Will You Be in Hospital Next Year: Leveraging Machine Learning in Improving Healthcare

Xiang Peng **Wentao Wu** **Jia Xu**

Department of Computer Sciences
University of Wisconsin-Madison
{csxpeng, wentaowu, jiaxu}@cs.wisc.edu

Abstract

According to some recent study from the American Hospital Association, more than 71 million individuals in the United States are admitted to hospitals each year. However, a considerable number of hospital admissions turn out to be unnecessary, and studies have shown that much money (e.g., over $30 billion in 2006) has been wasted on such unnecessary admissions.

In this paper, we study the problem of predicting how many days a patient will spend in a hospital in the next year. Such prediction is valuable, since once known, health care providers can develop new care plans and strategies to reach patients before emergencies occur, thereby reducing the number of unnecessary hospitalizations.

We focus on leveraging machine learning algorithms in this study. Specifically, we model the predictive problem in two different ways, either as a multi-class classification task, or as a regression task. We discuss the details of the algorithms we employ for each aspect, and present results from an extensive experimental evaluation.

1 Introduction

Today, more than 71 million individuals in the United States are admitted to hospitals each year, according to the latest survey from the American Hospital Association. However, a considerable number of hospital admissions turn out to be unnecessary, which causes huge wastes of money. For example, studies have pointed out that in 2006, over $30 billion was spent on unnecessary hospital admissions (HPN 2011). As a result, understanding how long a patient will spend in a hospital each year can be beneficial. Based on such knowledge, health care providers can develop new care plans and strategies to reach patients before emergencies occur, thereby reducing the number of unnecessary hospitalization.

Recently, the Heritage Provider Network (HPN) organizes an online competition (HPN 2011) for this predictive problem. Although the competition is still ongoing, the best result from the top teams on the leaderboard is around 45% Root Mean Squared Logarithmic Error (RMSLE, see Section 2), which is not very promising. As we will discuss in this paper, the problem turns out to be challenging in practice, for reasons from both the data we have and the problem itself. In short,

- Unlike most of the machine learning problems for which training data is scarce, the dataset (see Section 2) used in this problem consists plenty of records. However, the set of features used to characterize each record is limited, much less than the number of records we have. On the other hand, due to the highly skewed distribution of the target attribute (i.e., the number of days a patient will stay in a hospital) values (see Section 4.1), it is difficult to find indicative features that are highly correlated with the target attribute, even after applying some feature selection algorithms.
- Instead of predicting *whether* a patient will be in hospital next year, the problem requires to predict the *exact* number of days he/she will stay. Basically, this is a *regression* problem since we need to predict some *continuous* attribute. It is generally believed in machine learning research community that regression problems are more difficult than *classification* problems, for which we are only supposed to answer yes or no. In practice, fewer number of existing machine learning models can be applied to regression problems than classification problems.

In this paper, we leverage machine learning algorithms in developing predictive models for this problem. Our goal, however, is not to beat the existing results by getting some higher precision. Instead, we are interested in the performance of each individual machine learning algorithm, and try to understand why some approaches work better than the others. Therefore, we evaluated several existing models that are well accepted as among the *best* ones for general machine learning applications, and we gave detailed comparison and analysis for their performance. We think the lessons learned from this study can provide useful insights for future researches on this problem.

The rest of the paper is organized as follows. Section 2 analyzes the problem in detail. We focus on discussing the dataset we have, and metrics that could be used to measure the performance of the models. Section 3 describes the machine learning algorithms we will use in our experiments. We also discuss related issues such as the features employed to represent each data record. Section 4 reports the results

from our experimental evaluation, and Section 5 gives them a thorough analysis. We conclude the paper in Section 6.

2 Problem

In this section, we first describe the dataset in hand, and then discuss possible problem settings and performance metrics.

2.1 Dataset

The dataset is publicly available at (HPN 2011). Here we give it a brief overview. As shown in Appendix A, there are 5 tables in the dataset: *Members*, *Claims*, *Drug Count*, *Lab Count*, and *Outcome*:

- The *Members* table contains basic information such as member ID and gender for each patient. It also contains the member's age when his first claim was made.
- The *Claims* table contains information describing each claim. Except for attributes like the time and place that the claim was made, it also includes other important features such as the *primary condition group* and the *Charlson index* of the patient.
- The *Drug Count* table includes information about prescription drugs.
- The *Lab Count* table includes information about laboratory and pathology tests.
- The *Outcome* table contains information about the days the patient stay in hospital in the two years Y1 and Y2.

