MS&E 226 Project Part 2

Zhaoyu (Joe) Lou and Carina Zhang

November 2018

1 Data Exploration

Upon closer inspection of our dataset, we found several issues with the dataset which we previously had not seen. We were planning on using the FIPS number for each county to join the two datasets; however, we noticed that around 100 counties had invalid or missing FIPS numbers. Further, we noticed that there were around 200 missing counties from the election dataset. For the invalid and missing FIPS numbers, we addressed the issue by downloading a dataset of county name to FIPS number mappings, and cross referencing both datasets with this mapping to fill in FIPS numbers. This left us with the 200 missing counties. Our original plan was to ignore these rows and simply use the remaining counties, however, t testing revealed statistically significant demographic differences between the missing counties and the ones in the election dataset (income per capita, for instance, was different with $p = 10^{-40}$). To prevent systematic errors due to this difference, we obtained new primary election data on missing counties. Combining the new election dataset with the improved demographic dataset gave us complete data on all 3142 counties and county equivalents in the US, except for Alaska, which did not have county level data. We dropped the 11 rows pertaining to Alaska and did our analysis on all other counties.

2 Classification

For classification model, we chose the best model in terms of validation AUC under 5-fold cross validation. I chose to optimize AUC instead of minimizing 0-1 loss because of the high prevalence of Republican counties (around 80 percent). A secondary goal is to maintain good sensitivity, since we want to do well on the lower prevelance democratic counties as well (democrat was labeled as 1). We report sensitivity and specificity for best model selection.

2.1 Covariate Selection

Since we have many variables, some of which are linear combinations of each other(such as population in different years and annual change), best subset and backward selection is not compuationally possible. Instead, within each training fold we computed pairwise correlations, and one out of each pair of highly correlated covariates. We end up with 45 covariates (the correlations were quite similar across folds).

2.2 Modeling and Evaluation

When modeling, we tried logistic regression, Lasso (with different λ), Naive Bayes, and KNN (with different number of neighbors) models. The justification of choices of parameters will be in the analysis part. We report the ROC, sensitivity, specificity, and accuracy averaged across the 5 folds.
Here is the table for the best models of each category:

Model	AUROC	Sens	Spec	Accuracy
Baseline (logistic)	0.9592544	0.7064273	0.9597508	0.9112093
Logistic, 20-predictors	0.9616424	0.708555	0.961231	0.9128164
Lasso, $\lambda = 0.001599175$	0.9618414	0.6960526	0.9681888	0.9647137
Naive Bayesian	0.9091026	0.5828947	0.9478106	
kNN, k=15	0.9533569	0.6310088	0.9751515	
kNN, k=3	0.8951814	0.6981798	0.9607360	

2.3 Analysis

Predicting general election result through demographic data is a fairly easy classifier. If we predict every county to be Republican, we will still have an accuracy of around 0.8. We chose AUC instead of accuracy as a test error estimator because it better captures the sensitivity rate, which translates into better predicting Democrat while Democrat occurence is rare.
The logistic function with all variables in performed fairly well. However, when we use many variables in our model, we notice a considerable increase in variance over the smaller set of covariates.
The logistic model with 20-predictors gives the best AUROC among all the models, while maintains a similar and slightly better sensitivity. Compared to the logistic model with all predictors, this model decreases the model complexity, addressing the variance problem which the original logistic model faced. However, we did not see a drastic change in ROC or Sens, probably because of multicollinearity.
The best Lasso model using CV selects $\lambda = 0.001599175$, which shrinks the number of predictors to 29. This model gives us a similar estimated test ROC, worse Sens, and better Spec than the logistic model. This is because Lasso shrinks the coefficients on unimportant covariates, leading the model towards the high bias side of often predicting republican (which is more common). This model is in the middle in terms of bias-variance trade-off the first two models.
The Naive Bayes model does not perform well. We see a lower AUROC and a huge drop in specificity. We decide to drop Naive Bayesian and not compute the accuracy rate.
We ran k = 1-15 for kNN and found the optimal number for kNN is k = 15 in terms of AUROC optimization. However, when k = 13, kNN has the best Sensitivity measure. Interestingly, we can see that with KNN, AUROC increases as the number of neighbor increases, suggesting that variance is the dominating form of error for this model. Sensitivity, on the other hand, increases at first, peaks at k = 3, and then decreases. Again, we see this through the lens of bias - as we increase k, bias increases, leading the model to predict republican more often. This is reflected in the low sensitivity measure in k = 15.
Since our primary goal is to optimize ROC and our secondary goal is to maintain a relatively good sensitivity rate, we choose Logistic function with 20-predictors as our prediction model. The coefficients and ROC Curve for the this model are in appendix B. The test error will be slightly higher than CV estimate of test error for this model. The code for the modeling is included in appendix A.

