Error Prediction With Partial Feedback

William Darling, Cédric Archambeau,
Shachar Mirkin, and Guillaume Bouchard

Xerox Research Centre Europe
Meylan, France
firstname.lastname@xrce.xerox.com

Abstract. In this paper, we propose a probabilistic framework for predicting the root causes of errors in data processing pipelines made up of several components when we only have access to partial feedback; that is, we are aware when *some* error has occurred in one or more of the components, but we do not know which one. The proposed error model enables us to direct the user feedback to the correct components in the pipeline to either automatically correct errors as they occur, retrain the component with assimilated training examples, or take other corrective action. We present the model and describe an Expectation Maximization (EM)-based algorithm to learn the model parameters and predict the error configuration. We demonstrate the accuracy and usefulness of our method first on synthetic data, and then on two distinct tasks: error correction in a 2-component opinion summarization system, and phrase error detection in statistical machine translation.

Keywords: error modeling, user feedback, binary classification, EM

1 Introduction and Motivation

In this work we are interested in predicting the root cause of errors for data that have been processed through a pipeline of components when we only have access to partial feedback. That is, an input X goes through a series of components that ultimately results in an output Y. Each component in the processing pipeline performs some action on X, and each of the components might result in an error. However, the user often only has access to the final output, and so it is unclear which of the components was at fault when an error is observed in the final output. In cases where the user is aware of the intermediate results, it is also typically more complex to have to specify the exact component that was at fault when providing error feedback. Therefore, given only the fact that an error has occurred or not, we would like to predict the root causes of the error.

Pipeline processing, or Pipes and Filters, has been a stalwart of computing since the concept was invented and integrated into the UNIX operating system in the 1970's [1]. The simple idea is that complex processing and powerful results can be achieved by running an input through a series of more basic components to in turn produce a more intricate output than would have been possible with

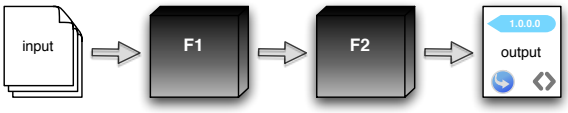

Fig. 1. Typical data processing pipeline with two components resulting in a marked-up output. The components are often black boxes that the user is unaware of which can render providing user feedback complicated.

a single method. This approach has seen increasing use in recent years as the outputs desired by users have become more complex. It is seen especially in Natural Language Processing (NLP) applications such as named entity recognition [2], text summarization [3], and recognizing textual entailment [4].

For example, the majority of comment or opinion summarization systems described in the literature make use of a collection of diverse techniques in a pipeline-like architecture [5, 6]. A first component might filter out spam comments and then a second could categorize the comments into aspects. This fits a typical data processing pipeline consisting of two components as is shown in Figure 1. It is also a common technique for applications such as identifying *evaluative* sentences [7] and MacCartney et al.'s three stage approach to textual inference: linguistic analysis (which consists of a pipeline itself), followed by graph alignment, ending with determining an entailment [8]. More generally, GATE provides a software architecture for building NLP pipelines [9]. Recent work has also shown that running two binary classifiers in a series can result in improved results over a more complex multi-class classification approach providing a further reason to consider problems associated with errors in processing pipelines [10].

We begin with a simple motivating toy example. Figure 2 visualizes a set of input data $\mathcal{X} = (x_1, x_2) \in \mathbb{R}^2$ (left) running through two affine transformation components in a pipeline. The first component translates the data by $\mathcal{T}_1 = x_1 + 6$, and the second component translates the output of the first component by $\mathcal{T}_2 = x_2 - 6$. However, each of the components have some region where they commit errors and an error causes the translation to be distorted by scaling it by uniform noise. Since the user only observes the final output (right), there are two classes of partial feedback: $f_i = 0$ means that for point $\mathcal{X}_i$, the data transformations were successful and no errors occurred; $f_i = 1$, on the other hand, means that there was some error, though it is not clear whether it was the result of the first component committing an error, the second component committing an error, or both.

The user is generally able to identify an overall error much more efficiently than having to specify its source. With K components, full feedback would require selecting from $2^K - 1$ distinct configurations of error. In many cases, it will be impossible to identify the source of error. In a Named Entity Recognition pipeline that consists first of a part-of-speech (POS) tagging component,

