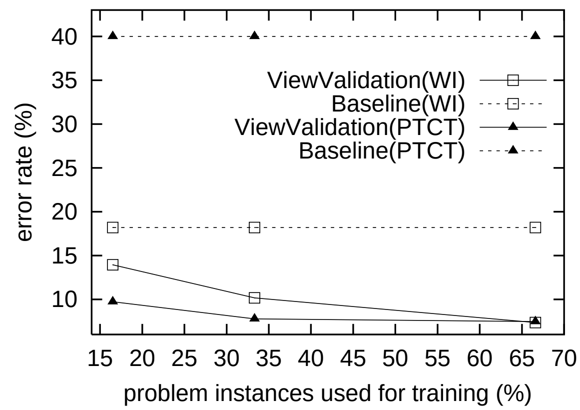

Figure 5: **View validation clearly outperforms a baseline algorithm that predicts the most frequent label.**

To evaluate view validation's performance, for both WI and PTCT, we partition the problem instances into *training* and *test instances*. For each such partition, we create the *training* and *test sets* for `C4.5` as follows: all $ExsPerInst = 20$ view validation examples that were created for a *training instance* are used in the `C4.5` *training set*; similarly, all 20 view validation examples that were created for a *test instance* are used in the `C4.5` *test set*. In other words, all view validation examples that are created based on the same problem instance belong either to the training set or to the test set, and they cannot be split between the two sets. In our experiments, we train on $\frac{1}{6}$, $\frac{1}{3}$, and $\frac{2}{3}$ of the instances and test on the remaining ones. For each of these three ratios, we average the error rates obtained over $N = 20$ random partitions of the instances into training and test instances.

Figure 5 shows the view validation results for the WI and PTCT datasets. The empirical results are excellent: when trained on 66% of the available instances, the view validation algorithm reaches an accuracy of 92% on both the WI and PTCT datasets. Furthermore, even when trained on just 33% of the instances (i.e., 11 and 20 instances for WI and PTCT, respectively), we still obtain a 90% accuracy. Last but not least, for both WI and PTCT, view validation clearly outperforms a baseline algorithm that simply predicts the most frequent label in the corresponding dataset.

The Influence of $ExsPerInst$ and $Size(T_k)$

The results in Figure 5 raise an interesting practical question: how much can we reduce the user's effort without harming the performance of view validation? In other words, can we label only a fraction of the $ExsPerInst$ view validation examples per problem instance and a subset of T_k, and still obtain a high-accuracy prediction? To answer this question, we designed two additional experiments in which we vary one of the parameters at the time.

To study the influence of the $ExsPerInst$ parameter, we keep $Size(T_k)$ constant (i.e., 6 and 70 for WI and PTCT, respectively), and we consider the values $ExsPerInst = 1, 5, 10, 20$. That is, rather than including all 20 view validation examples that we generate for each instance I_k, the

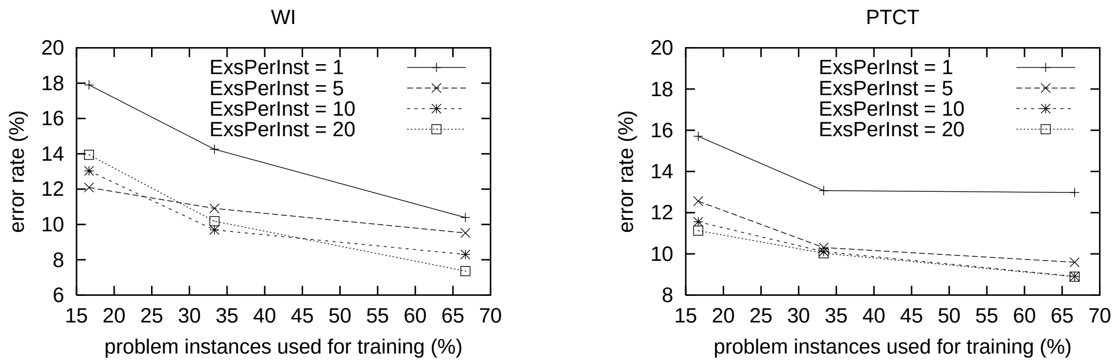

Figure 6: **We keep $Size(T_k)$ constant and vary the value of $ExsPerInst$ (1, 5, 10, and 20).**