2.2 Problem Analysis

The predictive problem is defined as to use a patient's record in the year Y1 to predict how many days he/she will stay in hospital in the year Y2. At a first glance, it is quite natural to model the problem as a *regression* task, since we are trying to predict *numerical* values instead of *categorical* labels. However, as shown in the *Outcome* table (see Table 15 in Appendix A), since the possible number of days that a patient can stay in hospital is limited to up to 15, it is also reasonable to model the problem as a *multi-class classification* task. In this paper, we will investigate learning algorithms in both ways, and we will compare the difference in performance when we view the problem in these two different aspects.

2.3 Performance Metrics

In machine learning literature, several performance metrics have been frequently used to measure the performance of the predictive models. For instance, the well-known off-the-shell Weka package (Hall et al. 2009) uses the following four metrics to indicate the errors caused by incorrect prediction:

- *Mean Absolute Error* (MAE), defined as:

$$E_{MAE} = \frac{1}{n}\sum_{i=1}^{n}|P_i - T_i|.$$

- *Root Mean Squared Error* (RMSE), defined as:

$$E_{RMSE} = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(P_i - T_i)^2}.$$

- *Relative Absolute Error* (RAE), defined as:

$$E_{RAE} = \frac{\sum_{i=1}^{n}|P_i - T_i|}{\sum_{i=1}^{n}|T_i - \bar{T}|}.$$

- *Root Relative Squared Error* (RRSE), defined as:

$$E_{RRSE} = \sqrt{\frac{\sum_{i=1}^{n}(P_i - T_i)^2}{\sum_{i=1}^{n}(T_i - \bar{T})^2}}.$$

Here P_i and T_i are the *predicted* value and *actual* value of the testing example i, respectively, while $\bar{T} = \frac{1}{n}\sum_{i}^{n}$ is the *mean* of the actual values of the testing examples.

Different performance metrics have different emphasis. Intuitively, *squared* errors will emphasize more on the predictions for the *outliers*, while *relative* errors try to take the inherent *variance* within the test examples into consideration. Therefore, they can give us different insights into the performance of a learning algorithm. In this study, for each algorithm we investigated, we will report all these performance metrics. Meanwhile, we also leverage a fifth metric named *Root Mean Squared Logarithmic Error* (RMSLE), which is defined as

$$E_{RMSLE} = \sqrt{\frac{1}{n}\sum_{i=1}^{n}[\ln(P_i + 1) - \ln(T_i + 1)]^2}.$$

The only difference of E_{RMSLE} from E_{RMSE} is that it applies the logarithmic function to both the predicted and actual values before calculating the root mean squared error. The "+1" here just ensures that the logarithmic function is always applied to a value that is greater than 0.

The effect of introducing the logarithm function is to balance the emphasis on *small* and *big* predictive errors. For example, consider a testing example with actual value 0. Suppose the predicted value is 15 (i.e., the upper bound of the target value in our predictive problem), and assume this is the only example we have in our testing set (i.e., $n = 1$ in the above formulas). Then $E_{MAE} = E_{RMSE} = 15$, while $E_{RMSLE} = 2.77$. On the other hand, if the predicted value is 1, then $E_{MAE} = E_{RMSE} = 1$, while $E_{RMSLE} = 0.69$. As we can see, metrics like E_{MAE} and E_{RMSE} are dominated by *big* predictive errors, while E_{RMSLE} is less affected by such kind of errors. Since *big* predictive errors often come from a small fraction of outliers in the testing data, E_{RMSLE} can justify the performance of the algorithm more fairly. Due to this reason, E_{RMSLE} is also used as the performance metric in the official online competition (HPN 2011).

3 Method

We describe our method in this section. The two major problems of leveraging machine learning in an application

are *feature selection* and *model selection*. Feature selection picks up a set of *relevant* features that are (ideally) statistically correlated with the target attribute to be predicted, and model selection chooses a good learning algorithm with respect to the characteristics of the available features and the constraints on available computational resources. Therefore, we will focus ourselves on discussing these two problems.

3.1 Feature Selection

The goal of feature selection in our problem is to select a set of relevant features that can effectively characterize each patient. Since the information about a patient is scattered across the five tables, we first join them based on the *MemberID* attribute in each table. We load the data into the MySQL database, and then issue a SQL query (see Appendix B) to perform this join operation. The result is a big table with each row containing the information about one claim made by one patient. However, since our prediction is for each patient, not for each claim, we need some way to aggregate the information from the claims made by the same patient. We next describe our aggregation method in detail.

