Data Science

Alan Gabriel Paredes Cetina
00279105
Dr. Alberto Muñoz Ubando

Final Project

November 28th, 2019

Resume

For this project I will be attending the social issue of diabetes, specifically the prediction of people having it. For that I will be using a Machine Learning algorithm that will analyze several health factors and predict which individuals are more likely to have it. This algorithm will be programmed in R using the library of healthcareai.

Tools and Resources Used

1. **RStudio:** is an integrated development environment (IDE) for R. It includes a console, syntax-highlighting editor that supports direct code execution, as well as tools for plotting, history, debugging and workspace management.
2. **Healthcareai:** is designed to streamline healthcare machine learning. They do this by including functionality specific to healthcare, as well as simplifying the workflow of creating and deploying models

The Algorithm & Analysis

Getting Started

In order to use healthcareai, first I had to upload it on my project

```
library(healthcareai)
```

This library comes with a built in dataset documenting diabetes among adult Pima females. Once I attached the package, the dataset is available in the variable pima_diabetes. I took a look at the data with the str function. There are 768 records in 10 variables including one identifier column, several nominal variables, and substantial missingness (represented in R by NA). We can see this dataset with the next line of code:

```
str(pima_diabetes)
```

```
# > Classes 'tbl_df', 'tbl' and 'data.frame':   768 obs. of  10
variables:
# >  $ patient_id    : int  1 2 3 4 5 6 7 8 9 10 …
# >  $ pregnancies   : int  6 1 8 1 0 5 3 10 2 8 …
# >  $ plasma_glucose: int  148 85 183 89 137 116 78 115 197 125
…
# >  $ diastolic_bp  : int  72 66 64 66 40 74 50 NA 70 96 …
# >  $ skinfold      : int  35 29 NA 23 35 NA 32 NA 45 NA …
# >  $ insulin       : int  NA NA NA 94 168 NA 88 NA 543 NA …
# >  $ weight_class  : chr  "obese" "overweight" "normal"
"overweight" …
# >  $ pedigree      : num  0.627 0.351 0.672 0.167 2.288 …
# >  $ age           : int  50 31 32 21 33 30 26 29 53 54 …
# >  $ diabetes      : chr  "Y" "N" "Y" "N" ...
```

Quick Machine Learning

`machine_learn` always gets the name of the data frame, then any columns that should not be used by the model (uninformative columns, such as IDs), then the variable to be predicted with `outcome =`.

```
quick_models <- machine_learn(pima_diabetes, patient_id, outcome
= diabetes)
```

```
# > Training new data prep recipe...
# > Variable(s) ignored in prep_data won't be used to tune
models: patient_id
# >
# > diabetes looks categorical, so training classification
algorithms.
```

```
# >
# > After data processing, models are being trained on 12
features with 768 observations.
# > Based on n_folds = 5 and hyperparameter settings, the
following number of models will be trained: 50 rf's, 50 xgb's,
and 100 glm's
# > Training with cross validation: Random Forest
# > Training with cross validation: eXtreme Gradient Boosting
# > Training with cross validation: glmnet
# >
# > *** Models successfully trained. The model object contains
the training data minus ignored ID columns. ***
# > *** If there was PHI in training data, normal PHI protocols
apply to the model object. ***
```

`machine_learn` has told us that it has created a recipe for data preparation (this allows us to do exactly the same data cleaning and feature engineering when you want predictions on a new dataset), is ignoring patient_id when tuning models as I told it to, is training classification algorithms because the outcome variable diabetes is categorical, and has executed cross validation for three machine learning models: random forests, XGBoost, and regularized regression. To see what the model looks like, I simply had to code `quick_models.`

```
quick_models
# > Algorithms Trained: Random Forest, eXtreme Gradient Boosting,
and glmnet
# > Model Name: diabetes
# > Target: diabetes
# > Class: Classification
# > Performance Metric: AUROC
# > Number of Observations: 768
# > Number of Features: 12
# > Models Trained: 2018-09-01 18:26:54
# >
# > Models tuned via 5-fold cross validation over 10 combinations
of hyperparameter values.
# > Best model: Random Forest
# > AUPR = 0.71, AUROC = 0.84
# > Optimal hyperparameter values:
# >   mtry = 9
```

```
# >    splitrule = extratrees
# >    min.node.size = 9
```

Everything looks as expected, and the best model is a random forest that achieves performance of AUROC = 0.84. Now that I had my models, I could make predictions using the `predict` function.

