Assignment 2 – Solutions

STAT30270 – Statistical Machine Learning

Exercise 1

Support vectors are those data points for which the equation of the plane/line is equal to -1 or +1 and are on the correct side of the hyperplane, i.e. those data points for which $y_i(w_0 + \mathbf{x}_i^\top \mathbf{w}) = 1$.

$$\mathbf{x}_1 : \ +1 \times (0.5 - 3\frac{1}{6} + \frac{3}{2}) = 1.5$$
$$\mathbf{x}_2 : \ -1 \times (0.5 - 3\frac{1}{3} - \frac{1}{2}) = 1$$
$$\mathbf{x}_3 : \ +1 \times (0.5 - 3\frac{1}{2} + 2) = 1$$
$$\mathbf{x}_4 : \ +1 \times (0.5 - 3\frac{1}{3} + 1) = 0.5$$
$$\mathbf{x}_5 : \ -1 \times (0.5 - 3\frac{1}{4} + \frac{3}{4}) = 0.5$$

Points $\mathbf{x}_2$ and $\mathbf{x}_3$ are the support vectors, in particular $\mathbf{x}_2$ is the support vector on the -1 class side, while $\mathbf{x}_3$ is the support vector on the +1 class side.

Exercise 2

(1)

A full iteration of k-means consists in an allocation and update step. Hence, we first compute the allocation of the observations for the given centroids. We compute all squared Euclidean distances from each data point from the centroid and allocate to cluster with minimum squared distance:

$\|\|\mathbf{x}_i - \hat{\mu}_1^{(t)}\|\|^2$	$\|\|\mathbf{x}_i - \hat{\mu}_2^{(t)}\|\|^2$	Allocation$^{(t+1)}$	$\mathbf{Z}^{(t+1)}$	
3.7268	14.0422	1	1	0
5.0525	6.5701	1	1	0
15.7770	6.5624	2	0	1
0.3773	3.6329	1	1	0
3.0238	1.0814	2	0	1

In the update step, we compute the mean of each cluster given the current allocation:

$$\mu_{11} = \frac{1.76 + 2.41 + 0.82}{3} = 1.6633 \qquad \mu_{12} = \frac{-1.30 + 0.66 + 0.64}{3} = 0 \qquad \mu_{13} = \frac{2.39 + 0.41 + 1.66}{3} = 1.4867$$
$$\mu_{21} = \frac{-2.63 - 0.73}{2} = -1.68 \qquad \mu_{22} = \frac{1.18 + 0.83}{2} = 1.005 \qquad \mu_{23} = \frac{-0.36 + 0.85}{3} = 0.245$$

Therefore we obtain the cluster centroids:

$$\hat{\mu}_1^{(t+1)} = (1.6633, 0.0000, 1.4867) \qquad \hat{\mu}_2^{(t+1)} = (-1.680, 1.005, 0.245)$$

(2)

Compute the k-means objective function at iteration (t) and iteration $(t+1)$.

	$\|\|\mathbf{x}_i - \hat{\mu}_1^{(t)}\|\|^2 * \hat{\mathbf{z}}_1^{(t)}$	$\|\|\mathbf{x}_i - \hat{\mu}_2^{(t)}\|\|^2 * \hat{\mathbf{z}}_2^{(t)}$	$\|\|\mathbf{x}_i - \hat{\mu}_1^{(t+1)}\|\|^2 * \hat{\mathbf{z}}_1^{(t+1)}$	$\|\|\mathbf{x}_i - \hat{\mu}_2^{(t+1)}\|\|^2 * \hat{\mathbf{z}}_2^{(t+1)}$
	3.7268	0	2.5154	0
	0	6.5701	2.152322	0
	0	6.5624	0	1.2992
	0.3773	0	1.1509	0
	3.0238	0	0	1.2992
Sum	7.1279	13.1325	5.8185	2.5983

Therefore, the objective function is given by the sum of the within cluster sum of squares at each iteration:

$$W^{(t)} = 7.1279 + 13.1325 = 20.2604$$

$$W^{(t+1)} = 5.8185 + 2.5983 = 8.4168$$

Clearly, $W^{(t+1)} < W^{(t)}$, since $8.4168 < 20.2604$.

Note:

- Students may have used R code to perform calculations. This is fine and it gives full marks as long as R is used as a calculator and all the steps are shown as per instructions.

