Predicting Healthcare Costs: Using a Machine Learning Approach

AUTHOR INFORMATION

Avedis Sarkisian, DePaul University, School of Computing, asarkis5@depaul.edu

Faizan Bhutto, DePaul University, School of Computing, fbhutto@depaul.edu

Gael Mota Hernandez, DePaul University, School of Computing, gmotaher@depaul.edu

Huzifa Noor, DePaul University, School of Computing, hnoor2@depaul.edu

ABSTRACT

Healthcare cost prediction is crucial for improving financial transparency in medical services. This study evaluates machine learning models for estimating healthcare (hospital) total charges, using a dataset of over two million patient discharges. Regression models, such as XGBoost, LightGBM and Neural Networks demonstrated high prediction accuracy, with XGBoost achieving the best performance (R^2=0.85). Additionally, a feature importance analysis suggested that the most influential feature is the time a patient stays in the hospital (length_of_stay), which logically makes sense. The findings highlight the value of machine learning in healthcare cost predictions, offering insights for patients, insurance companies, and hospitals to optimize their financial planning and decision-making processes.

KEYWORDS: Healthcare Cost, Prediction, Machine Learning, Regression, User Interface

INTRODUCTION

Healthcare costs in the United States have been rising steadily, creating uncertainty for patients when estimating out-of-pocket expenses. One of the critical gaps in healthcare management is the lack of transparency regarding the expected costs of services provided. To address this issue, our study aims to develop a machine learning-based predictive model that estimates the expected bill a patient may receive from their healthcare provider. This model can assist patients in financial planning and help insurance companies better assess coverage limits and expected payouts. By leveraging an extensive dataset containing patient demographics information, illnesses and services, and total costs, we aim to generate a robust predictive system that considers both provider-level and patient-level factors.

LITERATURE REVIEW

Predicting healthcare cost is a critical area of research in the medical industry, with applications in patient decision-making and insurance risk assessment, and financial planning. Several researchers have approached this study with various supervised learning techniques, treating it either as a regression or classification problem. This section reviews key studies in both approaches.

Regression approach

Most studies treat this problem as a regression task, where the objective is to estimate the total hospital charges (continuous target variable). Various machine learning models have been explored to improve the predictive accuracy over traditional statistical methods.

A study by Kulkarni et al. (2020) investigated the effectiveness of different regression techniques in predicting inpatient hospital charges. This research utilized secondary data from the New York State patient discharge records for 2017, applying several modeling techniques, such as Random Forest, Stochastic Gradient Descent (SGD), K-nearest Neighbors (KNN), eXtreme Gradient Boosting (XGBoost), and Gradient Boosting. Among these models, the Random Forest regressor achieved the best performance with an R^2 value of 0.7753. The study also found that length of stay, and age group were the most influential features in predicting the total hospital charges.

Prior research has attempted to address healthcare cost projections through statistical approaches. For instance, Multiple Linear Regression (MLR) has been a widely used technique, but it faces challenges such as multicollinearity and sensitivity to outliers (Patriche et al., 2011; Jones, 2010). To handle these issues, Generalized Linear Models have been explored (Konig et al., 2013); however, they are limited by their linear structure, making them less effective in capturing the non-linear complex relationships present in the features needed to predict healthcare costs.

Machine learning models have emerged as a powerful alternative, overcoming limitations of traditional statistical and linear models. Such researches compared different regression frameworks and found that boosted trees and decision trees outperformed the previously mentioned approaches (Duncan et al., 2016). More recent research has incorporated deep learning techniques to enhance predictive accuracy, combining Multilayer Perceptrons (MLPs) with Long Short-term Memory (LSTM) models (Kaushik et al., 2020). These approaches proved effective in capturing temporal dependencies in healthcare cost trends; however, these models require a lot of computational resources and lack of interpretability compared to simple tree-based approaches.

Building on the work from Kulkarni et al. (2020), this study also utilizes the 2017 New York State patient discharge data to predict total charges. By applying similar machine learning models, this study aims to validate previous findings, while exploring additional refinements, especially in terms of preprocessing, to improve the predictive power and accuracy.

Classification approach

While least common, some studies have treated healthcare cost prediction as a classification problem, following different approaches and setting different goals. For instance, some studies have categorized individuals into different cost groups (Lahiri et al., 2014), making it useful for predicting whether an individual's total yearly charges would increase the following year. By applying stacked generalization with multiple classifiers, such as Gradient Boosting, Conditional Inference Trees, Neural Networks, Support Vector Machine (SVM), Logistic Regression (for classification), and Naive Bayes. The best model achieved an accuracy of 78% and a recall rate of 80%, demonstrating that machine learning models can effectively segment patients based on several factors.

