Machine Learning and Extrapolation: Predicting University of Iowa Student Enrollment

Sarah Coleman (Concordia University Chicago) & Noah Johnson (Gustavus Adolphus College)

Faculty Mentor: Dr. Grant Brown

Outline

Introduction

- Purpose of Project
- Data

Julia

- Strengths and challenges

Modeling

- Penalized Regression: GLMNet
- Cross - validation & Extrapolation
- **MLJ**
- Model Performance over time

Future Explorations

Project Purpose

- The purpose of this project was to use classification machine learning models on enrollment data to predict whether an admitted student would enroll or not.

- This is important because every year there is a large pool of applicants and only a small subset ends up enrolling.

- Our goal was to see if we could create a model that could make accurate predictions

The Data

- The data was collected weekly from 2016 to 2021
- For each student applying, there were around 400 variables to describe them
- There was a lot of missing data
- With so much data we needed a new approach

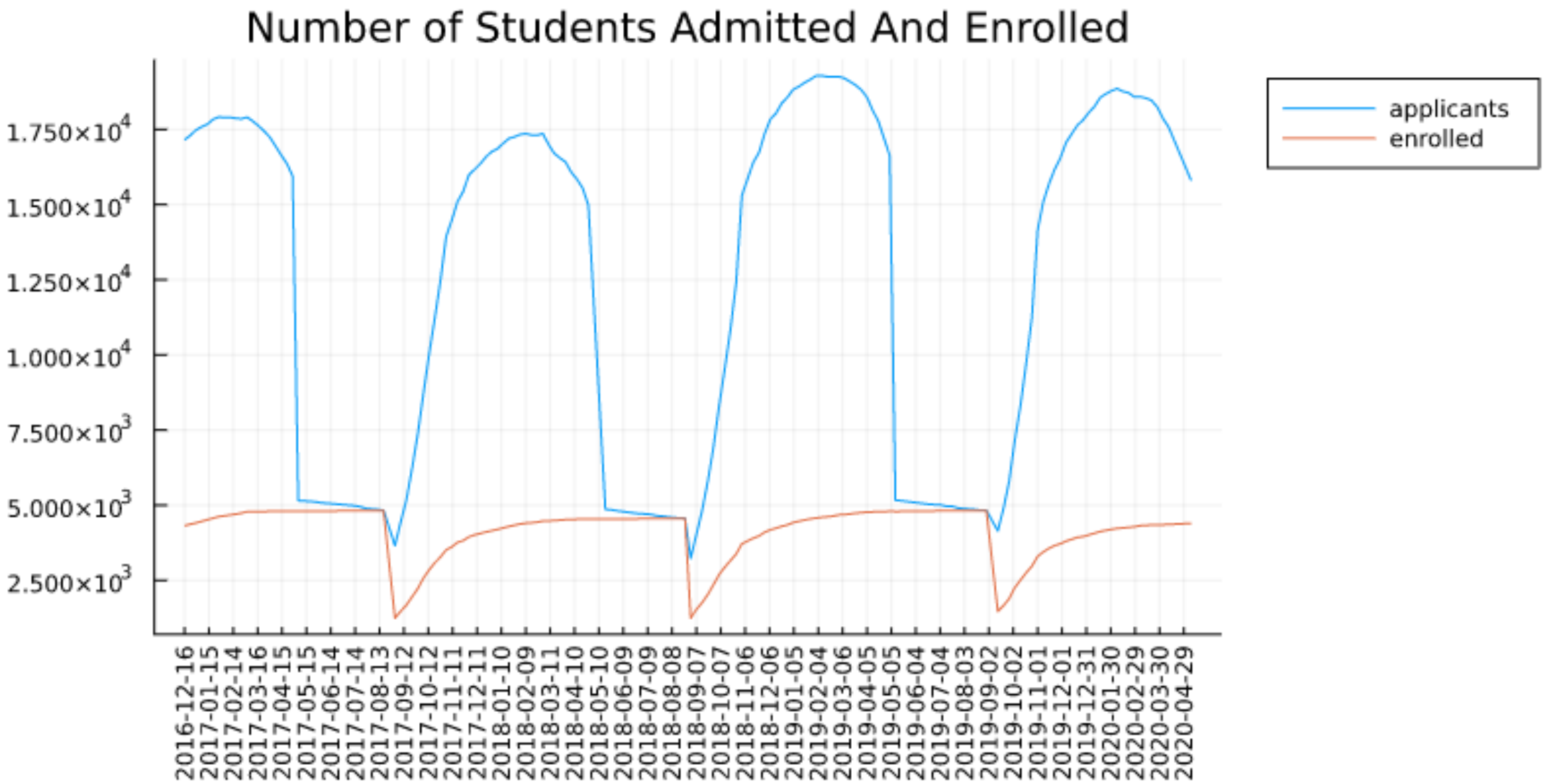
Variables

Categorical

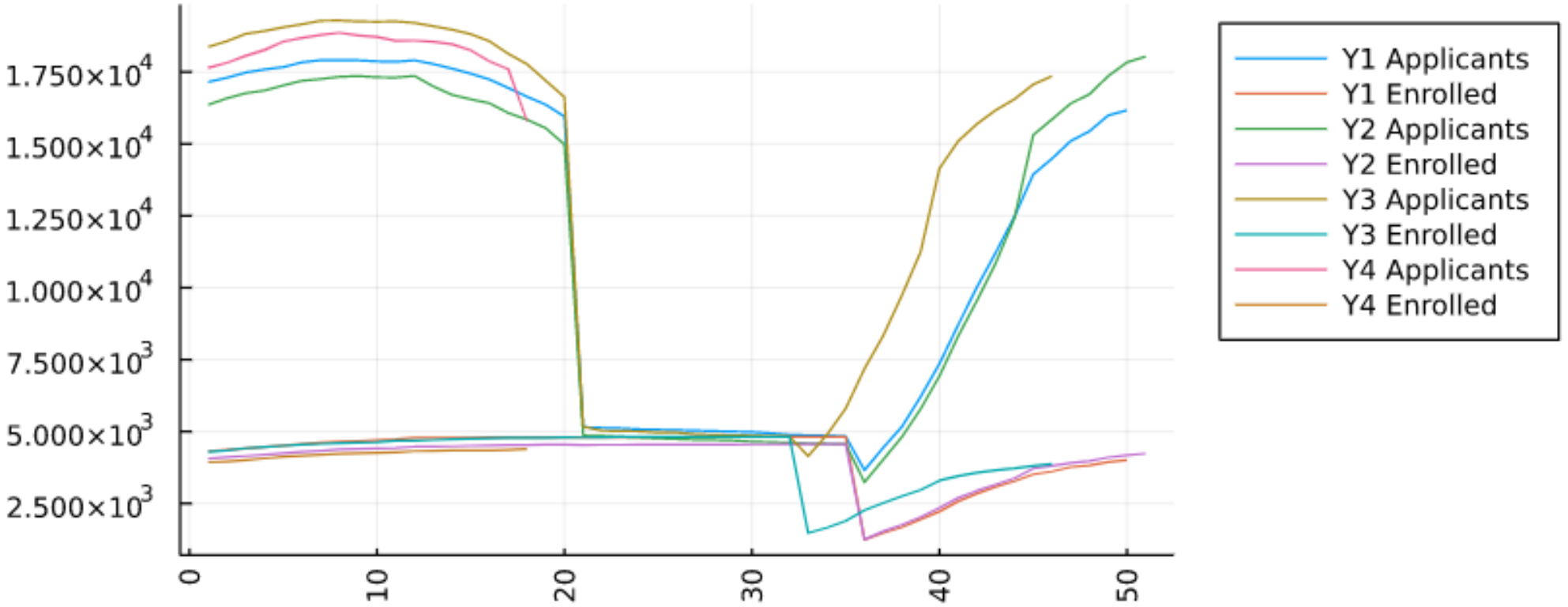
- Parents' Higher Education
- Foreign Language in HS
- Athlete Status
- Number of Youth Programs
- Administration Decision Type
- Application Source
- State Proxy
- Family Alumni
- Area of study