While some attributes such as *Sex* are single-valued, which can be directly used as features, most of the other attributes are multi-valued. For instance, there are 12 possible values for the attribute *Specialty*. For such attributes, we use multiple features to represent one single attribute, with one feature corresponding to one possible value of the multi-valued attribute. In addition to the attributes that can be directly obtained from the data set, we also add new features based on aggregated information such as the number of claims made by a patient. Table 1 to 3 list the set of features we used in detail.

Here, the features are divided into 3 groups, with the first group mostly including features from individual patient's personal information, the second group including features from the information of the place whether the treatment is served, and the third group mostly including features aggregated from claims made by each patient. The three groups contain 86, 22, and 30 features, respectively, and therefore we have 138 features in total. As shown in the tables, features can be either *binary* (denoted by BIN) or *numerical* (denoted by NUM). A binary feature can only take the vale 0 or 1, while a numerical feature can take any real number.

3.2 Model Selection

As mentioned in Section 2.2, our predictive problem can be modeled either as a regression task or a multi-class classification task. We discuss algorithms in both categories, respectively.

When deciding the algorithms to be used, besides its accuracy on prediction, we also need consider two other important factors. First, the conditions assumed by the algorithm should roughly hold on the available dataset. For example, Naive Bayes assumes that, given the target attribute, the features are independent of each other, while such assumptions may not hold on a particular dataset. Second, the algorithm should be relatively efficient, with respect to the given restriction on computational resources such as the available amount of memory. Considering the large amount of data we have (around 76,000 members and 780,000 claims for the year Y1), efficiency is an important factor affecting our choice on the learning algorithms.

Table 1: Individual Features

ID	Name	Type	Description
1 -10	*Age*	BIN	10 features indicating whether the age at the first claim is from 0 to 9, from 10 to 19, from 20 to 29, from 30 to 39, from 40 to 49, from 50 to 59, from 60 to 69, from 70 to 79, above 80, or missing.
11 -13	*Gender*	BIN	3 features indicating whether the gender is male, female, or missing.
14	*ClaimTrunc*	BIN	Whether the claims are truncated.
15 -60	*PCG*	NUM	45 counts for Primary Condition Groups of ICD-9-CM Codes, plus 1 count for missing PCG.
61 -78	*PG*	NUM	17 counts for Procedure Groups CPT Codes, plus 1 count for missing PG.
79	*N_Claim*	NUM	# of claims made.
80	*N_Provider*	NUM	# of providers.
81	*N_Vendor*	NUM	# of vendors.
82	*N_PCP*	NUM	# of PCP's.
83	*N_Spec*	NUM	# of specialty.
84	*N_Svc*	NUM	# of places of service.
85	*N_PCG*	NUM	# of PCG's.
86	*N_PG*	NUM	# of PG's.

Table 2: Service Features

ID	Name	Type	Description
87 -99	*Specialty*	NUM	12 counts for Specialty Groups of, plus 1 count for missing specialty.
100 -108	*PlaceSvc*	NUM	8 counts for Place of Service Groups, plus 1 count for missing value.

Multi-class Classification For the purpose of multi-class classification, we choose to use Support Vector Machines (SVMs) and Random Forests (RFs) as our learners.

- SVM (Cortes and Vapnik 1995) is well recognized as among the classifiers with best performance in general-purpose binary classification applications. However, using standard Quadratic Programming (QP) approaches to train SVM is too slow. For efficiency purpose, we use the Sequential Minimal Optimization (SMO) proposed in (Platt 1999) to train the SVM. The multi-class classification problem is reduced into multiple binary classification problems by leveraging a *one-versus-all* approach, namely, we build binary classifiers that distinguish be-

Table 3: Aggregation Features

ID	Name	Type	Description
109 -113	*LenOfStay*	NUM	Minimum, maximum, mean, range, deviation for the length of stay.
114	*N_LOS_M*	NUM	Counts for missing *LenOfStay*.
115	*N_LOS_K*	NUM	Counts for known *LenOfStay*.
116	*N_LOS_S*	NUM	Counts for supressed *LenOfStay*.
117 -121	*DSFS*	NUM	Minimum, maximum, mean, range, deviation for *DSFS*.
122 -126	*Charlson*	NUM	Minimum, maximum, mean, range, deviation for *Charlson* index.
127 -131	*LabCount*	NUM	Minimum, maximum, mean, range, deviation for *LabCount*.
132	*N_LabRec*	NUM	# of lab records.
133 -137	*DrugCount*	NUM	Minimum, maximum, mean, range, deviation for *DrugCount*.
138	*N_DrugRec*	NUM	# of drug records.

tween one of the classes to the rest. Classification is then done by a *winner-takes-all* strategy, in which the classifier with the highest output function assigns the class.