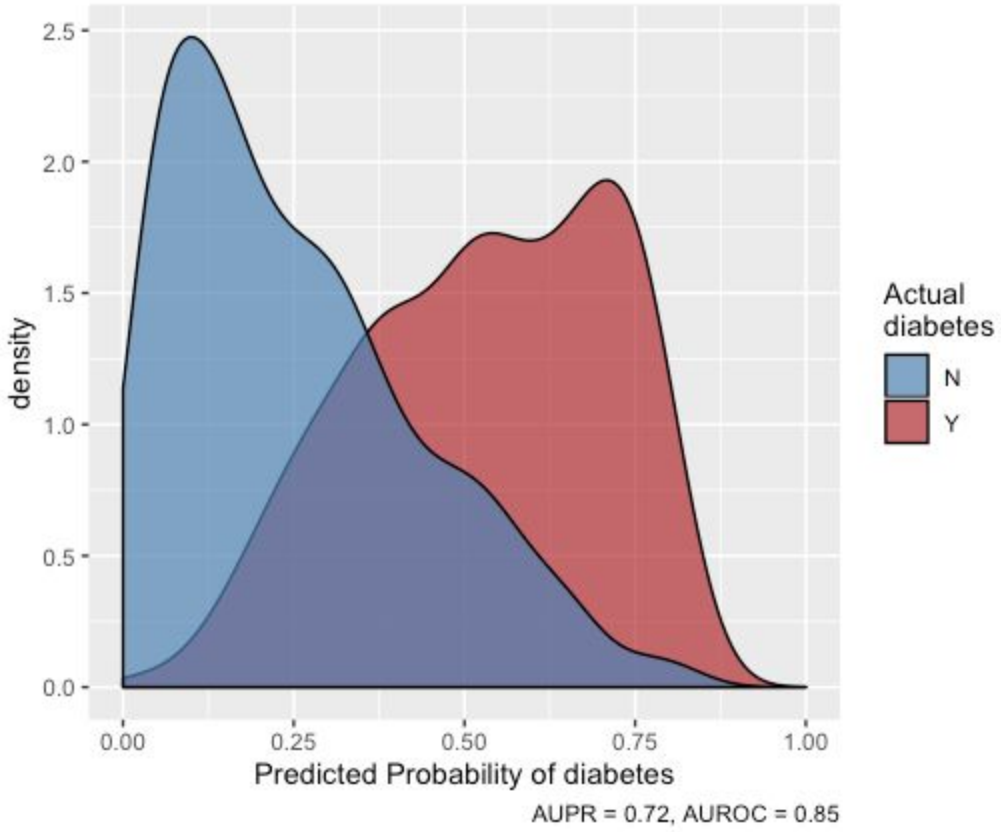
```
predictions <- predict(quick_models)

predictions
```

```
# > "predicted_diabetes" predicted by Random Forest last trained:
2018-09-01 18:26:54
# > Performance in training: AUROC = 0.84
# > # A tibble: 768 x 11
# >   diabetes predicted_diabe… patient_id pregnancies
plasma_glucose
# > * <fct>       <dbl>      <int>       <int>          <int>
# > 1 Y           0.765          1           6            148
# > 2 N           0.0506         2           1             85
# > 3 Y           0.518          3           8            183
# > 4 N           0.00349        4           1             89
# > 5 Y           0.637          5           0            137
# > # ... with 763 more rows, and 6 more variables: diastolic_bp
<int>,
# > #   skinfold <int>, insulin <int>, weight_class <chr>,
pedigree <dbl>,
# > #   age <int>
```

I got a message about when the model was trained and how well it performed in training, and I got back a data frame that looked sort of like the original, but had a new column `predited_diabetes` that contained the model-generated probability each individual has diabetes, and contained changes that were made preparing the data for model training, e.g. missingness had been filled in and weight_class had been split into a series of “dummy” variables. I then plotted how effectively the model was able to separate diabetic from non-diabetic individuals by calling the `plot` function on the output of `predict`.