Exercise 3 - Data analysis

Some notes, main aspects that the report should contain, and key points for grading:

- The assignment concerns a real world problem regarding prediction of the outcome of the county level winning candidate for the 2020 US presidential elections, using input features associated to voting patterns, economic, social, and demographic factors. This particular data application has been given to stimulate reasoning and critical thinking in relation to a real world data analysis problem.
- The main methodological task concerns the **appropriate implementation of a cross-validation procedure for comparing and tuning three (or more) classifiers**, and evaluating the generalization performance of the best one. In this solution document, the package `caret` is employed to implement the procedure, although it is not required to use this package for full marks.
- **At least 3 different supervised learning methods must be implemented for full marks**. Implementing only a single classifier and comparing 3 or more instances/hyperparameter configurations of the same will not give full marks. For example, considering only a SVM with GRBF kernel and C = (1, 10, 20) and `sigma` = (0.01, 0.05) does not correspond to implementing 6 models, but it corresponds to implementing only a single supervised learning method and it does not give full marks.
- In this solution we compare the classifiers: logistic regression with automated variable selection, logistic regression on different subset of input features, classification tree, random forests and bagging, support vector machine with polynomial kernel (including linear kernel), support vector machine with Gaussian radial basis function kernel. Three classifiers are sufficient for full marks, however here we consider more for completeness. As pointed out in class, random forest is based on ensemble of classification trees, so it will always outperform a single classification tree. **Implementing a set of three classifiers where two of them are random forest and classification trees will not give full marks**.
- **Some tuning must implemented for all classifiers considered for full marks**, and a brief discussion about how the grids of hyperparametes were set **must be provided for full marks**, i.e. running the tuning over a grid without justification does not give full marks. A trial and error approach and a few notes are sufficient to justify the selection of the hyperparameter grid. Students can restrict the number of classifiers trained in total if computing time is too long or because of limited resources, but **this needs to be pointed out in the text of the solutions for full marks**. In these solutions, a very large number of classifiers are trained for completeness (more than 25000).
- **For full marks, a discussion and the reasoning behind the various decisions taken in all stages of the analysis must be provided**. Students can use only a subset of the features of the data in the models, however, the choice and why some features are discarded must be clearly motivated for full marks.
- The classes are imbalanced to some extent, with a proportion of `biden` vs `trump` of about 16% to 84%. However, this is not an imbalanced classification problem, as the goal is to deploy a model capable of predicting the winning candidate, without any particular prior preference for one of the two (there is no real "positive" vs "negative" class dichotomy in this context). In this regard, **using precision/recall, AU-PR, and F_1 score for model evaluation is incorrect and does not give full marks**, as with these metrics focus will be placed on the detection of the arbitrary positive class. The appropriate metrics for model evaluation are those that account both classes, i.e. accuracy, sensitivity, specificity, and AU-ROC. In these solutions we employ AU-ROC, which would assess detection of either Biden or Trump as the winning candidate, in conjunction with sensitivity and specificity. Using only accuracy is okay to some extent, although not ideal, and still provide full marks (note that the selected classifier may change; results are not reported). **Using all the metrics introduced in the module** to assess the predictive performance/compare/tune the models **does not give full marks**, since it shows lack of understanding of which metrics are more appropriate for this type of problem.
- **The dataset must be split into a portion set aside for testing and a portion employed for training, tuning, and model comparison for full marks**. The split into test set and training/validation set can be performed within a re-sampling process. Any splitting procedure if implemented correctly provides full marks. In this solution, the test data is set aside at the start.
- **The classifiers must be compared within the same cross-validation procedure for full marks**, i.e. trained and validated on the same sets/partitions of the data. Running cross-validation on different splits does not provide full marks. If using `caret` the same random seed must be employed before training the models with `train`, to make sure that the classifiers are trained and validated on the same splits. If using standard for-loop

implementation, the splits must be defined appropriately and the classifiers trained and validated accordingly (compare for example previous lab code examples).

- Students can use different ways of implementing cross-validation, i.e. splitting in train and validation, or using k-fold cross-validation, or leave-one-out. In the code solutions below, k-fold cross-validation is employed using `caret`. In all cases, **for full marks the classifiers need to be appropriately compared computing the average validation predictive performance**.
- **The students must produce and comment a summary of the validation predictive performance for full marks**, and select the best classifier according to the average validation predictive performance. In this solutions outline here, we use sensitivity, specificity, and AU-ROC to compare the classifiers and boxplots to summarize visually the performance of the different classifiers.
- The classifiers must be compared appropriately, i.e. by comparing the average predictive validation performance. **Comparing the classifiers and select the best one according to training predictive performance or test predictive performance a is incorrect and will not provide full marks.**
- Multiple replications must be employed to account for sampling variability, i.e. **not running the procedure for multiple replications will not provide full marks**. It is not required that students run the procedure for a large number of replications (like 100), even a small number is ok (like 10, or even less) if this is clearly justified by limited computing and time resources.
- Input variables have very different scales and ranges of variation, **a transformation of the data must be considered and appropriately implemented for full marks**. The same transformation must be applied to training, validation, and test data, otherwise models will be incorrectly trained and evaluated on data on different scales/ranges. The transformation must be applied to minimize information leakage in the training/validation sets (see labs 3 and 5 for example). In this example solutions, standardization is applied.
- The application of logistic regression might return the warning `glm.fit: fitted probabilities numerically 0 or 1 occurred`. This is not a problem, since the main purpose is classification.
- In using logistic regression classifiers (or other probabilistic classifiers, like trees and random forest), students may tune/select the classification threshold to estimate the class labels. Multiple approaches could be taken: (1) Tune the threshold as an actual hyperparameter within a cross validation procedure; (2) Pre-select the threshold using the full training/validation data, then run the cross-validation with a fixed pre-selected threshold; (3) Set the threshold to a pre-specified value without selection. All these approaches, if implemented correctly and clearly motivated, will give full marks. In this solutions example, tuning of the threshold is not implemented, since the use of AU-ROC allows to evaluate the models regardless of the threshold.
- The definition of the "best" classifier can be given not only in terms of overall predictive performance according to a single metric, but also regarding certain aspects of the classifier and in relation to multiple metrics. For example, given that all classifiers provide similar performance, one could select the one that is simpler to interpret; another example, one could select the classifier that provides ideal performance on certain metrics. As long as the reasoning is provided and appropriately motivated, full marks are given.
- The selected classifier **must be re-trained on the full data before testing for full marks** (this is automatically done in `caret`, see also Lab 5), otherwise it would be trained not using all the available data for training.
- The definition of the "best" classifier can be given not only in terms of overall predictive performance according to a single metric, but also regarding certain aspects of the classifier and in relation to multiple metrics. For example, given that all classifiers provide similar performance, one could select the one that is simpler to interpret; another example, one could select the classifier that provides ideal performance on certain metrics. As long as the reasoning is provided and appropriately motivated, full marks are given.
- **The selected classifier must be tested on the test data with a brief discussion about the predictive performance**.
- As by instructions of the assignment, which requires using only numerical features for prediction, **the variables `state` and `county` should not be used as input variables to the classifiers for full marks**.
- **Submission of a report with unnecessary/verbose output** or **unnecessary large number of pages** will incur in penalization.

