Bundled Health Services

TEAM
David Rhine, Amos Gwa, and Jacob Granley

CLIENT
Wim De Pril

6/19/2017

TABLE OF CONTENTS

1. INTRODUCTION

The Zero Card works with companies that manage health insurance for their employees internally, and creates bundled health services in order to lower the prices both for the employer and employee. They are able to accomplish this goal by cutting out large amounts of administrative waste and can cut the prices of these procedures by an average of 25%. The Zero Card also offers free procedures to the companies's employees, which incentivizes the employees to use their service. The procedures that The Zero Card covers are episodal in nature, meaning that they only occur once at a time, and are not recurring. The Zero Card does not bundle emergency services, because those treatments are given as urgently as possible, and it does not offer bundled services for recurring treatments, such as dialysis.

The first goal of this project is to design a machine learning algorithm with the capability of flagging the procedures for which The Zero Card does aim to create bundled services. The majority of this project is research-based, in that we work on determining which machine learning algorithms are the most effective and which features we should use to obtain the highest accuracy score. In machine learning, a feature is an individual property of the dataset that a machine learning algorithm has the capability of observing. In our project specifically, the features we use are mostly medical codes used to describe the procedure with a numeric values, information about the patient such as their gender and and their zip code, and place of service codes, numeric values used to encode which part of a hospital the procedure took place.

The second goal of this project is to design a machine learning algorithm that can take the dataset produced by the first algorithm and then create the bundled services. Currently, The Zero Card has algorithms that complete both of these goals, but the reasoning for shifting to a machine learning algorithm is that as the models receive more data they will be able to get increasingly accurate, and if new rules are created for ignoring procedures, the model can respond dynamically and the code does not have to be changed.

2. REQUIREMENTS

2.1 Requirements Overview:

The goal of the project is to research various self-optimizing clustering and classification algorithms and determine the most accurate and performant algorithm. The project is oriented more towards data analysis than writing a software. There will be two types of algorithms: ignoring procedures that can't or won't be bundled by The Zero Card and building bundles. Our requirements can be divided up into functional and nonfunctional requirements as follows.

2.2 Functional requirements:

- A machine learning algorithm with the capability of flagging procedures for which The Zero Card does not offer a bundled service so that they can be ignored by the bundling algorithm.
- A self-optimizing machine learning algorithm that takes in the result dataset from the first algorithm, then creates the healthcare bundles.

2.3 Non-functional requirements:

- Using Python3 to format the data given to us by the client
- Using Pandas to create and handle the data frames
- With respect to time constraints and the scope of this project, we will use the Scikit-learn classifiers rather than building custom machine learning algorithms
- Using Slack for communication amongst the team and with the client

As always, there are a number of risks associated with completing the project. These are the issues we are most concerned about with our project:

- Overfitting data: the trained data only works on the provided training set of data. So, our algorithm might not work on the other data.
- Finding the right language: we were concerned that we may have needed to learn new languages based on the available frameworks for the machine learning. So, the timing might become difficult.
- Linguistics of the healthcare industry: as a group we have limited knowledge of the technicalities within the healthcare industry, so we may need to do some additional research on the vocabulary
- Limited knowledge: We didn't have any data science or machine learning knowledge. So we may have to spend time researching about machine learning and data mining.

In order to tell when we have thoroughly completed the project, as well as give us a concrete goal to strive for, we have decided on our own definition of done, outlined below:

- Propose a machine learning algorithm that effectively identifies services that cannot be bundled, the ignore model
- Propose two distinct clustering algorithms that automate the bundling process
- Implement these four algorithms and analyze their effectiveness
- Delivery: We will deliver to our client a program that ingests his input data, feeds it through our machine learning models, and outputs corrected data. Additionally, we will provide code for each individual model used, and documentation describing each model and its recall and precision

3. SYSTEM ARCHITECTURE

The design of these ignore and bundling models was by far the most important component of our project. Because of this, we have split the design up into two components, System Architecture, and Technical Design. This section gives a brief overview of the design of the model, as well as describe some of the design process we used. The technical design section will give a much more detailed discussion of our current best design, what hasn't worked in the past, and possible future alternatives, as requested by our client.