Another approach to treat this as a classification problem involves cost bucketing, where patients are assigned to a discrete cost group rather than being predicted on a continuous scale (Guo et al., 2015). This research developed a model to predict transitions between cost buckets, improving performance by 21% over baseline models. The key advantage of this approach is that it ensures that high-cost individuals are correctly bucketed, minimizing their impact on other patients.

One of the most effective classification-based approaches is the cost-on-cost approach. In this study, prior costs serve as primary features for future cost predictions (Bertsimas et al., 2008). This study demonstrated that using 22 cost-related features could achieve nearly the same predictive performance than using 1,500 clinical, and demographic features. This reinforces the idea that past costs are powerful indicators of future costs, making classification models parsimonious and effective.

THEORETICAL DEVELOPMENT

Dataset Description

The dataset utilized for this research was obtained from the New York Health Data government website. The dataset contains 2,343,569 patient discharges (observations) in 2017, which are described by 34 features. These 34 features, in text format, explained hospital demographics, patients' demographics, diagnosis and procedures, and payment information, including the total charges. Refer to appendix 1 to see all the features and their descriptions.

Data Cleaning

Removing Unnecessary Features

The dataset contained several descriptive features that were not necessary for the analysis. For instance, discharge year. The dataset contains information from 2017 discharges in New York, therefore, the value for this feature across the whole dataset is the same. Having the same value on a feature for all the observations adds no predictive power, on the contrary, it increases model complexity, as it requires unnecessary computations.

Furthermore, most machine learning models take numeric values to train the model. Several features on the dataset contained descriptive information, mostly for illness codes and procedures. These features are not only not accepted by the model given its nature, but also not necessary. While these features were not used on the model training, a mapping was saved and later used on the user interface (UI) created. See the mappings and the UI sections to find more information.

Other features like CCS Diagnosis Code, CCS Procedure Code, and APR DRG Code were removed due to their high number of unique values, 263, 224, and 320, respectively. If maintained, encoding these features would increase the dataset dimensions greatly, making it hard to process and train models due to computational constraints faced on the research. While the information provided by these features is important, APR MDC Code contains the same information grouped in broader groups, 26 unique values, reducing the number of final features after encoding greatly. See appendix 1 to find feature descriptions.

After removing all the unnecessary features, the dataset ended up with 18 features describing the 2,343,569 patient discharges.

Handling Missing Values and Outliers

To handle missing values, a simple approach was taken. We realized that several observations contained missing values on the features related to hospital information, therefore, to avoid inputting values and introducing bias, those observations were removed. After doing so, a few missing values remained on the risk of mortality feature, and since this is an important feature, those were removed as well. Finally, the feature explaining the second type of payment

contained some missing values. If the first type of payment covers the whole cost, the second payment type is empty. Therefore, we created another option for this feature, “Covered by 1st Payment”, to explain this relationship and fill the missing values.

Similarly, the approach to handle outliers was quite simple. Outliers are unusually high or low data points that significantly differ from the rest of the data. These data points may distort distributions and statistical measures such as mean, correlation, and variance, or even affect the model performance by introducing bias. To remove them, we utilized the interquartile range (IQR) approach, which is a statistical technique that keeps the data points based on the spread of the middle 50% of the data.

To use this technique, we first calculated the first quartile (Q1) and the third quartile (Q3) of the target feature. Then we subtracted Q1 from Q3 to get the IQR. To determine the lower bound of the interquartile range, 1.5 is multiplied by the IQR and the result is subtracted from Q1. Similarly, to determine the upper bound, 1.5 is multiplied by the IQR and the result added to Q3. While 1.5 is a commonly used heuristic, any other value can be used. This 1.5 multiplier (heuristic) is commonly used as it statistically balances the identification of extreme values while avoiding excessive removal of valid data points, working well with almost all data distributions. This approach detected 205,352 observations as outliers, which were removed.

Preprocessing

To prepare the datasets for model training, we conducted several preprocessing steps, which include scaling, encoding, and dimensionality reduction. Preprocessing is a critical step in any analysis, especially when working with predictions and machine learning as it improves data quality, consequentially enhancing model performance.