- **Submission of only code will not give any marks**.

Note: Not all possible scenarios have been tested in preparing these solutions, and it may happen that for certain random splits of the data, hyperparameter settings and grids, cross-validation process, the classifier chosen in this solution document may not be the best classifier. In the case where everything has been implemented appropriately as outlined above, this would provide full marks.

(1) We load the data and the relevant packages. We inspect certain characteristics of the data.

```
library(caret)
library(doParallel)

load("data_assignment_2.RData")
```

We look at the class distribution of `biden` and `trump`. The class distribution is unbalanced to some extent, with a proportion of `biden` vs `trump` of about 14% to 86%. However, we do not consider this an imbalanced classification problem, as the goal is to deploy a model capable of predicting the winning candidate, without any particular prior preference for one of the two and equal importance is given to the two classes `biden`/`trump`. For this reason, we use AU-ROC in conjunction with sensitivity and specificity to evaluate the predictive performance. These metrics will be computed specifying function `twoClassSummary` in the argument `summaryFunction` of `trainControl`. Additionally, the use of AU-ROC for assessment allows to evaluate the predictive performance of models regardless of a specified classification threshold, which makes things easier. The associated sensitivity and precision metric will be computed for a MAP classification (as it is default of `caret`).

```
# inspect class distribution
table(elections$winner)/nrow(elections)
```

```
##
##     biden     trump
## 0.1646707 0.8353293
```

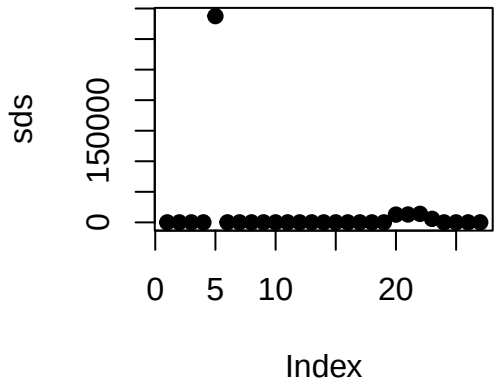
The data are on different scales and have different range of variation. Therefore standardization is applied, also because is more appropriate standardizing the data when using SVMs. The standardization is applied to minimize information leakage in the subsequent chunk of code, where data splitting is also implemented.

```
# inspect range and standard deviations
apply(elections[,-c(1:2,30)], 2, range)
```

```
##      romney_2012 obama_2012 trump_2016 clinton_2016 pop_2019 white_2019
## [1,]   0.0719343 0.03448276 0.04087474   0.03144654       98        8.3
## [2,]   0.9586207 0.90913114 0.94584838   0.90863822 10081570      100.0
##      black_2019 asian_2019 hispanic_2019 median_age_2019 age_over_18_2019
## [1,]        0.0        0.0           0.0            23.5             58.2
## [2,]       87.2       42.7          99.2            67.4             92.7
##      age_over_65_2019 women_16_to_50_birth_rate_2017 veterans_2019
## [1,]              3.2                             0.0           0.0
## [2,]             56.7                            28.1          26.2
##      bachelors_2019 uninsured_2019 unemployment_rate_2016
## [1,]            0.0            0.7                   1.67
## [2,]           75.3           40.9                  23.64
##      unemployment_rate_2017 unemployment_rate_2019 median_household_income_2016
## [1,]                   1.62                    0.0                         22045
## [2,]                  19.07                   25.3                        134609
##      median_household_income_2017 median_household_income_2019
## [1,]                        19264                        21504
## [2,]                       129588                       142299
##      median_individual_income_2019 poverty_2019 avg_family_size_2019
## [1,]                         12911          3.1                 2.23
## [2,]                         70148         42.4                 5.85
```

```
##      household_has_smartphone_2019 household_has_computer_2019
## [1,]                          25.0                         31.5
## [2,]                          92.4                         95.0
```

```r
sds <- apply(elections[,-c(1:2,30)], 2, sd)
plot(sds, pch = 19)
```