- RF (Breiman 2001) is an *ensemble* classifier that consists of a lot of decision trees and outputs the class by leveraging a simple *voting* strategy, i.e., it will output the class that receives the most number of votes from individual trees. The basic idea behind RF is to use *bagging* and random selection of features (as roots of subtrees) to increase the diversity of the trees so built. RF is also recognized as one of the best learning algorithms available. Another reason for us to choose it is its efficiency. RF can still run very fast even on datasets containing tens of thousands of training examples, which may be prohibitive for many other classifiers.

Regression Regression is generally admitted as more difficult than classification problems, and there are fewer mature approaches that are generally recognized as powerful. Therefore, we leverage two simple models with regarding to linear and nonlinear aspects of the problem. For the linear case, we use simple linear regression, while for the nonlinear case, we use regression trees. Regression trees share similar idiocrasy as decision trees, with the difference that they put numerical values in their leaves, instead of categorical labels as in decision trees. We consider both Reduced Error Pruning (REP) trees (Quinlan 1987), which is a variant of standard regression trees plus some pruning strategy, and Model trees (Quinlan 1992), which takes the idea to replace some subtrees with linear regression models.

Ensembles In addition to using a *single* model to do prediction, we also use ensembles of models to build more powerful predictive models. In particular, we focus on Boosting (Freund and Schapire 1995). The idea of boosting is to build a series of classifiers where subsequent classifiers built are tweaked in favor of those training examples misclassified by previous classifiers. The output is combined from individual classifiers with a *weighted-voting* strategy, where the output from each classifier is weighted according to its performance on the training data. *Bagging*, which is another popular ensemble approach, has been exploited in Random Forests, and hence we do not consider it again here when assembling the individual models we discussed in this section.

4 Experiments

We conduct extensive experiments to compare the performance of the models discussed in the previous section. In this section, we report and analyze the results we obtained.

4.1 Statistics of Data

As mentioned in previous sections, we use the data in Y1 as our training set, and use the data in Y2 as our testing set. In total, we have 76,038 records (i.e., members) in the training set, and 71,435 records in the testing set. Table 4 further shows the distribution of members, according to the days they are admitted in hospital in Y1 and Y2, respectively.

Table 4: Distribution of members

Days	# of Members (Y1)	# of Members (Y2)
0	64,269	60,706
1	4,835	4,464
2	2,366	2,182
3	1,453	1,429
4	977	842
5	565	528
6	373	287
7	256	218
8	173	143
9	148	115
10	106	103
11	80	65
12	73	62
13	61	50
14	42	23
15	261	218

As we can see from Table 4, the data exhibits two important characteristics. On one hand, the distributions of members in Y1 and Y2 are very similar, which means the number of days patients stay in hospital are statistically stable. This is good news if we wish to build some statistical model to do the prediction. However, on the other hand, the distribution of members in each year is highly skewed, with a dominant value covering most of the records. Here, each year, almost 85% of the patients are actually not admitted in hospital. This skewness, as we will see in later sections, raises great challenge for existing state-of-the-art machine learning algorithms.

4.2 Baseline Approaches

We leverage two naive approaches as our baselines. The first approach blindly predicts the target attribute with the most frequent value appearing in the training data. As shown in Table 4, among the 76,038 members with at least one claim in the year Y1[1], 64,269 (about 84.5%) of them were actually not admitted into hospital. Therefore, this approach will predict everything as 0, and we name it as *AllZero*. The second baseline approach simply uses the number of days a patient stay in hospital in the year Y1 as its prediction for the number of days this patient will stay in hospital in the year Y2. We name this approach as *SameAsY1*.

Table 5: Results of baseline approaches

Model	MAE	RMSE	RAE	RRSE	RMSLE
AllZero	0.460	1.642	0.591	1.042	0.520
SameAsY1	0.719	1.980	0.925	1.256	0.611

Table 5 shows the performance of the baseline approaches. *SameAsY1* performs worse than *AllZero*. This indicates that, although the distributions of members in Y1 and Y2 are close, for an individual patient, the days he/she stays in hospital each year can be quite different.