Scaling

Scaling is a preprocessing technique that transforms numerical features, so they have a consistent range or distribution. It is crucial because it ensures that all features contribute equally to the model’s learning process, preventing larger numerical values from dominating smaller ones. In fact, some machine learning algorithms like Neural Networks perform significantly worse if the data is not scaled.

The dataset contains a combination of 4 ordinal, 13 non-ordinal categorical, and one target feature. While the values for the non-ordinal categorical features range from 0 to 1 due to encoding [see the encoding section to find more information], the ordinal features, including the target, vary in distribution. The 4 ordinal features, and the target are scaled to prevent any issues when modeling. The scaler used in this research was the StandardScaler from Scikit-learn, which standardized features by removing the mean and scaling to unit variance. It transforms each feature using the formula:

$$X_{scaled} = (X - \mu) \div \sigma \quad (1)$$

where:

X is the original feature value, μ is the mean of the feature, and σ is the standard deviation of the feature. Unlike other scalers, the StandardScaler preserves the distribution of the data by shifting and scaling. After the transformation, each feature has a mean of 0, and a standard deviation of 1.

Encoding

Encoding is extremely important in machine learning because most models can only work with numerical data. Encoding basically transforms categorical variables into a format that models can understand. While a possible approach to encode non-ordinal categorical features is to assign a random number to each unique value, when the data is non-ordinal, this approach introduces false ordinality that misleads the models to think that one value is greater than other.

To avoid this issue, the 13 non-ordinal categorical features were encoded using the OneHotEncoder from Scikit-learn. This approach creates a new binary column for each category. While the ordinality issue is solved by implementing this approach, a new one arises. The dimensions of the dataset increase to the number of unique values in the 13 features times the number of observations, which in this case would be 397 times 2,132,821. Having such a large dataset is hard to manage, especially with the computational constraints faced during this research.

To solve the previously mentioned issue, the encoder was initialized to store the result as a sparse matrix instead of an array. A sparse matrix is a type of matrix where most values are zero. Instead of storing all the values explicitly, the sparse matrix only stores non-zero values and their positions, reducing memory usage.

Dimensionality Reduction

Dimensionality reduction is important in machine learning when you are working with big data. High dimensionality can not only make models struggle to learn patterns on the data effectively, in some cases, as ours, prevents model training if the computational power is not enough. This step helps simplify the complexity of a large dataset, while retaining essential information. Doing this step brings many benefits, such as the potential improvement on performance by preventing overfitting, the reduction of computational cost and improvement of data storage, and the improvement of interpretability, among others.

While dimensionality reduction can be applied to the whole dataset, to preserve the ordinal features, we decided to only apply it to the one-hot encoded features. The Truncated Singular Value Decomposition (Truncated SVD) from Scikit learn was used to perform this step. The key characteristic of this algorithm is that it does not center the data, making it effective for sparse datasets. The Truncated SVD decomposes the matrix into singular values and vectors, then selecting and retaining only the top k most significant ones, where k is a user defined parameter.

In this case, the Truncated SVD was first trained on a sample of the data (about 5%) to estimate the optimal number of components that capture 95% of the variance. Once the optimal number was obtained, the Truncated SVD was trained on the full data, adding a small buffer to the components obtained to account for the larger dimensions. After the dimensionality reduction, the encoded dataset contained 134 components.

Final Dataset

Once all the preprocessing steps were completed, the data frame containing the ordinal data was merged with the data frame containing the reduced non-ordinal one-hot encoded features. This resulted in a dataset with 2,132,821 observations and 139 features. This final dataset is used to train the models described below.

Figure 1. Final Dataset Dimensions

Rows	2132821
Features	139

Model Training

Our goal, as stated before, is to predict the healthcare cost (total charges) that a patient might receive from a healthcare provider. We approached this as a regression problem and evaluated multiple machine learning models. The data was split into 70% training and 30% testing sets. As seen in figure 2, the training sets consisted of 1,492,974 observations, and the testing sets of 639,847. Furthermore, the predictors set consisted of 138 features, and the predicted set had just one feature, “Total Charges”.

Figure 2. Train and Test datasets

Dataset	Rows	Features
Features Train	1,492,974	138
Features Test	639,847	138
Cost Train	1,492,974	Total Charges
Cost Test	639,847	Total Charges

Evaluation Metrics

When evaluating regression models, different metrics should be evaluated to measure how well the model predicts continuous values. In this case, we decided to evaluate based on the Mean Squared Error (MSE), Root Mean Squared Error (RMSE), and the Coefficient of Determination (R^2).

