Data Mining to Predict and Prevent Errors in Health Insurance Claims Processing

Mohit Kumar, Rayid Ghani, Zhu-Song Mei
Accenture Technology Labs
Chicago, IL, USA
mohit.x.kumar, rayid.ghani, zhu-song.mei@accenture.com

ABSTRACT

Health insurance costs across the world have increased alarmingly in recent years. A major cause of this increase are payment errors made by the insurance companies while processing claims. These errors often result in extra administrative effort to re-process (or rework) the claim which accounts for up to 30% of the administrative staff in a typical health insurer. We describe a system that helps reduce these errors using machine learning techniques by predicting claims that will need to be reworked, generating explanations to help the auditors correct these claims, and experiment with feature selection, concept drift, and active learning to collect feedback from the auditors to improve over time. We describe our framework, problem formulation, evaluation metrics, and experimental results on claims data from a large US health insurer. We show that our system results in an order of magnitude better precision (hit rate) over existing approaches which is accurate enough to potentially result in over $15-25 million in savings for a typical insurer. We also describe interesting research problems in this domain as well as design choices made to make the system easily deployable across health insurance companies.

Categories and Subject Descriptors

H.2.8 [**Database Management**]: Database Applications—*Data Mining*; H.4.2 [**Information Systems Applications**]: Types Of Systems—*Decision Support*

General Terms

Design, Experimentation, Performance

Keywords

Health insurance claims, Claim rework identification, Predictive system, Machine Learning

KDD'10, July 25–28, 2010, Washington, DC, USA.

1. INTRODUCTION

Health insurance costs across the world have increased alarmingly in recent years. These costs have been passed down to consumers and employer-sponsored health insurance premiums have increased 131 percent [11] over the last decade. A large proportion of these increases has been due to increase in the administrative costs of insurance providers. According to a study by McKinsey and Company [9], $186 billion of over-spending on healthcare in the US is related to high administrative costs.

The typical process for insured healthcare in the US is that a patient goes to a service provider (medical facility) for the necessary care and the provider files a claim with the patient's health insurance company for the services provided. The insurance company then pays the service provider based on multiple complex factors including eligibility of the patient at time of service, coverage of the procedures in the benefits, and contract status with the provider etc.

Payment errors made by insurance companies while processing claims often result in re-processing of the claim. This extra administrative work to re-process claims is known as *rework* and accounts for a significant portion of the administrative costs and service issues of health plans. These errors have a direct monetary impact in terms of the insurance company paying more or less than what it should have. [1] estimates from a large insurance plan covering 6 million members had $400 million in identified overpayments. In our discussions with major insurance companies, we have found that these errors result in loss of revenue of up to $1 billion each year. In addition to the direct monetary impact, there is also an indirect monetary impact since employees need to be hired to *rework* the claim and answer service calls regarding them. According to estimates by an Accenture study, 33% of the administrative workforce is directly or indirectly related to rework processing. These statistics make the problem of rework prevention extremely important and valuable to the healthcare industry and motivated the work described in this paper.

There are two industry practices currently prevalent for identifying payment errors: random quality control audits and hypothesis (rule) based queries. Random audits are not very effective at identifying this rework since the majority of claims are correct and most of the effort spent on audits is wasted. In our discussions and research, we found that somewhere between 2% and 5% of the claims audited are rework, making 95% to 98% of the effort spent in random audits a waste. Hypothesis based querying involves domain experts identifying several hypothesis about how rework occurs and

instantiates itself in claim data. They create rules which are then turned into SQL queries to find matching claims. Typical systems contain a few thousand rules that are run every day identifying thousands of suspect claims. This results in slightly better precision (or hit rate) than random audits but still require a lot of manual effort in discovering, building, updating, executing and maintaining the hypotheses and rules. These rules are also insurance plan specific and do not generalize well across companies thus making the deployment very time consuming and dependent on domain experts.

In this paper, we describe our system to help reduce these claims errors using machine learning techniques by predicting claims that will need to be reworked, and experiment with generating explanations to help the auditors correct these claims, feature selection, concept drift, and active learning to collect feedback from the auditors to improve over time. This Rework Prevention Tool has been developed in conjunction with industry experts from Accenture's Claims Administration group who currently work with most of the large insurance companies in the US. We have applied this system to two large US health insurance companies. In the rest of this paper, we describe our system framework, problem formulation, evaluation metrics, and experimental results on claims data from a large US health insurer. We show that our system produces an order of magnitude better precision (hit rate) over existing approaches which is accurate enough to potentially result in over $15-25 million in savings each year for a typical insurer. This in turn would have a large effect on the healthcare costs as well as help make the healthcare process smoother. We also describe interesting research problems in this domain as well as design choices made to make the system easily deployable across health insurance companies.

2. RELATED WORK

In general, claims processing is not an area where data mining techniques have been widely used. A lot of the work in claims has focused on fraud detection which is related but different area. [4] mentioned a system for detecting healthcare provider fraud in electronically submitted claims developed at Travelers Insurance. Health claim fraud is very different from Rework identification because the characteristics of fraudulent claims and Rework claims are different. Often, there is not much labeled data available for fraud and the magnitude of fraud is also typically smaller. Similarly, related work in the area of credit card fraud [2] is also different from Rework identification as the data characteristics and expectations are different. There has been some work in the area of claim overpayment identification [1] which is a subproblem of Rework identification as Rework constitutes both overpayment and underpayment of claims. Also the approach taken in the system is dependent on client-specific 'edits' or rule-based scenarios by training the models for each scenario. This makes the approach less generalizable as it requires the designing of these client scenarios or rules.