4.3 Results of Single Models

We extract features in the way described in Section 3.1, and we implement the models discussed in Section 3.2 by leveraging libraries from Weka. For support vector machines, we use the RBF kernel (i.e., Gaussian kernel). Parameters for each algorithm are set to be their default values in Weka.

Table 6 to 8 shows the results for single models without ensembles. Here, *SVM* is short for support vector machine, *RF* is short for random forest, *LR* is short for linear regression, *REP* is short for regression tree, and *M5P* is short for model tree. As discussed in Section 3.1, *SVM* and *RF* are multi-class classification algorithms, while *LR*, *REP* and *M5P* are regression algorithms. For SVM, we are not able to run it over the entire training set, and the results reported are based on a smaller training set consisting of 10,000 randomly sampled records.

Table 6: Results of single models (86 features)

Model	MAE	RMSE	RAE	RRSE	RMSLE
SVM	0.438	1.593	0.588	1.040	0.506
RF	0.449	1.610	0.603	1.051	0.509
LR	0.479	1.522	0.644	0.994	0.487
REP	0.479	1.535	0.644	1.002	0.488
M5P	0.492	1.529	0.661	0.998	0.490

We have some interesting observations from Table 6 to 8. First, regression algorithms perform better than multi-class classification algorithms. One possible interpretation for this may be that, in this problem, the target attribute is continuous by its semantic, although it only has discrete values. Multi-class classification algorithms usually treat target classes as completely irrelevant, which may be not true here. Second, the algorithms within each category (i.e., regression or multi-class classification) have close performance. Third, increasing the number of features has different effect for different algorithms. For *SVM* and *RF*, it seems that they are insensitive to the new features added. For *LR*, adding features slightly improves its performance. For *REP* and *M5P*, however, introducing new features actually downgrades their performance. Fourth, quite surprisingly, *LR* (i.e., linear regression), the simplest model we used, outperforms all the other more complicated models.

[1]Recall that we use the records from Y1 as training set, and the records from Y2 as testing set.

Table 7: Results of single models (108 features)

Model	MAE	RMSE	RAE	RRSE	RMSLE
SVM	0.438	1.593	0.588	1.040	0.506
RF	0.447	1.600	0.600	1.045	0.509
LR	0.479	1.521	0.644	0.993	0.487
REP	0.484	1.530	0.651	0.999	0.491
M5P	0.479	1.521	0.644	0.993	0.487

Table 8: Results of single models (138 features)

Model	MAE	RMSE	RAE	RRSE	RMSLE
SVM	0.438	1.593	0.588	1.040	0.506
RF	0.444	1.599	0.597	1.044	0.508
LR	0.482	1.513	0.647	0.988	0.485
REP	0.482	1.529	0.647	0.998	0.491
M5P	0.494	1.537	0.664	1.003	0.494

4.4 Results of Ensembles

We further tested ensemble algorithms. Since regression algorithms perform better than multi-class classification algorithms, we focus on applying Boosting to regression algorithms. Specifically, we use *Additive Regression* (Friedman 1999) (i.e., *Stochastic Gradient Boosting*), which is a variant of Boosting that can be applied to regression models. Table 9 summarizes the results.

Table 9: Results of additive regression

Model	MAE	RMSE	RAE	RRSE	RMSLE
86 Features					
LR + AR	0.479	1.522	0.644	0.993	0.487
REP + AR	0.480	1.528	0.645	0.997	0.486
M5P + AR	0.489	1.524	0.657	0.995	0.489
108 Features					
LR + AR	0.480	1.521	0.645	0.993	0.487
REP + AR	0.479	1.524	0.644	0.995	0.487
M5P + AR	0.479	1.520	0.644	0.992	0.487
138 Features					
LR + AR	0.482	1.513	0.647	0.988	0.485
REP + AR	0.473	1.537	0.635	1.003	0.487
M5P + AR	0.482	1.513	0.647	0.988	0.485

Basically, boosting performs no worse than the original models without boosting, as expected by the theoretical justification of ensemble algorithms that they usually outperform underlying weaker learners. Nonetheless, the perfor-

mance gain obtained by leveraging boosting seems not remarkable. Actually, in some cases, it does not enhance performance at all.

Finally, we test ensembles with 4 different underlying models: *RF*, *LR*, *REP*, and *M5P*. This gives us 0.482 RMSLE, which is the best performance we obtained so far.