Numerical

- Distance to Campus
- ACT Composite Score
- HS GPA
- Number of Applications
- Days since first Inquiry
- Awards & Scholarships


Number of Students Admitted And Enrolled
applicants
enrolled
1.750×10^4
1.500×10^4
1.250×10^4
1.000×10^4
7.500×10^3
5.000×10^3
2.500×10^3
2016-12-16
2017-01-15
2017-02-14
2017-03-16
2017-04-15
2017-05-15
2017-06-14
2017-07-14
2017-08-13
2017-09-12
2017-10-12
2017-11-11
2017-12-11
2018-01-10
2018-02-09
2018-03-11
2018-04-10
2018-05-10
2018-06-09
2018-07-09
2018-08-08
2018-09-07
2018-10-07
2018-11-06
2018-12-06
2019-01-05
2019-02-04
2019-03-06
2019-04-05
2019-05-05
2019-06-04
2019-07-04
2019-08-03
2019-09-02
2019-10-02
2019-11-01
2019-12-01
2019-12-31
2020-01-30
2020-02-29
2020-03-30
2020-04-29

Number of Students Admitted And Enrolled




julia

Julia Strengths

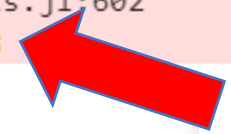
One of the faster computational languages

Easy to reproduce results (Juptyer Notebooks)

Open Source

```
(17355, 190)(16552, 190)
┌ Info: Training Machine{ProbabilisticTunedModel{Grid,…},…} @468.
└ @ MLJBase C:\Users\noahj\.julia\packages\MLJBase\diSrF\src\machines.jl:354
┌ Info: Attempting to evaluate 100 models.
└ @ MLJTuning C:\Users\noahj\.julia\packages\MLJTuning\Uj5Cx\src\tuned_models.jl:602
Evaluating over 40 metamodels: 100%[=========================] Time: 0:28:15
┌ Info: Only 40 (of 100) models evaluated.
│ Model supply exhausted.
└ @ MLJTuning C:\Users\noahj\.julia\packages\MLJTuning\Uj5Cx\src\tuned_models.jl:529
```

```
┌ Info: Training Machine{ProbabilisticTunedModel{Grid,…},…} @476.
└ @ MLJBase C:\Users\noahj\.julia\packages\MLJBase\diSrF\src\machines.jl:354
┌ Info: Attempting to evaluate 100 models.
└ @ MLJTuning C:\Users\noahj\.julia\packages\MLJTuning\Uj5Cx\src\tuned_models.jl:602
Evaluating over 10 metamodels:  20%[=====>                    ]  ETA: 2:44:38
```



Julia Drawbacks

- It can be slow at times because of the computation
- Took time to learn
- Tough to correct errors

Models -- Overview

- Classification
  - Training Data
  - Test Data
- GLMNet - LASSO
- Cross validation
- MLJ – DecisionTrees.jl
  - XGBoost
  - RandomForest
  - AdaBoostStumpClassifier
- Extrapolation