Shown below in *Figure 1* is a diagram of our model. This model is what our project is hoping to achieve. First, we take raw medical claims data, and run it through an ignore model, which filters out the rows that need to be ignored. Next, it goes through a bundling model, which extracts the relevant bundles, and return the bundled data back to our client

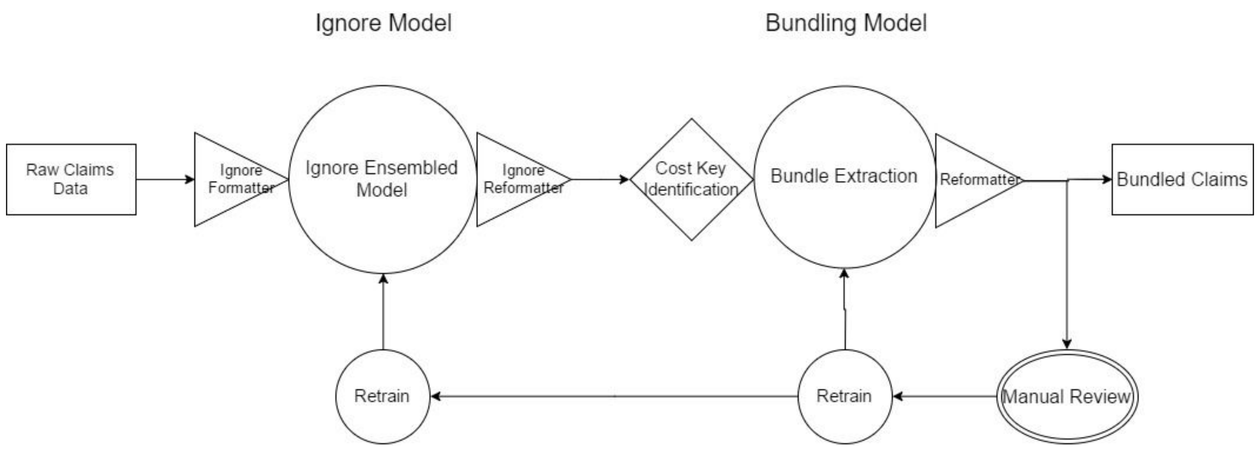

Figure 1: Overview of the two algorithms processing healthcare claims data

For our project, we decided to use Python as it has powerful machine learning and data mining libraries such as scikit-learn and pandas. Also, the language is easy to pick up, and we didn't have to learn a complicated new language to do the machine learning.

In the ignore machine learning model, the client wants an algorithm that filters procedures that cannot be bundled. Then, the output of the provided data is labeled as either 1 or 0, ignored or not ignored. We used various methods in supervised learning to predict the output. The client provided us with around 200,000 samples and we used 75% of the data as training set and 25% as the testing set. The dataset provided by the client contained huge categorical data such as CPT Code, UB Code, etc. These features have 1000 levels or more each, which makes machine learning complicated. In order to deal with this, the categorical data was grouped and then converted to numeric "dummy" representation, so that the classifiers provided by scikit-learn would be able to digest them. From there the data was be fed into our ignore model.

For the ignore model, we decided to use an ensembled model. This model uses random forest classifiers on a variety of different input datasets (each data set has individual sets of features from the complete data set). Each of these individual models makes a prediction about whether or not a line should be ignored, as well as give a confidence in that prediction. Finally, at the end, all of these models' predictions are put into another model, either logistic regression or another random forest. This model evaluates each prediction, and based on its confidence, determines the final prediction.

After the ignore model, the data undergoes more restructuring to be in the correct format for the bundling machine learning algorithm. In our proposed model, it first goes through cost key matching for each claim. Then, based on which cost keys matched, it applies a set of extraction rules. Again, we are using a 75% training data set and 25% test data set.

Finally, after the data has been through both models, it is restructured into a format similar to the original and delivered back to our client. For more information on either of these models, please see the technical design section.

4. TECHNICAL DESIGN

The project entails to produce two algorithms for ignoring and bundling health procedures. In this section, we explain each algorithm in detail.