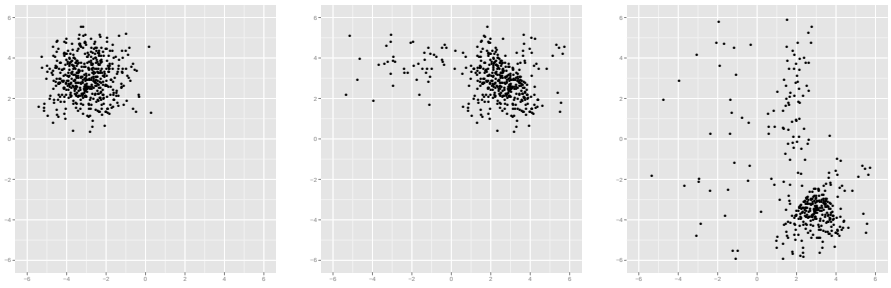

Fig. 2. Input data $\mathcal{X} = (x_1, x_2) \in \mathbb{R}^2$ (left) is first translated by $\mathcal{T}_1$ (centre) and then $\mathcal{T}_2$ (right). When either translation component commits an error, the translation fails and instead results in a translation distorted by uniform random error.

followed by a chunker, followed by a named entity classifier, an incorrectly identified named entity can easily be spotted, but it may very well be impossible to identify if it was the POS tagger, the chunker, or the classifier that was the root cause of the error. In Figure 3, the incorrectly translated output datapoints have been identified as red circles and the error-free datapoints as blue squares (right). We also show the same colour-coding on the input and intermediate results (left) and (centre). In these latter plots, we have also plotted the component-specific linear error predictor. Knowing this relationship, we can then predict the prior probability of a component committing an error given some input, and the posterior probability of error configuration given that an error has been observed. One could then take corrective measure by directing training data to the component at fault, or automatically attempting to rectify the error through a component wrapper.

In this paper, we propose a probabilistic framework that aims to uncover the predictors of error for each of the arbitrary number of components in a data processing pipeline, and predict the configuration of error for each data observation. After discussing some related work, we present our probabilistic model that is based on binary classification of error through logistic regression. We then present an Expectation Maximization (EM)-based algorithm to learn component-specific error model parameters and to estimate the configuration of error. We demonstrate the accuracy of our approach first on synthetic data, and then on two real-world tasks: a two-component opinion summarization pipeline and a phrase error prediction task for post-editing in machine translation. We conclude with some discussion and thoughts on future work.

2 Related Work

While our probabilistic error model has some connections to sigmoid belief networks (SBN) [11], the most closely related work with respect to improving performance in pipelines when errors occur comes from the NLP domain. In [12],

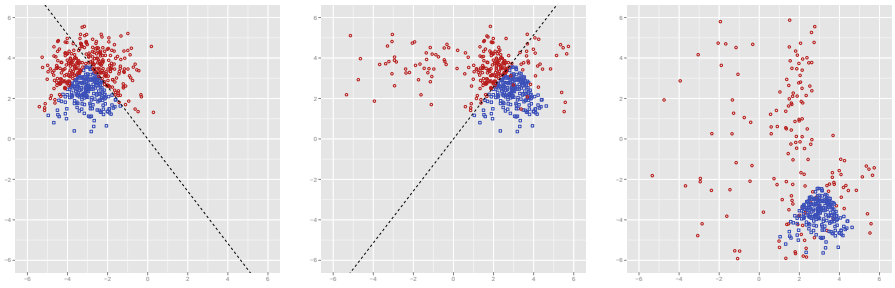

Fig. 3. Colour-coded data $\mathcal{X}$ where blue represents the feedback $f = 0$ (no error) and red represents the partial feedback $f = 1$ (error in one or more of the components). The linear predictors of error for $\mathcal{T}_1$ (left) and $\mathcal{T}_2$ (centre) are also plotted.

Marciniak and Strube explain how NLP problems can generally be cast as a set of several classification tasks, some of which are mutually related. A discrete optimization model is presented that is shown to deliver higher accuracy in a language generation task than the equivalent task implemented as solving classification tasks sequentially. However, they do not address either a general approach to improve the accuracy of each of the classifiers, nor do they consider how user feedback might be taken into consideration.

Finkel et al. show that modeling a pipeline as a Bayesian network where each component is seen as a random variable and then performing approximate inference to determine the best output can outperform a greedy pipeline architecture where a best decision is made at each node [4]. While a general method to solve any kind of multi-stage algorithm is proposed, a principal requirement is that each component must be able to generate samples from a posterior. The authors note:

> If ... all NLP researchers wrote packages which can generate samples from the posterior, then the entire NLP community could use this method as easily as they can use the greedy methods that are common today... [4]

Our proposed method is also a Bayesian network [13]. However, its aim is to predict the root causes of errors in a pipeline and it requires no changes to any of the underlying methods. Depending on the composition of the underlying components, knowing the cause of an output error could allow us to dynamically correct it by asking for a training label in an active learning setting, or flipping the erroneous prediction if the component consists of a binary classifier.

3 Probabilistic Model

For each component n in a pipeline processing system, we model the probability that it will commit an error e_n as a Bernoulli random variable modeled using

binary logistic regression: $p(e_n = 1|x, \beta) = \sigma(\phi_n(x)^\top \beta)$ where $\sigma(\cdot)$ is the logistic function and $\phi_n(\cdot)$ is a function that extracts the features required for component n. In this setting we address the case where the system only has access to *partial* feedback; that is, the only error observation, f, is with respect to the aggregate error. In this case, a user provides feedback only pertaining to whether *some* error occurred at an indeterminate set of components ($f = 1$), or that the output contains no errors at all ($f = 0$).