	Sensitivity	Specificity
	Float64	Float64
1	0.567662	0.873549

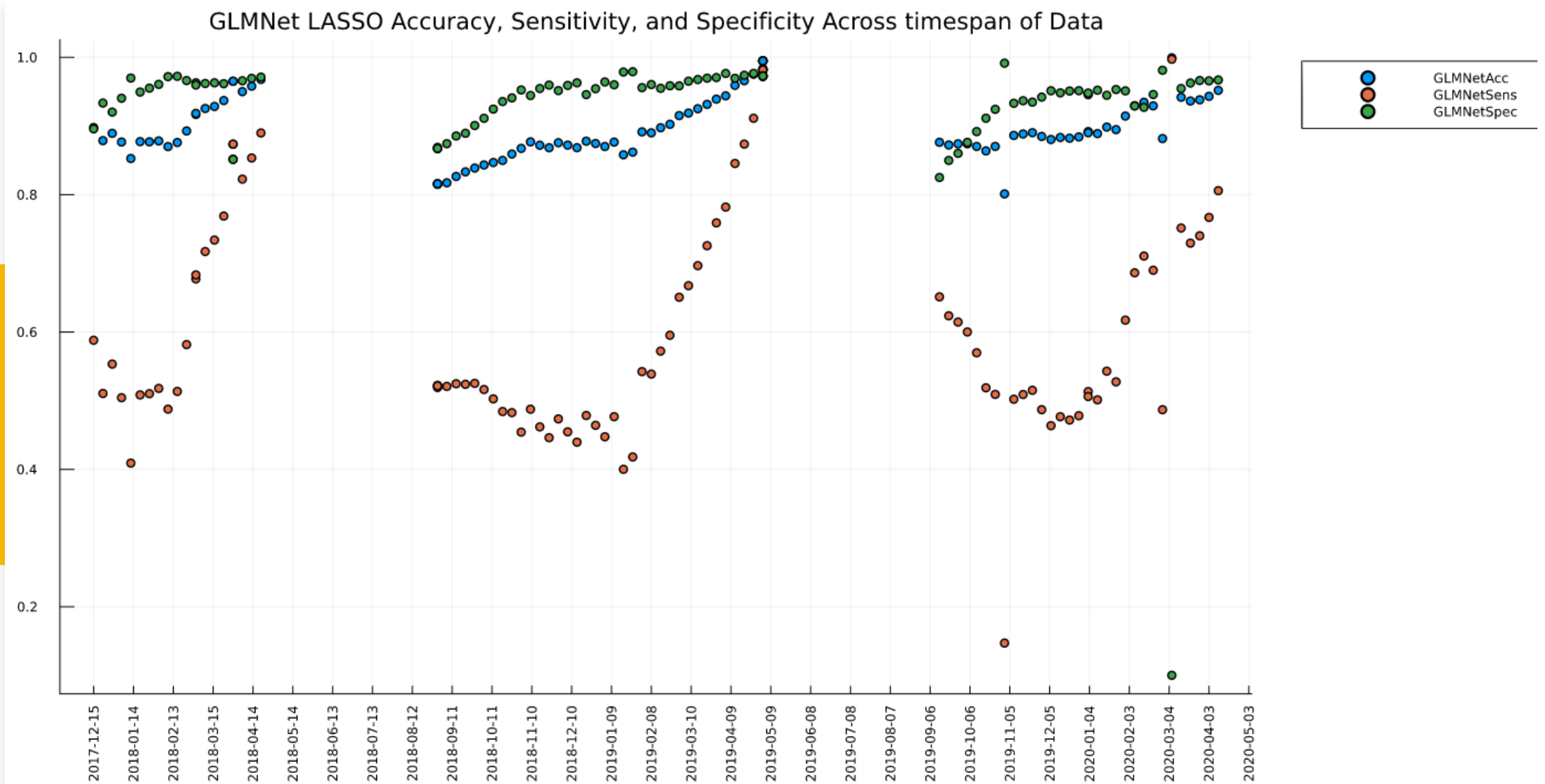
	Parameter	Estimate
	String	Float64
1	(Intercept)	0.0
2	Max_HS_FOUR_PT_GPA	-0.114389
3	Max_HS_FOUR_PT_GPA_missing	-0.101962
4	ACT_COMPOSITE	-0.00523698
5	ACT_COMPOSITE_missing	-0.144446
6	ADMN_DECISION_TYPE_EN: DEFERRED	-1.88605
7	ADMN_DECISION_TYPE_EN: NO_DECISION	-1.23244
8	StateProxy: IOWA	2.04616
9	StateProxy: NA	-2.20916
10	StateProxy: NATIONAL	-0.21616
11	StateProxy: OTHER	0.347326

