Predicting Readmission Rates of Diabetic Patients

Stephen Smith

Introduction:

For our project we designed and implemented multiple models and ran experiments on a real world dataset to predict whether a patient would be readmitted to the hospital after 30 days from their initial discharge. We used python and R to implement our models.

The data we used is a real world dataset, representing 10 years (1999-2008) of clinical care at 130 US hospitals and integrated delivery networks. The data contains 50+ attributes such as race, gender, age, HbA1c test result, diabetic medications.

Data Preprocessing:

Preprocessing in R

For logistic regression, each patient encounter must be independent of other encounters. To accomplish this in the data, I sorted the dataset by encounter ID, and then removed rows with duplicate patient ID's, keeping only the first encounter. Some columns in the dataset had over 50% missing values, and those were removed as well. Gender had 3 "Unknown/Invalid" values and those rows were removed. After removing columns with large amounts of missing data, any remaining rows with missing values were also removed, but the number of removed rows was very small. Columns with single values, ie "No" for every instance, were removed as well, as they do not provide any new information. For Logistic Regression, variables were coded as factors, and variables with many factor levels were collapsed into fewer levels. Icd9 codes were changed from numeric values to factors representing the diagnosis group name. After this, training and testing data was set aside for the unbalanced data set. At this point in time, the cleaned data had 71,515 instances, and the training and test data was split 60-40, with 42,909 and 28,606 instances, respectively.

Balancing the Data in R

The data set was incredibly unbalanced, with 91% of patients not being readmitted within 30 days of discharge. To fix this, R has two ways to balance the data, sampling up, and sampling down. Sampling up creates new data points similar to other points in the data, until the data set is balanced. When sampled up was implemented to our training data, the training data grew to include 78,352 instances, split 50-50 by readmittance class. Sampling down samples data points from existing data, and reduces the number of data points overall. After sampling down the training data, 7,466 data points remained. Both of these methods have advantages and disadvantages, but a loss of 90% of the instances of data is not good. The sample up method is the better method for this data set.

Preprocessing in Python

Working off of what we learned from R model, we had to clean the data for python models a little differently. We had to remove variables/columns that had higher than 50% missing data. We also removed variables that only had only a single unique attribute. We looked at imputating remaining missing values, but we found that there weren't many data observations with missing values, so we ultimately decided to delete the rows instead.
After cleaning the missing data, we did one-hot encoding using pandas get_dummies on all categorical data, and for discrete data we manually coded a dummy generation function.
Finally, we made sure all boolean variables were set to 0 or 1, and also the re-admitted variable was coded to 0 for not re-admitted and 1 for re-admitted.

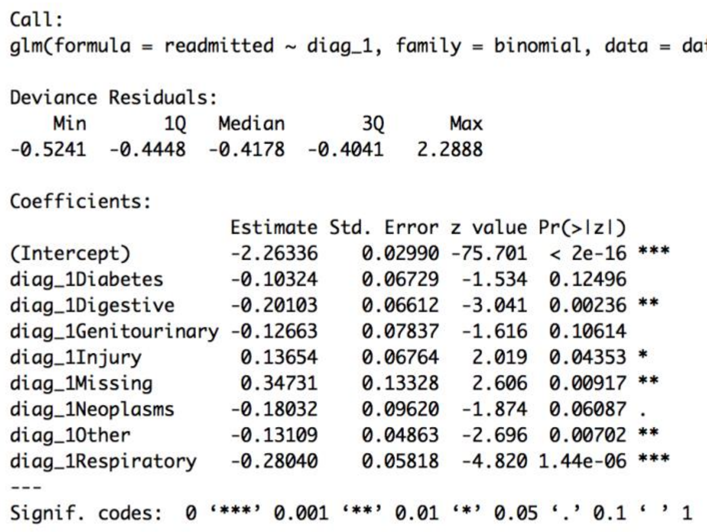
Balancing the Data in Python

For Python we used the SMOTE package from imblearn to perform balancing of the Test data. We applied the oversampling method in SMOTE.