We let $\boldsymbol{e} = (e_1, \ldots, e_N)$ be the collection of error random variables for each component, such that

$$p(f, \boldsymbol{e}|x, \beta) = p(\boldsymbol{e}|x, \beta)p(f|\boldsymbol{e}), \tag{1}$$

where the first term $p(\boldsymbol{e}|x, \beta)$ contains the probability of a given error configuration $\boldsymbol{e}$, and the second term $p(f|\boldsymbol{e})$ encodes how the user feedback f relates to the error configuration. In the general case of the former, we have

$$p(\boldsymbol{e}|x, \beta) = \prod_{i=1}^{N} p(e_i|x, \beta) \tag{2}$$

For the latter term in the standard case where 1 or more errors committed in the components leads to an observed final error $f = 1$ we have $p(f = 1|\boldsymbol{e}) = \delta(\sum_i e_i)$ and $p(f = 0|\boldsymbol{e}) = 1 - p(f = 1|\boldsymbol{e}) = 1 - \delta(\sum_i e_i)$ where $\delta(z) = 1$ if $z > 0$ and $\delta(z) = 0$ otherwise. Note that this term could be modeled more intricately by allowing a user to specify a *degree* of error or by leading the model in the general direction of error(s) without having to explicitly report them.

All errors are assumed to be conditionally independent given the features x: $p(\boldsymbol{e}|x, \beta) = p(e_1, \ldots, e_N|x, \beta) = \prod_{i=1}^{N} p(e_i|x, \beta)$, and the posterior probabilities of error are then given by

$$p(\boldsymbol{e}|f, x, \beta) = \begin{cases} \delta_{\{e_1=0,\ldots,e_n=0\}} & \text{if } f = 0, \\ \frac{\prod_{i=1}^{N} \sigma((2e_i-1)\phi_i(x)^\top \beta)}{1-\prod_{i=1}^{N} \sigma(-\phi_i(x)^\top \beta).} & \text{if } f = 1. \end{cases} \tag{3}$$

A graphical model depiction of the error model framework is shown in Figure 4.

4 Parameter Estimation

We can learn the component-specific error model parameters β by maximizing the likelihood which is obtained by integrating out the latent error variables e_i. The likelihood and its derivative can be computed in closed form and the parameters then optimized using gradient descent [14]. However, as the number of components grows, the terms in the gradient and the likelihood grow unwieldy. For simplicity, we therefore decompose the error estimation and parameter learning by turning to a stochastic EM-based approach [15].

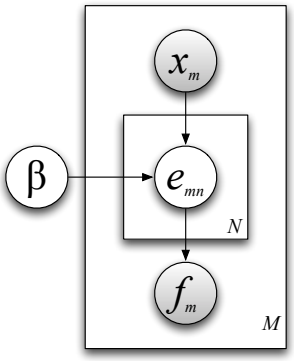

Fig. 4. Graphical model of error prediction framework. There are M observations and N components in the pipeline; β is a vector parameter of the component error models.

Where there are M observations and N components, the log likelihood is:

$$\begin{aligned}\ln \mathcal{L} = \ell(\beta) &= \sum_{m=1}^{M} \ln \sum_{e_1} \cdots \sum_{e_N} p(f_m, e_1, \ldots, e_N | x_m, \beta) \\ &= \sum_{m=1}^{M} \ln \sum_{\boldsymbol{e}} p(f_m, \boldsymbol{e} | x_m, \beta) \end{aligned} \tag{4}$$

which includes the log of a sum. By the Jensen inequality, however,

$$\begin{aligned}\ell(\beta) &= \sum_{m=1}^{M} \ln \sum_{e \in \boldsymbol{e}} p(f_m, e | x_m, \beta) \\ &\geq \sum_{m=1}^{M} \sum_{e \in \boldsymbol{e}} w_{m,e} \ln p(f_m, e | x_m, \beta) + \mathcal{H}(\mathbf{w_m}) \\ &= g(\mathbf{w}, \beta) \end{aligned} \tag{5}$$

where $\mathbf{w_m}$ contains a non-negative weight for each configuration of error (size $2^N - 1$), $\sum_{e \in \{\boldsymbol{e} \setminus e_{0\ldots0}\}} w_{m,e} = 1$ and $\forall e, w_{m,e} \geq 0$. $g(\mathbf{w}, \beta)$ is then a lower bound for the log likelihood.

Because $g(\mathbf{w}, \beta)$ is a lower bound for the log likelihood, maximizing $g(\mathbf{w}, \beta)$ will also maximize $\ell(\beta)$. However, we now have the latent parameters $\mathbf{w}$ so we iteratively maximize $\mathbf{w}$ (E-step) and β (M-step).