The MSE metrics measure the average squared difference between the actual and predicted values. The RMSE measures the square root of the MSE, making it easier to interpret as it is in the same unit as the target variable. These two metrics quantify the overall error of the model’s predictions, and can be calculated with the following formulas:

$$MSE = \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2 \quad (2)$$

$$RMSE = \sqrt{MSE} = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2} \quad (3)$$

where:

n is the number of datapoints, y_i is the observed value, and $\hat{y}_i$ is the predicted value.

On the other hand, the R^2 metric measures the proportion of variance explained by the model, essentially indicating how well the model fits the data and how much of the variation can be predicted by the model. The closer the R^2 value is, the better the model fits the data. This metric can be calculated with the following formula:

$$R^2 = 1 - \frac{RSS}{TSS} = 1 - \frac{\sum(y_i - \hat{y}_i)^2}{\sum(y_i - \bar{y}_i)^2} \quad (4)$$

where:

y_i is the observed value, $\hat{y}_i$ is the predicted value, and $\underline{y_i}$ is the mean of all actual target values.

Decision Tree Regressor

Decision tree regressors are machine learning algorithms used to predict continuous target variables. It makes the prediction by recursively splitting the dataset into smaller subsets based on feature values. The goal is to create homogeneous groups where the target variable is as consistent as possible within each group. It starts at the root and selects a feature that best splits the data based on some criterion. In this case, the split criterion used was the MSE, which basically minimizes the MSE for the child nodes. This step is repeated recursively until a stopping condition is met, which in this case was set either a maximum depth of 10, minimum sample per split of 10, or minimum samples per leaf of 5.

Random Forest Regressor

Random forest regressors are ensemble learning methods that improve the accuracy and stability of a decision tree regressor by combining multiple trees. Instead of relying on a single decision tree, it builds multiple and averages their predictions to reduce overfitting and improve accuracy. While the training process of each tree is quite similar to the one for decision trees, the main difference is that this one randomly samples the dataset with replacement to create multiple subsets and uses each subset to train a different decision tree. After a grid search to find the optimal from a small set of parameters, the random forest was set to train 300 different decision trees on the GPU, with a maximum depth of 20 each, a minimum of 2 samples per split, and the square root of the number of features to consider when looking for the best split.

eXtreme Gradient Boosting Regressor (XGBoost)

Gradient boosting is a machine learning algorithm that sequentially builds decision trees, where each new decision tree tries to correct the errors of the previous one, aiming to minimize the overall error. It iteratively combines the prediction of many “weak” decision trees. XGBoost is a powerful, optimized implementation of the gradient boosting algorithm that splits all nodes at the same depth of the tree first. Some of the optimizations include improving regularization techniques to prevent overfitting and achieve the best balance between accuracy and simplicity. Furthermore, it is designed to be highly scalable, enabling efficient training on large datasets.

The model is trained on the GPU, setting the learning objective to MSE, the evaluation metric to RMSE, the way the model builds trees to “hist” which bins continuous features into buckets to speed up the tree construction, the learning rate which controls how much each tree contributes to the final prediction to 0.05, and the maximum depth of the trees to 10.

Light Gradient Boosting Machine Regressor (LightGBM)

LightGBM is a machine learning algorithm that also optimizes the gradient boosting algorithm. The main difference with XGBoost is that it grows the most promising leaf on the tree first. This difference makes training the model much faster also using less memory; however, it also makes it more prone to overfitting if not the parameters are not tuned correctly. The model was trained on the GPU, setting the evaluation metric to RMSE, the learning rate to 0.05, the maximum depth of the trees to 10, the maximum number of leaves per tree to 400, and the friction of features to use at each split to 80% of the features (0.8).

Neural Network Regressor (MLP)

Neural Network is a machine learning model inspired by the human brain. It consists of layers of interconnected nodes (neurons) that process input features and pass information through weights and activation functions to make predictions and decisions. The input layer takes numerical inputs, passes them to the hidden layers so they can apply a weighted sum and the activation functions, and finally passes it to the output layer to make the prediction. This neural network is trained on the GPU using the MSE loss function and the Adam optimizer to minimize the loss function. During training, the model takes input features, passes them through three hidden layers with 128, 64, and 32 nodes respectively, all using ReLu activation to learn non-linear patterns. The model learns to minimize prediction errors by adjusting weights and biases using backpropagation and gradient descent. Dropout layers are included to prevent overfitting, and early stopping ensures training stops when validation loss stops improving. The final layer (output layer) produces continuous numeric predictions.