5 Discussion

Based on the results from our experiments, the best performance is about 0.482 in terms of RMSLE. This sounds not very exciting, since the baseline approach *AllZero* already gives performance at about 0.520. On the other hand, however, since the best result known so far is 0.454 (HPN 2011), our result seems already not bad.

The problem, of course, is then, why is it so difficult to improve the performance over the baseline? Actually, this question is not difficult to answer. As we have mentioned in Section 4.1, according to Table 4, around 85% of the patients will finally not be admitted in hospital. This is the same for both the training and testing data. As a result, if we measure the performance in terms of accuracy instead of RMSLE, *AllZero* can predict 85% of the patients correctly, which is already an amazing number. Therefore, a classifier that wants to beat the baseline has to predict correctly for more than 85% of the patients.

Recall that the two key steps in developing a good predictive program is to first pick a set of good features and then pick a good machine learning algorithm. Hence, to enhance performance, a natural question to ask is whether our features and learning algorithms can be improved. We next discuss these two factors one by one.

5.1 Features

With respect to the results from the previous section, different sets of features do affect the performance, although the effect may be either positive or negative, depending on the specific learning algorithms employed. As a result, it is possible that by carefully picking a subset of features, we can achieve better performance.

We developed a simple *greedy* feature selection algorithm (see Algorithm 1). Basically, it starts with a set S of *seed* features, and repeatedly iterates over all the features that are not picked yet. During each iteration, it tries to find some feature f that minimizes RMSLE of the model $\mathcal{M}$ on features $F \cup \{f\}$ (line 5 to 14). If the minimum RMSLE ϵ_{min} is less than the RMSLE ϵ from $\mathcal{M}$ on current features F, f is then included into F (line 15 to 17). This procedure terminates when F does not change any more.

Table 10 gives some results by leveraging this simple feature selection algorithm on different sets of seed features. Here, linear regression is used as the specific machine learning model for the purpose of this study on feature selection. The numbers in S and F are the ID's of the features listed in Table 1 through 3. Due to space limitation, for performance metrics, we only report RMSLE by leveraging linear regression with the set of features F on the testing set.

As can be seen, starting from different seed features yields different set of final features, and linear regression can have

Algorithm 1: Feature Selection

Input: S, the set of seed features; $\mathcal{M}$, the machine learning model
Output: F, the set of features minimizing RMSLE on $\mathcal{M}$

1 Train $\mathcal{M}$ with features S;
2 Compute RMSLE ϵ on $\mathcal{M}$;
3 $F \leftarrow S$;
4 **repeat**
5 $\quad \epsilon_{min} \leftarrow \infty$; $f_{min} \leftarrow null$;
6 $\quad$ **foreach** *feature* f **do**
7 $\quad\quad$ **if** $f \notin F$ **then**
8 $\quad\quad\quad$ Train $\mathcal{M}$ with features $F \cup \{f\}$;
9 $\quad\quad\quad$ Compute RMSLE ϵ' on $\mathcal{M}$;
10 $\quad\quad\quad$ **if** $\epsilon' < \epsilon_{min}$ **then**
11 $\quad\quad\quad\quad$ $\epsilon_{min} \leftarrow \epsilon'$; $f_{min} \leftarrow f$;
12 $\quad\quad\quad$ **end**
13 $\quad\quad$ **end**
14 $\quad$ **end**
15 $\quad$ **if** $\epsilon_{min} < \epsilon$ **then**
16 $\quad\quad$ $\epsilon \leftarrow \epsilon_{min}$; $F \leftarrow F \cup \{f_{min}\}$;
17 $\quad$ **end**
18 **until** *no changes to* F;
19 **return** F;

Table 10: Results of feature selection (with *LR*)

S	F	RMSLE
$\emptyset$	{8, 14, 21, 32, 35, 38, 54, 55, 97, 100, 101, 106, 134}	0.4892
{1-14}	{1-14, 16, 18, 30, 37, 41, 43, 45, 52, 55, 57, 68, 71, 72, 76, 81, 89, 90, 100, 109, 122, 133, 135}	0.4832
{15-78}	{15-78, 1, 5, 9, 10, 13, 81, 84, 90, 91, 97, 102, 116, 120, 124, 132, 134, 135}	**0.4819**
{1-78}	{1-78, 81, 86, 102, 124, 132, 135}	0.4843

different performance with different "best" features. The 0.4819 RMSLE is the best performance observed so far in our study (the previous 0.482 is indeed 0.4821 before rounding). However, we are a bit over optimistic here since we are using the testing set to help us pick the features, instead of using a tuning set or cross validation on the training set. Nonetheless, the current results still serve our purpose of showing that different selections of features do affect the performance. It is interesting to see that with more strict and complicated feature selection algorithms, whether there is some other set of features that can produce even better results, and we treat it as one of our future work.