E-step Where $e \in \boldsymbol{e}$ is one of the $2^N - 1$ permutations of $e_1 e_2 ... e_N$ where there is at least one error we have, for each observation m:

$$w_{m,e} = \frac{p(f_m, \boldsymbol{e} = e | x_m, \beta)}{\sum_{e' \in \boldsymbol{e}} p(f_m, \boldsymbol{e} = e' | x_m, \beta)} \tag{6}$$

Therefore, for the example where there are $N = 3$ components in an observation, there will be $2^3 - 1 = 7$ w's for each configuration of error (e_1, e_2, e_3): w_{001}, w_{010}, w_{100}, w_{110}, w_{101}, w_{011}, and w_{111}. Each w is a weight in the sense that it represents the probability of the given configuration (for observations where there is no error, $f = 0$, the weight $w_{0\ldots0} = 1$). This exponential explosion of error combinations can be managed for medium numbers of components, which is reasonable for many applications. For large numbers of components, an approximate E-step could be derived using a variational EM algorithm.

M-step The M-step is a weighted maximum likelihood of the following:

$$\begin{aligned} g(\mathbf{w}, \beta) &= \sum_{m=1}^{M} \sum_{e \in \boldsymbol{e}} w_{m,e} \ln p(f_m, \boldsymbol{e} | x_m, \beta) \\ &= \sum_{m=1}^{M} \sum_{e \in \boldsymbol{e}} w_{m,e} \sum_{i=1}^{N} [e_i \ln \sigma(\phi_i(x_m)^\top \beta) + (1 - e_i) \ln(1 - \sigma(\phi_i(x_m)^\top \beta))] \end{aligned} \tag{7}$$

where each e_i takes on its value assigned by the permutation indexed by e. For example, if $N = 2$, then $\boldsymbol{e} = (e_1, e_2) = \{1 : (0,1), 2 : (1,0), 3 : (1,1), 4 : (0,0)\}$. Therefore, each observation m with $f_m = 1$ requires 3 w calculations, and contributes 3 weighted samples to the maximum likelihood. Things are further complicated by the fact that β will generally be different for each component. We get around this issue by having each feature vector $\phi_i(x)$ be of size $D \times N$ where there are D features and place zeros in the components that align with β values not considered by this component. A dot product between a sparse feature vector and the parameters that pertain to the given component can be efficiently computed. For the M-step we run a small number of iterations of SGD or batch gradient descent (depending on the application) at each step.

It is well known that EM algorithms are often highly sensitive to how the parameters are initialized [16]. Our algorithm is no different and we empirically observed falling into local minima for certain initializations. We overcome this problem by initializing the model parameters to those obtained by running an independent logistic regression with the observed labels being the overall feedback for the entire pipeline. In other words, for observation $\mathcal{X}$ with 2 components, we learn β_i for component i with features $\phi_i(\mathcal{X})$ and label f, even though $f = 1$ is partial as it could imply any of the following configurations: $(e_1 = 1, e_2 = 0), (e_1 = 0, e_2 = 1), (e_1 = 1, e_2 = 1)$. This initialization seems to discourage local minima that could trap the algorithm with a random initialization.

5 Experiments

We demonstrate the viability of our method on three separate tasks. First, we show that our model and inference algorithm are sound by learning the error

configuration and model parameters on synthetic data for different lengths of pipelines and numbers of feedback observations. We then show results on improving a 2-stage opinion summarization system by learning the probability of two static components committing an error given partial feedback. Finally, we describe the results of a semi-synthetic experiment on phrase error prediction for post-editing in machine translation where we predict the phrases most likely to contain translation errors given that we know there is some error in the translation.

5.1 Synthetic Data

To demonstrate how our model is able to learn the probability of a component committing an error with access only to partial feedback, we revisit the motivating $N = 2$ component example that we presented in the introduction. Here, we draw up to $M = 500$ datapoints with $d = 2$ features from a multivariate normal distribution with $\mu = (-3, 3)$ and $\Sigma = \mathbb{I}_2$. We randomly select a true β parameter for each component which corresponds to what we hope to learn. An error matrix $E \in \{0, 1\}^{M \times N}$ is generated where $e_{m,n} = 1$ implies that $\phi_n(x_m)$ would result in an error for component n. Each element $e_{m,n}$ is computed by drawing a random Bernoulli variable with parameter modeled by a binary logistic regression resulting in data with some added noise. This tends to result in a dataset that is roughly balanced between $f_m = 0$ and $f_m = 1$ observations. The translation $\mathcal{T}_n$ is applied for point x_m if $e_{m,n} = \text{Bernoulli}(\sigma(\phi_n(x_m)^\top \beta)) = 0$, otherwise the translation is scaled by some noise and the data gets translated randomly. The observations are then $(x_m, f_m)_{m=1}^M$ where $f_m = 1$ if any of $e_{m,n} = 1$ and $f_m = 0$ otherwise. Before proceeding all $e_{m,n}$ are removed and the algorithm learns β (and eventually $e_{m,n}$) given only $x_{m,n}$ and f_m. This is synthetic data experiment 1.