Outside the research community, there are some commercial product vendors such as Enkata that provide tools to analyze past claims in order to discover root causes for claims rework. These tools do not predict future rework using machine learning techniques and do not help prevent errors automatically. Typically, these tools help with understanding the factors of historical rework and require managers to fix the back-end system.

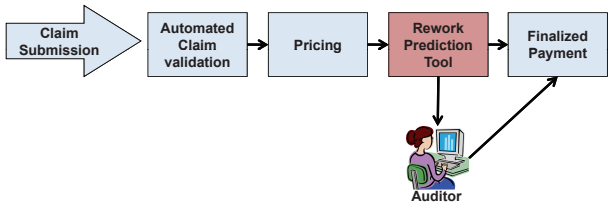

Figure 1: Claim Processing Pipeline

3. PROBLEM FORMULATION

We formulate the problem of Rework prediction as a classification problem and generate a ranked list of claims that need to be manually reviewed. We give the details of the problem formulation in the sections below.

3.1 Claim Processing Overview

We start by giving an overview of the claims processing workflow (Figure 1) and describe how rework prediction fits in this workflow. Claims are created by service providers and submitted to the insurance company. They go through automatic validation checks followed by pricing using benefit rules and contracts. This takes place automatically in some cases and in other cases, manual intervention is required. Once the pricing is applied, claims gets finalized and payment is sent to the service provider. The system we describe in this paper is the box placed after the pricing is applied to detect potential issues with the claim before it's finalized so it can be corrected before payment is sent.

3.2 Requirements

We worked with domain experts to come up with requirements that would make a rework prediction solution practical and useful for insurance companies. These requirements are described below and motivates our solution.

- *Prepayment prediction*: We need to identify rework claims *before* payment is sent with the information that is available at the time of claims submission. This seems obvious to data mining experts but the industry norm today is that most of this analysis happens *after* payment is made. The problem for insurance companies then becomes how to *recover* the payment (or pay penalties if there was an underpayment). Insurance companies have recovery departments tasked with recovering payments that have been made incorrectly. Since the motivation of the system is to reduce healthcare costs, the goal is to pay the claim correctly the first time and without the extra effort later to correct it.

- *Generalization*: We need to identify a wide variety of rework, not be limited to manually identified rules, and should be easily deployable across companies. Designing rules for identifying errors in payment requires deep domain knowledge. These rules are also very specific to individual companies making the process of transferring the system to new companies difficult and expensive. Thus our framework needs to be easily deployable across clients with minimal extra effort.

- *Accuracy*: We should flag suspect claims with high accuracy to make it financially feasible for a human auditor to examine the claim and correct it. Since examining a claim can take considerable time (ranging from 20 minutes to over an hour), the accuracy of our system has to be high enough to justify this extra cost.

- *Explanations*: We should provide a means of fixing the claim errors quickly and economically. It should be able to communicate to the auditors the reasons for which the system is predicting the claim as Rework.

- *Adaptability*: The framework should adapt to changes in environment and the healthcare ecosystem due to changes in legislations, insurance plans, contracts, and pricing. We need to make sure our system is able to adapt accordingly. The system is also expected to improve its accuracy over time as it observes more and more data. Thus the system should actively solicit and use the feedback from the auditors to keep adapting to the changes and improving its performance.

3.3 Our Formulation

We have formulated the problem of Rework identification as a binary classification problem and predict whether a claim will end up requiring Rework in the future or not. We use the confidence score of the binary classifier to prioritize and order the claims for review by the auditors.

'Prepayment prediction': We look at the claim before payment and predict whether it is likely to be Rework. All our features are based on the information that is available at the point in time when the claim was adjudicated and ready for payment (but not paid).

'Generalization': translates to having a company independent 'universal' data model to which a company's data is mapped making the feature extraction step for the framework company-independent. The system also has the flexibility to include any company specific feature which may not be present in the 'universal' data model. There is no preprocessing of the data based on the manual rules that may exist for the client, so the rework prediction is not tied to client-specific rules.

'Accuracy': We prioritize the claims based on the classification confidence and aim to optimize the predictions for the top ranked claims. This is justified in real life as the daily volume of claims coming into the pipeline for adjudication is huge, typically 50,000 claims per day for a 3.5 million member plan, and it is practically only possible to manually examine only a fraction of the claims.

'Explanations': We have come up with a novel User Interface that communicates the system's recommendation to the user, highlighting the underlying reasons for the recommendations. The UI gives the auditors an overall likelihood score for the claim being rework as well as the data fields responsible for the prediction making their task simpler and more efficient.

'Adaptability': We have built a detailed feedback mechanism in the UI that solicits feedback from the auditor to confirm or deny the system's recommendation. We also get feedback about the system's reasoning for the Rework. The system uses this feedback to improve its accuracy over time. For adaptability over time, we have built in the experimentation pipeline a model selection framework that selects the best model.

4. SYSTEM OVERVIEW

Our system consists of the following components: Data Collection, Feature Construction, Model Learning and Selection, item Scoring, Explanation Generation, User Feedback, and User Interface.

4.1 Data Collection

As shown in the figure 1 we capture data from the claims processing pipeline when it is priced and ready for finalization and payment. The data we operate on is the entire claims data warehouse which contains all the claims that have been submitted and processed in the past several years.

