STAT 440 Module 1: COVID-19 Time To Hospitalization

Team Machine: *Justin Heer*, *Carl Zou*, *Leo Chen*, *Neil Shah*

1 INTRODUCTION

The problem we are tackling is accurately predicting the duration of time between symptom onset and hospitalization for confirmed hospitalized cases of COVID-19 in January and February 2020. Our chosen metric to measure success is the RMSE of our predicted duration[1].

The patient-level training data (training dataset) we are working with comes from the public domain, and more specifically from various government disease control centres around the world. To give a bit more context on the training dataset, it is a subset of a larger resource including 13,199 patients. From this total pool, features and durations were collected for 576 patients. Those 576 patients' data were then separated into a training set of 219 patients, public testing set of 200 patients, and private testing set of 157 patients[1].

The broad approach we undertook for this project was:

(1) Data exploration
(2) Data preprocessing
(3) Model training, tuning and validation
(4) Iterate through step 1 - step 3 for further model prediction improvements
(5) Create an ensemble from models we deemed exceptionally predictive

Initially we tried a shotgun approach and trained as many different types of models as possible to identify which has the most potential. Through data exploration and further analysis of the models we initially trained, we concluded that linear models seemed to perform best on the dataset.

To combat potential overfitting of the public test set, we also decided to create an ensemble model that includes regularized (Ridge, Lasso, ElasticNet) linear regression, Bayesian regression, and XGBoost. The criteria we used to determine if a model was suitable for our ensemble is Kaggle RMSE less than 4.7. Due to this criteria, we did not include SVM and Random Forest models in the ensemble, though they were implemented and tested.

As a result of our efforts, we combined **25** models into one ensemble, and the resulting Kaggle RMSE we achieved was **4.431** on the public leader-board.

2 DATA EXPLORATION

At the beginning of the module, we manually looked into the dataset and obtained a basic understanding of each feature. In order to improve our understanding further, we systematically explored features through correlation analysis [Fig 1], feature importance generation via XGBoost [Fig 2], clustering via KMeans and Gaussian Mixture Models algorithms, as well as pairwise plots. These methods provided further insight into how the data was structured as well as any important underlying relationships between variables. The fruits of this exploration formed the basis upon which our models were built.

3 PREPROCESSING

Prior to training any machine learning algorithm, careful preprocessing is required to convert data into the appropriate form. In our dataset, many features were not structured homogeneously or consisted of string variables. In the subsections below, we detail our preprocessing steps for the age feature, symptoms feature, and all other categorical variables.

3.1 Age

The two main complications for the age feature were the presence of ranges and NaN values. To handle the range format issue, we filtered the age column for all non-numeric values, then we split each value at '-' and finally adopted the range's mean as the final age value. To handle NaN values, our two options were to either remove all NaN values, or replace all NaN values with the group mean. The second approach preserved more of the original dataset, which was small to begin with, hence it was decided that this was more suitable than removing NaN values.

3.2 Symptoms

To retain maximum information for different symptoms while minimizing dimensionality we first preprocessed the symptoms column by splitting each symptom into its own feature column and then combined synonymous symptoms together into groupings.

The initial splitting was required because the symptoms column had many NaN values and unwanted characters (unicode). To achieve this, we used one-hot encoder to create binary columns for each symptom.[7] To deal with NaN values, we replaced all NaN values with 'Asymptomatic'. In hindsight, we recognize that the absence of recorded symptoms does not necessarily imply an individual did not experience symptoms; it is likely that this is the result of a data collection error. However, since the purpose of this analysis is not to make claims regarding the predictive abilities of features on hospitalization times, the specific encoding of NaN symptoms is meaningless, thus it was left as "Asymptomatic" for the purpose of our study.