Into an example LASSO

Model output

	Sensitivity	Specificity
	Float64	Float64
1	0.567662	0.873549

	observed	prediction	fitted_value
	Any	Int64	Float64
1	0	1	0.57137
2	0	1	0.602332
3	0	0	0.144293
4	1	1	0.539983
5	0	0	0.133821
6	1	1	0.551146
7	0	1	0.569109
8	0	1	0.601627
9	1	1	0.569078
10	0	1	0.590242


GLMNet LASSO Accuracy, Sensitivity, and Specificity Across timespan of Data
1.0
0.8
0.6
0.4
0.2
GLMNetAcc
GLMNetSens
GLMNetSpec
2017-12-15
2018-01-14
2018-02-13
2018-03-15
2018-04-14
2018-05-14
2018-06-13
2018-07-13
2018-08-12
2018-09-11
2018-10-11
2018-11-10
2018-12-10
2019-01-09
2019-02-08
2019-03-10
2019-04-09
2019-05-09
2019-06-08
2019-07-08
2019-08-07
2019-09-06
2019-10-06
2019-11-05
2019-12-05
2020-01-04
2020-02-03
2020-03-04
2020-04-03
2020-05-03

Machine Learning Coding Patterns

GLMNet Code

```
function get_glmnet_cv_results(X,y,K)
    # Assign every observation to a CV fold randomly
    fold_indices = [(i % K)+1 for i in randperm(size(y)[1])]
    # Create a vector to store the prediction probability
    pred_prob = zeros(Float64, size(y)[1])
    # Create a vector to keep track of the lambdas which are used
    lambdas = zeros(Float64, K)
    # Iterate over the folds
    for fold_idx in 1:K
        # Identify test and traning data
        X_test = X[findall(fold_indices .== fold_idx),:]
        X_train = X[findall(fold_indices .!= fold_idx),:]
        Y_test = y[findall(fold_indices .== fold_idx)]
        Y_train = y[findall(fold_indices .!= fold_idx)]
        # Within the trainin data, find the optimal lambda
        #  value with an additional round of cross validation
        cv = glmnetcv(X_train, Y_train)
        best_lambda = cv.lambda[argmin(cv.meanloss)]
        lambdas[fold_idx] = best_lambda
        # Fit the model at the chosen lambda value
        fit = glmnet(X_train, Y_train,lambda = [best_lambda])
        # Make a prediction for the test data
        probs = GLMNet.predict(fit, X_test,outtype = :prob)
        pred_prob[findall(fold_indices .== fold_idx)] = probs
    end

    # Fit a final model for extrapolation
    fit_overall = glmnet(X,y, lambda = [mean(lambdas)])
    Dict("prediction_summary" => DataFrame(observed = y,
              prediction = 1 .* (pred_prob .> 0.5),
              fitted_value = pred_prob),
         "overall_model" => fit_overall)
```

XGBoost Classifier – Pipeline Approach

```
# Function to read in, process, and analyze data
bst = @load XGBoostClassifier
function run_xgboost_analysis(fname_train, fname_test; manual_variable_drops = [], K=10)
    # Read in training and test/extrapolation data sets.
    training_data = getAnalyticData_v2(fname_train,manual_variable_drops)
    test_data = getAnalyticData_v2(fname_test,manual_variable_drops)
    # In case the variables have changed, make sure to keep only the intersection
    names_train = training_data["coefnames"]
    names_test = test_data["coefnames"]

    train_keep_vars = findall([x in intersect(names_train, names_test) for x in names_train] .-> 0)
    test_keep_vars = findall([x in intersect(names_train, names_test) for x in names_test] .-> 0)

    MM_train = training_data["model_matrix"][:, train_keep_vars]
    MM_test = test_data["model_matrix"][:, test_keep_vars]

    # Update columnames
    names_train = names_train[train_keep_vars]
    names_test = names_test[test_keep_vars]
    # Check if any variables have zero variability and need to be excluded, skipping intercept
    zerovar_train = findall([var(MM_train[:,x + 1]) < 1e-12 for x in 1:(size(MM_train)[2] - 1)] .-> 0) .+ 1
    zerovar_test = findall([var(MM_test[:,x + 1]) < 1e-12 for x in 1:(size(MM_test)[2] - 1)] .-> 0) .+1
    zerovar_either = union(zerovar_train, zerovar_test)

    both_keep_vars = setdiff(1:(size(MM_train)[2]), zerovar_either)
    MM_train = MM_train[:,both_keep_vars]
    MM_test = MM_test[:,both_keep_vars]
     # Update columnames
    names_train = names_train[both_keep_vars]
    names_test = names_test[both_keep_vars]

    print(size(MM_train))
    print(size(MM_test))

    # Triple check that everything is lined up
    if (!all(names_train .== names_test))
        error("Columns don't match after initial variable exclusion: \n "*fname_test*"\n "*fname_train)
    end
```