We learn the parameters with varying number of error observations and then test the precision and recall of predicted prior probability of error on a separate test set of 500 observations drawn from the same distribution. For each number of observations (10 to 500), we run 5 trials and report the average precision and recall. Figure 5 (left) shows that we do very well even from very few observations and we predict essentially perfectly from 250 observations on.

Next, we examine the precision and recall statistics for another synthetic experiment that considers how our model performs as the number of components varies. This is synthetic data experiment 2. Here, to simplify things, independent features are drawn for each component from a standard multivariate normal distribution. We randomly select error parameters and generate ground truth labels. Our algorithm then observes the features and only the partial feedback f for each observation. We are then interested in how many observations are required for different lengths of processing pipelines. We consider between $N = 2$ and $N = 6$ components. For testing, we again draw data from the same distribution but with $M = 100$ observations; this will amount to MN values of e_i to be predicted. We show F1-scores for different lengths of pipelines as the number of observations grows in Figure 5 (right). This shows that even with up to 6

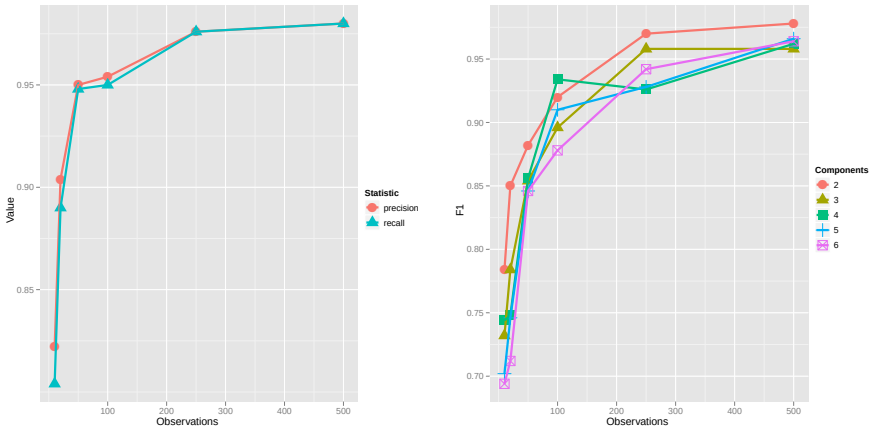

Fig. 5. Precision and Recall values for synthetic data experiment 1 (left) and 2 (right). Both experiments demonstrate that the precision and recall increase with the number of observations, as expected, but also that we approach perfect prediction with only very few labels.

components, we can learn the error model parameters very well with few observation examples. Also, the number of required observations for good predictive performance does not seem to heavily depend on the number of components at least for medium numbers of components as we tested.

5.2 Opinion Summarization

Next, we present a simple 2-component deterministic opinion summarization system that first filters out comments that do not contain opinion, and then labels the comments with up to K category labels. For determining opinionated texts, we use the MPQA Subjectivity Lexicon [17]. Here, among other designations, words can be described as *strong_subj* and *weak_subj* for being commonly associated with strong and weak subjectivity, respectively. Our intuition is that strongly subjective words result in opinionated texts. For each text, if a word is marked as *strong_subj* it scores 1.0, if it is marked as *weak_subj* it scores 0.5, and all other words score 0. The opinion score is the average word score for the text, and a text is considered opinionated if its opinion score is above some threshold Γ_O.

For determining whether a text can be labeled with some category marker c_k, we use a method that is common in text summarization: average word probability [18]. We use LDA [19] to learn word distributions for each category and then consider a text's average word probability under a word distribution for each category. Again, we consider a text to be a positive example for a category if its average word probability for that category is above some threshold Γ_C. The

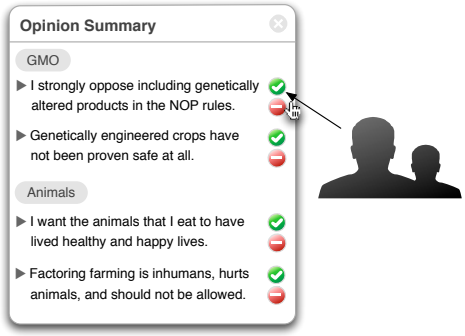

Fig. 6. The base components produce a summary and the user gives partial feedback by stating whether a given sentence either contains opinion and is in the correct category ($f = 0$), or that one or both of these is incorrect ($f = 1$).

underlying methods are relatively basic but our aim is to demonstrate how we can predict when each of the components has commit an error given that the final observation resulted in an error. Because each component is made up of binary classifiers, we can improve the system in the light of user feedback without modifying the underlying components. We wrap each of the components in an error model *wrapper* such that when the error model predicts that the current input would result in an error, we flip the prediction.