4.1 Ignore Algorithm

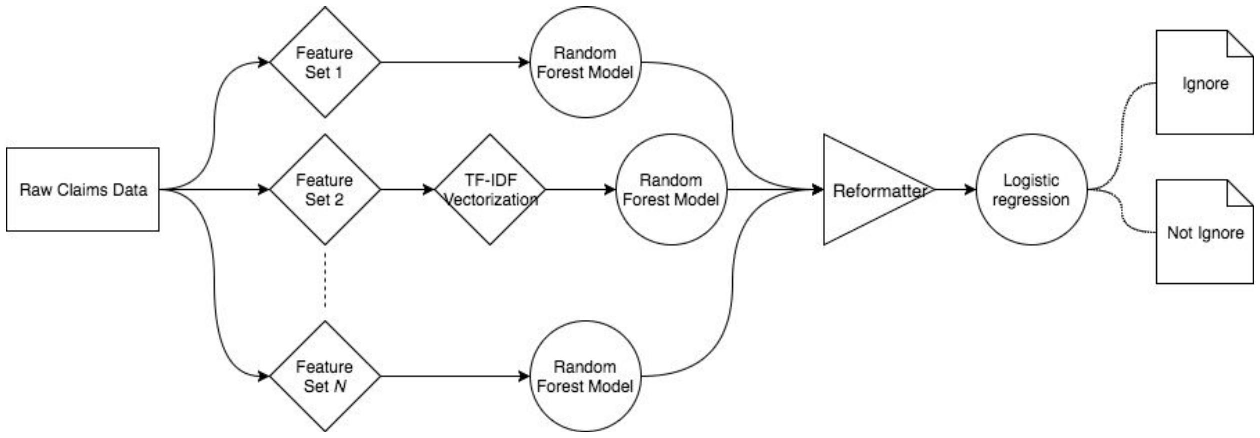

Figure 2: Technical design of the Ignore Algorithm implementation

Ignore model filters out procedures that cannot be bundled. The provided raw claim data were labeled with “ignored” or “not ignored” by the client’s deterministic algorithm. Our Ignore model has to outperform the accuracy of the client’s algorithm which is roughly 98%. We tried different classifiers to find an algorithm that predicts the highest accuracy. The results can be seen in the result section.

Initially, we were using built in classifiers from the Scikit-learn library without tuning the parameters or deliberate selection of the features. The results were not satisfying. Before we started feeding the data into the classifiers, we cleaned up the data into the format that our chosen library can understand as we explained in the system architecture. Then, we selectively chose combination of feature sets that have high importance in the determination of the labels.

In determination of the features sets, we looked at the client’s deterministic algorithm and paired the features based on the rules of the algorithms. For example, the CPT Code was seen paired with other code such as POS Code and UB Code in determining if a procedure should be ignored or not. Thus, we used combination of CPT Code, POS Code, and UB Code as the subset of features. We also used the descriptions of the codes as additional features. Since the descriptions are text, we have to convert the text to numeric representation by vectorizing with Term Frequency Inverse Document Frequency (TF-IDF) technique and filter out the stop words (words with no significance) [1].

Using the selected features, we created set of data to be trained. Then, we used a technique called “boosting” to increase the accuracy of the initial attempt where we used built in classifiers. Boosting is a technique to make weak learners stronger; the prediction and confidence of the algorithms with certain set of features were appended back to the training data and the data was fed into another classifier and so on. This increases the accuracy; however, it overfit the data. Therefore, we use came up with the design as shown in *Figure 2*. We called it stacking technique where we ensemble the output of the each model with certain set of features. Then, we run logistic regression or random forest classifier on each output of the models to classify if a procedure should be ignored or not.

4.2 Bundling Algorithm

4.2.1 Introduction

The purpose of this section is to provide a detailed, high level design for a bundling algorithm. This algorithm will take a list of medical claims data and group these procedures and claims into bundles that The Zero Card supports. It will use machine learning to not only learn how to extract bundles, but also to adapt to new bundles in the future. This will create a feedback loop, so that any time the model receives claims data that it does not know how to process, this data will be flagged for manual review. After it has been manually reviewed, it will be stored, and can be used to retrain the algorithm to work on this new data.

Two implementations of this algorithm will be described, with a discussion of the benefits and risks of each. The first is the model that was discussed with our client, which will use the cost keys associated with each claim to train a rules based machine learning model on the extraction rules for each cost key. The second is the model proposed by Tabor, a student at Galvanize Data Science. This model would use natural language processing (NLP) and TF-IDF vectorization to create string representations of bundles, and from that a traditional machine learning algorithm could be ran [1]. From here on, these implementations will be referred to as the cost key model and the NLP model.

4.2.2 Cost Key Implementation

Model Overview

This model uses cost keys and rules based machine learning in order to create and maintain an optimized set of extraction rules to bundle. It has two main components, an online bundling engine, and an offline review/training feedback cycle. The model is shown below in *Figure 1*, with the ignore model shown for completeness.