Preliminary Analysis:

Logistic Regression

The first logistic regression model we ran was very simple. Given only the first diagnosis, could the model predict if a patient was readmitted. R automatically converts categorical variables to dummy variables, so we did not have to preprocess the data in additional ways.



```
Call:
glm(formula = readmitted ~ diag_1, family = binomial, data = datatrain)

Deviance Residuals:
    Min       1Q   Median       3Q      Max
-0.5241  -0.4448  -0.4178  -0.4041   2.2888

Coefficients:
                    Estimate Std. Error z value Pr(>|z|)
(Intercept)         -2.26336    0.02990 -75.701  < 2e-16 ***
diag_1Diabetes      -0.10324    0.06729  -1.534  0.12496
diag_1Digestive     -0.20103    0.06612  -3.041  0.00236 **
diag_1Genitourinary -0.12663    0.07837  -1.616  0.10614
diag_1Injury         0.13654    0.06764   2.019  0.04353 *
diag_1Missing        0.34731    0.13328   2.606  0.00917 **
diag_1Neoplasms     -0.18032    0.09620  -1.874  0.06087 .
diag_1Other         -0.13109    0.04863  -2.696  0.00702 **
diag_1Respiratory   -0.28040    0.05818  -4.820 1.44e-06 ***
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1
```

Although the model outputs Z scores and p-values that look significant, when we look at the accuracy of the model, we see the following from our confusion matrix.

Simple Regression Testing Data		
Predicted/Actual	No	Yes
No	26046	2560
Yes	0	0
Accuracy	0.910508285	
Precision	0.910508285	
Recall	1	

Training accuracy for this model is 91.3%, and testing accuracy for this model is 91.1%. Although this might sound good at first, it is not, as the model is just predicting “No” for every outcome, and the variations in accuracy is due to the variations in sampled training vs testing data. We then ran the model with 23 attributes, with the original unbalanced data, and then with both balanced data sets, to be discussed in our results section.

List of Python Training Models used in this project:

- Adaptive Boosting

- Bagging
- Extra-Trees
- Gradient Boost
- Random Forest
- Ridge
- SGD
- KNeighbors
- Multilayer Perceptron
- Tensorflow NN

After reviewing the results from R process, we decided for this project to test many different models to determine which models performed the best. We hypothesized that tree based models would work best for this dataset. We also wanted to include all models we learned about in class. We did not include SVM in our final model set because of the amount of time needed for SVM to run on this dataset.

For each model we ran the dataset first without fixing unbalanced data. Then we used SMOTE over_sampling, and ran our models on the balanced data set. After we identified the best model, we tuned the model to try and achieve a better fit.

Logistic-Regression coded in python was the best performing model, with the following parameters: LogisticRegression(class_weight='balanced', solver='lbfgs', n_jobs=4). I also tried to implement tensorflow NN, however our model kept getting killed by the OS for overflow of memory. This happened both in personal computers and on the erdos server. After some further research, it seems the cross_entropy function was holding the model hostage. We ran out of time to further investigate and fix the problem, so we had to abandon the model. However, we believe that a tensorflow NN would have performed well if we were able to tailor the network to our dataset.

Results and Conclusions:

Logistic Regression

Unbalanced

After the failure of our first simple model, we ran logistic regression again on our unbalanced dataset, this time including 23 categorical attributes as predictors.

Unbalanced Regression		
Predicted/Actual	No	Yes
No	26046	2560
Yes	0	0
Accuracy	0.910508285	
Precision	0.910508285	
Recall	1	

Our training and testing accuracies remain unchanged, as the model is still predicting "No" for every instance. The data is too unbalanced for our model to make reasonable predictions. With this in mind, we tested both balanced datasets.

Balanced Datasets

Sample Down

Balance Down Regression		
Predicted/Actual	No	Yes
No	14038	1117
Yes	12008	1443
Accuracy	0.541180172	
Precision	0.926294952	
Recall	0.538969515	

After balancing the dataset, our accuracy decreased overall, but is more meaningful. Instead of predicting “No” for every patient, we see that the model now predicts “Yes” for some patients. 54% accuracy is not ideal, however, and remember that for the sample down balancing process, we lost nearly 90% of our training data points.

Sample Up

Balance Up Regression		
Predicted/Actual	No	Yes
No	14472	1122
Yes	11574	1438
Accuracy	0.556177026	
Precision	0.92804925	
Recall	0.555632343	