Our data to summarize consists of a subset of public comments on the US Department of Agriculture's (USDA) proposed National Organic Program (NOP) ("USDA-TMD-94-00-2").[1] These are comments by concerned citizens with respect to a proposed rule on what standards would apply to allow products to be designated as *organic*. This data fits our problem nicely because a sizable portion of the data consists of no opinion, and most of the texts can be sensibly placed into different categories given what aspect of the proposed legislation a citizen was referring to (animal well-being, genetically modified organisms, human health, etc.). 650 texts were manually labeled as either containing opinion or not, and for membership in up to 6 categories. We randomly select 100, 300, and 500 texts for training and leave the rest aside for testing. In this experiment, the feedback is whether a comment is correctly identified as containing opinion and labeled with the correct category ($f = 0$), or some labeling error exists. Figure 6 visualizes the setting, and the features are simple bag-of-words. We are interested in the accuracy of all predicted labels for the "wrapped" system. That is, we use the base system described above, run the testing data through the pipeline, and at each component if our error model predicts an error, we flip that prediction. We run each experiment 5 times with different random permutations of training and testing data and report the average accuracy. Figure 7

[1] http://erulemaking.cs.cmu.edu/Data/USDA.

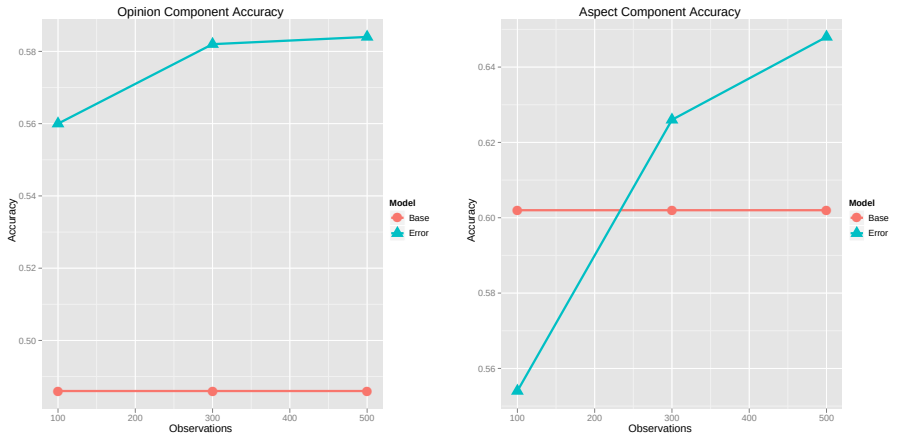

Fig. 7. Accuracy results for the opinion (left) and aspect (right) components in our augmented summarization system. The "base" curve shows the accuracy of the base components and the "error" curve shows the accuracy when our error model is applied.

shows the opinion component accuracy (left) and the aspect component accuracy (right) as the number of feedback examples varies.

With 100 and more partial feedback examples, the error model-wrapped opinion component does better than the base component. For the aspect labeling component, 100 examples was not enough to provide adequate predictive accuracy to do better than the base component. However, with 300 labels and more, it handily beats the base component. The reason for this discrepancy is data sparsity; each feedback example is only with respect to one aspect label and with 6 labels, a training set perfectly balanced amongst the 4 different error combinations would only include $100 \times \frac{1}{6} \times \frac{1}{4} \approx 4$ training examples per context. Of course, in practice we never achieve this perfect balance and certain contexts will be over-represented while others will have no training examples at all. Nevertheless, even with a relatively small amount of feedback, we can see that the system is able to predict the error configuration and therefore improve the accuracy of the overall system. In practice, when the error model is used as a wrapper in such an experiment, it would only be activated once an appropriate amount of training data was obtained.

5.3 Error Detection in Machine Translation

For our final experiment, we describe a semi-synthetic experiment in that the features are true data, but the labels are partially generated. Machine Translation (MT) quality has yet to reach a state where translations can be used reliably without user supervision. Therefore, when a high quality translation is required, a *post-editing* stage is typically conducted. In post-editing, professional transla-