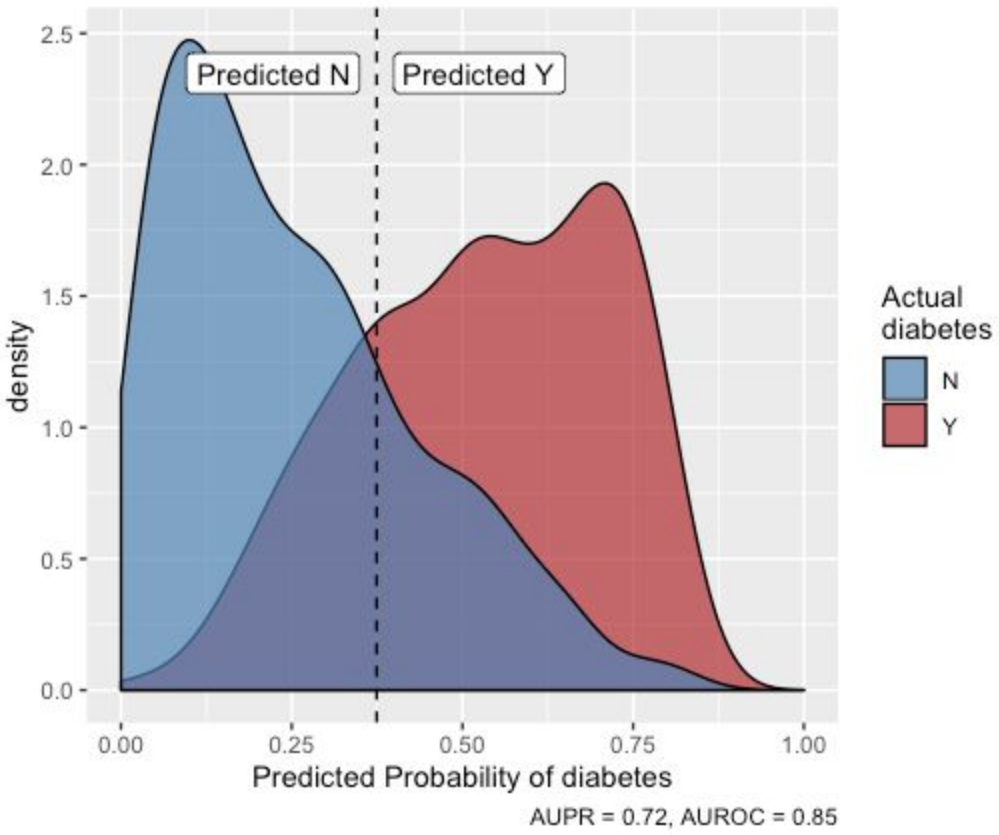
```
plot(predictions)
```



Plot 1. Predicted Probability of Diabetes

To outcome-class predictions in addition to predicted probabilities, the `outcome_groups` argument accomplishes that. If it is `TRUE` the overall accuracy of predictions is maximized. If it is a number, it represents the relative cost of a false-negative to a false-positive outcome. In this algorithm below says that one false negative is as bad as two false positives.

```
quick_models %>% predict(outcome_groups = 2) %>%  plot()
```

