Capstone Two: 21 models tested on the WDBC data

Thomas J. Haslam

March 12, 2019

Overview

This project uses the well-known Breast Cancer Wisconsin (Diagnostic) Data Set (WDBC), available from the UCI Machine Learning Repository, Center for Machine Learning and Intelligent Systems, University of California, Irvine.[1]

Data Set Characteristics

The data set is multivariate, consisting of 569 observations with 32 attributes (variables), with no missing values. It generally conforms to the tidy format: each variable, a column; each observation, a row; each type of observational unit, a table.[2]

The first two variables are the ID number and Diagnosis ("M" = malignant, "B" = benign). The next 30 variables describe ten features of each cell nucleus examined: radius, texture, perimeter, area, smoothness, compactness, concavity, concave points, symmetry, fractal dimension. Since the overall concern is predicting whether a cell is malignant or benign, the primary ML task is classification. Of the 569 observations, for the true values the class distribution is 357 benign ("B"), 212 malignant ("M").[3]

For modelling and evaluation purposes, I have set "M" or the diagnosis of malignant as the positive value, but with `trainControl` for `caret` set as `summaryFunction = twoClassSummary`. As a result, the accuracy scores will reflect both results for "M" and "B" based on their respective true values, and not the just correct percentage of results for "M".

Project Goals

This project uses an ensemble approach to evaluate 21 different ML alogorithms (models) as follows: the models are evaluated for **Accuracy** and **F Measure** (which considers both precision and recall)[4] on two different training and test splits, and three different preprocesing routines.

For the train/test splits, the first run: 50/50. The second: 82/18. For the three different preprocessing routines, Prep_0 (or NULL) uses no preprocessing. Prep_1 centers and scales the data. Prep_2 uses Principal Component Analysis (PCA), after first removing any near-zero-variant values (NZV) and centering and scaling the data. (In other words, Prep_2 uses a standard stack for the Caret package preProcessing option: in this case, "`nzv`, `center`, `scale`, `pca`"; likewise common, "`zv`, `center`, `scale`, `pca`").[5]

Research Questions

In total, each of the 21 models is run 6 times (2 splits; 3 preps each) for a total of 126 prediction/classification results. In evaluating the results, the relevant questions are as follows: Which models perform best overall? How do the different preprocessing routines affect each model? Which models deal best with limited data (the 50/50 split)? Which models learn the best (show significant improvement) when given more data (the 82/18 split)?

[1] WDBC data set available from https://archive.ics.uci.edu/ml/datasets/Breast+Cancer+Wisconsin+(Diagnostic); main UCI Machine Learning Repository site @ http://archive.ics.uci.edu/ml/index.php

[2] "Tidy data", *R for Data Science*, Wickham and Grolemund (2017) @ https://r4ds.had.co.nz/tidy-data.html

[3] Infomation about data set from "Wisconsin Diagnostic Breast Cancer (WDBC)" @ https://archive.ics.uci.edu/ml/machine-learning-databases/breast-cancer-wisconsin/wdbc.names

[4] "F Measure" (redirects to "F1 score"), *Wikipedia* @ https://en.wikipedia.org/wiki/F1_score

[5] Lesson 4, "Preprocessing your data", in *Machine Learning Toolbox*, Kuhn and Deane-Mayer (2018) @ https://www.datacamp.com/courses/machine-learning-toolbox

Ensuring Reproducibility

To ensure both valid comparisions between models and reproducibility of results, all models (for all runs and all preprocessing routines) share the same `caret` command for `trainControl` as follows:

```r
set.seed(2019)
seeds <- vector(mode = "list", length =  1000)
for(i in 1:1000) seeds[[i]] <- sample.int(1000, 800)

###  Does NOT change: same for all models
myControl <- trainControl(
  method = "cv", number = 10,
  summaryFunction = twoClassSummary,
  classProbs = TRUE, # IMPORTANT!
  verboseIter = FALSE,
  seeds = seeds
)
```

Please note that when using `caret` training multiple models , the `seeds` option must be set in `trainControl`[6] to ensure reproducibility of results: it is not enough to `set.seed()` outside of the modelling function. Otherwise, each time `caret` performs cross validation (or bootstrapping or sampling), it will randomly select different observation rows from the training data.