5.2 Models

The choice on models is more tricky. In this study, we pick models based on our own knowledge and experience, and in Section 3.2 we have justified our rational for choosing these models. However, several other kinds of models have not been tried in this study may have potential in gaining better

performance. In particular, we are interested in leveraging Inductive Logic Programming (ILP) (Muggleton and Raedt 1994) and Statistical Relational Learning (SRL) approaches (e.g., Probabilistic Relational Models (PRM) (Friedman et al. 1999)) for this problem. The motivation for trying these two kinds of algorithms is quite natural. For ILP, intuitively there should be certain kind of rules that can work quite well on the data. For instance, a patient with heart disease or cancer is more likely to stay longer in hospital than a patient with skin infection. At present, a practical difficulty for us to apply ILP is that Weka has not included a library for ILP yet. The motivation for using SRL is also straightforward, since the dataset we have has already been organized into relational format (i.e., a set of relations with schemas). However, due to the large number of data we have, directly running existing SRL algorithms may be intractable in practice. Therefore, we leave these two directions as our future work.

6 Conclusion

In this paper, we studied the problem of predicting the number of days that a patient will be admitted in hospital the next year, by leveraging his/her admission records in the past year(s). We modeled the predictive problem as either a multi-class classification task or a regression task, and we tested various machine learning algorithms in both categories. We also discussed related problems such as feature selection and ensembles. Experimental results show that, modeling the problem as a regression task can slightly outperform the way of modeling the problem as a multi-class classification task, and leveraging ensembles such as Boosting can slightly outperform individual models. We hope our results can provide useful insights into this problem for future researchers.

References

Breiman, L. 2001. Random forests. *Machine Learning* 45(1):5–32.

Charlson, M. E.; Pompei, P.; Ales, K. L.; and MacKenzie, C. R. 2008. A new method of classifying prognostic comorbidity in longitudinal studies: development and validation. *Journal of Chronic Diseases* 40(5):373–383.

Cortes, C., and Vapnik, V. 1995. Support-vector networks. *Machine Learning* 20(3):273–297.

Escobar, G. J.; Greene, J. D.; Scheirer, P.; Gardner, M. N.; Draper, D.; and Kipnis, P. 2008. Risk-adjusting hospital inpatient mortality using automated inpatient, outpatient, and laboratory databases. *Medical Care* 46(3):232–239.

Freund, Y., and Schapire, R. E. 1995. A decision-theoretic generalization of on-line learning and an application to boosting. In *EuroCOLT*, 23–37.

Friedman, N.; Getoor, L.; Koller, D.; and Pfeffer, A. 1999. Learning probabilistic relational models. In *IJCAI*, 1300–1309.

Friedman, J. 1999. Stochastic gradient boosting. Technical report, Stanford University.

Gordy, T. R. 2006. *CPT 2006 Current Procedural Terminology*.

Hall, M.; Frank, E.; Holmes, G.; Pfahringer, B.; Reutemann, P.; and Witten, I. H. 2009. The weka data mining software: an update. *SIGKDD Explorations* 11(1):10–18.

HPN. 2011. The heritage health prize competition, http://www.heritagehealthprize.com.

Muggleton, S., and Raedt, L. D. 1994. Inductive logic programming: Theory and methods. *J. Log. Program.* 19/20:629–679.

Platt, J. C. 1999. *Fast training of support vector machines using sequential minimal optimization*. 185–208.

Quinlan, J. R. 1987. Simplifying decision trees. *International Journal of Man-Machine Studies* 27(3):221–234.

Quinlan, R. J. 1992. Learning with continuous classes. In *5th Australian Joint Conference on Artificial Intelligence*, 343–348. Singapore: World Scientific.

A Dataset Description

The description of the dataset is from (HPN 2011). We include it here for convenience of reference. As shown from Table 11 to Table 15, the dataset contains five tables: *Members*, *Claims*, *Drug Count*, *Lab Count*, and *Outcome*. A piece of brief description for each table can also be found in Section 2.1.