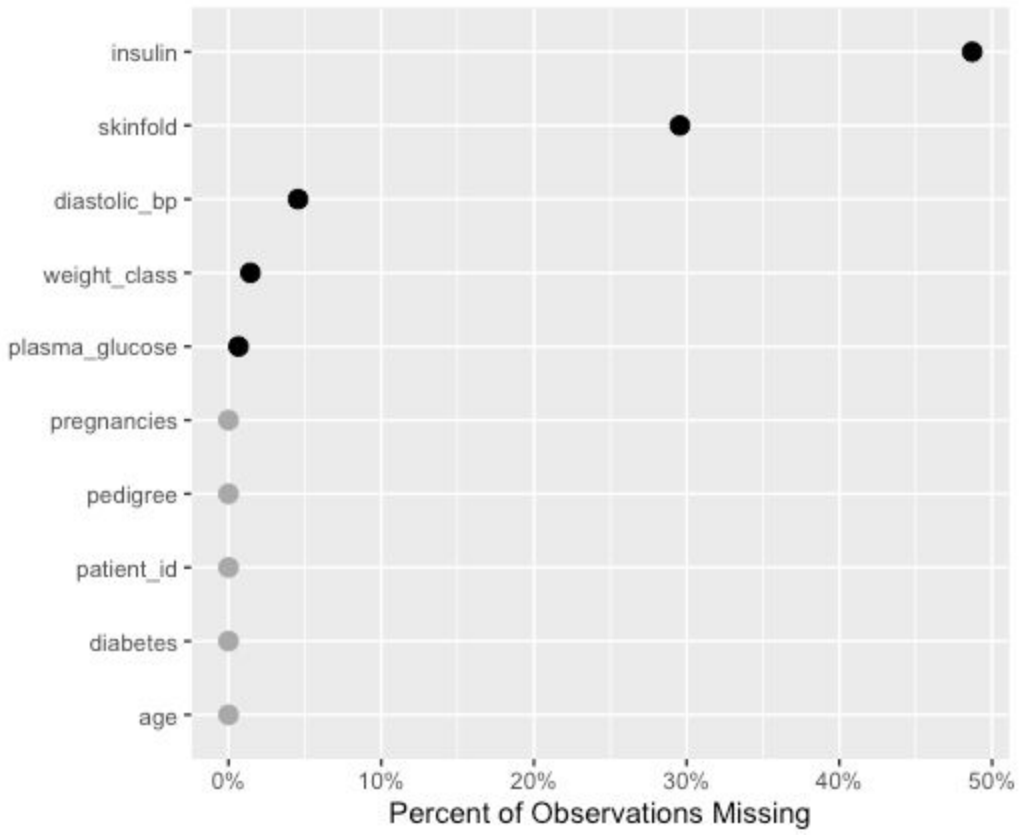


Plot 2. Predicted Probability of Diabetes With Outcome Groups

Data Profiling

It is always a good idea to be aware of where there are missing values in data. The `missingness` function helps with that. In addition to looking for values R sees as missing, it looks for other values that might represent missing, such as "`NULL`", and issues a warning if it finds any. Healthcareai has a plot method that allows me to inspect the results more quickly and intuitively by passing the output to `plot`.

```
missingness(pima_diabetes) %>% plot()
```



Plot 3. Percentage of Observations Missing

It's good that I didn't have any missingness in my ID or outcome columns. Let's see how missingness in predictors is addressed further down.

Data Preparation

To get an honest picture of how well the model performs (and an accurate estimate of how well it will perform on yet-unseen data), it is wise to hide a small portion of observations from model training and assess model performance on this "validation" or "test" dataset. The `split_train_test` function simplifies this, and it ensures the test dataset has proportionally similar characteristics to the training dataset. By default, 80% of observations are used for

training; that proportion can be adjusted with the p parameter. The seed parameter controls randomness so that I can get the same split every time I run the code.

```
split_data <- split_train_test(d = pima_diabetes, outcome =
diabetes, p = .9, seed = 84105)
```

One of the major workhorse functions in healthcareai is prep_data. Here, I customized preparation to scale and center numeric variables and avoid collapsing rare factor levels into "other". The first arguments to prep_data are the same as those to machine_learn: data frame, ignored columns, and the outcome column. Then I specified prep details.

```
prepped_training_data <- prep_data(split_data$train, patient_id,
outcome = diabetes, center = TRUE, scale = TRUE,
collapse_rare_factors = FALSE)

# > Training new data prep recipe...
```

The "recipe" that the above message refers to is a set of instructions for how to transform a dataset the way I just transformed the training data. Any machine learning that I do (within healthcareai) on prepped_training_data will retain that recipe and apply it before making predictions on new data.

Model Training

For this part, I used tune_models function which searches over hyperparameters to optimize model performance. I tuned all three available models: random forests ("RF"), regularized regression (i.e. lasso and ridge, "GLM"), and gradient-boosted decision trees (i.e. XGBoost, "XGB"). To optimize model performance, I cranked tune_depth up a little from its default value of ten. That will tune the models over more combinations of hyperparameter values in the search for the best model. I also select "PR" as my model metric. That optimizes for area under the precision-recall curve rather than the default of area under the receiver operating characteristic curve ("ROC").