Failure Tables

Finally, *Failure Tables* were generated for each set of model results. The relevant questions here are as follows: Where the failures in diagnosis random? Or, were they specific to certain cell nucleus observerations as determined by the id variable? If so, were these simply much more challenging for the models in general? Or, for certain types of models (for example, *linear* vs. *random forest* models)? Did data preprocessing, or the lack thereof, contribute classification failure on particular observations.

The *Failure Tables* allow us to drill down in much more detail, if needed. For this project, it might seem overkill. But in the context of medical research, it can be highly valuable to identify which set of characteristics typically get misdiagnosed. The *Failure Tables* are a useful step in that direction, and in developing better models or refining existing ones.

Methods / Analysis

The project starts with *EDA*, exploring the data particularly for variance. This will provide insight into what data preprocessing might offer (if anything) for modelling.

The modelling itself involves supervised learning[7], testing the results against known true values. But the *EDA* stage makes use of unsupervised learning[8], particularly cluster analysis, to explore how data processing might affect outcomes. (I learned this approach from Hank Roark's course "Unsupervised Learning in R"[9] at DataCamp).

The models are evaluated in terms of their *Accuracy* and *F Measure* scores. When the only the accuracy results appear in a table, they are ranked in descending order (top to bottom) in accordance with the matching *F Measure* results.

The model selection itself was based largely on what I learned in the *Harvard edX Machine Learning course*[10], with two algorithms eliminated during the development process: `gam` and `wsrf`. The baseline *Generalized*

[6] "trainControl", *RDocumentation* @ https://www.rdocumentation.org/packages/caret/versions/6.0-81/topics/trainControl

[7] "Supervised learning", *Wikipedia* @ https://en.wikipedia.org/wiki/Supervised_learning

[8] "Unsupervised learning", *Wikipedia* @ https://en.wikipedia.org/wiki/Unsupervised_learning

[9] "Unsupervised Learning in R", Roark (2018) @ https://www.datacamp.com/courses/unsupervised-learning-in-r

[10] "34.5 Ensembles", *Introduction to Data Science*, Irizarry, et al (2018) @ https://rafalab.github.io/dsbook/machine-learning-in-practice.html#ensembles

Additive Model[11] `gam`, unlike `gamboost` and `gamLoess`, simply took too long to run and returned generally inferior results. The *Weighted Subspace Random Forest for Classification*[12] `wrfs` also returned consistently inferior results, and I had other more commonly used *random forest*[13] models as valid alternatives.

EDA: Base R, Tidyverse, Unsupervised ML

Exploratory Data Analysis (EDA), obviously, should precede data modelling. If we take the old school approach, we might cast the dataframe `wdbc_data` into a matrix `wdbc_mx` and then check (to start) the mean and sd of each predictor variable with code such as `colMeans2(wdbc_mx)` or `apply(wdbc_mx, 2, sd)` This returns useful but messy output as follows:

```
apply(wdbc_mx, 2, mean) # mean per variable
```

```
##             radius_mean            texture_mean          perimeter_mean 
##            14.127291740            19.289648506            91.969033392 
##               area_mean         smoothness_mean        compactness_mean 
##           654.889103691             0.096360281             0.104340984 
##          concavity_mean     concave_points_mean           symmetry_mean 
##             0.088799316             0.048919146             0.181161863 
##  fractal_dimension_mean               radius_se              texture_se 
##             0.062797610             0.405172056             1.216853427 
##            perimeter_se                 area_se           smoothness_se 
##             2.866059227            40.337079086             0.007040979 
##          compactness_se            concavity_se       concave_points_se 
##             0.025478139             0.031893716             0.011796137 
##             symmetry_se    fractal_dimension_se            radius_worst 
##             0.020542299             0.003794904            16.269189807 
##           texture_worst         perimeter_worst              area_worst 
##            25.677223199           107.261212654           880.583128295 
##        smoothness_worst       compactness_worst         concavity_worst 
##             0.132368594             0.254265044             0.272188483 
##    concave_points_worst          symmetry_worst fractal_dimension_worst 
##             0.114606223             0.290075571             0.083945817
```