Feature Selection

As seen in table 1, the five models were trained with all the features available, 138. A feature importance analysis was performed in every model. For the decision tree, XGBoost, and LightGBM models, the feature importance was extracted directly from the model. However, for the Random Forest and Neural Network models, the feature importance is not available in the library used; therefore, the permutation importance technique was used to extract them. This technique measures the increase in model prediction error when a feature’s values are randomly shuffled. As seen in the figures below, the length of stay and SVD_Component_5 were two most significant features in every model.

Extracting the feature importance allows to train new models with fewer features and compare their performances. All five machine learning models were retrained several times, using different numbers of features set to capture 95%, 90% and 85% of the variance. However, after comparing their performance with the initial models’ performance, all of them had lower/worse results, so we decided to continue forward with the original five models.

Figure 3. Top 15 features in Decision Tree

Figure 4. Top 15 features in Random Forest

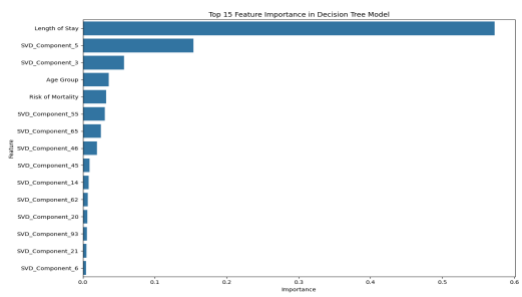


Figure 5. Top 15 features in XGBoost

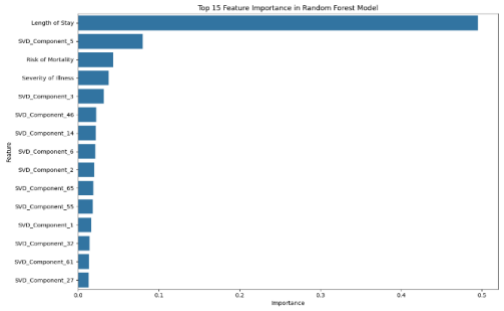
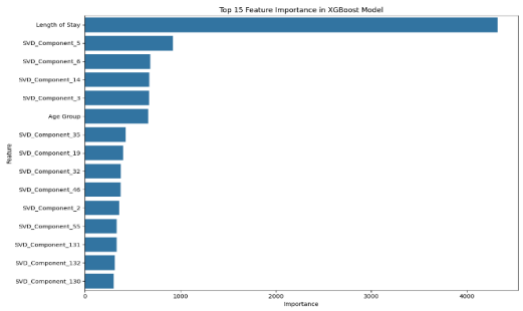


Figure 6. Top 15 features in Light BGM



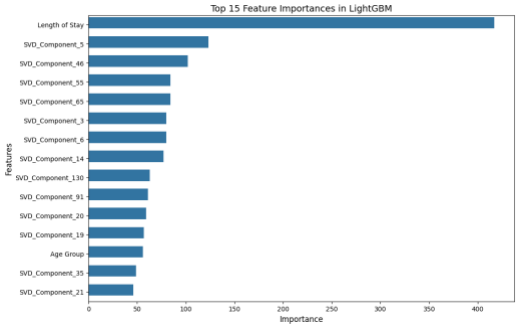
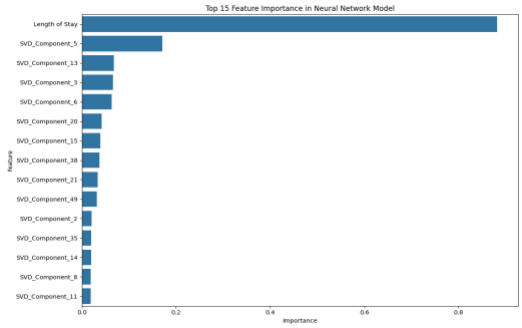


Figure 7. Top 15 features in Neural Network



SHINY APP

A Shiny App is an interactive web application built in R, which is also available for python using the Shiny for Python framework. It allows you to explore and manipulate data in real-time. Shiny apps are commonly used for data visualization, dashboards, interactive reports, and machine learning model deployment. It consists of three main components: user interface, server, and app launch. In this research, the Shiny app is used to let users input their details and get an estimated hospital/healthcare cost in real-time.