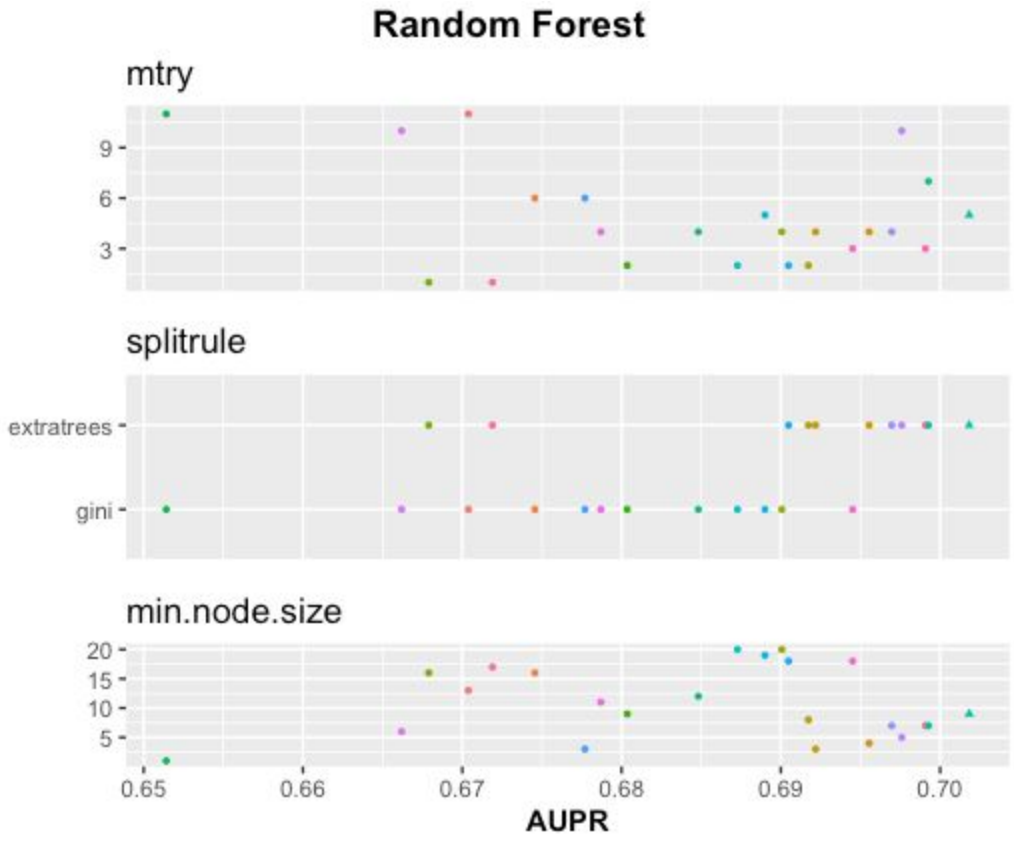
```
models <- tune_models(d = prepped_training_data, outcome =
diabetes, tune_depth = 25, metric = "PR")
# > Variable(s) ignored in prep_data won't be used to tune
models: patient_id
# >
# > diabetes looks categorical, so training classification
algorithms.
# >
# > After data processing, models are being trained on 13
features with 692 observations.
# > Based on n_folds = 5 and hyperparameter settings, the
following number of models will be trained: 125 rf's, 125 xgb's,
and 250 glm's
# > Training with cross validation: Random Forest
# > Training with cross validation: eXtreme Gradient Boosting
# > Training with cross validation: glmnet
# >
# > *** Models successfully trained. The model object contains
the training data minus ignored ID columns. ***
# > *** If there was PHI in training data, normal PHI protocols
apply to the model object. ***
```

Next I used `evaluate` function in order to compare performance across models.

```
evaluate(models, all_models = TRUE)
# > # A tibble: 3 x 3
# >   model                      AUPR AUROC
# >   <chr>                     <dbl> <dbl>
# > 1 Random Forest             0.715 0.841
# > 2 eXtreme Gradient Boosting 0.711 0.827
# > 3 glmnet                    0.697 0.823
```

