Insurance assurance:
Predicting Health Insurance Premiums using Regression

Calvin Tan 730625638
Lee Qi An 730625510
Lee Yi Hern 730625511

Notebook: https://github.com/itsleeqian/COMP562-Project

1. Background

1.1. Motivation

Getting your own health insurance can be a chore, particularly for those not on a company plan. With so many factors like out-network coverage, referral requirements, deductibles, copays and more, the insurance marketplace can be a nightmare to navigate. On top of that, the insurance's premium might not be transparent and vary depending on factors like your age, health condition and number of family members. Knowing an approximate cost of your future health insurance would empower individuals to make more informed decisions in selecting their plans. In this project, we used Linear Regression, Ridge Regression, Random Forests, Neural Networks and Support Vector Regression to predict insurance premiums based on simple user data.

1.2. The Data

We obtained our data from Kaggle, a reputable online source for datasets, but was originally posted on GitHub. Although the data is available on the public domain, it was used in a textbook "Machine Learning with R" published by Brett Lantz in October 2013. The data has been cleaned to match the formatting of the textbook, so it can be assessed as credible and accurate. The data was said to be based on the demographic statistics from the US Census Bureau, though the official archive of the data could not be found.

The dataset has 1338 observations, and 7 variables. The variables discussed are age, sex, Body Mass Index (BMI), smoking status and health insurance charges (premiums).

	age	sex	bmi	children	smoker	region	charges
0	19	female	27.900	0	yes	southwest	16884.92400
1	18	male	33.770	1	no	southeast	1725.55230
2	28	male	33.000	3	no	southeast	4449.46200
3	33	male	22.705	0	no	northwest	21984.47061
4	32	male	28.880	0	no	northwest	3866.85520

Table 1. Sample of the dataset

1.3. Overview of Research Methodology

In our study, we will be trying to predict insurance premiums using the various predictors.

In section 2, we will further analyze our data, and pre-process it by cleaning it. After cleaning the data, we will deploy 5 different regression algorithm models and compare the performances of each of them. We will be comparing the Root-Mean-Square errors of each algorithm.

We will be using Linear Regression, Ridge Regression, Random Forests, Neural Networks and Support Vector Regression. We will then use the best Regression model to calculate the relative weights of each predictor for insurance prices.

2. Data Pre-processing

2.1. Preliminary Investigation

Despite the data being pre-cleaned, we double checked the data to ensure that it was properly formatted. We first analyzed our data by checking for any null values in the columns. We then checked for columns with only 1 unique value to reduce redundancy. As expected, we found zero abnormalities in the data

2.2. One Hot Encoding and Conversion of Binary data

Our next step was to convert all data with only 2 values into binary, and to one hot encode our categorical data. For example, we converted “sex” and “smoking status” into binary form, and we one-hot-encoded the “region” column as it had only 4 possible values (northeast, northwest, southeast, southwest).

2.3. Training-Test Split and Correlation finding

Following that, we used `train_test_split` from sklearn to split our training and test data using an 80/20 split. Before starting on our regression analysis, we also checked the variable correlation values to get a sense of which variables are likely to be heavy factors in our insurance cost prediction.

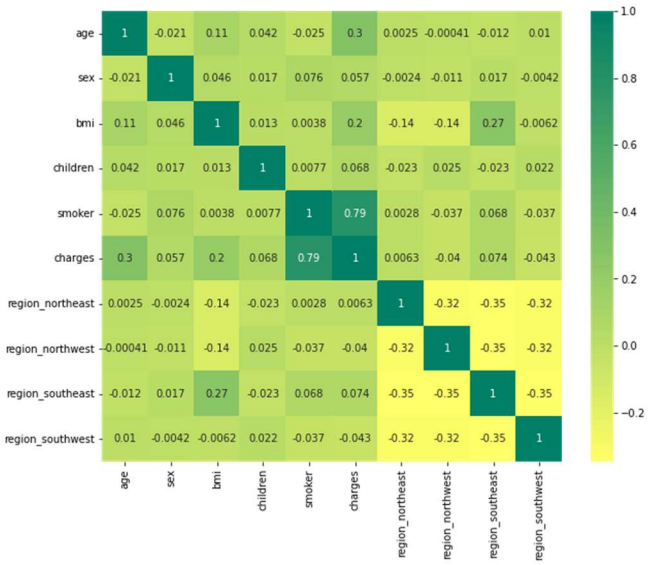

Figure 1. Heatmap of variable correlation

2.4. Multi-linear Regression

Finally, we used sklearn's `StandardScaler` which standardizes features by removing the mean and scaling to unit variance. We chose this scaler as deep learning algorithms often require zero mean and unit variance. Regression models also benefit from normally distributed data.

3. Regression Models

3.1. Multi-linear Regression

We started with a relatively simple linear regression model and obtained a model by minimizing the root mean square error. As discussed earlier, we converted the binary (yes/no) entries in the dataset to a binary (1/0) one, and we used a one-hot encoding representation of categorical features. We then ran a linear regression using sklearn's `LinearRegression` function using all of the features. By starting with a naive model, it allows us to verify that regression can be performed on this dataset to obtain a meaningful model without requiring complicated calculations. However, by using linear regression, it meant that the features with low correlation with *charges* resulted in an nonoptimal model. For example, we found that the region had almost zero coefficient in the final model, but cannot be safely disregarded due to potential multicollinearity.