We see a high degree of variance, which makes a case for centering and scaling the data, as I will demonstrate shortly. But it also might be better to capture this data as `tidyverse` summary statistics. The `tidy` output as follows, showing only the first 6 rows for sake of brevity:

Table 1: wdbc: Summary Stats [first 6 variables shown]

Variable	Avg	SD	Min	Max	Median
radius_mean	14.1272917	3.5240488	6.98100	28.1100	13.37000
texture_mean	19.2896485	4.3010358	9.71000	39.2800	18.84000
perimeter_mean	91.9690334	24.2989810	43.79000	188.5000	86.24000
area_mean	654.8891037	351.9141292	143.50000	2501.0000	551.10000
smoothness_mean	0.0963603	0.0140641	0.05263	0.1634	0.09587
compactness_mean	0.1043410	0.0528128	0.01938	0.3454	0.09263

Moreover, capturing the data makes it easier for us to the summarize the summary stats.

[11] "Generalized Additive Model", *Wikipedia* @ https://en.wikipedia.org/wiki/Generalized_additive_model

[12] "wsrf: Weighted Subspace Random Forest for Classification", *CRAN* @ https://cran.r-project.org/web/packages/wsrf/index.html

[13] "Random Forests", *UC Business Analytics R Programming Guide* @ https://uc-r.github.io/random_forests

Table 2: wdbc: Range of mean & sd values

min_mean	max_mean	avg_mean	sd_mean	min_sd	max_sd	avg_sd	sd_sd
0.0037949	880.5831	61.89071	195.8614	0.0026461	569.357	34.90472	119.6758

High Variance: A Case for Centering and Scaling?

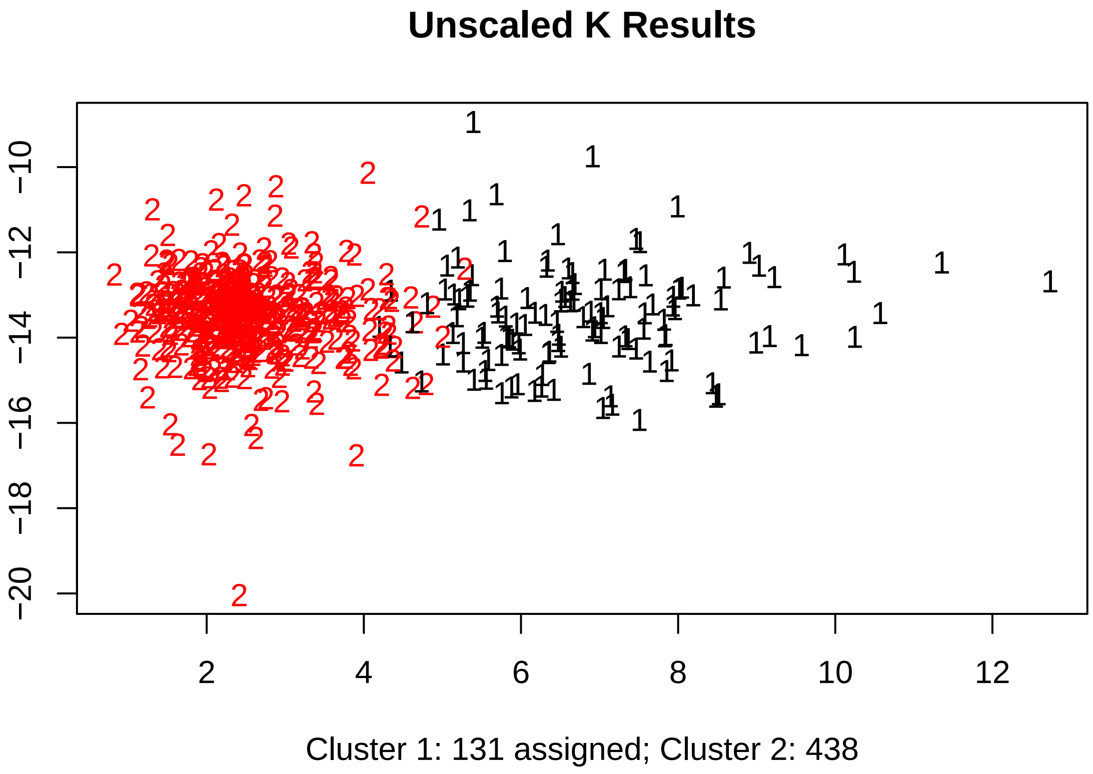
Examining the variance in the data set for all predictor values, we can see the SD of the mean is over 3 times the average of the mean, and the range runs from approximately 0.004 min to 880 max. The SD stats likwise show a considerable range.