Online Bundling Engine

The top part of this diagram is the online bundling engine. This engine could be used in real time to take any amount of correctly formatted medical claims and build bundles. Since this model inherently uses cost keys to do this, pricing these built bundles would be incredibly easy, as each cost key has an associated price. Thus, this online model could deliver priced, bundled claims data. In order to do this, the input data must have already been processed, as well as passed through an ignore model, to ignore lines that are not eligible for bundling. Additionally, this model will be able to flag bundles that it is unsure about for manual review.

This model works by splitting the input csv file into claims, and then identifying which cost keys are applicable to each claim. This cost key identification step creates a new csv where each line is a claim, with a 1 in each column where a cost key is applicable. The bundle extraction engine will have a set of extraction rules stored for each cost key. It will then look at each claim, find the corresponding rules for the matched cost keys, and apply those rules to the claim, extracting individual lines to make bundles. At this time, no bundles are ever created including lines from different claims, so this approach is valid. If in the future, bundles are created that include lines from multiple claims, then this model could be expanded to identify cost keys per patient instead of per claim. For now, leaving it so cost keys are identified per claim makes the bundling engine easier to implement and faster/easier to train.

Offline Training

The initial training and retraining of the bundling model will be offline processes, which must have manual oversight.

The initial training of this model is somewhat complicated, and the field of machine learning used is not nearly as developed as traditional machine learning methods. There are two main ways that this grouping could be accomplished: using supervised clustering algorithms, or rules based machine learning algorithms, or more specifically, learning classifier systems.

A typical supervised clustering algorithm would be difficult to implement on this data, because clustering algorithms use a distance function to group data. The medical claims data that this model will be operating on is pretty much only categorical, and distance functions don’t really make much sense. Instead, a different type of clustering algorithm must be used. There is some research on this type of supervised clustering algorithm such as k-modes [2], agglomerative clustering [3], and other models [4]. It does not appear, however, that any one of these models would give exactly what's wanted without modification. Specifically, these models require highly specialized inputs, and would not be able to handle the different sets of clustering rules based on different cost keys. A separate model could be made for each cost key, but that would be extremely computationally expensive, and it isn't clear that any of these models would be able to perform at the level desired, due to their stochastic nature. Finally, the level of customization required to make these models work for our data requires very advanced mathematics and data science knowledge, which we did not have time to learn in the few weeks required for this project. For these reasons, we decided to not focus on supervised clustering algorithms.

A rules based machine learning approach would still be complicated, but could be much more manageable in terms of knowledge necessary and implementation. Learning classifier systems (LCS's) are models that use a rule discovery component with a supervised learning component in order to identify a set of context dependent rules to classify or cluster. Some advantages of this system are that the rule discovery component can relatively easily be extensively customized to any problem, it works well with highly categorical data, and it can effectively deal with complex extractions rules. Additionally, since it does not map inputs to outputs like normal machine learning, but instead learns rules, this model is better at applying itself to new situations, and also at learning the new rules for new situations.