A 5-fold cross-validation procedure is implemented for model comparison and tuning, repeated 20 times for each classifier to account for sampling variability. A portion of 80% of the data are used for training + validation, while the remaining 20% is employed for testing. The test data are set aside at the start and only used at the end for testing. The test data is standardize separately from the training/validation data, using the summary statistics computed on the training/validation set. This minimizes any "leakage" of information from test to training/validation sets.

Variables `county` and `state` are removed from the data, as they are not employed for implementing the classifiers.

```r
# split into training + validation and test data
# and remove 'state' and 'county' columns
set.seed(445577)
set_train_val <- createDataPartition(elections$winner, p = 0.80, list = FALSE)
data_train_val <- elections[set_train_val,-c(1:2)]
data_test <- elections[-set_train_val,-c(1:2)]

# standardize test and training data separately to reduce information leakage
pre_scale <- preProcess(data_train_val, method = c("center", "scale"))
data_train_val <- predict(pre_scale, data_train_val)
data_test <- predict(pre_scale, data_test)
```

We compare the following classifiers:

- Logistic regression with automatic variable selection (via backward stepwise search and AIC). The reason is to check if through a formal model selection procedure certain features can be discarded and lead to a more parsimonious model with good predictive performance.
- Logistic regression models implemented considering different subset of the variables, based on the type of features: all features, voting features, population-related and demographic features, socio-economic features. The idea in this case is to identify if subset of variables defined by their context provide adequate predictive performance.
- Classification tree classifier. This classifier is only introduced for comparison with others. It is already expected that classification tree classifier will not outperform a random forest classifier. However, a classification tree classifier may have better performance compared to logistic regression and SVM, and also provide good interpretability.
- Random forests and bagging. These classifiers are used as black-box ensemble methods that are known to have high predictive power. As the main task is classification, in selecting these classifiers we favor predictive power over interpretability. A random forest with a number of variables considered in a split equal to 27 (number of input predictors) corresponds to a bagging classifier, so this value is included in the set of hyperparameters. In implementing random forest, we tune both the size of the ensemble and the number of variables considered at each split.
- Support vector machine with polynomial and linear kernel. A SVM with polynomial kernel can account for

nonlinear decision boundaries between the classes, which may prove useful. We include also a SVM with linear kernel, i.e. simple linear decision boundaries, by considering a degree value of 1. We select the hyperparameter grid to consider overlapping, shifted and/or nonlinear boundaries. Also in using this classifier we opt for predictive power over interpretability.

- Support vector machine with Gaussian radial basis function kernel. A SVM with GRBF kernel can account for nonlinear decision boundaries between the classes, which may prove useful for the type of features in the data. We select the hyperparameter grid to consider overlapping and nonlinear boundaries. Also in using this classifier we opt for predictive power over interpretability.

For all hyperparameter-based classifiers, tuning is performed in conjunction with the cross-validation procedure. In this solutions, the hyperparameter grids of the different classifiers have been set using some trial runs. The lower bound of the grids has generally been defined in correspondence of minimum (or almost minimum) complexity of a classifier, while the upper bound has been set via trial and error. The approach used is that if the best model corresponded to an hyperparameter equal to the upper bound of a specified interval, then the interval considered is enlarged, since it could be the case that the "true" optimal hyperparameter value is actually larger, but the optimal value has been selected because is the largest in the interval considered.

During the trial runs, we found that for the SVMs classifiers, the selected cost hyperparameter kept increasing with negligible improvement in the validation predictive performance. For this reason, in order to limit computing time and model space, we settled with a maximum value of this hyperparameter equal to 300.

We note that the definition of the grids can also been implemented balancing exploration of the model space and computing time. With the grids and the models considered here, around 30000 classifiers have been trained. This large number has been considered for completeness of these solutions.