Next I plotted only the best model's performance over hyperparameter by extracting it by name. It looks like extratrees is a superior split rule for this model.

```
models["Random Forest"] %>% plot()
```



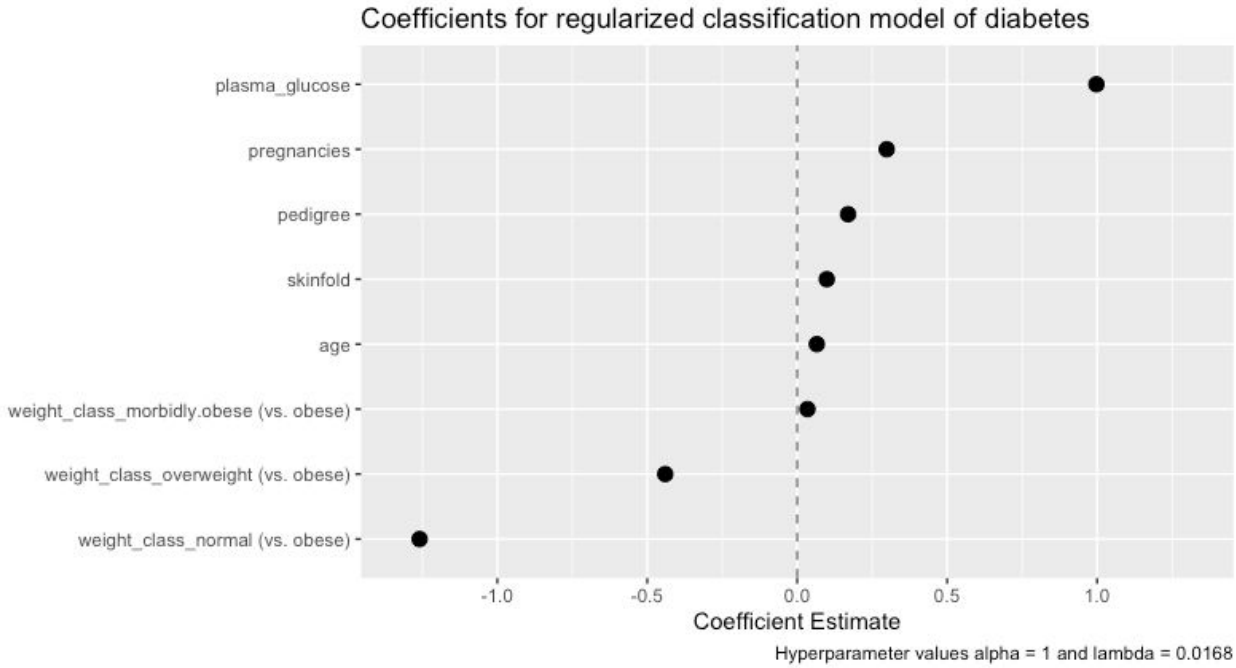
Plot 4. Random Forest Performance over Hyperparameters

Interpretation

With the `interpret` function I got coefficient estimates from a regularized logistic or linear regression model.

```
interpret(models) %>% plot()
```

```
# > Warning in interpret(models): Interpreting glmnet model, but
Random Forest
# > performed best in cross-validation and will be used to make
predictions. To
# > use the glmnet model for predictions, extract it with
x['glmnet'].
```