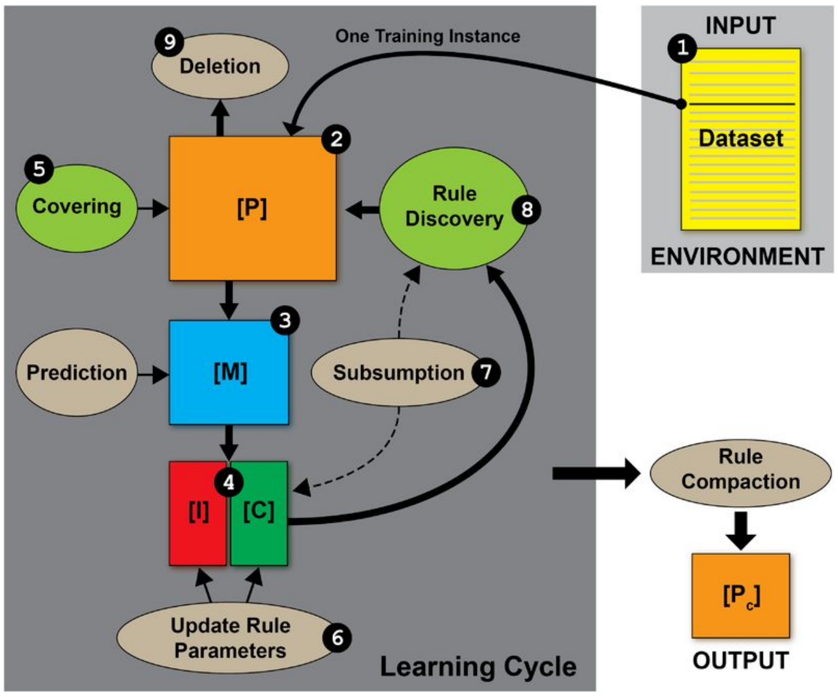

Figure 3: Michigan Style Learning Classifier System

The next major design decision is which kind of LCS to use. There are two major styles of LCS's, Michigan Style (shown above in *Figure 3*) and Pittsburgh Style. Either one could be applicable to our problem, but would have to be utilized a different way. A Michigan Style LCS learns one set of rules, called classifiers, and each one can apply to one context. For our problem, if a traditional LCS were to be used, we would have to train a Michigan Style LCS on every type of cost key. This would be computationally expensive, but should still be manageable, and we have a large amount of resources available. Pittsburgh LCS's, however, learn a rule set for each context. For our problem, the matching cost keys would identify the context, and the Pittsburgh LCS would learn extraction rules for each context. Thus, there would not have to be a separate model for each cost key, making the training process much easier and less expensive. This type of LCS, however, would be harder to design, and there are many fewer working implementations available.

The final design decision for a LCS is whether or not to train for each cost key, or for each combination of cost keys. Since each claim can match with multiple cost keys, this is an important distinction. If a model were to train for an individual cost key, and attempt to apply the set of rules for each cost key, then an extraction rule priority would have to be developed, because otherwise extracting one bundle could interfere with the extraction of the next bundle. The other option would be to train a set of rules for each combination of cost keys. This would be hard, since every combination of cost keys is obviously not present, and may never be present. In this situation, if a new combination of cost keys were to be encountered, each of the individual model's rules could be attempted to be used, and if conflicts arise, then it would be ignored and marked for manual review. Training this way would be advantageous in multiple cost key situations, but designing the training algorithm would be much more difficult. Also, not many claims in the database currently match to more than one or maybe two cost keys, so this might be a problem better dealt with in the future.

After much research, my recommendation on which model to use would be either an existing Michigan Style LCS, trained on each individual cost key, with a set of cost key priorities, or a custom Michigan Style LCS, where each rule is an extraction string similar to the existing extraction rules. This custom LCS would also only have to be trained once on the entire dataset, instead of requiring a separate model for each cost key. It would still require a set of cost key priorities for ideal performance

In the first model each rule would be a string identifying one line in the bundle to be extracted. There are some implementations of LCS's that may be worth exploring, but it is uncertain whether they would be customized enough for this specific situation [5].

The other option would be to make a custom model. Our recommendation would be to then make a custom Michigan Style LCS, where each rule is a string, which tells how bundles are extracted. This would be almost exactly the same as the existing extraction strings. This offers a huge advantage, because the LCS's initial set of rules can be set to the existing cost keys (identification) and extraction rules (extraction). The LCS will take these, create any new rules needed, and modify the existing rules to better fit the data.

Custom Model

If a custom model should be made, the following is an outline on how it could function:

1. Preprocessing
    a. Script for this is already written. It matches claims to cost keys, and establishes a new dataset. The script is called cost_key_bundler.py
    b. This script is designed to be multithreaded, so running on a more powerful computer will give much better (and significantly faster) results
    c. Although cost_key is already given in the dataset given, that is generated from algorithm output, and does not filter out ignores, so it made sense to do everything in one centralized data processing script
2. LCS Training
    a. For each claim in the training dataset, identify all groups that are not the default group. If there is more than one, look for the group that that consistently appears in all claims. This is the group that this rule should be able to extract
    b. Look for any matching rules already existing for that group
    c. If no matching rules exist or they are inaccurate, then apply the covering algorithm
        i. This algorithm is responsible for identifying new rules
        ii. If there is no existing classifier, then a new rule will have to be found.
            1. To do this, it will take all relevant features from the claim lines (cpt code, pos code, etc)
            2. Set the classifiers rule's value equal to whatever the lines' features are. This rule may be too strict (for example the rule actually might allow any pos code not just the one given), but future iterations will take care of this
        iii. If there is an existing rule, but it is not accurate to this situation (for example it says to extract only pos code 20 but in this group pos code 10 was extracted)
            1. Identify which features on the extraction rule are inaccurate
            2. Update these features in the extraction rule to accommodate the new claims features
    d. Rule subsumption
        i. LCS's tend to create redundant rules, so subsumption combines redundant rules into one more general rule
        ii. Simply check to see if there are multiple rules for each cost key, if so see if there is a way to combine any redundant rules
3. Store the set of classifiers and cost keys to a permanent location
    a. Done through pickling normally, would have to be a custom save process for this model