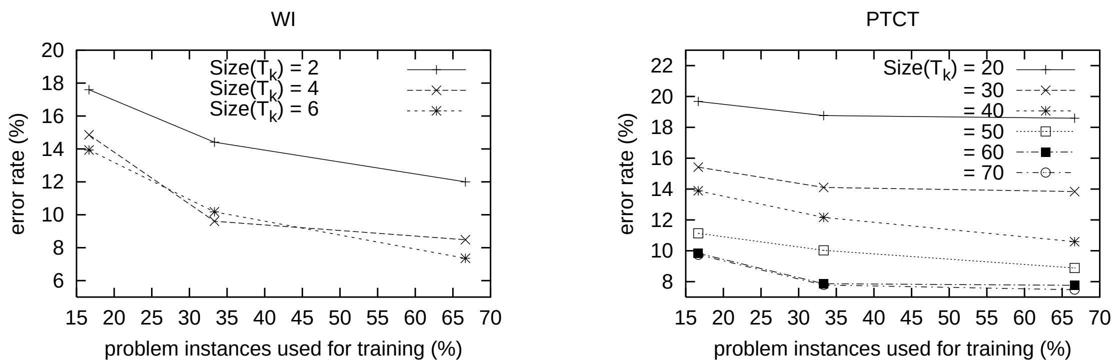

Figure 7: **For $ExsPerInt = 20$, we consider several values for $Size(T_k)$: 2/4/6 for WI, and 20/30/40/50/60/70 for PTCT.**

C4.5 training sets consist of (randomly chosen) subsets of one, five, 10, or 20 view validation examples for each training instance. Within the corresponding C4.5 test sets, we continue to use all 20 view validation examples that are available for each test instance.

Figure 6 displays the learning curves obtained in this experiment. The empirical results suggest that the benefits of increasing $ExsPerInst$ become quickly insignificant: for both WI and PTCT, the difference between the learning curves corresponding to $ExsPerInst = 10$ and 20 is not statistically significant, even though for the latter we use twice as many view validation examples than for the former. This implies that a (relatively) small number of view validation examples is sufficient for high-accuracy view validation. For example, our view validation algorithm reaches a 90% accuracy when trained on 33% of the problem instances (i.e., 11 and 20 training instances, for WI and PTCT, respectively). For $ExsPerInst = 10$, this means that C4.5 is trained on just 110 and 200 view validation examples, respectively.

In order to study the influence of the $Size(T_k)$ parameter, we designed an experiment in which the hypotheses h_1 and h_2 are learned based on a fraction of the examples in the original set T_k. Specifically, for WI we use two, four, and six of the examples in T_k; for PTCT we use 20, 30, 40, 50, 60, and 70 of the examples in T_k. For both WI and PTCT, we keep $ExsPerInst = 20$ constant.

Figure 7 shows the learning curves obtained in this experiment. Again, the results are extremely encouraging: for both WI and PTCT we reach an accuracy of 92% without using all examples in T_k. For example, the difference between $Size(T_k) = 4$ and 6 (for WI) or $Size(T_k) = 60$ and 70 (for PTCT) are *not* statistically significant.

The experiments above suggest two main conclusions. First, for both WI and PTCT, the view validation algorithm makes high accuracy predictions. Second, our approach requires a modest effort from the user's part because both the number of view validation examples and the size of the training sets T_k are reasonably small.

The distribution of the errors

In order to study the errors made by the view validation algorithm, we designed an additional experiment. For both WI and PTCT, we use for training all-but-one of the problem instances, and we test the learned decision tree on the remaining instance.[3] This setup allows us to study view validation's performance on each individual problem instance.

The graphs in Figure 8 display the results on the WI and PTCT datasets, respectively. On the x axis, we show the number of view validation examples that are misclassified

[3]For each problem instance we use the entire training set T_k and all $ExsPerInst = 20$ view validation examples.