The classifiers have been trained and validated/tuned using the same splits by setting the same random seed before each call to the function `train`. This ensures that the classifiers are compared appropriately in the cross-validation procedure.

```
# set parallel computations
cl <- makeCluster(8)       # use 8 cores
registerDoParallel(cl)     # start parallel resuting backend

# implement 5-fold cross-validation, 20 replications
train_ctrl <- trainControl(method = "repeatedcv",
                           number = 5, repeats = 20,
                           classProbs = TRUE,
                           summaryFunction = twoClassSummary)

# logistic regression with stepwise variable selection -----------------------
set.seed(112233)
fit_log_step <- train(winner ~ ., data = data_train_val,
                      method = "glmStepAIC",
                      family = "binomial",
                      direction = "backward",
                      trControl = train_ctrl,
                      metric = "ROC",
                      trace = FALSE)   # turn off verbose output

# logistic regression on subsets of features ---------------------------------
# all features
set.seed(112233)
fit_log_all <- train(winner ~ .,
                      data = data_train_val,
                      method = "glm",
                      trControl = train_ctrl,
                      metric = "ROC")

# voting features
```

```r
set.seed(112233)
fit_log_sub1 <- train(winner ~ romney_2012 + obama_2012 + trump_2016 + clinton_2016,
                      data = data_train_val,
                      method = "glm",
                      trControl = train_ctrl,
                      metric = "ROC")

# population and demographic features
set.seed(112233)
fit_log_sub2 <- train(winner ~ white_2019 + black_2019 + asian_2019 + hispanic_2019 +
                        median_age_2019 + avg_family_size_2019,
                      data = data_train_val,
                      method = "glm",
                      trControl = train_ctrl,
                      metric = "ROC")

# economic and social features
set.seed(112233)
fit_log_sub3 <- train(winner ~ bachelors_2019 + uninsured_2019 +
                        unemployment_rate_2019 + median_household_income_2019 +
                        median_individual_income_2019 + poverty_2019 +
                        household_has_smartphone_2019 + household_has_resuter_2019,
                      data = data_train_val,
                      method = "glm",
                      trControl = train_ctrl,
                      metric = "ROC")

# classification tree ------------------------------------------------------
tune_grid <- expand.grid( cp = c(0.00001, 0.00005,
                                 0.0001, 0.0002, 0.0003, 0.0004, 0.0005,
                                 0.001, 0.002, 0.003, 0.005) )
#
set.seed(112233)
fit_ct <- train(winner ~ ., data = data_train_val,
                method = "rpart",
                trControl = train_ctrl,
                tuneGrid = tune_grid,
                metric = "ROC")

# random forest + bagging --------------------------------------------------
# sqrt(27) = 5.196152
# bagging is equivalent to mtry = 27
ntree_set <- c(100, 200, 300, 500, 1000, 2000, 3000)
tune_grid <- expand.grid( mtry = c(2, 3, 4, 5, 8, 10, 15, 20, 25, 27) )
#
out <- vector("list", length(ntree_set))
for ( j in 1:length(ntree_set) ) {
  set.seed(112233)
  out[[j]] <- train(winner ~ ., data = data_train_val,
                    method = "rf",
                    trControl = train_ctrl,
                    tuneGrid = tune_grid,
                    ntree = ntree_set[j],
                    metric = "ROC")
  # print(j)
}
```

```
# svm ---------------------------------------------------------------------
# svm with polynomial and linear kernel
tune_grid <- expand.grid(C = c(5, 10, 30, 50, 70, 100, 200, 300),
                         scale = c(0.1, 0.5, 1, 1.5, 2),
                         degree = c(1, 2, 3, 4))
#
set.seed(112233)
fit_svm_poly <- train(winner ~ ., data = data_train_val,
                      method = "svmPoly",
                      trControl = train_ctrl,
                      tuneGrid = tune_grid,
                      metric = "ROC")

# svm with GRBF kernel
tune_grid <- expand.grid(C = c(5, 10, 30, 50, 70, 100, 200, 300),
                         sigma = c(
                           0.0001, 0.0002, 0.0005,
                           0.001, 0.002, 0.005,
                           0.01, 0.02, 0.05,
                           0.1))
#
set.seed(112233)
fit_svm_grbf <- train(winner ~ ., data = data_train_val,
                      method = "svmRadial",
                      trControl = train_ctrl,
                      tuneGrid = tune_grid,
                      metric = "ROC")


# close cluster connection
stopCluster(cl)
```