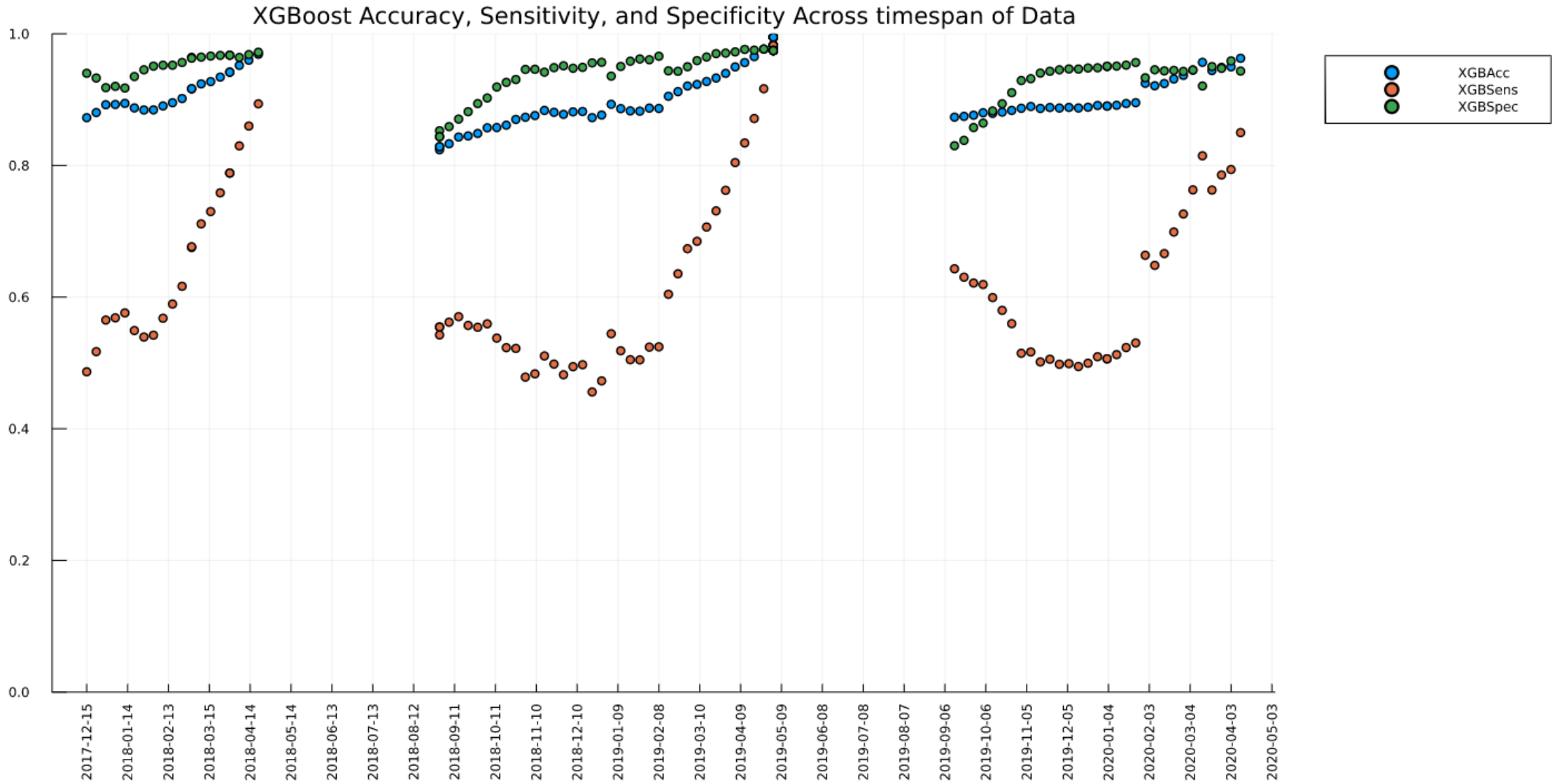
```
    # Set up pipeline. OneHot not needed, data is already processed
    xgbpipe = @pipeline(bst())
    # Hyperparameter tuning ranges
    ranges = [range(xgbpipe, :(xg_boost_classifier.max_depth), lower = 1, upper = 4),
              range(xgbpipe, :(xg_boost_classifier.eta), lower = 0.01, upper = 0.5, scale = :log)]
    xgbpipe.xg_boost_classifier.num_round = 500
    xgbpipe.xg_boost_classifier.subsample = 0.8
    xgbpipe.xg_boost_classifier.colsample_bytree = 0.25

    tuned_xgb_model = TunedModel(model = xgbpipe, resampling = CV(nfolds = K),
                                 tuning = Grid(), range = ranges,
                                 measure = [LogLoss()], n = 100)


    m = machine(tuned_xgb_model, MLJ.table(MM_train[:,2:(size(MM_train)[2])]),
                coerce(training_data["outcome"] .* 1, OrderedFactor))

    fit!(m, rows= 1:(size(MM_train)[1]))
    #fitted_params(m).best_model
    #evaluate!(m,resampling=CV(nfolds=5),measure=[LogLoss()], verbosity=1)
    fits_extrapolate = MLJ.predict(m, MLJ.table(MM_test[:,2:(size(MM_test)[2])]))
    preds_extrapolate = [1*(pdf(x, 1) >= 0.5) for x in fits_extrapolate]


    Dict("summary" => get_pred_perf(test_data["outcome"] .* 1, preds_extrapolate, "XGBoost Model Extrapolation"),
         "model" => m,
         "training_data" => training_data,
         "test_data" => test_data)
end
```