We also need labels for this data to train our models. There are two labels we need to assign: *rework* or *correct*. The claims assigned the label *rework* are those that were manually examined in the past and contained errors. Those assigned the label *correct* are ones that were manually examined in the past and found to be correct. The process of getting these labeled claims is fairly difficult in insurance companies. The labeled data exists but it's distributed across several systems and is collected through different business processes.

The labels we typically come from three primary sources. The first source is the Quality Control Audit system which contains all the claims that have been manually audited by auditors for quality control. The class distribution in this data is typically 2-5% rework and 95-98% correct claims which is the overall distribution of the entire population of claims. The second source is the Provider Dispute system. This source contains claims that are discovered by the service providers as erroneously paid and sent back to the insurance company for re-processing. Although this happens after claim is paid, we can use these as labeled claims. Most of the claims that come through this system are *rework* (typically underpayments). The third source is the financial recovery process. This process is also initiated after the claim is paid to recover overpayments the insurance company makes to providers. Since the number of claims in the provider dispute and financial recovery systems is much larger than that in the quality control system, the entire training set contains more *rework* examples (≈60-80%) than *correct* examples (≈20-40%). This raises interesting research issues since the training distribution is very different from the distribution that would occur in real data after deployment. The real data distribution is typically the opposite - 95% rework, 5% correct since that is the distribution of the entire population of claims.

4.2 Feature Construction

Once the data is collected and labeled, feature construction is the next step that takes place. There are four classes of information in each claim: Member information, Provider information, Claim Header, and Claim Line Details. Our features are extracted from these classes of information. Member and Provider information span the entire claim and provide information about the Member (patient), and the provider (hospital, doctor, or medical facility). The claim header gives information about the entire claim as well. Contract information, Amount billed, Diagnosis codes, Dates of service are some examples of data fields in the claim header. The last source is the Claim Line Details. This gives the details of each line in the claim which is used to itemize the claim for each procedure that was conducted on

the patient. For each procedure, the claim line includes the amount billed, the procedure code (CPT), the counter for the procedure (quantity). Since we are currently focusing on predicting the likelihood of the entire claim as being rework, we aggregate claim line details to the overall claim level. For example, we create features using Min, Max, Average and standard deviation functions for each numeric data field in each line. We also create more aggregate features that are specific to procedure codes.

We derive some more features that are specific to the claims processing domain. For example, we calculate the time (in days) between the date of service and the date of submission of the claim. The intuition for this feature is to figure out if the claim is valid as there is a time limit within which the claims should be filed with the plan and also to see if there is a correlation between Rework and late/early submission of claims. Overall, we end up with $\tilde{1}$5,000 categorical and numerical features to build our models.

4.3 Model Learning & Selection

We experimented with SVMs as the main learning algorithms. SVMs have been shown to be robust for large data mining tasks with large feature sets. Since SVMs are not able to handle categorical data we had to create binary features from our categorical features which led to feature explosion with nearly 110,000 features. We used SVMperf [7] for SVMs. We also used SVMs from other packages Weka [5], KNIME but neither of them were able to handle our dataset and run experiments efficiently. In our experience, SVMperf is the most efficient package which can handle large datasets quickly.

We also chose SVMperf because of the extremely fast training time. One of our goals was to create a system that requires minimal machine learning expertise to deploy. We wanted to automate the selection of classifier parameters and the ideal feature set size empirically using hold-out sets. This required us to run thousands of experiments varying the training data, classifier parameters, feature set sizes, and evaluation metrics to optimize. SVMperf, because of it's fast training time, was ideal for our needs since it allowed us to automate the model selection process.

Another aspect of model selection is dealing with concept drift and change in the target function over time. The obvious approach is to use all the training data available at any given time to train the models. If there is a lot of concept drift over time, more recent training data is more useful and retaining all history may end up hurting the overall performance. Our model selection experiments take this into account and empirically estimate the best subset of the data that is useful to predict rework in the near future. We give more details in the experiments section later.

4.3.1 Feature Selection

Since we have more than 110K features after creating binary features from the categorical data, we wanted to see if feature selection would be useful. We wanted to see if we can reduce the storage requirements for the features as well as explore the effect of reducing the feature size on SVM accuracy. Although it has been shown that SVMs are able to compensate for feature noise and can handle large feature sets, we wanted to confirm that for our problem and data. We experimented with Information Gain measure but the runtime for our data set was impractically high so we used a frequency-based feature selection technique. Since converting categorical features into binary features creates a sparse feature space, frequency-based feature selection was useful in making our system more efficient both in terms of execution time and storage requirements.

4.4 Scoring

Once we have picked the best model(s), we run that model to classify all the unlabeled data in our database. The scores assigned to each claim by the model(s) are aggregated and stored in our database to be used by the user interface as well as other components of our system.

4.5 Generating Explanations

Accurately identifying rework and presenting it to auditors is one aspect of our system. Another equally important aspect is to give the auditors the ability to quickly determine if the claim being presented to them is rework or not. Currently, auditors in most companies are given a claim without any hints about why a claim could be rework. The goal of this component is to *explain* the predictions of the classifier to the auditor to reduce the time it takes them to determine the label of the claim.