In order to reduce dimensionality we combined symptoms that were synonymous or represented similar conditions. For example, 'Dyspnea', meaning labored respiration or 'Shortness of Breath' [3], was combined with other symptoms indicating similar conditions. 'Myalgia', describing muscle aches and paints[4] was joined with its corresponding similar symptoms. We repeated this process for every symptom, the specific groupings themselves are listed in our

publicity available code repository. Grouping symptoms together reduced dimensionality significantly; further, this improved the overall predictive abilities of our models as well, reducing RMSE scores on both validation and test sets.

3.3 Categorical Variable Handling

We used one-hot encoding process on the data to handle all categorical variables. This process takes categorical variables such as 'fever' from symptoms column and converts it into numerical representation without an arbitrary ordering. The categorical variables 'city' and 'province' had little predictive value as there was significant co-linearity between features, thus extensively analysis was performed to remove a majority of these elements from our final predictive feature list. This resulted in reduced model variance and improved RMSE.

3.4 Cross Validation

Cross-validation is an organized way to do repeated data splitting, it produces estimates of RMSE with less bias than a traditional train-test split. In this module, we randomize and split the training dataset so that 80% of the data we used for model training, and the remaining 20% of the data for validation. Each iteration of this splitting is called a "fold". In our cross validation, we usually specified five folds. Each fold served as a testing set one time and training set for predicting each of the other four folds. This algorithm is repeated for each fold and at the end we obtained the average RMSE.

4 MODELS TRAINED

In total, we trained nine different types of machine learning models. Five of them relate to linear regression, and the rest were Random Forest, SVM, XGBoost, and Ensemble. There were two reasons why we trained so many types of models. One, it was an initial approach to explore which model gave us the best results. Two, we wanted to utilize them in an ensemble, in hopes to fight the clear potential for overfitting due to the small dataset.

4.1 Linear Regression: Least squares

Linear regression is the most basic method for modelling the linear relationship between predictors and the dependent outcome variable, this served as our baseline model. The least squares method (least squares) is a form of mathematical regression analysis used to determine the line of best fit for a set of data points.

4.2 Linear Regression: Lasso

Lasso performs two main tasks: regularization and variable selection. Lasso applies a regularization or shrinkage process on the least squares, where it attempts to minimize regressors' coefficients. The strength of regularization is controlled by the tuning parameter λ. Larger λ value implies more coefficients will shrink to zero. Non-zero coefficients after variable selection will be selected to be included in the model. Since our dataset has many indicator variables, Lasso should perform better than least squares by removing redundant variables to reduce variance. The best result we got through Lasso was a public test RMSE of 4.44.

4.3 Linear Regression: Ridge

Ridge regression applies a different regularization method to regressor coefficients. The main difference in comparison with Lasso is that coefficients are reduced toward zero, but will never reach zero. In terms of variable selection, Ridge cannot reduce the number of parameters in the model. It performs better than Lasso regression