3.2. Ridge Regression with Cross Validation

Ridge regression is a variation of linear regression specifically adapted for data that shows heavy multicollinearity. Ridge regression applies a L2 penalty term on the weights in the loss function. It is well-suited for datasets that have an abundant amount of features which are not independent (collinearity) from one another. By using this, we aim to mitigate the potential multicollinearity mentioned in Section 3.1. We followed an identical approach to the regular linear regression by again using sklearn's `RidgeCV` function, which performs ridge regression with Leave-One-Out Cross-Validation to automatically select

the best hyperparameters instead of having to manually fine-tune them. By using ridge regression, we are trading variance from bias to lower the standard errors.

As expected, we found that the root mean square error for the ridge regression model was lower than that of regular linear regression. However this difference is not very substantial as when tested across 100 different train/test splits, ridge regression's average root mean square error was only 0.05% lower than that of linear regression. Moreover, for linear and ridge regression, we also used R^2 to give us another way to compare the two different algorithms. Similarly, we found that the R^2 value for regression was consistently higher than that of linear regression.

3.3. Random Forests

Next, we use random forests, which is an ensemble of decision trees. Our input vector is run through multiple decision trees. For regression, the output value of all the trees is then averaged. We chose to use a random forest as the large number of decision trees reduces overfitting. We opted for sklearn's `RandomForestRegressor` and chose to use 500 trees in the random forest classifier as it was sufficient, and any more trees did not improve results. With this large number of trees, it ensures a better regression for the final model. Moreover, bootstrapping and random feature selection also vastly improves the results.

The combination of all these allowed random forests to get the lowest root mean square error and the highest R^2 score, indicating that it produced the best model.

3.4. Neural Network

We then used a neural network, which are multiple layers of interconnected nodes, with the connections modeled as weights between nodes. We decided to use Keras and its `Sequential` and `Dense` layers. We trained our neural network with 100 epochs and with a batch size of 32. We first started with 48 nodes on our first layer, and halved it down every layer. We used 2 hidden layers as it was the perfect balance between minimizing our root mean squared error and not adding unnecessary complexity to our model.

We also chose the mean squared error for our loss function, and the Rectified Linear Unit (ReLU) for our activation function. The main benefit of using ReLU over sigmoid is it being more computationally efficient as sigmoid requires more expensive exponential operations. ReLU also has a lower likelihood of the gradient vanishing, when $a > 0$. This generally results in faster learning if ReLUs are used. We then used the stochastic gradient-based ADAM optimizer, which is generally considered by industrial experts to be the best among the adaptive optimizers.

3.5. Support Vector Regression

For the final model, we decided to use sklearn's `SVR(Support Vector Regression)`, which is similar to Support Vector Machines taught in class, but with the added benefit of being able to be used with continuous data like in regression. For SVR, we plot each data item as a point in 9-dimensional space (since we have 9 features) with the value of each feature being the value of a particular coordinate. Then, we perform regression by finding the hyper-plane that differentiates the two classes.

For SVR, we used three different kernels - *linear*, *poly* and *Radial-Basis Function* (rbf) to determine which kernel would work best for our dataset. We found that the *linear* kernel worked best for this dataset. However, when compared to the other models, this is by far the worst in terms of root mean square error and R^2 score and thus the least effective model to use for predicting the charges. This could be due to SVR having a higher error tolerance than the higher models, thus producing a worse model.

4. Summary of Findings

4.1. Results

As mentioned above, we used both root mean square error and R^2 score to compare the different regression algorithms. We consolidated the results in the table below

	Linear	**Ridge**	**Random Forest**	**Neural Network**	**SVR**
RMSE	6307.23	6307.87	5002.83	12032.27	13112.87
R^2	0.77448	0.77444	0.85812	0.17928	0.02524

Table 2: RMSE and R^2 results from the various Regression algorithms

After obtaining the results, we re-ran the code multiple times to ensure that the results were not a fluke. We were able to confirm that for this dataset, Random Forest works the best and produced the most accurate model among the algorithms tested. With an R^2 score of 0.85, it is fairly accurate and can be used to predict the prices of premiums.

Linear and Ridge regression produced very identical results, and constantly traded blows with each other in terms of RMSE and R^2 score depending on the split.
Surprisingly, the neural network produced poor results. This could be due to the model not using an ideal number of hidden layers and lack of tuning the hyperparameters.
SVR also produced poor results, with the best kernel (linear), having the worst RMSE and R^2 score. We suspect that this might be due to the data being unsuitable for SVR to work well.

4.2. Limitations

Admittedly, the dataset contained only a subset of the actual factors used when determining the premium prices. For instance, it does not include the type of insurance and its benefits, which is a key factor affecting the final price of the premium. Nevertheless, the model we obtained implicitly handled these and aggregated the insurance benefits into the charges. Thus, this model provides a good baseline for approximating insurance premiums.

Another limitation is that the data is from 2013, which is almost 10 years old. The current health insurance marketplace might have its prices drastically adjusted since 2013 due to various factors. In lieu of

obtaining a more up to date dataset, we can attempt to make the model obtained more in line with today's prices, we should adjust the charges accounting for inflation, which is about 28%.

4.3. Conclusion

We obtained a fairly accurate model which can be used to predict insurance premiums. In addition to fixing the above mentioned limitations, going forward we can aim to make the model more accessible by creating an online calculator where users can easily enter their details to get an approximate of their premiums.

5. References

- https://www.kaggle.com/datasets/mirichoi0218/insurance
- https://qz.com/health-insurance-prices-went-up-nearly-30-over-the-pas-1849655576#:~:text=US%20health%20insurance%20prices%20 went%20up%2028.2%25%20 since%20 September%202022
- https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.StandardScaler.html
- https://keras.io/api/models/sequential/