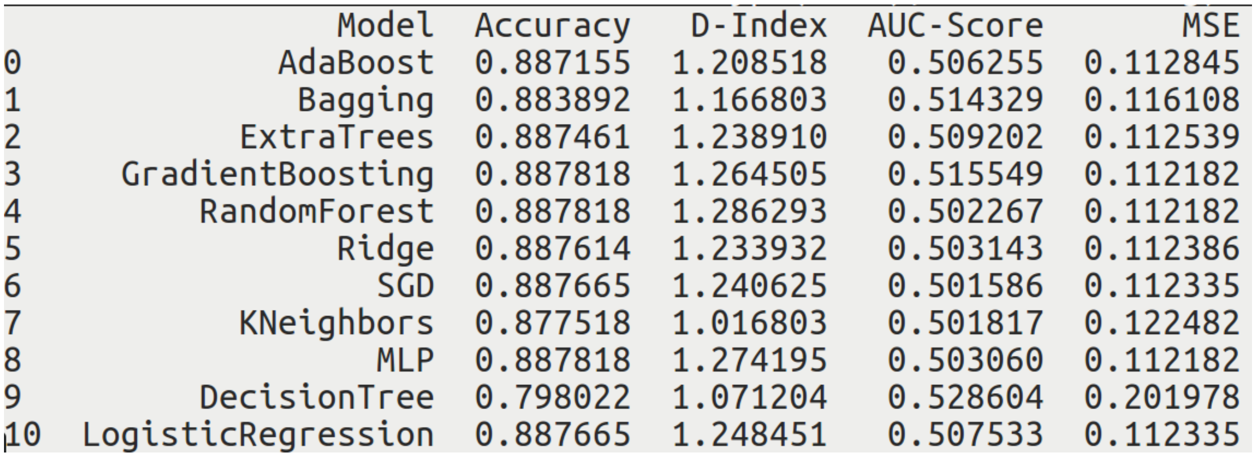
After balancing the dataset with the sample up balancing process, we have slightly better accuracy than the sample down training set. We had more data points available in this data set, but still only predicted 1% better than the model trained on a tiny sample of the data points.

Logistic Regression Conclusion:

With the time limitations of our project, we were unable to create a successful regression model in R. None of our R models performed particularly well in any measurable aspect.

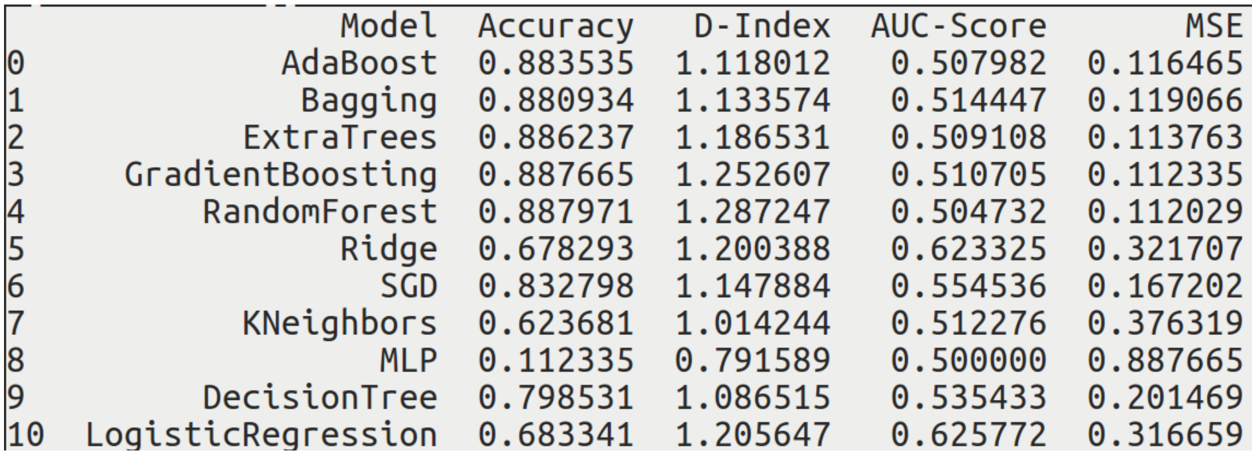
Best Python Models:

Unbalanced Results



```
                 Model  Accuracy   D-Index  AUC-Score       MSE
0             AdaBoost  0.887155  1.208518   0.506255  0.112845
1              Bagging  0.883892  1.166803   0.514329  0.116108
2           ExtraTrees  0.887461  1.238910   0.509202  0.112539
3     GradientBoosting  0.887818  1.264505   0.515549  0.112182
4         RandomForest  0.887818  1.286293   0.502267  0.112182
5                Ridge  0.887614  1.233932   0.503143  0.112386
6                  SGD  0.887665  1.240625   0.501586  0.112335
7           KNeighbors  0.877518  1.016803   0.501817  0.122482
8                  MLP  0.887818  1.274195   0.503060  0.112182
9         DecisionTree  0.798022  1.071204   0.528604  0.201978
10  LogisticRegression  0.887665  1.248451   0.507533  0.112335
```