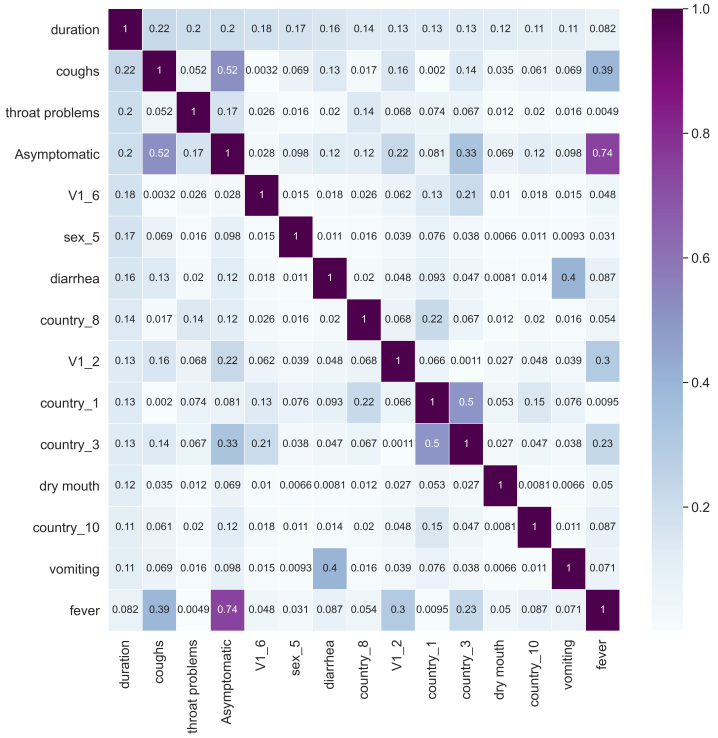

Figure 1: Correlation Analysis Heatmap

when most of the explanatory variables are highly correlated. It can also effectively eliminate collinearity, improve accuracy, and provide more interpretable parameter estimates. The best result we got through Ridge was Kaggle RMSE of 4.41.

4.4 Bayesian Ridge Regression

Judging from the success of the regular linear regression models, we theorized that testing other linear models could be beneficial. In Bayesian Ridge, the response variable is not estimated as a single value, but assumed to be drawn from a probability distribution. The output is generated from a normal distribution. The goal is not to find the single best value for the model parameters, but to determine the posterior distribution for the model parameters; thus the parameters are assumed to come from a distribution as well[6]. Although Bayesian Ridge tends to perform better with more data, our tuned model delivered an acceptable Kaggle RMSE of 4.626, fitting the criteria for inclusion in our ensemble.

4.5 Linear Regression: ElasticNet

ElasticNet is a combination of L1 and L2 regularized regression. It overcomes some L1 regression limitations, those being n variable saturation and general inability to select multiple parameters (albeit, sometimes a strength as well). We expected ElasticNet to perform better than Lasso, however in practice the performance was slightly worse, but still good enough to include in our ensemble.

4.6 Random Forest

Random Forest is an ensemble learning method for classifications and regression. It fits a number of classifying decision trees or sample subsets of dataset and uses averaging to improve accuracy

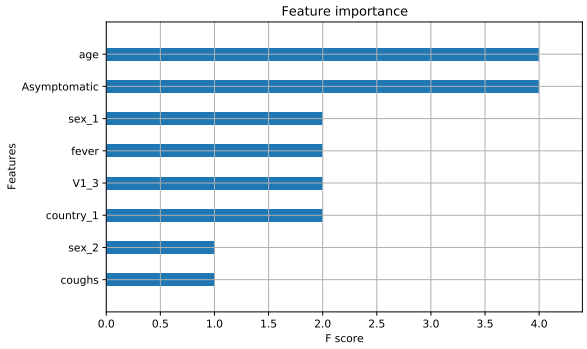

Figure 2: Feature Importance Generated by XGBoost F score calculation

and control overfitting [2]. We decided to implement a Random Forest model as we hoped it would overfit less compared to our other linear models. However, even after tuning the hyper-parameters for the model we did not find it especially predictive, and it did not meet our Kaggle RMSE criteria of 4.7 for the ensemble model.

4.7 Support Vector Machine

Support Vector Machine (SVM) learns the importance of each training data points in relation to representing the decision boundary between different classes. Only those training points that lie on the decision boundary matter, and they are called the support vectors[5]. We tried SVM as part of the initial shotgun approach to see which models are the most promising. StandardScaler was applied to preprocessed data, as SVM is very sensitive to feature scaling. Next, GridSearchCV was utilized to find the best parameters for C, gamma, and epsilon. The best result we got through SVM was Kaggle RMSE of 4.77. In an attempt to reduce the RMSE further, we tried feature selection through both SelectPercentile and RandomForestClassifier. Unfortunately this did not improve the RMSE score. We did not include any SVM models in our ensemble because none made the cutoff public test RMSE of 4.7.