3 Regression

For the regression task, we decided that an interesting continuous outcome to predict would be voter turnout in each county, as a proportion of the total population. The metric that we tracked in this task was 100 times the RMSE, which gave the average deviation of the estimate in percentage points. Ideally, we'd also like to have an interpretable model, so we can understand how we might be able to improve voter participation in future elections.

3.1 Covariate Selection

We took four different approaches for selecting covariates. First, we plotted scatterplots of voter turnout against each variable. Based on these scatterplots, we proposed data transformations as well as whether or not to keep the covariate. This gave us a set of 13 covariates which we call the "handpicked" covariates.
Second, we applied lasso regression with a regularization parameter of 0.002. We kept all the covariates with nonzero coefficients in the resulting model, which left us with a set of 25 covariates which we will call the "lasso" covariates.
Third, we augmented the lasso covariates in two steps. First, any covariates which were counts, rather than proportions, were transformed with a logarithm. This was because we noticed these covariates tended to have a heavily right skewed distribution. Second, for any covariates that looked like they were related to each other we added interaction terms (for example, foreign born persons and foreign language speakers had an interaction term). This gave us a set of 33 covariates, which we will call the "augmented lasso" covariates.
Finally, we included all covariates, which gave us a set of 55 covariates which we will call the "all" covariates.
Each of these approaches was applied individually in 5 fold cross validation, but we found that in practice the covariate sets were the same each time, so we fixed the covariate sets to avoid having to recompute the same covariates 500 times over. This fixing may lead to a slight underestimate of test error, but we always saw the same covariates across folds so we think it should be a minor point (and it saves a lot of computational time).

3.2 Modeling

For each of these covariate sets, we fit 25 different models to the data using 5-fold cross validation. We tracked the average training and validation error of each model across the 5 folds. A selection of validation results are tabulated below; each cell is 100 times the cross validated RMSE on that particular model - covariate pair. When hyperparameters

are not specified, they are set to sklearn defaults. Code for the fitting procedure, including the best model, is included in Appendix C.

Model	Handpicked	Lasso	Aug. Lasso	All
Constant (Mean)	8.314	8.314	8.314	8.314
OLS	5.169	5.294	5.197	6.229
Ridge, $\lambda = 0.1$	5.168	5.282	5.152	6.171
Ridge, $\lambda = 1.0$	5.162	5.255	5.101	6.160
Ridge, $\lambda = 10.0$	5.160	5.182	5.145	6.117
Lasso, $\lambda = 0.001$	5.178	5.103	5.272	5.642
Lasso, $\lambda = 0.01$	5.626	5.190	5.342	5.686
Lasso, $\lambda = 0.1$	6.409	6.050	6.597	6.802
Bayesian Ridge	5.167	5.140	5.234	6.014
kNN, k=5	5.877	7.692	6.903	7.986
kNN, k=10	5.813	7.513	6.773	7.867
Support Vector Regress.	8.243	8.348	8.348	8.348
Random Forest, n=10	5.391	5.354	5.544	5.311
Random Forest, n=100	5.073	5.182	5.205	5.065
Ex. Rand Forest, n=10	5.436	5.518	5.277	5.434
Ex. Rand Forest, n=100	5.158	5.127	5.089	5.112
XGBoost, n=100	5.023	5.056	5.040	4.922
XGBoost, n=1000	5.168	5.289	5.301	5.076
Neural Network	396.9	1.5×10^7	4371	3.0×10^7

3.3 Analysis

We note that most of the models have similar performances. This suggests that either the models are approaching the Bayes error, or the covariates we are using may not be the optimal covariates for the task.