Define User Interface (UI)

The UI component is responsible for defining the structure and layout of the application. It specifies how input and output elements are arranged and displayed on the app. While there is no guide on how the app should look like, usually it is done in a way that creates interactive responsive ways for the user to input the data and view the results. This component does not include any logic or preprocessing, its primary function is to define the app's visual structure and elements.

In our app, most of the elements are structured as dropdown menus, where you can either select the desired option or type the first few letters and find the desired option. This application consists of a single page, containing the aforementioned dropdown menus, a prediction button that triggers the prediction once clicked, and a table showing the prediction history (the values input by the user historically).

Define Server

The server component is the backend logic of the app. It is where all the computations, preprocessing, and updates take place. It is defined by a function that takes two arguments: input (which captures the user inputs from the UI) and outputs (which renders the outputs displayed in the UI). Essentially, the server acts as the brain of the app, processing user interactions and returning the results.

In our app, there are two main components within the logic process, which are mapping the dropdown options to the actual values used in the model, and applying the same preprocessing steps applied when training the model.

Variable Mapping

As previously mentioned, the dataset contains several descriptive features. These features were used to map the codes with the descriptions. Two mappings were created using two features, Permanent Facility Id → Facility Name, and APR MDC Code → APR MDC Description. Three additional mappings were created by making age group, risk of mortality, and severity of illness ordinal. Since the risk of mortality and the severity of illness had the same values, a single mapping was created for both.

These mappings were created to add interpretability to the UI. They allow the user to input/select an option from the “raw” data, rather than selecting an insignificant value that is only known by the model.

Apply Preprocessing Steps

Once the user input is translated to known values for the model (mapped), the preprocessing steps start. For the inputs from an ordinal feature, the server will scale them using the same StandardScaler used to train the model. On the other hand, for the inputs from non-ordinal categorical features, the server will encode them using the same OneHotEncoder used in to train the model and then will apply the same TruncatedSVD used to train the model to assign it to the same component. Once the prediction is generated, the server will unscale the “Total Cost” to display it in the original scale and make it interpretable for the user. See the preprocessing section to find detailed information about preprocessing.

Launch App

The shinyApp() function combines the UI and server components and launches the application. It takes the UI and server as arguments and runs them together to create the interactive web application. This step is what makes the app accessible in a web browser and serves as the entry point to bring the app to life, allowing the user to interact with it.

RESULTS

The results table below presents the performance of the top model trained for each machine learning algorithm. As shown in the table, the best performing model was the XGBoost model, achieving the highest R^2 value of 0.8499, and the lowest MSE and RMSE values of 0.15 and 0.39 respectively. The Neural Network achieved a similar performance, with a R^2 value of

0.8436, MSE of 0.16 and RMSE of 0.4. Following closely in terms of performance, the LightGBM model achieved a R^2 value of 0.8173, MSE of 0.18, and RMSE of 0.43. The performance of these three models is acceptable, and while some changes could be done to improve it, as they are right now, they are considered good.

The same cannot be said about the Random Forest and Decision Tree. The Random Forest model achieved a R^2 value of 0.7637, MSE of 0.24, and RMSE of 0.49, showing a moderate performance. On the other hand, the performance of the Decision Tree model was the weakest, achieving a R^2 value of 0.6257, MSE of 0.37, and RMSE of 0.61. While these values might be considered moderate in other fields and research, compared to the other model this one has a weak predictive power.

Table 1. Results Table

Scoring Metrics				
Model	# of Features	MSE	RMSE	R^2
Decision Tree	138	0.37	0.61	0.6257
Random Forest	138	0.24	0.49	0.7637
XGBoost	138	0.15	0.39	0.8499
LightGBM	138	0.18	0.43	0.8173
Neural Network	138	0.16	0.40	0.8436

DISCUSSION

Strengths

Our study effectively leveraged a large and diverse dataset of over two million patient discharges, integrating hospital and patient-level features to predict healthcare costs. By using a variety of machine learning models, we ensured a robust comparison of different modeling techniques. Furthermore, we implemented well-structured preprocessing steps to enhance model performance and mitigate computational challenges. Finally, the integration of a user-friendly UI also provides a practical application for patients and healthcare providers, allowing for real-time estimation based on their input.

Limitations

Despite the strengths, several limitations should be acknowledged. One key limitation in our approach was the decision to exclude procedure codes (i.e. CCS Procedure Code) due to their high dimensionality. While this step reduced computational complexity greatly, it also eliminated information that could potentially play a significant role in determining healthcare costs. By solely relying on diagnosis codes, our model contemplated the cause (diagnosis) but overlooked the solution (procedures).