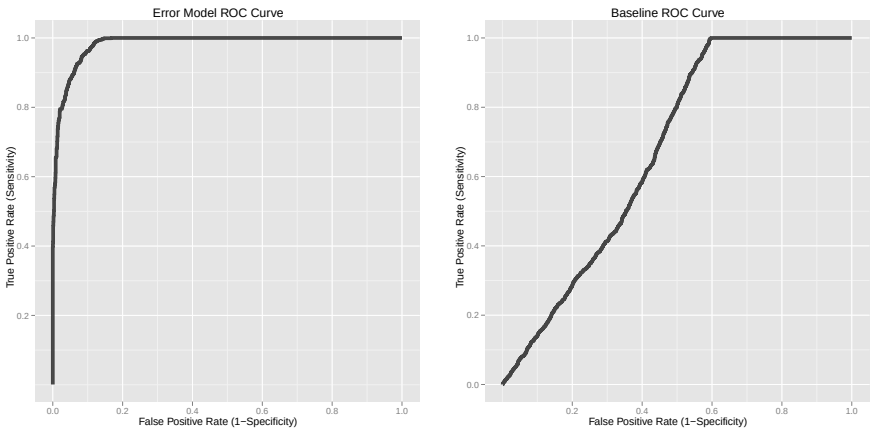

Fig. 8. ROC Curves for predicting prior probability of an incorrect phrase translation learned with the partial feedback error model (left) vs. the baseline which assigns an error to all phrases in a sentence when $f = 1$ (right).

tors review and correct a translation before it is used. Error detection is therefore an important subject in machine translation [20, 21]. It is a useful means of reducing post-editing effort by directing the translator to specific segments in the translation which are estimated to be erroneous. This could also be used within the MT system itself, by avoiding erroneous translations and reverting to the next best alternatives proposed by the system in the light of a predicted error.

Here, we use our error model framework to predict the phrases in a translated sentence that are most likely to contain errors. Connecting the setting to the previous examples, each phrase is a component, and each sentence is a pipeline. Feedback consists of either a perfectly translated sentence ($f = 0$) or a sentence that contains at least one error ($f = 1$). There are simply 4 features for this experiment: the probability of the source phrase given the target phrase; the lexical weighting of the source phrase given the target phrase; the probability of the target phrase given the source phrase; and the lexical weighting of the target phrase given the source phrase. Each of these features is computed automatically using the Moses phrase-based SMT system [22].

Because we need phrase-specific error labels for testing, we take a synthetic approach to labeling. We manually labeled ~ 400 translated phrases as either containing or not containing an error and then learned an independent binary classifier on this fully-labeled data. Using this classifier, we then generated labels for a set of 5000 sentences that are segmented into phrases. We then took all of the sentences that contained 6 phrases or less to end up with 1002 training sentences. Each of these sentences receives a label $f = 1$ if any of its phrases contain errors, and $f = 0$ otherwise. We learn the error model and then predict the prior probability of each phrase-pair containing an error.

We compare our model to a simple baseline. The baseline learns a binary logistic regression classifier on phrases where the labels are simply the partial feedback f. That is, when $f = 0$, each phrase is an independent example with the (correct) label 0. When $f = 1$, each phrase is also an independent example but now the label will only sometimes be correct. In fact, it will rarely be correct because most translated sentence errors are confined to 1 or 2 phrases. The behavior of the baseline is best understood by showing its ROC curve. The ROC curves for each method are shown in Figure 8 and demonstrate that there is little problem with choosing a discrimination threshold in our method (left) and that the baseline (right) is a poor method especially when the density of errors in a pipeline is low.

6 Conclusions and Future Work

In this paper, we have described a probabilistic error model framework that aims to predict the configuration of error in components in a pipeline and therefore learn the probability of each component committing an error given only partial feedback. In many cases it is difficult or time consuming for a user to provide full feedback to a system when the output is the result of decisions made by numerous components. Conversely, it is generally easy to be able to tell if the output is perfect versus containing some error. In our model, we are able to probabilistically infer the component-specific error model parameters from partial feedback and use that information to either dynamically improve the system if it consists of binary classifiers (opinion summarization example), alert the user to components that should be examined (translation post-editing example), or properly direct the feedback for further training.

In the near future, we plan to apply it to other natural language processing tasks, such as named entity recognition, which could benefit a great deal from human feedback, and which would be very difficult for a human to pinpoint the exact cause of error. We also plan to combine it with active learning techniques to choose which examples to show to the user. At the moment, this is done randomly, irrespective of the quality of the individual components and the predictions we make. This strategy is clearly suboptimal. In principle, we could reduce the amount of feedback to be provided by the user for a same level of accuracy if we choose the examples to provide feedback on in a sensible way. Finally, we are also interested in exploring non-linear versions of the error model.

Acknowledgments

The research leading to these results has received funding from the European Commission Seventh Framework Programme (FP/2007-2013) through the projects Fupol and Fusepool.