Inspect results of tuning for the classifiers.

```
# logistic regression classifiers
fit_log_step
```

```
## Generalized Linear Model with Stepwise Feature Selection
##
## 2405 samples
##   27 predictor
##    2 classes: 'biden', 'trump'
##
## No pre-processing
## Resampling: Cross-Validated (5 fold, repeated 20 times)
## Summary of sample sizes: 1924, 1924, 1924, 1924, 1924, 1924, ...
## Resampling results:
##
##   ROC        Sens       Spec
##   0.9977798  0.9419082  0.9918119
```

```
summary(fit_log_step)
```

```
##
## Call:
## NULL
##
## Coefficients:
##                               Estimate Std. Error z value Pr(>|z|)
```

```
## (Intercept)                     14.3800     1.5107   9.519  < 2e-16 ***
## trump_2016                       9.9320     1.5207   6.531 6.52e-11 ***
## clinton_2016                    -2.8450     1.1150  -2.552 0.010725 *
## pop_2019                        -0.4098     0.2784  -1.472 0.140935
## white_2019                       1.7135     0.3226   5.312 1.09e-07 ***
## women_16_to_50_birth_rate_2017   0.4549     0.2520   1.805 0.071092 .
## bachelors_2019                  -1.0761     0.3393  -3.171 0.001518 **
## unemployment_rate_2019           0.3645     0.2343   1.556 0.119748
## avg_family_size_2019             0.5252     0.2094   2.509 0.012114 *
## household_has_smartphone_2019    0.8594     0.3455   2.488 0.012864 *
## household_has_computer_2019     -1.4830     0.4149  -3.575 0.000351 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 2151.57  on 2404  degrees of freedom
## Residual deviance:  191.88  on 2394  degrees of freedom
## AIC: 213.88
##
## Number of Fisher Scoring iterations: 11
```

```
fit_log_all
```

```
## Generalized Linear Model
##
## 2405 samples
##   27 predictor
##    2 classes: 'biden', 'trump'
##
## No pre-processing
## Resampling: Cross-Validated (5 fold, repeated 20 times)
## Summary of sample sizes: 1924, 1924, 1924, 1924, 1924, 1924, ...
## Resampling results:
##
##   ROC        Sens       Spec
##   0.9976182  0.9415396  0.991339
```

```
fit_log_sub1
```

```
## Generalized Linear Model
##
## 2405 samples
##    4 predictor
##    2 classes: 'biden', 'trump'
##
## No pre-processing
## Resampling: Cross-Validated (5 fold, repeated 20 times)
## Summary of sample sizes: 1924, 1924, 1924, 1924, 1924, 1924, ...
## Resampling results:
##
##   ROC        Sens       Spec
##   0.9971632  0.9352405  0.9915886
```

```
fit_log_sub2
```

```
## Generalized Linear Model
##
## 2405 samples
##    6 predictor
```

```
##    2 classes: 'biden', 'trump'
##
## No pre-processing
## Resampling: Cross-Validated (5 fold, repeated 20 times)
## Summary of sample sizes: 1924, 1924, 1924, 1924, 1924, 1924, ...
## Resampling results:
##
##   ROC        Sens       Spec
##   0.8758451  0.4954478  0.9778485
```

```
fit_log_sub3
```

```
## Generalized Linear Model
##
## 2405 samples
##    8 predictor
##    2 classes: 'biden', 'trump'
##
## No pre-processing
## Resampling: Cross-Validated (5 fold, repeated 20 times)
## Summary of sample sizes: 1924, 1924, 1924, 1924, 1924, 1924, ...
## Resampling results:
##
##   ROC        Sens       Spec
##   0.8814397  0.4242231  0.971254
```

```
# classification tree
plot(fit_ct)
# fit_ct$finalModel  # inspect structure of selected tree

# random forest -- select optimal ntree and keep corresponding
# rf caret object
roc <- sapply(out, "[[", c("results", "ROC"))
colnames(roc) <- ntree_set
rownames(roc) <- tune_grid$mtry
roc
```