Let's first consider the linear models. For these models, we see that careful covariate selection is quite important for controlling variance. Indeed, OLS performs best on the smallest set of covariates (handpicked), and it's performance declines dramatically when we move to using all covariates. Even on the handpicked covariates we find that regularization improves our validation accuracy, suggesting that our model error is still dominated by variance rather than bias. As we increase the amount of covariates and especially when we use all covariates, strong regularization is required to get even close to the results of OLS on the handpicked covariates. Lasso is extremely sensitive; even with fairly small values of λ, lasso's validation error increases over that of OLS (and train error, not shown here, is similarly high), showing significant bias without a corresponding decrease in variance. The fact that even as we sweep λ from creating a model with high variance to a model with high bias without seeing the loss drop below 5.1 further suggests that either we are approaching the Bayes error for linear models, or the covariates are suboptimal (causing high bias regardless of regularization). Bayesian ridge regression outperforms both ridge and lasso, perhaps due to it's prior causing initially strong but adaptive regularization.

Moving on to the nonlinear models, we first note that SVM and neural network methods, which are generally quite data hungry, do extremely poorly on this small dataset. KNN performs passably well when using the small set of handpicked coordinates, but it's performance declines sharply on larger covariate sets. This is likely due to the model's inability to ignore insignificant features, introducing variance. For the random forest models (which perform the best), covariate selection seems to be less important, and using all covariates generally improves performance over the smaller covariate sets. This attests to the fact that these methods are able to learn more complex correlations between nonlinear functions of the features even in covariates which were not useful for the linear models, and thus are able to model the data with reduced bias compared to other approaches. Variance, however, remains a problem - as we increase the number of trees performance declines, and looking at the training error (not displayed) shows that these methods are still overfitting the data.

The best model is a forest of 100 gradient boosted decision trees (XGBoost) which uses all covariates. This model is advantageous in both of the aspects with which we evaluated models. First, its validation RMSE was the lowest among all the models, outperforming the error of the constant prediction model by 41% and the error of the baseline OLS model trained on all covariates by 21%. Second, it is highly interpretable. Decision trees give a clear hierarchy of decisions which were used to make predictions, so it is easy to see exactly how the model is making it's predictions. We expect that our validation error will be an underestimate of the test error, since we picked the model which performed best on the validation. The test error will probably be around 5.