To demonstrate how this much variance could affect ML classification, I will use `kmeans` do unsupervised cluster analysis first on the raw data, and then on the centered and scaled data.

Unsupervised ML Exploration

In this exercise, we are simply trying to identify distinguishable clusters in the data. We are not (yet) predicting the classification of "M" (malignant) or "B" (benign). Rather, we want to know what general groupings or patterns occur. I will cheat a little, however, and set the number of clusters to 2. So the question then becomes which approach, no data preprocessing or centering and scaling the data, brings us closer the to right numbers for each group: 357 "B" (benign) and 212 "M" (malignant).

The results from `kmeans(wdbc_mx, centers = 2, nstart = 20)` have been saved as `unscaled_K`, plotted as follows:



Running either `table(unscaled_K$cluster)` or `unscaled_K$size` gives us the breakdown of K1:131; K2: 438. If we assume that the larger cluster 2 is "B" and so cluster 1 "M", an assumption we will test shortly, then `unscaled_K` has failed to idenfity a miminum of 81 malignant cells. This number rises if cluster 2 contains cells that should have been diagnosed as benign but instead were diagnosed as malignant. So our best possible accuracy rate is 0.857645 (488/569).

Let's test whether centering and scaling the data has a meaningful impact on the outcome. For comparison purposes, we will create a second matrix: `wdbc_mx_sc <- scale(sweep(wdbc_mx, 2, colMeans(wdbc_mx)))`.