4.8 XG Boost

XGBoost is a decision tree based ensemble algorithm that uses a gradient boosting framework[8]. It is mainly used to handle unstructured non linear data where a neural network would overfit too easily. It utilizes decision trees, bagging and boosting, which is the process of minimizing the errors of the previous models while boosting the influence of better performing models[8]. For this module, we trained a XGBoost model using a RMSE loss function. It did not outperform our other models, however we attribute this to a lack of time for hyper-parameter tuning. There are many parameters to tune and utilizing a randomized grid required time on the order of days. The final model outputted by XGBoost was suitable for inclusion in our ensemble.

4.9 hyper-parameter tuning

To determine the best parameters for our models, we utilized a randomized grid search algorithm with cross validation, this was a computationally expensive but rewarding task as it provided optimal hyper-parameters for model initialization.

5 RESULTS

5.1 Ensemble

In order to create a model that would generalize well to the private leaderboard we created an ensemble of our most successful methods. This process involved randomized testing of 20 combinations of 37 models for their predictive accuracy on the public test dataset. After testing, the models in the best combinations were examined, and the intent was to ensure sufficient model variety existed in the ensemble. We examined the features used by the models and fine tuned our final ensemble to include models that used a wide selection of features. Then we looked at the model type itself and ensured there was no significant bias towards a single model/feature combination.

5.2 Submitted Model

Out of all the models we trained, we decided to combine **25** specifically chosen models into an ensemble. All of them are some variety of the standard regression model except one XGBoost model. The type of algorithm, key features, number of features, validation (CV) RMSE, and public test RMSE are listed below in **Table 1**.
Validation RMSE ranges from **3.696 to 4.155**
Test RMSE ranges from **4.410 to 4.642**
The final test RMSE is **4.431**

5.3 Lessons learned

5.3.1 Teammate: Carl. In this module, we explored the implementation of a variety of machine learning models on the dataset. The ensemble technique is the method that I learned, which seems the most optimal method for this module. Since certain models do well in modelling one aspect of the data, while others do well in modelling another, it's tough to balance the biases and variances. I learned that the ensemble finds a way to combine multiple machine learning models into one predictive model in order to decrease the variance and bias. This approach permits composite prediction, where the final accuracy is better than the accuracy of individual models. In other words, the ensemble learning combined the strength of the models to offset individual model variances and biases. This improves the stability and predictive power of the model while also avoid overfitting.

5.3.2 Teammate: Justin. This module presented a new challenge not of the programming sort. Since I was already quite familiar with Python (our chosen language for a majority of the module) I spent a lot more time helping my teammates to debug their Python scripts and teaching some of the tips and tricks I've learned throughout the years of programming in Python. This helped me further cement my understanding of the various functions and Pythonic programming styles. In order to test the accuracy of our model more reliably, I implemented cross validation for the first time. This taught me the importance of using cross validation to get a more accurate representation of model performance when comparing various models. Lastly, in this module I was able to spend some time learning the XGBoost library. Previously I have only utilized the Scikit-Learn wrapper for the library, but this time around I took the time to learn how to use the base library and this has helped me better understand how to perform hyper-parameter tuning with XGBoost.

5.3.3 Teammate: Neil. This is the first time I have done a competition and module where we would apply machine learning algorithms. I learned how to implement these algorithms in python.