4 Appendices

A Classification Code

The following R code was used for the classification portion of the project.

```
library(leaps)
train <- read.csv("/Users/chujunzhang/Documents/Stanford Classes/2018 Fall/MS&E 226/project/train.csv")

# clean up the train data before standardization
train <- read.csv("/Users/chujunzhang/Documents/Stanford Classes/2018 Fall/MS&E 226/project/train.csv")
drop1 =c("fips")
datamatrix= train[ ,!(names(train)%in%drop1)]
drop2 =c("area_name")
datamatrix = datamatrix[ ,!(names(datamatrix)%in%drop2)]
drop3 =c("state_abbreviation")
datamatrix= datamatrix[ ,!(names(datamatrix)%in%drop3)]
drop4 =c("state_abbreviation", "votes_dem", "votes_gop",
"per_dem", "per_gop", "diff", "per_point_diff", "total_votes")
datamatrix= datamatrix[ ,!(names(datamatrix)%in%drop4)]

library(mlbench)
library(caret)

k = 5
set.seed (123)
folds=sample (1:k,nrow(datamatrix),replace =TRUE)

#create dummy variable
suppressWarnings(datamatrix[,ncol(datamatrix)] <- sapply(datamatrix[,ncol(datamatrix)], switch,
"Democrat"= 0,"Republican" = 1))

for (i in 1:k) {
  # calculate correlation matrix
  correlationMatrix <- cor(datamatrix[folds!=i,])
  # find attributes that are highly corrected (ideally >0.75)
  highlyCorrelated <- findCorrelation(correlationMatrix, cutoff=0.8)
  # print indexes of highly correlated attributes
  print(highlyCorrelated)
}

drop4 =c("population_2010_base", "population_2010","foreign_born_perc","housing_units",
"nonfarm_establishments", "nonemployer_establish")
datamatrix= datamatrix[ ,!(names(datamatrix)%in%drop4)]

#standardize all the numerical data
datamatrix[] <- lapply(datamatrix, function(x) if(is.numeric(x)){
                     scale(x, center=TRUE, scale=TRUE)
                      } else x)

library(caret)
set.seed(123)

ctrl <- trainControl(method = "cv",
                     number = 10,
                     summaryFunction=twoClassSummary,
                     classProbs= TRUE,
                     savePredictions = TRUE)
```

```
datamatrix$party <- as.factor(datamatrix$party) # transform the outcome variable as factors

levels(datamatrix$party) <- c("Democrat", "Republican")

mod_fit_base <- train(party ~.,
                  trControl=ctrl,
                  data = datamatrix,
                  method="glm",
                  family="binomial",
                  metric = "ROC",
                  preProcess = c("center","scale"))

print(mod_fit_base)

obs = mod_fit_base$pred$obs
confusionmatrix = table(mod_fit_base$pred$pred, mod_fit_base$pred$obs)
confusionmatrix
accuracy = (confusionmatrix[1,1] + confusionmatrix[2,2])/nrow(mod_fit_base$pred)
accuracy

#Plot the ROC curves for base
library(ROCR)
# install.packages("pROC")
library(pROC)
plot(roc(predictor = mod_fit_base$pred$Republican, response = mod_fit_base$pred$obs))

# estimate variable importance
importance <- varImp(mod_fit_base, scale=FALSE)
# summarize importance
print(importance)
# plot importance
plot(importance)

ctrl <- trainControl(method = "cv",
                     number = 10,
                     summaryFunction=twoClassSummary,
                     classProbs= TRUE,
                     savePredictions = TRUE)

datamatrix$party <- as.factor(datamatrix$party) # transform the outcome variable as factors
levels(datamatrix$party) <- c("Democrat", "Republican")

mod_fit_red <- train(party ~ occupied_housing_value + pop_over_65  + high_school_perc + bachelors_perc
+ same_house_1_year_perc + people_per_household + housing_multi_unit_perc + income_per_capita +
pop_under_18 + pop_percent_change + white_not_hispanic_perc + homeownership + households + land_area
+  women_firms  + hispanic_perc + pacific_island_firms +  nonfarm_employ_change_perc + female_perc + firms,
                  trControl=ctrl,
                  data = datamatrix,
                  method="glm",
                  family="binomial",
                  metric = "ROC")

print(mod_fit_red)
exp(coef(mod_fit_red$finalModel))

# Print out confusion matrix and ROC curve for the model results
obs = mod_fit_red$pred$obs
confusionmatrix = table(mod_fit_red$pred$pred, mod_fit_red$pred$obs)
confusionmatrix
```

```
accuracy = (confusionmatrix[1,1] + confusionmatrix[2,2])/nrow(mod_fit_red$pred)
accuracy

#Plot the ROC curves for base
library(ROCR)
# install.packages("pROC")
library(pROC)
plot(roc(predictor = mod_fit_red$pred$Republican, response = mod_fit_red$pred$obs))
coef(mod_fit_red$finalModel)

fit_KNN <- train(party ~ .,
             method     = "knn",
             tuneGrid   =  expand.grid(k = 1:15),
             trControl  =  ctrl,
             metric     =  "ROC",
             data       =  datamatrix)
print(fit_KNN)

## Print out confusion matrix and ROC curve for the model results
obs = fit_KNN$pred$obs
confusionmatrix = table(fit_KNN$pred$pred, fit_KNN$pred$obs)
confusionmatrix
accuracy = (confusionmatrix[1,1] + confusionmatrix[2,2])/nrow(fit_KNN$pred)
accuracy

#Plot the ROC curves for kNN
library(ROCR)
# install.packages("pROC")
library(pROC)
plot(roc(predictor = fit_KNN$pred$Republican, response = fit_KNN$pred$obs))

library("glmnet")
x = model.matrix(party ~ . - 1, data = datamatrix)
y = datamatrix$party

## cv for lasso
set.seed (123)
cv.out = cv.glmnet(x, y,
                   family = "binomial",
                   alpha = 1,
                   type.measure="auc",
                   nfolds = 5) #k = 10 default
plot(cv.out)
bestlam =cv.out$lambda.min
bestlam
summary(cv.out)

#Create a LASSO grid with cross validated lambda choice
set.seed(123)
glmgrid <- expand.grid(.alpha=1, .lambda = bestlam)

## Now run CV to estimate the test error for Lasso model
set.seed(123)

datamatrix$party <- as.factor(datamatrix$party)

ctrl <- trainControl(method = "cv",
                     number = 5,
                     summaryFunction=twoClassSummary,
```

```
                    classProbs= TRUE,
                    savePredictions = TRUE)

levels(datamatrix$party) <- c("Democrat", "Republican")

modelfit<- train(party ~ ., data = datamatrix,
             method="glmnet",
             tuneGrid = glmgrid,
             trControl=ctrl,
             metric="ROC")

print(modelfit)
coef(modelfit$finalModel)
obs = modelfit$pred$obs
confusionmatrix = table(modelfit$pred$pred, modelfit$pred$obs)
confusionmatrix
accuracy = (confusionmatrix[1,1] + confusionmatrix[2,2])/nrow(modelfit$pred)
accuracy

#Plot the ROC curves for lasso
library(ROCR)
# install.packages("pROC")
library(pROC)
plot(roc(predictor = modelfit$pred$Republican, response = modelfit$pred$obs))
predictors(modelfit)

# Naive Bayes
set.seed(123)
fit_Bayes <- train(party ~.,
                   data = datamatrix,
                   method = "nb",
                   trControl = ctrl,
                   metric = "ROC")
print(fit_Bayes)
```