In general, this task is extremely important for machine learning algorithms. People typically use decision trees because they are easy to explain the classification to domain experts. Often, this comprehensibility comes at the expense of accuracy. In our case, we use the feature weights learned by the SVM to generate the explanations. Linear SVMs use a prediction function of the form : $prediction(x) = sign(w^T x + b) = sign(\sum_i w_i * x_i + b)$ with features i. Features with large (absolute) values have a large effect on the classification while features with weights w_i close to zero have little effect. [12] has shown that the features with high weights are the features that are influential in determining the width of the margin when learning with SVMs. [10] have also used a similar approach to perform feature selection by retaining features with high weights.

To generate an explanation for claim C, we calculate an influence score for each feature C_i in the feature vector. $InfluenceScore(C_i) = w_i * value_{C_i}$ where $value_{C_i}$ is the value of the feature i in claim C. We treat this score as the influence of feature i in classifying the current claim C. This score is embedded in to the user interface that is provided to the auditors to focus the attention of the auditors on the most influential data fields. We describe this in more detail in the User Interface section later.

In addition to influence scores, we also experimented with using structured codes that were assigned as error codes to the claims in the labeling process to generate explanations. We also plan to explore the use of text comments that are entered by auditors as part of the auditing process to generate templates for explanation.

4.6 User Feedback

Getting feedback from auditors to improve future predictions is an important requirement for our system. We get feedback from the auditors in three ways:

1. Claim Feedback: For each claim that is presented to the auditor, the auditor examines the claim and decides if it's rework or not. In each case, the feedback the auditor provides is a binary judgement as well as a Reason category (from a predefined set of categories).

In addition, we also provide a text comment box where the auditor can enter a free-text comment that we can analyze to create further reason categories.

2. Active Learning: Typically the auditors will review claims that have high probability of rework in order to maximize efficiency and find as many rework claims as possible. One drawback of this approach is that the learning algorithm usually gets positive reinforcement from this feedback and may not improve much over time. We also implement active learning strategies in our system. The key idea behind active learning is that a machine learning algorithm can achieve greater accuracy with fewer training labels if it is allowed to query the user about specific unlabeled examples. Since our problem domain contains millions of unlabeled examples, there is a cost to obtaining examples, and we want our system to improve while minimizing the labeling cost, active learning is an appropriate technique to use. In the experiments section, we describe the active learning strategies we use.

3. Feature Feedback: In addition to getting more labeled data through user feedback, we also want to get finer-grained *structured* feedback from the auditors. Although we have the text comment field to get unstructured feedback, we wanted to make it easy for auditors to give us structured feedback that can be directly used by the learning algorithms. Using our User Interface (described in the next section), when a claim gets presented to the user with the *high influence* fields highlighted, we allow the auditor to toggle the highlighted fields signifying that field was *not* a cause for rework in that claim. We also allow auditors to click on any other fields which highlights it, telling the system that this field is influential in that claim being rework or not. This structured feature-level feedback can be incorporated back into the learning algorithm and improve the models over time.

4.7 User Interface

Our system is designed to be used by auditors in health insurance companies as part of their every day work. This makes the user interface critical for adoption by users. We needed the user interface to be simple and intuitive for the auditors and at the same time, useful for us to collect the necessary feedback from the auditors. To achieve these goals, we worked with domain experts and designed an interface that was as close to what auditors typically use (a claim form) when auditing the claim but enhancing it with some new features. Figure 2 shows a screenshot of the user interface for the auditors. This interface provides the auditors with a list of claims in their work queue with a rework likelihood score assigned to each claim that is generated by the classifier. In addition, we use the influence scores described earlier to highlight the data fields. The intensity of the highlight color is proportional to the influence score assigned by the classifier. This feature has been designed to focus the attention of the auditor to the influential data fields and help them audit the claim faster. As mentioned earlier, we also allow the auditors to click on any field to *unhighlight* it (if it's highlighted) or to highlight it and give us feedback that field is influential (or not).

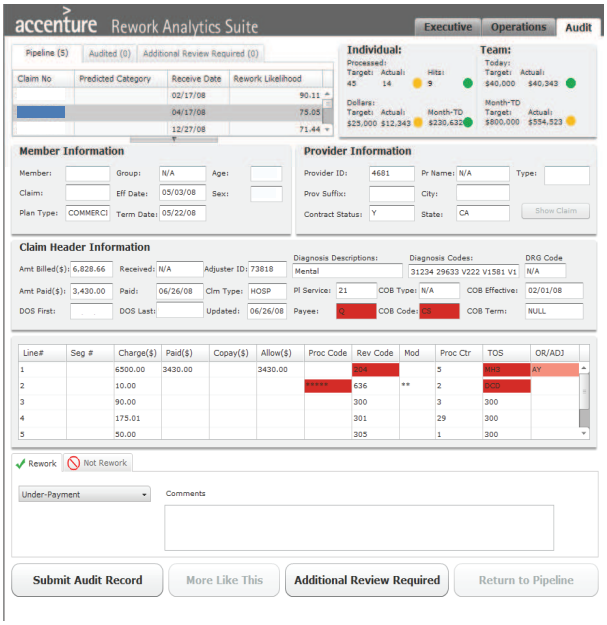

Figure 2: User Interface with the Auditor view (sensitive PHI fields are blanked out).

5. EXPERIMENTAL RESULTS

5.1 Data

In this paper, we present experimental results using real claims data from a major insurance company in US as part of our project with them. We got approximately 3.5 million claims (14 million claim lines) spanning 2 years. Of these 3.5 million claims, 121,000 had been manually audited and found to be either Rework or not, forming our labeled data set. In this labeled data set, around 40% claims were Rework and the rest Correct. It is important to remember that this distribution does not hold in the operational world since the labeled data has a skewed class distribution (described in section 4.1). For feature extraction, we take the state of the claim from the historical data at the moment when they are priced and are ready for final payment. Since SVMs are not able to handle categorical data we create binary features from the categorical features resulting around 110,000 features based on all the 3.5 million claims data. Of these 110,000 features, only 33,000 are present in our labeled data set.