Sample Up Results



```
                 Model  Accuracy   D-Index  AUC-Score       MSE
0             AdaBoost  0.883535  1.118012   0.507982  0.116465
1              Bagging  0.880934  1.133574   0.514447  0.119066
2           ExtraTrees  0.886237  1.186531   0.509108  0.113763
3     GradientBoosting  0.887665  1.252607   0.510705  0.112335
4         RandomForest  0.887971  1.287247   0.504732  0.112029
5                Ridge  0.678293  1.200388   0.623325  0.321707
6                  SGD  0.832798  1.147884   0.554536  0.167202
7           KNeighbors  0.623681  1.014244   0.512276  0.376319
8                  MLP  0.112335  0.791589   0.500000  0.887665
9         DecisionTree  0.798531  1.086515   0.535433  0.201469
10  LogisticRegression  0.683341  1.205647   0.625772  0.316659
```

Confusion Matrix: (For Models that performed well in different respects)

Extra-Trees	
Training Conf Matrix	Test Conf Matrix

	T	F
P	69579	0
N	0	69579

	T	F
P	17330	78
N	2153	50

Gradient-Boosting	

Training Conf Matrix			Test Conf Matrix		
	T	F		T	F
P	69515	64	P	17354	54
N	8396	61183	N	2149	54

Ridge					
Training Conf Matrix			Test Conf Matrix		
	T	F		T	F
P	48529	21050	P	12085	5323
N	29926	39653	N	986	1217

Logistic Regression					
Training Conf Matrix			Test Conf Matrix		
	T	F		T	F
P	48812	20767	P	12186	5222
N	29564	40015	N	988	1215

Further Tuning of Linear Regression Model Results:

```
                Model  Accuracy   D-Index  AUC-Score       MSE
0  LogisticRegression  0.710571  1.186791    0.60721  0.289429
```

Logistic Regression	
Training Conf Matrix	Test Conf Matrix

	T	F
P	51606	17973
N	33695	33695

	T	F
P	12891	4517
N	1159	1044

For our results, we were more concerned with our value from D-Index (which is a function of accuracy, sensitivity and precision values by using values from the confusion matrix) and also AUC-Score (Area under the curve Score). As these values were a much better indication of the performance of our models than simply accuracy. As we can see from the results above before we resample the dataset to balance the data, we were able to get high values for accuracy, but we were unable to achieve high AUC-scores. Thus, after balancing the data, although our accuracy was lower, we had higher AUC-scores for Logistic Regression and Ridge Classifier. Taking the model with the best AUC-score (Logistic Regression), and relatively higher accuracy, we decided to try and tune the model to improve our score. However it seems that even though we were able to increase the accuracy slightly, it ended up decreasing the D-index and AUC-score. In conclusion, even though we could see logistic regression performed better as a model, it's results were not impressive in python either. Though tree based models had great accuracy, they were unable to archive meaning, sensitivity, or precision from the test data.

Strengths and Weaknesses:

Strengths

Random Forest combines the idea of bagging and the random selection of features, introduced independently in order to construct a collection of decision trees with controlled variation. Logistic regression is intrinsically simple, it has low variance and so is less prone to overfitting.

Weaknesses

Random Forest is quite slow to create predictions once trained. More accurate ensembles require more trees, which means using the model becomes slower. There can be situations where run-time performance is important and therefore other approaches would be preferred. A weakness that plagues all the "tree" models is that they don't predict false positives and another weakness of our logistic regression models is that none of our models performed particularly well.

Future Work:

In the future I would like to map the ICD-9 codes to the to the diagnosis codes provided in the dataset to see certain types of diagnosis could drive a patient to be readmitted. The diagnosis codes should be rolled up to higher level because there are hundreds of them. I hypothesize that certain diagnosis would correlated with other attributes would be a leading factor in determining whether or not a patient would be readmitted. For example, a patient who goes in for heart surgery would be more likely to be readmitted to the hospital if they had a weight problem but a different patient may have been admitted for a broken bone but there weight should no influence the likelihood of the patient needing to be readmitted.

Another analysis that could be done is logistic regression with the inclusion of interaction terms. It is likely that the attributes of the patient are not independent, so running additional models that account for that may improve model accuracy. In addition, to improve the performance of the logistic model, selection of important attributes could be done to eliminate unnecessary columns from the data.

Works Cited

Strack, Beata, et al. “Impact of HbA1c Measurement on Hospital Readmission Rates: Analysis of 70,000 Clinical Database Patient Records.” *BioMed Research International*, vol. 2014, 2014, pp. 1–11., doi:10.1155/2014/781670.