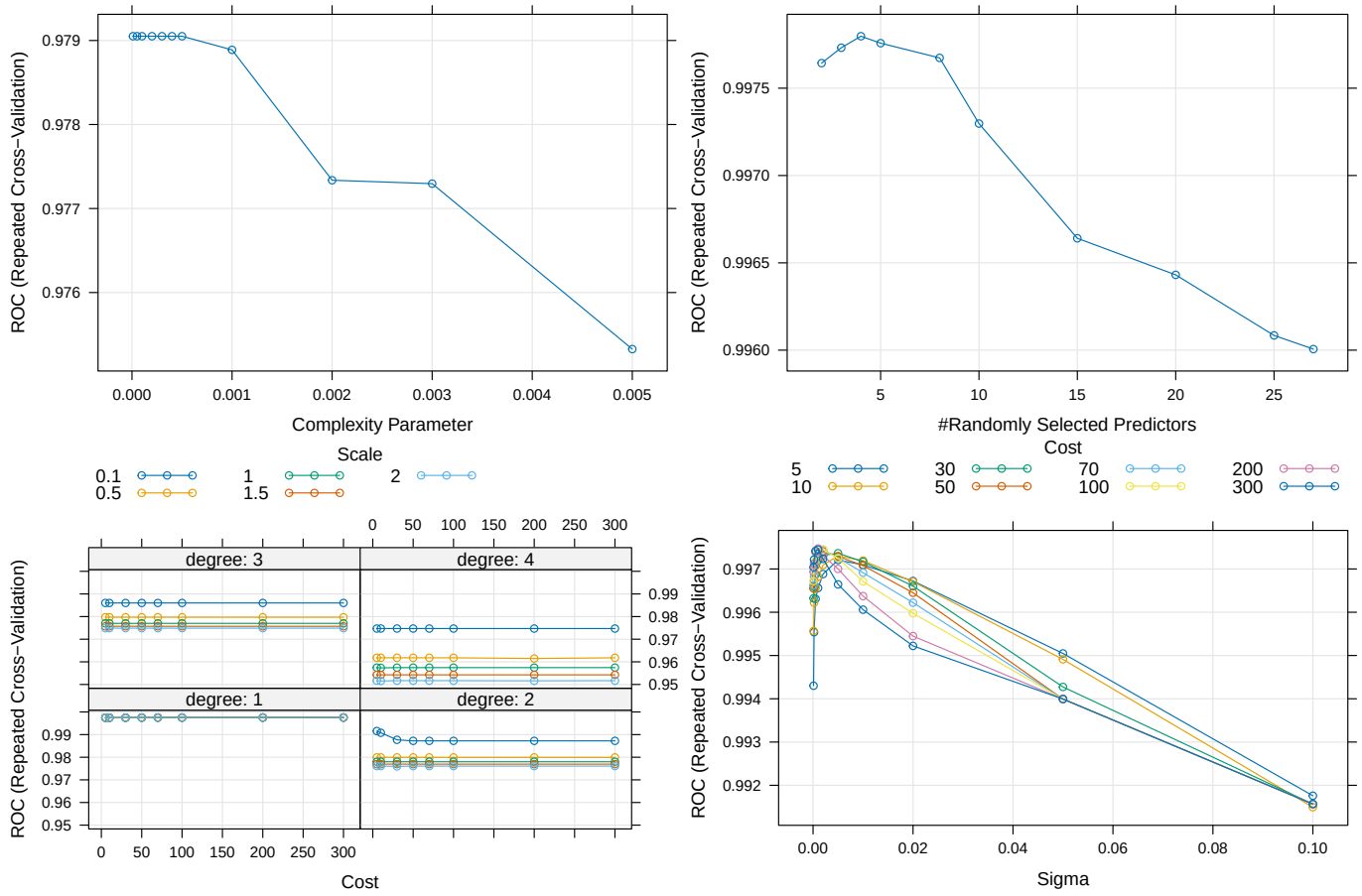
```
##             100       200       300       500      1000      2000      3000
##  [1,] 0.9974744 0.9976007 0.9976072 0.9976353 0.9976436 0.9976527 0.9976555
##  [2,] 0.9972458 0.9975682 0.9976468 0.9976808 0.9977316 0.9977614 0.9977545
##  [3,] 0.9975102 0.9976203 0.9976679 0.9977668 0.9977976 0.9977948 0.9977970
##  [4,] 0.9974146 0.9976165 0.9976760 0.9977605 0.9977579 0.9977960 0.9977888
##  [5,] 0.9969153 0.9971766 0.9972786 0.9973710 0.9976735 0.9976720 0.9976775
##  [6,] 0.9966968 0.9968344 0.9969039 0.9971340 0.9972974 0.9974933 0.9975259
##  [7,] 0.9964929 0.9966308 0.9966166 0.9966092 0.9966405 0.9967100 0.9968882
##  [8,] 0.9957985 0.9961194 0.9963159 0.9963517 0.9964311 0.9964041 0.9964374
##  [9,] 0.9954744 0.9959125 0.9959882 0.9960702 0.9960843 0.9961219 0.9961217
## [10,] 0.9951211 0.9957287 0.9958110 0.9958677 0.9960060 0.9960086 0.9960196
```

```
#
tune_grid <- expand.grid( mtry = c(2, 3, 4, 5, 8, 10, 15, 20, 25, 27) )
sel <- which(roc == max(roc), arr.ind = TRUE)
c( mtry = tune_grid$mtry[sel[1]], ntree = ntree_set[sel[2]] )
```

```
##  mtry ntree
##     4  1000
```

```
#
fit_rf <- out[[ sel[2] ]]   # select
plot(fit_rf)
```

```
# svm
plot(fit_svm_poly)
plot(fit_svm_grbf)
```



In general, the best predictive performance evaluated in terms of AU-ROC is obtained for hyperparameter values and model specifications corresponding to simple classifiers. In all cases, variations of the hyperparameters within a certain range seem to cause small (even very small) variations in the predictive performance.

The logistic regression with automatic variable selection includes a mix of voting, demographic, and socio-economic features. Interestingly, for the logistic regression classifiers, a simple model including only the voting features already provides an excellent out-of-sample predictive performance, similar to that one of a model with all the features and that one of a model determined with automatic variable selection. The logistic regression models using solely demographic and socio-economic features have worse predictive performance, indicating that the voting features are relevant for determining the winning candidate accurately.

For the classification tree classifier, the optimal hyperparameter corresponds to a tree with moderately complex structure. Interestingly, however, only the voting features are actually employed to construct the tree. This result is in line with the considerations made above in rlation to the logistic regression classifiers.

For random forests, the optimal number of variables considered at each split is 4, close to the default of $\sqrt{27} \approx 5$. As the number of variables considered for a split increases towards the bagging classifier, the performance gradually reduces. The optimal number of trees is quite large, equal to 1000. The selected classifier appears to correspond to a situation where trees are significantly different from each other in the ensemble, hence reducing considerably the variance.

For the polynomial SVM, the optimal degree is 1, indicating that actually a simple linear support vector classifier is sufficient to attain good predictive performance. This classifier has a predictive performance on par with that one of a GRBF SVM with non linear decision boundaries, indicating that the GRBF SVM may be unnecessarily complex. The selected hyperparameters for the SVM classifiers correspond to support vector classifiers that allow for overlapping of the classes. Larger values of the cost `C` induced negligible improvements and variations in the predictive performance as measured by AU-ROC.