Furthermore, our approach to encode the diagnosis codes with the rest of the non-ordinal categorical variables using the OneHotEncoder might have inadvertently lowered their importance. Alternative encoding methods, such as embedding layers, which maps each value to a dense vector of fixed size containing real numbers rather, could potentially preserve more meaningful relationships within the data.

Finally, a common limitation across the whole research were computational constraints. The sheer size of the dataset posed computational challenges, limiting our ability to explore additional feature engineering techniques. While dimensionality reduction helped manage

feature space, it is possible that this transformation negatively impacted how meaningful some features were to the prediction.

Future Studies

Future studies could address these limitations in several ways. For instance, approaching encoding with more efficient techniques will not only allow to represent the features with their original importance (i.e. APR MDC Code), but also include the potentially important removed features (i.e. CCS Procedure Code) without excessive dimensionality expansion. Overall, this could provide a more comprehensive understanding of cost drivers. Additionally, including other features like insurance claim history could further refine the model's predictive accuracy.

By addressing these areas, future research can improve the accuracy, reliability, and interpretability of the machine learning models trained for healthcare cost predictions. Leading to more transparent, personalized, and accurate estimations for patients.

CONCLUSION

This study demonstrates the potential of machine learning models in accurately predicting healthcare costs, helping both patients and insurers manage financial planning and risk assessment. By leveraging a large dataset and implementing rigorous preprocessing steps, our analysis confirms that ensemble learning methods, particularly XGBoost, outperform traditional regression models. Additionally, classification-based cost prediction approaches offer practical alternatives for segmenting patients into cost categories avoiding the exact prediction of a cost, especially when using historical cost information. While our models show strong predictive performance compared to previous research, future research should explore more advanced techniques and the addition of contextual variables not only to improve the performance, but also to enhance the model's interpretability.

APPENDIX

Appendix 1. Initial Features and Description

Column	Feature name	Description	Data Type
Hospital Service Area	hospital_service_area	Health Service Area where the hospital is located.	Text
Hospital County	hospital_county	County where the hospital is located.	Text
Operating Certificate Number	operating_certificate_number	Operating Certificate Number assigned by the Department of Health.	Text
Permanent Facility Id	permanent_facility_id	Facility identifier.	Text
Facility Name	facility_name	Name of the facility.	Text
Age Group	age_group	Age in years at the time of discharge.	Text
Zip Code - 3 digits	zip_code_3_digits	First 3 digits of the zip code.	Text
Gender	gender	Patient's gender.	Text
Race	race	Patient's race.	Text
Ethnicity	ethnicity	Patient's ethnicity.	Text
Length of Stay	length_of_stay	Number of days the patient was in the facility. (Discharge Date - Admission date) + 1	Text
Type of Admission	type_of_admission	Manner in which the patient was admitted to the facility.	Text
Patient Disposition	patient_disposition	Patient's destination after discharge.	Text
Discharge Year	discharge_year	Year (2017) of discharge.	Text
CCS Diagnosis Code	ccs_diagnosis_code	AHRQ Clinical Classification Software (CCS) Diagnosis Category Code.	Text
CCS Diagnosis Description	ccs_diagnosis_description	AHRQ Clinical Classification Software (CCS) Diagnosis Category Code description.	Text
CCS Procedure Code	ccs_procedure_code	AHRQ Clinical Classification	Text

		Software (CCS) ICD-9 Procedure Category Code.	
CCS Procedure Description	ccs_procedure_description	AHRQ Clinical Classification Software (CCS) ICD-9 Procedure Category Code description.	Text
APR DRG Code	apr_drg_code	APR DRG Classification Code.	Text
APR DRG Description	apr_drg_description	APR DRG Classification Code description.	Text
APR MDC Code	apr_mdc_code	All Patient Refined Major Diagnostic Category (APR MDC) Code.	Text
APR MDC Description	apr_mdc_description	All Patient Refined Major Diagnostic Category (APR MDC) Code description.	Text
APR Severity of Illness Code	apr_severity_of_illness_code	All Patient Refined Severity of Illness Code: 1, 2, 3, 4	Text
APR Severity of Illness Description	apr_severity_of_illness	All Patient Refined Severity of Illness description: Minor (1), Moderate (2), Major (3), Extreme (4)	Text
APR Risk of Mortality	apr_risk_of_mortality	All Patient Refined Risk of Mortality (APR ROM) description: Minor (1), Moderate (2), Major (3), Extreme (4)	Text
APR Medical Surgical Description	apr_medical_surgical	Whether the treatment was Medical, Surgical, or Not Applicable.	Text
Payment Typology 1	payment_typology_1	Description of the first type of payment.	Text
Payment Typology 2	payment_typology_2	Description of the second type of payment.	Text
Payment Typology 3	payment_typology_3	Description of the third type of payment.	Text
Birth Weight	birth_weight	Neonate’s birth weight.	Text
Abortion Edit Indicator	abortion_edit_indicator	Flag to indicate if the discharge contains any	Text