5.2 Metrics

For the large insurance provider, the estimated capacity for auditing the claims is about 200 claims per day (121K claims in 2 years) whereas the volume of claims that need to be processed daily is approximately 5000 claims per day (3.5 million claims in 2 years). The audit capacity is approximately 4-5% of the total daily volume of the claims which is the case, in general, for most of the insurance companies.

Standard metrics such as accuracy, precision, or recall are useful in comparing different models and approaches, but not enough to minimize the number of claims errors, which is the eventual metric. Based on the audit capacity and discussions with domain experts, we decided that we need a metric that evaluates performance for the top 5-10% scored

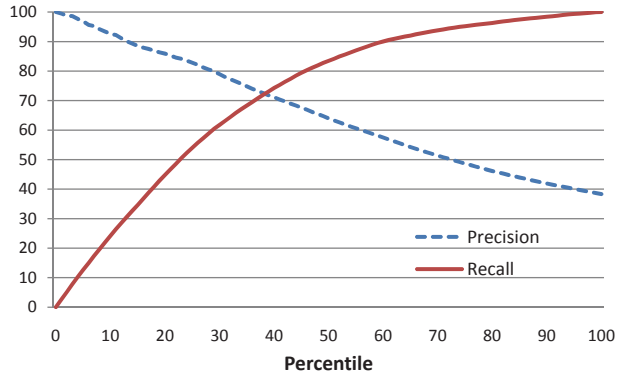

Figure 3: Precision Recall for test set

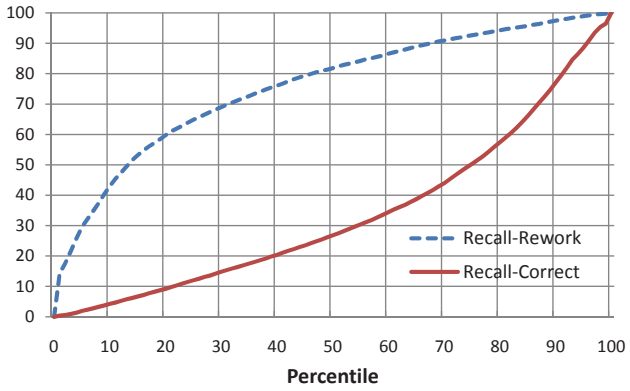

Figure 4: Recall on unlabeled data

claims to maximize benefits in real scenarios. Hence, we use Precision (or hit rate) at 10th Percentile as our metric. This is similar to the evaluation metric, Precision at top 10 documents, popularly used in the Information Retrieval community. In our experimental framework, we do have the flexibility to modify the metric easily and do model selection based on different metrics. In the results reported below, we show precision recall graphs where appropriate, as well as our precision at top 10% metric when comparing different models.

5.3 Is our system accurate?

Before we can deploy our system, we needed to run some experiments to show the effectiveness of our system without live audits. We randomly split the labeled data in 70%-30% training and test sets. Figure 3 shows the Precision-Recall graph for this experiment averaged over 10 random 70-30 splits. The average precision at 10th percentile is 92.9%. Compared to the 2-5% hit rate that the insurance companies get in their auditing practice currently, this is a significant boost in performance. However, the results are not expected to translate directly to real life because the class distribution in this labeled data is skewed and not reflective of the real life distribution. In our dataset, 40% of the claims are Rework whereas in real-life only 2-5% of claims are expected to be rework. Even so, the baseline performance on our data set corresponding to a random ranking is 40% whereas our system gives almost 93% precision at top 10.

In order to estimate the performance of our model in a more realistic scenario, we design the following experiment. We mix the labeled test set (36,000 claims) from the previous experiment in the unlabeled data (3.4 million claims) to form a new data set. We score the claims from this new data set using the model trained on the training set and rank order them based on the rework score. We expect the Rework examples from the labeled test set to be towards the top of this list and the Correct examples from the labeled test set to be towards the bottom, with the unlabeled examples scattered throughout based on their rework score. We calculate the Recall of both known Rework and Correct from the labeled test set in this new dataset as shown in figure 4. This graph helps us in verifying that the Rework claims are being ranked higher in general than the Correct claims. The interesting quantitative measure is that 42% of the known Rework can be detected in the top 10% scored claims. Based on our discussion with industry experts and [1], the estimated recall for the current industry practice, i.e. percentage of identified rework compared to the total unidentified rework, is about 10% which makes our recall score of 42% much higher.

Once we have shown in our experiments that we can effectively detect labeled rework at high levels of precision and recall, we give a sample of the highly scored unlabeled claims to auditors. As mentioned earlier, it takes 20 minutes to an hour for the auditor to audit a claim. Since we have 3.4 million unlabeled claims, there is no practically feasible sample size that can give us statistically significant confidence estimates about the system's performance on the unlabeled dataset. We selected a small sample of highly scored unlabeled claims, which took them 3 person days to audit, and achieved an overall precision (hit rate) of 65%. This 65% precision was significantly better than the random audits as well as the hypothesis (rule) based audit tools that are currently in use that result in 5%-10% precision.