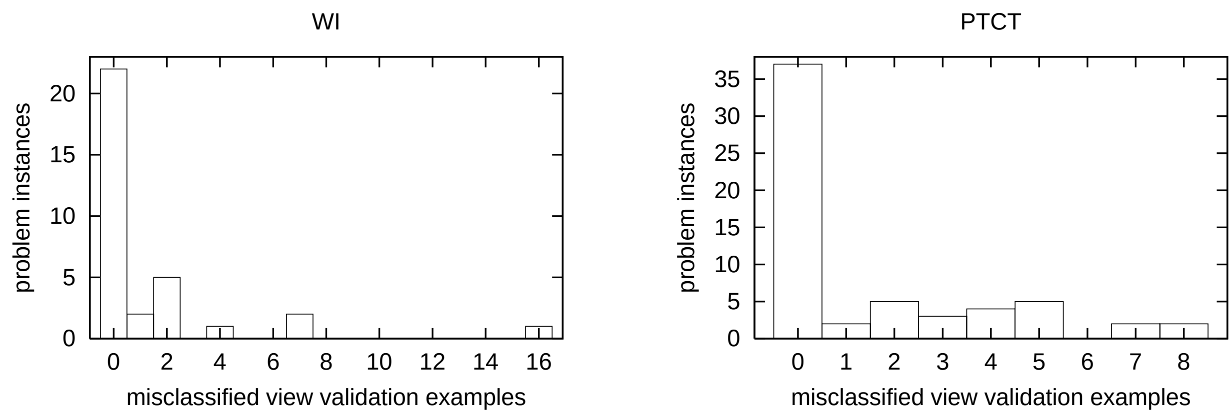

Figure 8: **The distribution of the errors for WI (left) and PTCT (right).**

by view validation (remember that each test set consists of the $ExsPerInst = 20$ view validation examples generated for the problem instance used for testing). On the y axis we have the number of problem instances on which our algorithm misclassifies a particular number view validation examples.

Consider, for example, the graph that shows the results on the 33 problem instances in WI (see Figure 8). The leftmost bar in the graph has the following meaning: on 22 of the problem instances, our algorithm makes *zero* errors on the view validation examples in the corresponding 22 test sets; that is, view validation correctly predicts the labels of all $ExsPerInst = 20$ examples in each test set. Similarly, the second bar in the graph means that on two other problem instances, view validation misclassifies just *one* of the 20 examples in the test set.

These results require a few comments. First, for more than half of the problem instances in both WI and PTCT, our algorithm labels correctly *all* view validation examples; i.e., regardless of the particular choice of the sets T_k and U_k that are used to generate a view validation example, our algorithm predicts the correct label. Second, for most instances of WI and PTCT (29 and 44 of the 33 and 60 instances, respectively), view validation has an accuracy of at least 90% (i.e., it misclassifies at most two of the $ExsPerInst = 20$ view validation examples). Last but not least, for all but one problem instance, our algorithm labels correctly *at least* 60% of the view validation examples generated for each problem instance.

Conclusions and Future Work

In this paper we introduce the first approach to view validation. We use several solved problem instances to train a classifier that discriminates between problem instances for which the views are *sufficiently* and *insufficiently compatible* for multi-view learning. For both wrapper induction and text classification, view validation requires a modest amount of training data to make high-accuracy predictions. In the short term, we plan to apply the view validation to new domains and to investigate additional view validation features. Our long-term goal is to create a *view detection* algorithm that partitions the domain's features in views that are adequate for multi-view learning.

References

Blum, A., and Mitchell, T. 1998. Combining labeled and unlabeled data with co-training. In *Proc. of the Conference on Computational Learning Theory*, 92–100.

Collins, M., and Singer, Y. 1999. Unsupervised models for named entity classification. In *Proceedings of Empirical Methods in NLP and Very Large Corpora*, 100–110.

Joachims, T. 1996. A probabilistic analysis of the ROCCHIO algorithm with TFIDF for text categorization. In *Computer Science Tech. Report CMU-CS-96-118*.

Muslea, I.; Minton, S.; and Knoblock, C. 2000. Selective sampling with redundant views. In *Proc. of National Conference on Artificial Intelligence*, 621–626.

Muslea, I.; Minton, S.; and Knoblock, C. 2001a. Hierarchical wrapper induction for semistructured sources. *J. Autonomous Agents & Multi-Agent Systems* 4:93–114.

Muslea, I.; Minton, S.; and Knoblock, C. 2001b. Selective sampling + semi-supervised learning = robust multi-view learning. In *IJCAI-2001 Workshop on Text Learning: Beyond Supervision*.

Muslea, I.; Minton, S.; and Knoblock, C. 2002. Active + semi-supervised learning = robust multi-view learning. *Submitted at ICML-2002*.

Nigam, K., and Ghani, R. 2000. Analyzing the effectiveness and applicability of co-training. In *Proc. of Information and Knowledge Management*, 86–93.

Pierce, D., and Cardie, C. 2001. Limitations of co-training for natural language learning from large datasets. In *Proc. of Empirical Methods in NLP*, 1–10.

Sarkar, A. 2001. Applying co-training methods to statistical parsing. In *Proc. of NAACL 2001*, 175–182.