		abortion (“Y” = yes, “N” = no)	
Emergency Department Indicator	emergency_department_indicator	Flag to indicate the quality of care within an emergency department (if the revenue code is equal to 045X, the indicator is “Y” = yes, otherwise, “N” = no).	Text
Total Charges	total_charges	Total charges for the discharge.	Number
Total Costs	total_costs	Total estimated charges for the discharge.	Number

AUTHOR CONTRIBUTIONS

Avedis Sarkisian

I built and trained the Light GBM model.

Faizan Bhutto

I built and trained the XGBoost model.

Gael Mota

I found the dataset we used for this research and took care of all the preprocessing steps described above. Additionally, I built and trained the Random Forest Regressor, helped Huzifa training the Neural Network, and ensured consistency across all the model training files.

Huzifa Noor

I proposed the idea of working with healthcare and financial information to address the issue of unpredictable medical costs, which my group members agreed was a meaningful direction. At first, I scraped three datasets from the Centers for Medicare and Medicaid Services website. We were planning on merging those datasets and using them as one, but because of project requirements and issues with that data, we ended up using a different dataset.

Additionally, I built and trained machine learning models, including a Random Forest Regressor and a Neural Network, to predict medical expenses and provide cost transparency for consumers and insurance companies.

REFERENCES

Bertsimas, D., Bjarnadóttir, M. V., Kane, M. A., Kryder, J. C., Pandey, R., Vempala, S., et al. (2008). “Algorithmic prediction of health-care costs”, Operations Research. 56(6):1382-92.

Duncan, I., Loginov, M. and Ludkovski, M. (2016). “Previous cost model: testing alternative regression frameworks for predictive modeling of health care costs”, North American Actuarial Journal, Vol. 20 No. 1, pp. 65-87.

Guo, X., Gandy, W., Coberley, C., Pope, J., Rula, E., Wells, A. (2015). “Predicting health care cost transitions using a multidimensional adaptive prediction process”, Population health management. 18(4):290-9.

Jones, A. (2010). “Models for Health Care. Technical report Department of Economics”, University of York.

Kaushik, S., Choudhury, A., Sheron, P.K., Dasgupta, N., Natarajan, S., Pickett, L.A. and Dutt, V. (2020). “AI in healthcare: time-series forecasting using statistical, neural, and ensemble architectures”, Frontiers in Big Data, Vol. 3. Retrieved from: www.frontiersin.org/article/10.3389/fdata.2020.00004.

Konig, H., Leicht, H., Bickel, H., et al. (2013). “Effects of multiple chronic conditions on health care costs: an analysis based on an advanced tree-based regression model”, BMC Health Services Research. Vol. 13 No. 1, pp. 219.

Kulkarni, S., Ambekar S. S., Hudnurkar, M. (2020). “Predicting the Inpatient Hospital Cost Using a Machine Learning Approach”. International Journal of Innovation Science. Vol. 13, no. 1, pp. 87–104. Retrieved from: https://www.emerald.com/insight/content/doi/10.1108/ijis-09-2020-0175/full/html.

Lahiri, C., Agarwal, N. (2014). “Predicting healthcare expenditure increase for an individual from medicare data”. Proceedings of the ACM SIGKDD Workshop on Health Informatics.

Morid, M. A., Kawamoto, K., Ault, T., Dorius, J., Abdelrahman, S. (2017). “Supervised Learning Methods for Predicting Healthcare Costs: Systematic Literature Review and Empirical Evaluation.” AMIA Annual Symposium Proceedings, p. 1312. Retrieved from: https://pmc.ncbi.nlm.nih.gov/articles/PMC5977561/.

Patriche, C. V., Pirnau, R. G. and Rosca, B. (2011). “Bulletin UASVM agriculture: comparing linear regression and regression trees for spatial modeling of soil reaction in dobrovat basin (Eastern Romania)”.