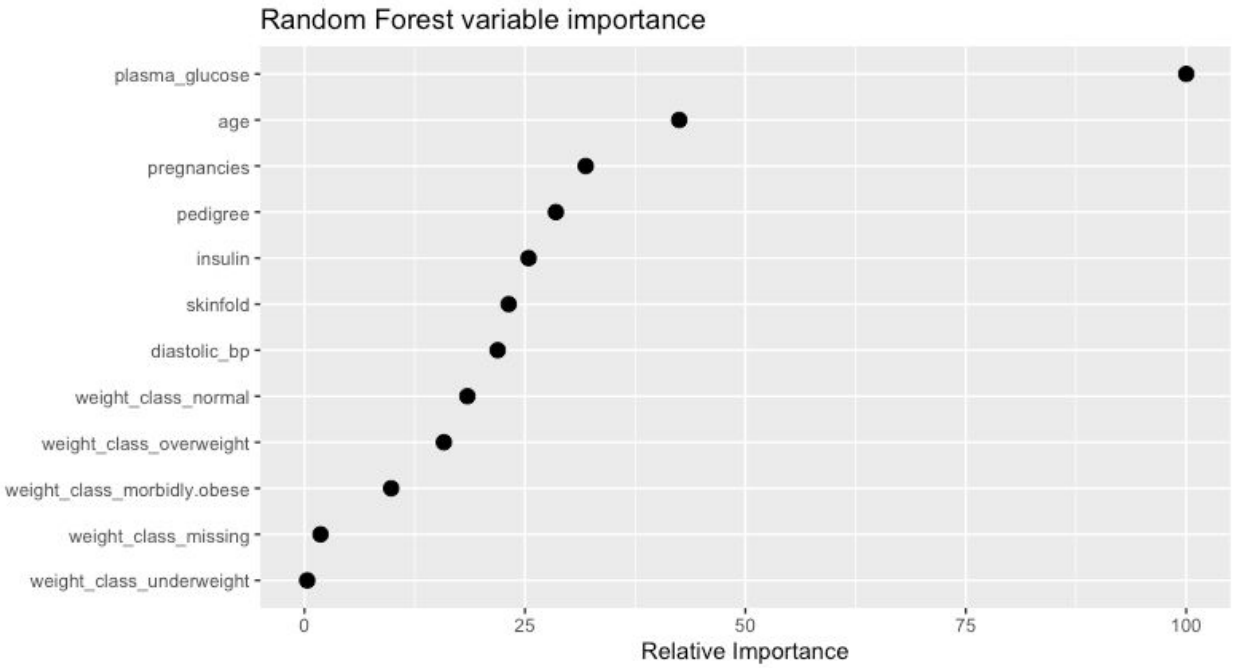
Plot 5. Coefficients for Regularized Classification Model of Diabetes

In the plot above, a unit increase in plasma glucose corresponds to an expected log-odds increase of diabetes of just over one. n this plot, the low value of `weight_class_normal` signifies that people with normal weight are less likely to have diabetes. Similarly, plasma glucose is associated with increased risk of diabetes after accounting for other variables.

Variable Importance

Tree based methods such as random forest and boosted decision trees can't provide coefficients like regularized regression models can, but they can provide information about how important each feature (aka predictor, aka variable) is for making accurate predictions. I plotted these "variable importances" by calling `get_variable_importance` on the model object.

```
get_variable_importance(models) %>% plot()
```



Plot 6. Importance of Variables in Random Forest Model

Prediction

`predict` will automatically use the best-performing model from training (evaluated out-of-fold in cross validation).

```
predict(models)
# > "predicted_diabetes" predicted by Random Forest last trained:
2018-09-01 18:27:33
# > Performance in training: AUPR = 0.71
# > # A tibble: 692 x 11
# > diabetes predicted_diabe… patient_id pregnancies
plasma_glucose
# > * <fct>          <dbl>      <int>       <int>          <int>
# > 1 N            0.0874           2           1             85
# > 2 Y            0.409            3           8            183
# > 3 N            0.0134           4           1             89
```