5.4 Does removing features hurt the classifiers?

The experiments described above were done using all features. Since we run multiple models during experimentation, training and scoring times are crucial factors for our system. One way to speed up the I/O is to reduce the number of features in our data. Another reason to experiment with feature selection is to verify that the classifiers we are using are robust to feature noise and do not suffer by using all the features we create. As mentioned in System Overview section, we use frequency based feature selection techniques. Figure 5 shows the experiments with multiple minimum frequency thresholds. We find that a threshold of 10 gives slightly better accuracy than using all the features and reduces the feature set to about 8200 features. This gives us the flexibility to reduce the number of features if speed and storage requirements become important and also reassures us that we can easily add more features without the SVM precision suffering much.

5.5 Is recent labeled data more useful than all labeled data?

There are two goals we are trying to achieve with our system. The first one is at initial deployment of our system; it needs to identify claims that are in the historical claims database that have been processed incorrectly. The experiments described so far optimize that scenario where we randomly split the data into training and test sets. Once all

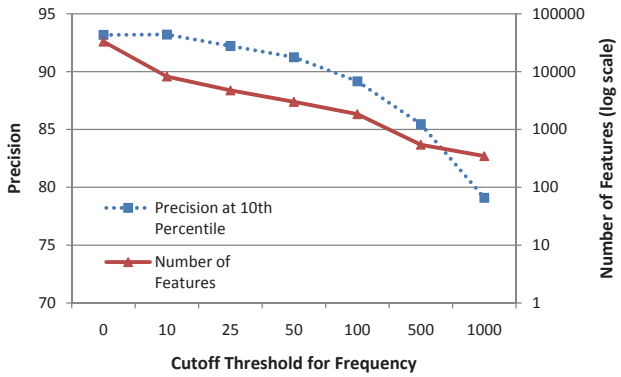

Figure 5: Feature Selection Results

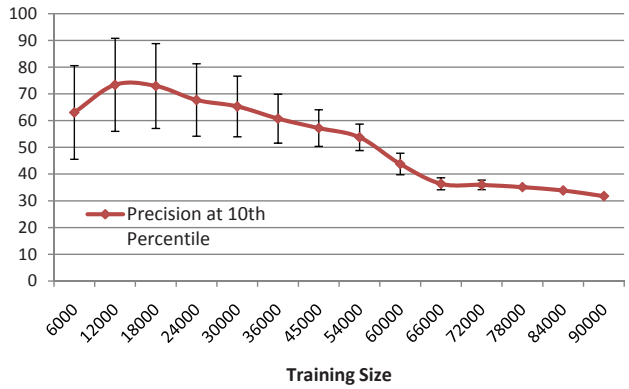

Figure 6: Temporal experiment results

historically processed claims have run through our system, the next goal is to continuously run new claims that come in to the system and score them with the rework likelihood. To show that our system is capable of doing well on future data as well, we need to take into account that the nature of rework can change over time. In data mining literature, this problem has been termed as *concept drift*.

There are two approaches to take into account changes in the nature of rework over time. We can either create features that take into account temporal changes or create models that are trained on specific time periods. We chose to implement the latter approach in our system. The goal is to understand and evaluate how well the model will do on future data based on historical data. It is possible that using all the training data available might inject noise into our classifier since the nature of rework a year ago might be very different. It is also possible that only taking recent data into account won't give us enough training data to have high levels of precision and recall.

In our experiments, we take all the data for the 2 year span. We then fix a size for the time window as the test set and empirically evaluate the optimal preceding time window for the training set. We build models using all time windows before the test set and select the best performing one. The size of this test window is varied from 1 to 8 months. Figure 6 shows the result of one set of temporal experiments where we fix the test window to have 36000 temporally contiguous data points while the training set is varied to include more and more of the history starting from the data closest to the test set. For example, the last point in the figure

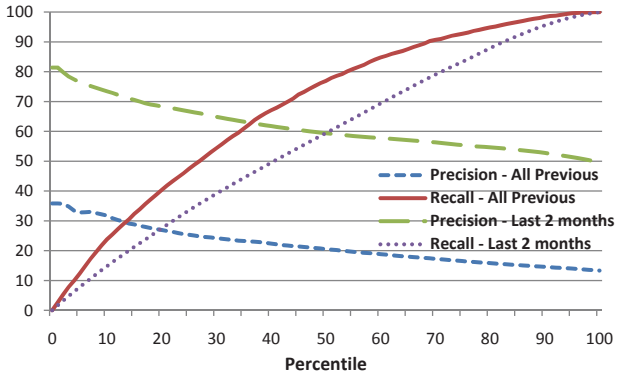

Figure 7: Precision Recall for Temporal experiments

has training size 90,000 which means all the data up to the last 36000 examples is included in the training set and the last 36000 examples make up the test set. The first point on this graph are prediction at 10% results averaged over every 36000 example window as the test set and the 6000 examples just before each test set as the training set. This experiment is designed to show if there is an optimal time window on which to train the classifiers to optimize future performance. Figure 6 shows that creating a training set using recent claims is better than using all the claims.

Figure 7 shows detailed precision recall results for two of the experiments from figure 6. We select the experiment using the first 90000 examples as training and the last 36000 as test (the last point on the graph) and the experiment which results in the highest average precision at top 10% (12000 examples training set and the same 36000 example test set as before). It is interesting to note that the more recent training set does result in better precision than using all the training data but performs worse in terms of recall. We believe that adding more history injects more noise (reducing precision) but gets more coverage (increasing recall). Although these results are specific to the data we are using, we have build an automated experimentation framework in our system that can run all these experiments with different time windows and optimize the metric of choice for the business users. This also reinforces our choice in SVMperf as the learning algorithm due to its fast training time as it allows us to run these experiments quickly to select the best model.