Table 1: Ensemble Architecture

Algorithm	Key features	# of features	Val RMSE	Test RMSE
c_LASSO_10	λ=0.0495	11	3.696	4.443
c_LASSO_3	λ=0.1659	35	3.888	4.498
c_LASSO_7	λ=0.0949	22	3.846	4.483
c_LASSO_9	λ=0.0949	13	3.732	4.475
c_LM_3	N/A	33	4.136	4.642
c_LM_7	N/A	14	4.128	4.551
c_LM_8	N/A	8	4.154	4.441
c_LM_9	N/A	12	4.066	4.438
c_ridge_10	λ=90.9	11	4.028	4.465
c_ridge_1	λ=142.65	38	N/A	4.438
c_ridge_2	λ=149.05	35	N/A	4.451
c_ridge_3	λ=136.05	33	N/A	4.486
c_ridge_4	λ=129.25	32	N/A	4.486
c_ridge_5	λ=115.2	24	N/A	4.497
c_ridge_6	λ=122.65	22	N/A	4.456
c_ridge_7	λ=98.8	14	N/A	4.474
c_ridge_8	λ=97.45	8	N/A	4.412
c_ridge_9	λ=89.15	12	N/A	4.410
j_BayesianRidge_1	BayesianRidge(n_iter =300, alpha_1=1e-10, alpha_2 = 400, lambda_1= 1e-10, lambda_2=2)	19	4.046	4.602
j_BayesianRidge_2	BayesianRidge(n_iter =300, alpha_1=1e-10, alpha_2 = 400, lambda_1= 1e-10, lambda_2=4)	19	3.922	4.625
j_ElasticNet_1	ElasticNet(alpha = 0.035, l1_ratio=0.04)	19	3.924	4.619
j_lasso_1	λ = 0.05	19	3.986	4.643
j_lasso_2	λ = 0.06	19	3.915	4.645
j_ridge_1	λ = 12	19	3.943	4.553
j_xgboost_1	'objective':'reg:squarederror', 'eta':0.5,'gamma':6, 'max_depth':9, 'min_child_weight':40, 'max_delta_step':2, 'subsample':1, 'lambda':0.5, 'alpha': 5	19	4.054	4.612

This module actually helped me learn python, specifically Numpy, Pandas, Matplotlib and scikit-learn libraries. My teammates were experienced with modules like this, so they constantly helped me learn techniques like SVM, Ridge Regression which were quite new to me. They also often helped me debug my code and improve my python skills. During the time of the module, I learned about ensemble technique from both prof. Lloyd and my teammates. Learning about these tools took a lot of time, as quiet often I would to get confused in the sense what is happening or why we are doing this particular model but then my teammates would explain it for me to learn. I learned lot of new thing about statistical models and machine-learning algorithms and also had an opportunity to improve my Python skills.

5.3.4 Teammate: Leo. This module was my first systematic introduction to machine learning and I learned a lot through the class lectures, my teammates, and my own trial and error. From a broad perspective, I now have a better understanding of the general process in utilizing machine learning as a problem solving tool. In addition, this group-work setup helped me feel comfortable communicating with teammates in a machine learning context. For specific skills and knowledge, I learned about data exploration, data preprocessing, best practices in splitting training data and cross validation, parameter tuning, feature selection techniques, how different algorithms work, and more specifically spent some time exploring the support vector machine algorithm.

REFERENCES

[1] URL: https://www.kaggle.com/c/stat440-20-module1.

[2] *3.2.4.3.2. sklearn.ensemble.RandomForestRegressor¶*. URL: https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestRegressor.html.

[3] *Definition of DYSPNEA*. Merriam-webster.com, 2020. URL: https://www.merriam-webster.com/dictionary/dyspnea.

[4] *Definition of MYALGIA*. www.merriam-webster.com. URL: https://www.merriam-webster.com/dictionary/myalgia (visited on 10/09/2020).

[5] Aurelien Geron. *Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow: Concepts, Tools, and Techniques to Build Intelligent Systems*. O'Reilly Media, Incorporated, 2019.

[6] Will Koehrsen. *Introduction to Bayesian Linear Regression*. Apr. 2018. URL: https://towardsdatascience.com/introduction-to-bayesian-linear-regression-e66e60791ea7.

[7] Will Koehrsen. *Random Forest in Python*. Jan. 2018. URL: https://towardsdatascience.com/random-forest-in-python-24d0893d51c0.

[8] Vishal Morde. *XGBoost Algorithm: Long May She Reign!* Apr. 2019. URL: https://towardsdatascience.com/https-medium-com-vishalmorde-xgboost-algorithm-long-she-may-rein-edd9f99be63d.