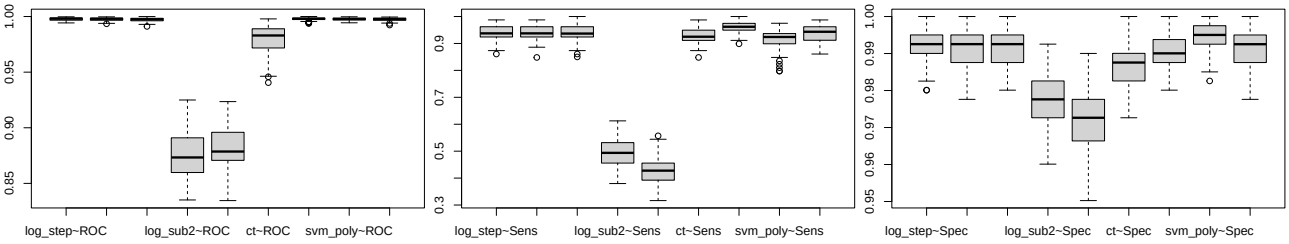
We now compare the different classifiers for the given selected optimal hyperparameter values. The comparison also includes the logistic regression models fitted on subsets of the input features.

```
res <- resamples(list(log_step = fit_log_step,
                      log_all = fit_log_all,
                      log_sub1 = fit_log_sub1, log_sub2 = fit_log_sub2, log_sub3 = fit_log_sub3,
                      ct = fit_ct, rf = fit_rf,
                      svm_poly = fit_svm_poly, svm_grbf = fit_svm_grbf))

# extract entries corresponding to the different metrics
roc <- grep("ROC", names(res$values))
sens <- grep("Sens", names(res$values))
spec <- grep("Spec", names(res$values))
#
# plot
boxplot( res$values[,roc] )
boxplot( res$values[,sens] )
boxplot( res$values[,spec] )

# rank models by average validation AU-ROC
smry <- summary(res)
sort( smry$statistics$ROC[,"Mean"] )
```

```
##  log_sub2  log_sub3        ct  log_sub1  svm_grbf  svm_poly   log_all  log_step
## 0.8758451 0.8814397 0.9790509 0.9971632 0.9974730 0.9975592 0.9976182 0.9977798
##        rf
## 0.9977976
```



The logistic regression models defined on socio-economic and demographic subset of the variables have the lowest AU-ROC scores. These are followed by the classification tree classifiers. These three models have also the largest variance in predictive performance.

Apart from these three models, the logistic regression with variable selection, the logistic regression with all the features, the logistic regression with voting features, the random forest, and the SVMs classifiers attain an average AU-ROC of above 0.997, indicating excellent predictive performance. They also are all characterized by high sensitivity and high specificity, indication of their ability of appropriately detecting both Trump and Biden as the winning candidate. All these models provide very similar predictive performance, with no striking winner, as shown by the overlapping of the AU-ROC boxplots. These models also have reduced variance in predictive performance, which tends to be stable.

Interestingly, the two more "interpretable" models, i.e. the logistic regression with the voting features and that one with selected features, provide an out-of-sample predictive performance which is not too far off or almost on par with the other black box classifiers (random forest and SVMs).

The top three models ranked by average AU-ROC are random forest, logistic regression with automated variable selection, and logistic regression with all the features. The SVM with linear SVM is just outside the podium.

Using the AU-ROC as the main reference metric, the random forest classifiers is the best model overall. This classifier provides the most satisfactory predictive performance given that the main task is the correct identification of the winning candidate. However, it can be argued that the classifiers corresponding to the logistic regression with the selected variables and the logistic regression with the voting features could be also chosen as the best model, given their simplicity and ease of interpretation and that they provide a predictive performance close to that one of the black-box random forest classifier.

(2) We apply the selected model to predict the labels of the test data and report its estimated generalized predictive performance on the test set.

```
class_hat <- predict(fit_rf, newdata = data_test)

# compute metrics
confusionMatrix(class_hat, data_test$winner, mode = "sens_spec")
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction biden trump
##      biden    95     6
##      trump     4   496
##
##                Accuracy : 0.9834
##                  95% CI : (0.9696, 0.992)
##     No Information Rate : 0.8353
##     P-Value [Acc > NIR] : <2e-16
##
##                   Kappa : 0.94
##
##  Mcnemar's Test P-Value : 0.7518
##
##             Sensitivity : 0.9596
##             Specificity : 0.9880
##          Pos Pred Value : 0.9406
##          Neg Pred Value : 0.9920
##              Prevalence : 0.1647
##          Detection Rate : 0.1581
##    Detection Prevalence : 0.1681
##       Balanced Accuracy : 0.9738
##
##        'Positive' Class : biden
##
```

The classifier obtains a sensitivity of 0.96, a specificity of 0.99, and an accuracy of 0.98. These are all indication of accurate prediction, both in terms of accurate identification of counties whose preferred candidate is Trump and counties whose preferred candidate is Biden. Overall, only 10 counties are incorrectly classified in the test data, indicating very good generalized predictive performance and that the classifier is reliable in a real world scenario, like prediction of 2024 election county-level outcomes.