Unprocessed vs. Centered and Scaled

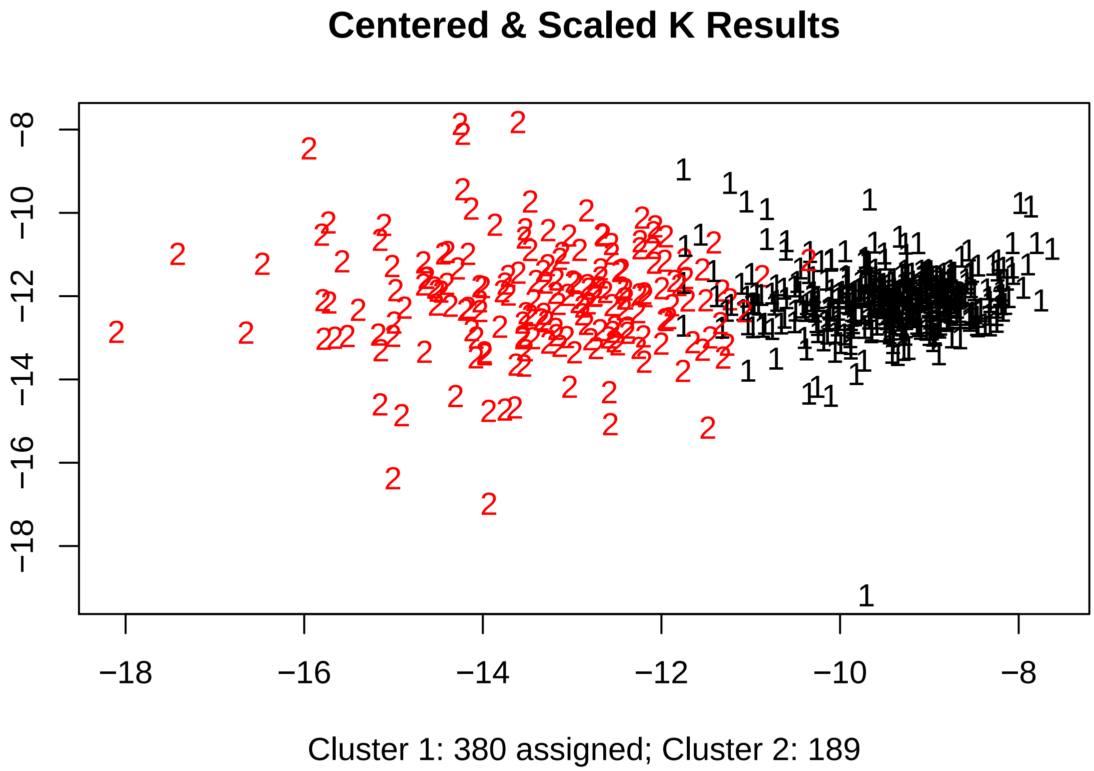
Table 3: Raw Data: wdbc_mx: First 8 Vars. for 5 Cases

Variable	842302	842517	84300903	84348301	84358402
radius_mean	17.9900	20.57000	19.6900	11.4200	20.2900
texture_mean	10.3800	17.77000	21.2500	20.3800	14.3400
perimeter_mean	122.8000	132.90000	130.0000	77.5800	135.1000
area_mean	1001.0000	1326.00000	1203.0000	386.1000	1297.0000
smoothness_mean	0.1184	0.08474	0.1096	0.1425	0.1003
compactness_mean	0.2776	0.07864	0.1599	0.2839	0.1328
concavity_mean	0.3001	0.08690	0.1974	0.2414	0.1980
concave_points_mean	0.1471	0.07017	0.1279	0.1052	0.1043

Table 4: C-S Prep: wdbc_mx_sc: First 8 Vars. for 5 Cases

Variable	842302	842517	84300903	84348301	84358402
radius_mean	1.0960995	1.8282120	1.5784992	-0.7682333	1.7487579
texture_mean	-2.0715123	-0.3533215	0.4557859	0.2535091	-1.1508038
perimeter_mean	1.2688173	1.6844726	1.5651260	-0.5921661	1.7750113
area_mean	0.9835095	1.9070303	1.5575132	-0.7637917	1.8246238
smoothness_mean	1.5670875	-0.8262354	0.9413821	3.2806668	0.2801253
compactness_mean	3.2806281	-0.4866435	1.0519999	3.3999174	0.5388663
concavity_mean	2.6505418	-0.0238249	1.3622798	1.9142129	1.3698061
concave_points_mean	2.5302489	0.5476623	2.0354398	1.4504311	1.4272370

With the predictor variable values centered (on zero) and scaled (now measured in SDs from the mean), we will rerun `kmeans` now on `wdbc_mx_sc` and save the results as `scaled_K`.



Running either `table(scaled_K$cluster)` or `scaled_K$size` gives us the breakdown of K1: 380; K2: 189. In contrast to `unscaled_K`, and again an assumption we will test shortly, if we assume that the larger cluster 1 is "B" and so cluster 2 "M", then `scaled_K` has failed to idenfity a miminum of 23 malignant cells. So now our best possible accuracy rate is 0.9595782 (546/569), but likely lower.