5.6 Does active learning help improve the models?

As mentioned in Section 4.6, we include active learning strategies in our system to obtain labels for the most *informative* examples. Our setting is a natural setting for pool-based active learning since we have a large unlabeled pool of claims in our data. The aim is to select optimal samples from the unlabeled pool in order to improve classifier performance. There are a number of sample selection strategies developed in active learning literature. Initially, we have used simple uncertainly sampling, where the active learner queries those instances for which it is least certain. Figure 8 shows the comparison of random versus uncertainly sampling. The results are not very encouraging but since it's been shown that uncertainly sampling can end up focusing on the outliers in the data (especially with little amounts of labeled data in the beginning of the active learning process),

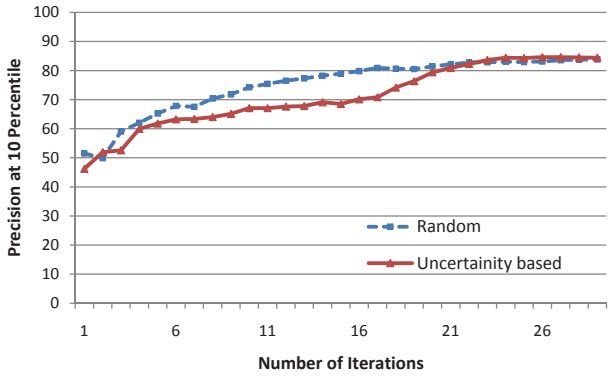

Figure 8: Active Learning Results

we plan to augment it with a density-based sample selection strategy which selects instances with high uncertainty and inhabiting densely populated regions of the unlabeled space. We are currently exploring better active learning approaches in our system.

6. DISCUSSION AND FUTURE WORK

Our experimental results show that the Rework Prevention tool is accurate and allows health insurance companies to save tens of millions of dollars every year. In this section, we discuss some of the alternative problem formulations we considered and experimented with as well as some design choices we made in our system to make it easy to deploy and practical to use in the real-world.

6.1 Alternative Problem Formulations

Although our initial formulation of rework prediction was that of a binary classification problem, we think there are several alternative ways to formulate the problem that lead to interesting research directions. We present some of these alternative formulations below.

Rework Prediction as a Ranking problem: We are using the classification confidence to rank the claims and aim to optimize the precision for top ranked claims. Thus an intuitive alternate formulation is the classical ranking set up where we learn some utility function to rank the claims. The utility function can be based on not only the likelihood of rework but also monetary impact of that rework such as 'recovery amount from overpayment and interest amount for underpayments'. We have done initial experiments for this formulation but the run-time for the system based on RankSVM is impractical for the size of data we are currently working with. We don't present the details of the experiments and results due to space constraints in this paper but we plan to investigate this approach in future work.

Multi-class classification: In general, when a claim is reworked, the auditors document what was wrong with the claim. They attach different error categories after auditing the claim, chosen from a list of predefined ones (totaling about 50). Based on these categories, we can model the rework prediction problem as a multi-class classification problem where we try to predict the error category for the claim. There are two motivations for this formulation: 1) using different error categories makes the learning problem easier as the classifier does not have to learn possibly disjoint functions and can model each category separately. 2) if we can predict the error category, it will help us in explaining to the auditors what is wrong with the claim in a manner similar to what they are used to reporting themselves. We have done initial experiments on this formulation using direct multiclass, one vs. all, as well as ECOC (error-correcting output codes) [3] but the results were not as good as binary classification. We plan to experiment with hierarchical classification algorithms to further evaluate the feasibility of multiclass formulation.

Semi-supervised/Transductive learning: In our problem setting, only a fraction of the claims are labeled as reworked or correct. More than 90% of the claims in typical insurance companies are unlabeled. This allows us to explore semi-supervised modeling techniques such as EM, Co-training, transductive SVMs, and graph-based methods. We did initial experiments using transductive SVMs [6] but the size of our unlabeled data made the run-time of the system impractical. An interesting future work for us is to intelligently sample the unlabeled data and then use the transductive learning or come up with alternative, more efficient formulations of transductive SVMs.

Multi-Instance Learning: In all the formulations we have described so far, the goal has been to predict the entire claim as being rework or not. In general, when a claim is adjusted i.e. Rework we might also know which individual lines of the claim were adjusted. An alternate problem formulation is to predict which line(s) are likely to be adjusted later instead of the entire claim. This helps us in giving a finer-grained 'explanation' to the auditors by pointing out which lines are candidates for Rework. Also we can aggregate the line-level predictions to generate claim level predictions. For the current data set, the labeled data was at the level of claims, and not lines. We can still do line level classification by assigning all the lines in a claim the same label in the training set but classify lines in the test/unlabeled set individually and then aggregating the predictions to generate claim-level predictions. We experimented using this formulation and used different functions for combining the line level recommendation to obtain claim level recommendation such as Max, Average, and Median line scores. We found the Max function to give best performance. More detailed discussion is beyond the scope of this paper.