Table 11: Members

Attribute	Description
MemberID	Member pseudonym.
AgeAtFirstClaim	Age in years at the time of the first claim's date of service computed from the date of birth; Generalized into ten year age intervals.
Sex	Biological sex of member: M = Male; F = Female.

B SQL Queries

We use the following SQL query to join the 5 relations in our dataset to obtain the information for each patient.

```
SELECT *
FROM Members M, Claims C,
     DaysInHospital_Y1 D1,
     DaysInHospital_Y2 D2,
     DrugCount DC, LabCount LC
WHERE M.MemberID = C.MemberID
AND M.MemberID = D1.MemberID
AND M.MemberID = D2.MemberID
AND M.MemberID = DC.MemberID
AND M.MemberID = LC.MemberID
```

Before issuing this query, we create a database in MySQL. We then create 5 tables based on the schema specified in Appendix A, and load the data into each table accordingly. The SQL statements for creating tables and loading data are so straightforward that we omit the details here.

Table 12: Claims

Attribute	Description
MemberID	Member pseudonym.
ProviderID	Provider pseudonym.
Vendor	Vendor pseudonym.
PCP	Primary care physician pseudonym.
Year	Year in which the claim was made: Y1; Y2.
Specialty	Generalized specialty.
PlaceSvc	Generalized place of service.
PayDelay	Number of days delay between the date of service (the date the actual procedure was performed or service provided) and date of payment. Values above 161 days (the 95% percentile) are top-coded as "162+".
LengthOfStay	Length of stay (discharge date - admission date + 1), generalized to: days up to six days; (1-2] weeks; (2-4] weeks; (4-8] weeks; (8-12 weeks]; (12-26] weeks; more than 26 weeks (26+ weeks).
DSFS	Days since first claim, computed from the first claim for that member for each year, generalized to: [0-1] month, (1-2] months, (2-3] months, (3-4] months, (4-5] months, (5-6] months, (6-7] months, (7-8] months, (8-9] months, (9-10] months, (10-11] months, (11-12] months.
PrimaryConditionGroup	Broad diagnostic categories, based on the relative similarity of diseases and mortality rates, that generalize the primary diagnosis codes (ICD-9-CM) (Escobar et al. 2008).
CharlsonIndex	A measure of the affect diseases have on overall illness, grouped by significance, that generalizes additional diagnoses. Scores greater than zero are carried forward (for up to a year) in subsequent claims with a comorbidity score of zero (Charlson et al. 2008).
ProcedureGroup	Broad categories of procedures, grouped according to the hierarchical structure defined by the Current Procedural Terminology (CPT) (Gordy 2006).
SupLOS	Indicates if the NULL value for the *LengthOfStay* variable is due to suppression done during the de-identification process. A value of 1 indicates that suppression was applied.

Table 13: Drug Count

Attribute	Description
MemberID	Member pseudonym.
Year	Year in which the drug prescription was filled: Y1; Y2.
DSFS	Days since first service (or claim), computed from the first claim for that member for each year, generalized to: [0-1] month, (1-2] months, (2-3] months, (3-4] months, (4-5] months, (5-6] months, (6-7] months, (7-8] months, (8-9] months, (9-10] months, (10-11] months, (11-12] months.
DrugCount	Count of unique prescription drugs filled by DSFS. No count is provided if prescriptions were filled before DSFS zero. Values above 6, the 95% percentile after excluding counts of zero, are top-coded as "7+".

Table 14: Lab Count

Attribute	Description
MemberID	Member pseudonym.
Year	Year in which the drug prescription was filled: Y1; Y2.
DSFS	Days since first service (or claim), computed from the first claim for that member for each year, generalized to [0-1] month, (1-2] months, (2-3] months, (3-4] months, (4-5] months, (5-6] months, (6-7] months, (7-8] months, (8-9] months, (9-10] months, (10-11] months, (11-12] months.
LabCount	Count of unique laboratory and pathology tests by DSFS. Values above 9, the 95% percentile after excluding counts of zero, are top-coded as "10+".

Table 15: Outcome

Attribute	Description
MemberID	Member pseudonym.
DaysInHospital_Y1	Days in hospital, the main outcome, for members with claims in Y1. Values above 14 days (the 99% percentile) are top-coded as "15+".
DaysInHospital_Y2	Days in hospital, the main outcome, for members with claims in Y2. Values above 14 days (the 99% percentile) are top-coded as "15+".
ClaimedTruncated	Members with truncated claims in the year prior to the main outcome are assigned a value of 1, and 0 otherwise.