XGBoost Accuracy, Sensitivity, and Specificity Across timespan of Data
1.0
0.8
0.6
0.4
0.2
0.0
2017-12-15
2018-01-14
2018-02-13
2018-03-15
2018-04-14
2018-05-14
2018-06-13
2018-07-13
2018-08-12
2018-09-11
2018-10-11
2018-11-10
2018-12-10
2019-01-09
2019-02-08
2019-03-10
2019-04-09
2019-05-09
2019-06-08
2019-07-08
2019-08-07
2019-09-06
2019-10-06
2019-11-05
2019-12-05
2020-01-04
2020-02-03
2020-03-04
2020-04-03
2020-05-03
XGBAcc
XGBSens
XGBSpec

Extrapolation Comparison for one date

Model	Accuracy	Sensitivity	Specificity
GLMNet (LASSO)	0.9346	0.7107	0.9272
XGBoost	0.9245	0.6661	0.9437
RandomForest	0.9098	0.6011	0.9607
AdaBoostStumpClassifier	0.9051	0.5803	0.9612

Conclusion

- Predicting enrollment with dynamic data
- Using GLMNet and MLJ provided through Julia
- Comparison of models

Future Explorations

- Calculate random forests and Ada Boost Stump Classifier over all dates
  - Too computationally intensive with what we have

- Other Models

Acknowledgments

- ISIB Program sponsored by the National Heart Lung and Blood Institute (NHLBI), grant # HL-147231.

http://www.nhlbi.nih.gov/about/org/logos

Questions?