References

1. Ritchie, D.M.: The evolution of the unix time-sharing system. Communications of the ACM **17** (1984) 365–375
2. Ritter, A., Clark, S., Mausam, Etzioni, O.: Named entity recognition in tweets: An experimental study. In: Proceedings of the 2011 Conference on Empirical Methods in Natural Language Processing, Edinburgh, Scotland, UK., Association for Computational Linguistics (July 2011) 1524–1534
3. Ly, D.K., Sugiyama, K., Lin, Z., Kan, M.Y.: Product review summarization from a deeper perspective. In: Proceedings of the 11th annual international ACM/IEEE joint conference on Digital libraries. JCDL '11, New York, NY, USA, ACM (2011) 311–314
4. Finkel, J.R., Manning, C.D., Ng, A.Y.: Solving the problem of cascading errors: approximate bayesian inference for linguistic annotation pipelines. In: Proceedings of the 2006 Conference on Empirical Methods in Natural Language Processing. EMNLP '06, Stroudsburg, PA, USA, Association for Computational Linguistics (2006) 618–626
5. Blair-goldensohn, S., Neylon, T., Hannan, K., Reis, G.A., Mcdonald, R., Reynar, J.: Building a sentiment summarizer for local service reviews. In: In NLP in the Information Explosion Era. (2008)
6. Lu, Y., Zhai, C., Sundaresan, N.: Rated aspect summarization of short comments. In: Proceedings of the 18th international conference on World wide web. WWW '09, New York, NY, USA, ACM (2009) 131–140
7. Zhai, Z., Liu, B., Zhang, L., Xu, H., Jia, P.: Identifying evaluative sentences in online discussions. In: AAAI. (2011)
8. MacCartney, B., Grenager, T., de Marneffe, M.C., Cer, D., Manning, C.D.: Learning to recognize features of valid textual entailments. In: Proceedings of the main conference on Human Language Technology Conference of the North American Chapter of the Association of Computational Linguistics. HLT-NAACL '06, Stroudsburg, PA, USA, Association for Computational Linguistics (2006) 41–48
9. Cunningham, H., Humphreys, K., Gaizauskas, R., Wilks, Y.: Software infrastructure for natural language processing. In: Proceedings of the fifth conference on Applied natural language processing. ANLC '97, Stroudsburg, PA, USA, Association for Computational Linguistics (1997) 237–244
10. Lamb, A., Paul, M.J., Dredze, M.: Separating fact from fear: Tracking flu infections on twitter. In: Proceedings of NAACL-HLT. (2013) 789–795
11. Neal, R.M.: Connectionist learning of belief networks. Artificial intelligence **56**(1) (1992) 71–113
12. Marciniak, T., Strube, M.: Beyond the pipeline: discrete optimization in nlp. In: Proceedings of the Ninth Conference on Computational Natural Language Learning. CONLL '05, Stroudsburg, PA, USA, Association for Computational Linguistics (2005) 136–143
13. Koller, D., Friedman, N.: Probabilistic Graphical Models: Principles and Techniques. MIT press (2009)
14. Bottou, L.: Large-scale machine learning with stochastic gradient descent. In Lechevallier, Y., Saporta, G., eds.: Proceedings of the 19th International Conference on Computational Statistics (COMPSTAT'2010), Paris, France, Springer (August 2010) 177–187
15. Liang, P., Klein, D.: Online em for unsupervised models. In: Proceedings of Human Language Technologies: The 2009 Annual Conference of the North American

Chapter of the Association for Computational Linguistics. NAACL '09, Stroudsburg, PA, USA, Association for Computational Linguistics (2009) 611–619
16. Fayyad, U., Reina, C., Bradley, P.S.: Initialization of iterative refinement clustering algorithms. In: Proceedings of the Fourth International Conference on Knowledge Discovery and Data Mining. (1998) 194–198
17. Wilson, T., Wiebe, J., Hoffmann, P.: Recognizing contextual polarity in phrase-level sentiment analysis. In: Proceedings of the conference on Human Language Technology and Empirical Methods in Natural Language Processing. HLT '05, Stroudsburg, PA, USA, Association for Computational Linguistics (2005) 347–354
18. Nenkova, A., Vanderwende, L., McKeown, K.: A compositional context sensitive multi-document summarizer: exploring the factors that influence summarization. In: SIGIR '06: Proceedings of the 29th annual international ACM SIGIR conference on Research and development in information retrieval, New York, NY, USA, ACM (2006) 573–580
19. Blei, D.M., Ng, A.Y., Jordan, M.I.: Latent dirichlet allocation. J. Mach. Learn. Res. **3** (2003) 993–1022
20. Xiong, D., Zhang, M., Li, H.: Error detection for statistical machine translation using linguistic features. In: Proceedings of the 48th Annual Meeting of the Association for Computational Linguistics. ACL '10, Stroudsburg, PA, USA, Association for Computational Linguistics (2010) 604–611
21. Ueffing, N., Ney, H.: Word-level confidence estimation for machine translation. Comput. Linguist. **33**(1) (March 2007) 9–40
22. Koehn, P., Hoang, H., Birch, A., Callison-Burch, C., Federico, M., Bertoldi, N., Cowan, B., Shen, W., Moran, C., Zens, R., Dyer, C., Bojar, O., Constantin, A., Herbst, E.: Moses: Open source toolkit for statistical machine translation. In: Proceedings of ACL, Demo and Poster Sessions. (2007)