Creating the training set by assigning the same label to every line in a claim has the risk of introducing label noise which could make it difficult for typical supervised learning algorithms to learn accurately. We can formulate this as a multi-instance classification problem [8] by treating the claim as a bag and the lines being the instances. We assign the claim labels to the bag and then standard multi-instance learning algorithms can be used to learn and classify unlabeled data. Unfortunately, we were not able to experiment with this setting for our data because of impractically long run-time of current MIL implementations (including those in Weka[5])

In general, alternative formulations had issues with scalability and complexity of existing implementations. Rank SVM, Transductive SVM, and the multi-instance classification approaches were extremely interesting to us but ran into scalability issues. We still believe that those directions are promising and will explore them further in future work. The semi-supervised learning direction is also promising but given that most approaches are better suited to scenarios where training data is much more limited than in our domain, it may not be immediately applicable.

6.2 Design Choices for Ease of Deployment & Adoption

We made a variety of design choices when building the rework prevention tool that were motivated by the need to make the entire system easy to deploy across several companies as well as to maximize it's adoption.

- Generic Claims Rework Data Model: We designed a generic data model for claims rework that would allow us to map a specific company's data to our model and automatically do feature selection, model learning and selection. This was extremely important in getting the system running at a new company quickly and make deployment easier.

- Flexible feature construction: Our feature construction process generates over 100,000 features. Our feature construction architecture was designed so that new features can be added very easily and fed into the experimental pipeline using feature selection techniques to select the best model.

- Automated Experimental Framework: We have set up a framework where thousands of models are created by varying training data size, features, time window for the training set, and parameters for the classifier. By automating the design and execution of these experiments we are able to do fast and automated model selection without the need for an expert. We can also optimize different evaluation metrics for model selection.

- Claims Workflow Integration: Although we have focused on the rework prediction and auditor tool in this paper, an important component of our overall system is integration with the claims workflow system. We have designed an interactive tool that, among other functions, allows an audit team manager to look at the claims queue and the predictions of the system and assign claims to the work queues of individual auditors

- Executive Dashboard: Claims rework is a significant problem in the health insurance industry today. For a system to be adopted, it is important to quantify it's benefits to the executives of the insurance company. We have designed a dashboard that monitors the overall performance of the rework/audit department using our system and presents charts showing the trends for overall rework, rework prevented, as well as the monetary benefit accumulated due to this tool over time.

  The features we described above are not research contributions but are important to point out since they are critical to the adoption of a system such as the ones we describe in this paper.

7. CONCLUSION

In this paper, we described our system to help reduce claims processing errors using machine learning techniques by predicting claims that will need to be reworked, and experiment with generating explanations to help the auditors correct these claims, feature selection, concept drift, and active learning to collect feedback from the auditors to improve over time. This Rework Prevention Tool has been developed in conjunction with industry experts from Accenture's Claims Administration group who currently work with most of the large insurance companies in the US. We have applied this system to two large US health insurance companies. We described our system framework, problem formulation, evaluation metrics, and experimental results on claims data from a large US health insurer. We show that our system produces an order of magnitude better precision (hit rate) over existing approaches which is accurate enough to potentially result in over $15-25 million in savings each year for a typical insurer. This in turn would have a large effect on the healthcare costs as well as help make the healthcare process smoother. We also described interesting research problems in this domain as well as design choices made to make the system easily deployable across health insurance companies.

8. ACKNOWLEDGMENTS

We would like to thank Chad M. Cumby, Dmitriy Feferman, Prof Jaime Carbonell and Prof Alexander I. Rudnicky for helpful suggestions and comments. We would also like to thank our partners from Accenture's Health Claims Payer practice particularly Lindsey J. Lizardi and Laura J. Jantzen for insightful domain discussions.

9. REFERENCES

[1] A. Anand and D. Khots. A data mining framework for identifying claim overpayments for the health insurance industry. *INFORMS Workshop on Data Mining and Health Informatics*, 2008.

[2] C. Curry, R. L. Grossman, D. Locke, S. Vejcik, and J. Bugajski. Detecting changes in large data sets of payment card data: a case study. *KDD*, 2007.

[3] T. G. Dietterich and G. Bakiri. Solving multiclass learning problems via error-correcting output codes. *JAIR*, 1995.

[4] U. M. Fayyad, G. Piatetsky-Shapiro, P. Smyth, and R. Uthurusamy. Advances in knowledge discovery and data mining. *AAAI*, 1996.

[5] M. Hall, E. Frank, G. Holmes, B. Pfahringer, P. Reutemann, and I. H. Witten. The weka data mining software: An update. *SIGKDD Explorations*, 2009.

[6] T. Joachims. Transductive inference for text classification using support vector machines. *ICML*, 1999.

[7] T. Joachims. Training linear svms in linear time. *KDD*, 2006.

[8] O. Maron and T. Lozano-Pérez. A framework for multiple-instance learning. *NIPS*, 1998.

[9] McKinsey and Company. Accounting for the cost of us health care a new look on why americans spend more. 2007.

[10] D. Mladenic and J. Brank. Feature selection using linear classifier weights: interaction with classification models. *SIGIR*, 2004.

[11] National Coalition on Health Care. Health care facts: Costs. `http://nchc.org/sites/default/files/resources/Fact%20Sheet%20-%20Cost.pdf`, Date URL checked: 16th May 2010.

[12] K. Shih, Y. han Chang, L. Shih, J. Rennie, and D. Karger. Not too hot, not too cold: The bundled-svm is just right! *ICML Workshop on Text Learning*, 2002.