It may be noted that the rule discovery step from *Figure 3* is absent from this list. In traditional LCS's, at the end of a learning cycle (arbitrary number of claims in this case), every rule in the population is evaluated based on its "fitness" (in our case accuracy or f-score). Rules with too low of fitness go through a genetic algorithm where “offspring” rules are produced with characteristics of the parent rule, but with some changed characteristics to increase the fitness. This is a more advanced LCS feature, and while it could improve accuracy, it is very complicated to implement, so it may not immediately beneficial to build.

Retraining

With the majority of machine learning models, retraining would require re-running the initial training algorithm with the new expanded dataset. This isn’t really retraining then, but completely replacing the old model. If new data is frequently coming in, this can become a huge toll on resources to constantly retrain. LCS’s, however, can be trained to handle new data easily, as they just add it to their list of data and complete another learning cycle. With a LCS, it would actually be possible to bring the retraining process online as well, so that as soon as data is manually reviewed, the model would automatically retrain itself on it.

4.2.3 NLP Implementation

Model Overview

This model would be identical in external functionality to the cost key implementation, but instead of learning rules based on cost keys, natural language processing would be used. Cost key may (and probably should) be a parameter in the NLP, but the design is not centered around it. Instead, the model uses TF-IDF vectorization on line and bundle string. It could then use a random forest or another typical model on the TF-IDF vectors.

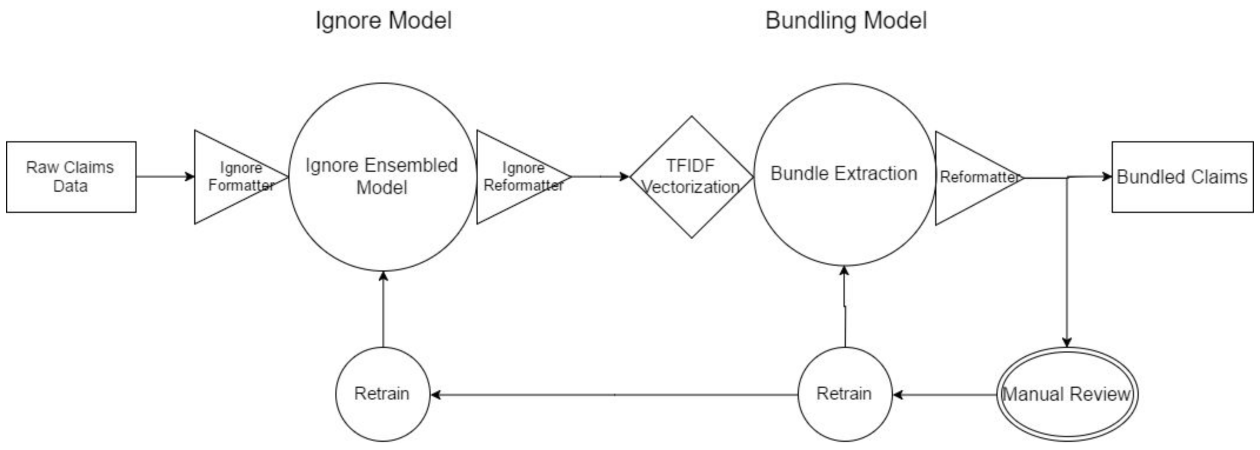

Figure 4

Online Bundling Engine

This model would take in the same output as the cost key model, as outputted from the ignore model. It would then take each line, and concatenate relevant features into one line string. TF-IDF vectorization would then be performed on these line strings, turning each line string into a vector containing the frequency of each term multiplied by the inverse document frequency. This is basically creates a list with a number stored for each word. The number will be higher if the word is deemed more “important”, by logic of being frequent in this line, but infrequent in the entire data set. Thus, components of the vector with high values signify distinguishing words from this line. Next, potential bundles will be created, with the TF-IDF vector-izations of both combined into a single line. All of the possible combinations would have to be considered. This would likely be one of the most complicated and expensive parts of this model. The model would have to not only consider all possible bundle combinations, but it would have to find a mutually disjoint set of bundles that cover as much of the claim as possible. If there are multiple bundles, there really isn’t a way to control which one is extracted first (without extensive customization of the random forest or neural network).