B Logistic Regression Model

```
exp(coef(mod_fit_red$finalModel))
```

```
##               (Intercept)     occupied_housing_value
##                23.5763149                  0.2656482
##               pop_over_65          high_school_perc
##                 2.1976562                  0.2790329
##            bachelors_perc     same_house_1_year_perc
##                 0.2427508                  0.4309910
##       people_per_household    housing_multi_unit_perc
##                 2.0986897                  0.3178524
##         income_per_capita              pop_under_18
##                 6.9512228                  1.7627967
##        pop_percent_change    white_not_hispanic_perc
##                 1.7684979                 14.8586987
##             homeownership                households
##                 0.7183003                  0.2407705
##                land_area               women_firms
##                 1.0930745                  0.7929933
##             hispanic_perc      pacific_island_firms
##                 0.8170172                  0.2513536
## nonfarm_employ_change_perc               female_perc
##                 0.9166073                  0.8694145
##                    firms
##                 4.3137289
```

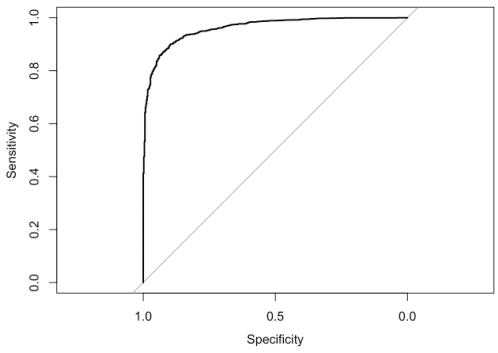

Figure 1: Left: Coefficients for the best logistic regression model. Right: ROC for this model.