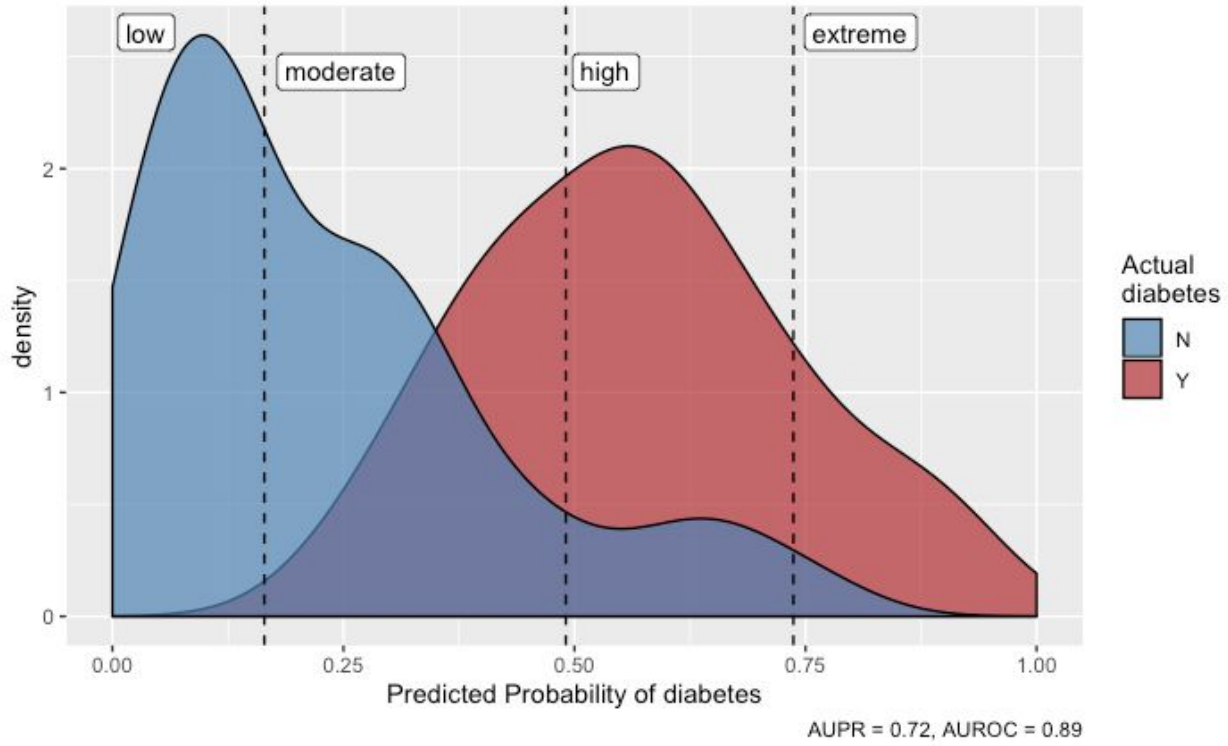
```
# > 4 Y            0.559           5           0
137
# > 5 N            0.202           6           5            116
# > # ... with 687 more rows, and 6 more variables: diastolic_bp
<int>,
# > #   skinfold <int>, insulin <int>, weight_class <chr>,
pedigree <dbl>,
# > #   age <int>
```

I plotted the predictions to see how well the model was doing, and I saw that it's separating diabetic from non-diabetic individuals pretty well, although there is a fair number of non-diabetics with high predicted probabilities of diabetes. This may be due to optimizing for precision recall, or may indicate pre-diabetic patients.

Finally, I made risk-group predictions, defining four risk groups (low, moderate, high, and extreme) containing 30%, 40%, 20% and 10% of patients, respectively.

```
test_predictions <- predict(models, split_data$test,risk_groups = c(low
= 30, moderate = 40, high = 20, extreme = 10))
# > Prepping data based on provided recipe

plot(test_predictions)
```



Plot 7. Risk-Groups Predictions

Conclusions

Healthcareai is a great library for medical purposes, it made possible a huge and importante analysis in a few lines of code with great accuracy. As for the problem, it was possible at the end to make predictions of people having diabetes though the analysis of medical data from patients. This could have great benefits if applied to yucatecan citizens, if we could have the data to train a machine learning model as I did in this example we could effectively determine if a patient could have diabetes or not, and with that we could take a course of actions in order to avoid it. The algorithm had an outstanding performance with and AUROC up to 84% and an AUPR of 71%.

References

[1] M. Levy, M. Mastanduno, T. Miller & L. Thatcher. “Tools for Healthcare Machine Learning”. *Healthcareai.* 2019. Retrieved from https://docs.healthcare.ai on november 20th, 2019.

[2] S. Narkhede. “Understanding AUC - ROC Curve”. *Towards Data Science: Medium.* 2018. Retrieved from https://towardsdatascience.com/understanding-auc-roc-curve-68b2303cc9c5 on november 20th, 2019.