A pre-trained random forest, or ideally neural network, would then be applied to this data. While a random forest would likely give the best initial results, a neural network could give better results, especially in the future. This is because neural networks can be made with hidden layers, and are then categorized in deep learning. Deep learning is extremely complicated, but can give very good results, especially with very large data sets. Deep learning also scales to large data sets much better than traditional models; deep learning accuracies will continue to rise with more data, while traditional models accuracies will plateau after a certain amount of data.

Initial Training

The initial training for the NLP model is somewhat more complex. The process is outlined in the steps below:

1. Turn each line into a line string
2. Apply TF-IDF vectorization to each line string
3. Find every bundle in the data, and create a TF-IDF vector for that bundle, containing the TF-IDF vectors for each line in that bundle. Create a line in a new data set containing this combo TF-IDF vector and label it as TRUE
4. Make a whole bunch of bad bundles, with lines that do not form bundles from the initial data set. Get the combo TF-IDF vectors for these bad bundles, and put them in the new data set, labelling this line as FALSE
5. Train either a random forest or neural network on this new set of data

Unlike with the LCS's, retraining this model would mean having to do the entire process again, and rebuild the model from scratch with the new updated data set.

5. DESIGN DECISIONS

We had a variety of important design decisions to make for our project, including those already mentioned in the design sections above. First, we had to decide on which sklearn model to use for each individual model in our ensembled model. Below is a list of all the models we considered and why we chose to use or not to use each one.

- *K nearest neighbor*
  - Doesn't make sense, no "Distance" with our categorical data
- *Naive bayes*
  - Hard to fit to our highly categorical data
- *SVC*
  - Way too much data, model takes way too long and is too complex. Considered looking into kernel trick but seemed too complicated
- *Logistic Regression*
  - The data is noisy, and it's non-linear
- *Gradient boosted regression trees*
  - About the same accuracy as random forests, but cannot be split amongst multiple cores for testing, so much slower
- *Random forest*
  - Best accuracy and efficiency, avoids overfitting

The next important thing to consider was Efficiency vs Accuracy. Since there really isn't much of a need to be extremely fast or lightweight, so we decided to optimize accuracy, even if the model takes a huge amount of memory and processing power. For this reason, most of our code is now best run on mines's isengard server or something similar.

Next, we had to decide how to create an ensemble of the models. We first decided to use boosting, which appends each model's output to the previous ones, and the next model uses it as an input, making a progressively better model as it goes. This, however, seemed to cause overfitting, so we decided to switch to stacking, which is where each model sends its vote to the end, where another model is applied

When using these assembled models, we had to choose which features or combinations of features to use for each individual model. We considered a wide range of possibilities including vanilla features, combination features (combos of multiple features made into one), NLP, encoding multi line features, taking out features that weren't useful, and grouping huge categorical features to make the data more manageable

Finally, we had to decide how to do the more complicated parts of the project, such as increasing accuracy and doing machine learning across multiple lines. One of the ways we could have achieved this is deep learning. Deep learning could have been very powerful, but the algorithms are extremely complex, involving advanced mathematics, and have to be extensively customized. For this reason we decided that it was outside the scope of this project. Instead, we looked at some other technologies such as rules based machine learning, which can still be powerful, but are not nearly as complex as deep learning.

For a lot of our design decisions for the bundling model, since we were not making an actual implementation, we decided to list the options and discuss the advantages and disadvantages of each instead of actually making the decision. This way our client can look at the options and choose for himself which route to pursue in the future.

6. RESULTS

In the end, we ended up being able to deliver multiple designs for both parts of the model, but were only able to get a working implementation for the ignore model. Along the way, both us and our clients realized that the problem we were trying to solve was much more complicated than we originally anticipated, so although we didn't accomplish everything that we originally set out to do, we still did a large amount of useful work for our client. Specifically, we had the following unimplemented features/goals:

- Did not get the ignore model up to 99% accuracy
- No optimized implementation of the bundling model, only a the cost key matching algorithm and a detailed but theoretical design
- Couldn't get existing LCS's to work on our data
- The starting goal of a larger application that ignores and bundles the claims and has a machine learning feedback cycle

Our client, however, was not looking for just a working implementation, but also wanted to know all the things we tried; what worked and what didn't. Thus, we have compiled the following list of all the different things we tried when implementing the ignore model:

6. 1 Classifier Performance

Classifier	Accuracy	Comment
Random Forest w/ cpt categorization	0.9884	Used algorithm output
Random Forest w/out cpt categorization	0.9880	Used algorithm output
Gradient Boosted Regression Trees w/ cpt categorization	0.9887	Used algorithm output
Logistic Regression	0.9878	Used algorithm output
Custom Ensembled Model	0.9899	Used algorithm output
Random Forest	0.9345	
Gradient Boosted Regression Trees	0.9215	
Logistic Regression	0.9064	

Table 1 : List of all individual classifiers and their accuracy

6.2 Model Performance

Model	Accuracy	Comment
CPT Description	0.7913	NLP using TF-IDF
CPT Mod Description	0.7160	NLP
POS Code Description	0.7258	NLP
UB Description	0.7125	NLP
Cpt Code	0.7986	
Diagnosis Code	0.7626	
Allowed Amount	0.8548	
All but cpt and diagnosis, with added multi line features	0.9246	
1st Ensembled Model	0.9380	Used boosting
Combo ensembled Model	0.9587	2 models with combo features

Table 2: List of individual models and their accuracy

We also explored deep learning and using LCS's for this, but did not have time to develop any models using these technologies. Also, note that the high accuracies initially achieved were using our clients algorithm output. Our client wanted us to steer away from using the algorithm output, so we sacrificed the increased accuracy for better usability

If we could, in the future, we would like to continue working on a multitude of things, including continuing researching LCS's, building a working implementation of the bundling model, improve the accuracy of the ignore model, and do more research on more advanced techniques, such as deep learning or neural networks.

Nonetheless, we have still learned some very valuable skills while working on this project. In addition to learning the in's and out's of machine learning, especially in python, we also learned about the agile working environment and gained an appreciation of scrum meetings, learned various networking skills by going to data science meetups and Galvanize Data Science seeking help, figured out how to set an agenda and run meetings, how to organize effectively through email and slack, as well as learning some more advanced concepts such as LCS's, deep learning, and NLP. Overall, we've definitely learned a lot from this project, and hopefully were able to give our client some useful designs and information as well.

7. APPENDICES

7.1 Useful Links

The machine learning course:
https://www.coursera.org/learn/machine-learning/home/welcome

Discussion of machine learning algorithms
http://www.kdnuggets.com/2016/08/10-algorithms-machine-learning-engineers.html

We want classification algorithms:
https://en.wikipedia.org/wiki/Statistical_classification

Machine learning in go:
http://www.infoworld.com/article/3121694/artificial-intelligence/googles-go-language-ventures-into-machine-learning.html
https://github.com/ryanbressler/CloudForest

Medical Coding:
https://en.wikipedia.org/wiki/Current_Procedural_Terminology
http://www.medicalbillingandcoding.org/medical-billing-coding/
http://www.vbh-pa.com/provider/info/claimsdept/UB04_Type_of_Bill_Codes.pdf

Deep learning / feature learning:
https://en.wikipedia.org/wiki/Feature_learning
http://machinelearningmastery.com/what-is-deep-learning/
http://www.deeplearningbook.org/

Machine learning rule learning
https://en.wikipedia.org/wiki/Rule-based_machine_learning
https://www.hindawi.com/archive/2009/736398/abs/
Learning classifier system
https://en.wikipedia.org/wiki/Learning_classifier_system

Saving model: http://scikit-learn.org/stable/modules/model_persistence.html

8. REFERENCES

[1]“Tf-idf :: A Single-Page Tutorial - Information Retrieval and Text Mining,” Tf-idf :: A Single-Page Tutorial - Information Retrieval and Text Mining. [Online]. Available: http://www.tfidf.com/. [Accessed: 10-Jun-2017].
[2]“nicodv/kmodes,” GitHub, 06-Jun-2017. [Online]. Available: https://github.com/nicodv/kmodes. [Accessed: 16-Jun-2017].
[3]“2.3. Clustering,” 2.3. Clustering — scikit-learn 0.18.1 documentation. [Online]. Available: http://scikit-learn.org/stable/modules/clustering.html. [Accessed: 16-Jun-2017].
[4]N. Zeidat, C. F. Eick, and Z. Zhao, “Supervised Clustering: Algorithms and Application.”.
[5]R. J. Urbanowicz, G. Bertasius, and J. H. Moore, “An Extended Michigan-Style Learning Classifier System for Flexible Supervised Learning, Classification, and Data Mining.”