C Regression Code

The following Python code was used to fit regression models and report training/validation errors.

```
from sklearn.dummy import DummyRegressor
from sklearn import linear_model, svm
from sklearn.neighbors import KNeighborsRegressor
from sklearn.ensemble import RandomForestRegressor, ExtraTreesRegressor
from sklearn.neural_network import MLPRegressor
from sklearn.metrics import mean_squared_error
from sklearn.model_selection import cross_val_score
import numpy as np
import xgboost as xgb
import warnings

def get_model_zoo():
    # Defines all models and names them.
    mean = DummyRegressor()
    ols = linear_model.LinearRegression()
    ridge1 = linear_model.Ridge(alpha=100)
    ridge2 = linear_model.Ridge(alpha=1.0)
    ridge3 = linear_model.Ridge(alpha=10.0)
    lasso1 = linear_model.Lasso(alpha=0.001)
    lasso2 = linear_model.Lasso(alpha=0.01)
    lasso3 = linear_model.Lasso(alpha=0.1)
    bayes = linear_model.BayesianRidge()
    k5 = KNeighborsRegressor(n_neighbors=5)
    k10 = KNeighborsRegressor(n_neighbors=10)
    k20 = KNeighborsRegressor(n_neighbors=20)
    svr = svm.SVR()
    forest_10 = RandomForestRegressor(n_estimators=10)
    forest_50 = RandomForestRegressor(n_estimators=50)
    forest_100 = RandomForestRegressor(n_estimators=100)
    eforest_10 = ExtraTreesRegressor(n_estimators=10)
    eforest_50 = ExtraTreesRegressor(n_estimators=50)
    eforest_100 = ExtraTreesRegressor(n_estimators=100)
    boost_trees_100 = xgb.XGBRegressor(n_estimators=100)
    boost_trees_500 = xgb.XGBRegressor(n_estimators=500)
    boost_trees_1000 = xgb.XGBRegressor(n_estimators=1000)
    boost_trees_50_deep = xgb.XGBRegressor(n_estimators=50, max_depth=5)
    boost_trees_500_deep = xgb.XGBRegressor(n_estimators=500, max_depth=5)
    nn = MLPRegressor()

    models = {"mean": mean, "ols": ols, "ridge_100": ridge1, "ridge_1.0": ridge2, "ridge_10.0": ridge3,
    "lasso_0.001": lasso1, "lasso_0.01": lasso2, "lasso_0.1": lasso3, "bayesianRidge": bayes, "5NN": k5,
    "10NN": k10, "20NN": k20, "SVM": svr, "randForest_10": forest_10, "randForest_50": forest_50,
    "randForest_100": forest_100, "extraRandForest_10": eforest_10, "extraRandForest_50": eforest_50,
    "extraRandForest_100": eforest_100, "XGBoost_100": boost_trees_100, "XGBoost_500": boost_trees_500,
    "XGBoost_1000": boost_trees_1000, "XBBoost_50_deep": boost_trees_50_deep,
    "XGBoost_500_deep": boost_trees_500_deep, "neuralNet": nn}
    return models

warnings.catch_warnings()
warnings.simplefilter("ignore")
# Suffixes indicating dataset being used
datasets = ["handpicked", "lasso", "augmentedlasso", "all"]

best_overall = ["None", "None", 1000]
for suffix in datasets:
    # Keep track of best model
```

```
    best = ["None", 1000]

    # Load training and validation datasets. Split into covariates and labels.
    train_name = "train_" + suffix + ".csv"
    val_name = "test_" + suffix + ".csv"
    train_data = np.loadtxt(train_name, delimiter=',', skiprows=1)
    val_data = np.loadtxt(val_name, delimiter=',', skiprows=1)
    train_covariates = train_data[:,:-1]
    train_labels = train_data[:,-1]
    val_covariates = val_data[:,:-1]
    val_labels = val_data[:,-1]

    models = get_model_zoo()

    for name, model in models.items():
        train_errs = cross_val_score(model, train_covariates,
        train_labels, scoring="neg_mean_squared_error", cv=5)
        val_errs = cross_val_score(model, val_covariates,
        val_labels, scoring="neg_mean_squared_error", cv=5)

        val_rmse = 100 * np.sqrt(-np.mean(val_errs))
        print("Training average error for %s, %s covariates: %f%%" % (name, suffix, 100 *
        np.sqrt(-np.mean(train_errs))))
        print("Validation average error for %s, %s covariates: %f%%" % (name, suffix, val_rmse))

        if val_rmse < best[1]:
        best = [name, val_rmse]
        if val_rmse < best_overall[2]:
        best_overall = [suffix, name, val_rmse]
    print("Best model for %s covariates: %s, with validation average error of %f%%" %
    (suffix, best[0], best[1]))

print("Best overall model: %s on %s covariates, with validation average error of %f%%" %
(best_overall[1], best_overall[0],
best_overall[2]))
```