Time to test our assumptions.

Check against true values

So we will convert our *unsupervised learning* run into a raw *supervised learning* attempt. First, I will establish the vector of true values; second, do a quick check to make sure the matrix id numbers match the true value id numbers; and then finally run a `confusionMatrix` test for the *Accuracy* and *F Measure* scores

```
true_values <-  as.factor(wdbc_data$diagnosis) %>%
  relevel("M") %>% set_names(wdbc_data$id) # Set malignant as POSITIVE
# Assign clusters to most likely true value classifications
unscaled_k_pred <- if_else(unscaled_K$cluster == 1, "M", "B") %>%
   as.factor() %>% relevel("M") %>% set_names(wdbc_data$id) # match id to index#
scaled_k_pred <- if_else(scaled_K$cluster == 1, "B", "M") %>%
   as.factor() %>% relevel("M") %>% set_names(wdbc_data$id) # match id to index#
```

The following table results from comparing `true_values[90:97]` , `unscaled_k_pred[90:97]`, and `scaled_k_pred[90:97]`, making sure we are indeed testing the same id numbers.

Table 5: IDs and Indexs Match: sample[90:97]

	861598	861648	861799	861853	862009	862028	86208	86211
True_Values	2	2	1	2	2	1	1	2

	861598	861648	861799	861853	862009	862028	86208	86211
Unscaled_K	2	2	2	2	2	2	1	2
Scaled_K	1	2	2	2	2	1	1	2

So far, so good. The ID numbers match the index order for both K runs, and `scaled_k` appears a bit more accurate than `unscaled_k`. Let's run the `confusionMatrix` for both, check the key results, and wrap up this EDA experiment.

confusionMatrix Results

Table 6: Unscaled and Scaled K: Accuracy and F Measure results

unK_Acc	unK_F1	scalK_Acc	scalK_F1
0.8541301	0.7580175	0.9103691	0.872818

By centering and scaling the data, we improved the *Accuracy* by over 5% and the *F Measure* by over 11%. Importantly, in this context, the `scaled_K` model did much better as not mistaking malignant cells for benign: the potentially more dangerous outcome.

- `scaled_K`: 51 total failures versus 83 for `unscaled_K`.
- 'scaled_K': 45 more malignant cells correctly identified.

Table 7: L-R: True Values, Unscaled K, Scaled K

	TrV	M	B	M	B
M	212	130	1	175	14
B	357	82	356	37	343

So proof of concept. For some ML approaches, as we learned in the *Harvard edx* ML course, centering and scaling the data yields better results.

Thus far, we have two preparations. The null model, so to speak: no preprocessing. And the first, centering and scaling. This brings us to another common data preprocessing routine for ML, *Principal Component Analysis* (PCA).[14]

Principal Component Analysis (PCA)

PCA is particularly well-suited to dealing with large data sets that have many observations and likely collinearity among (at least some of) the predictor variables. *PCA* helps deal with the "curse of dimensonality"[15] by using an orthogonal transformation to produce "a set of values of linearly uncorrelated variables" or principal components.[16] Typically, if *PCA* is a good fit, only small number of principal components will be needed to explain the vast majority of variance in the data set.

The `wdbc_data` does have 30 predictor variables for describing 10 features, and it likewise seems correlation must exist among such variables as `area_mean`, `radius_mean`, and `perimeter_mean`. But the data set only has 569 observations. So *PCA* might not add as much value as it would for a larger data set.

[14] Lesson 4, "Preprocessing your data", in *Machine Learning Toolbox*, Kuhn and Deane-Mayer (2018) @ https://www.datacamp.com/courses/machine-learning-toolbox

[15] "Curse of dimensionality", *Wikipedia* @ https://en.wikipedia.org/wiki/Curse_of_dimensionality

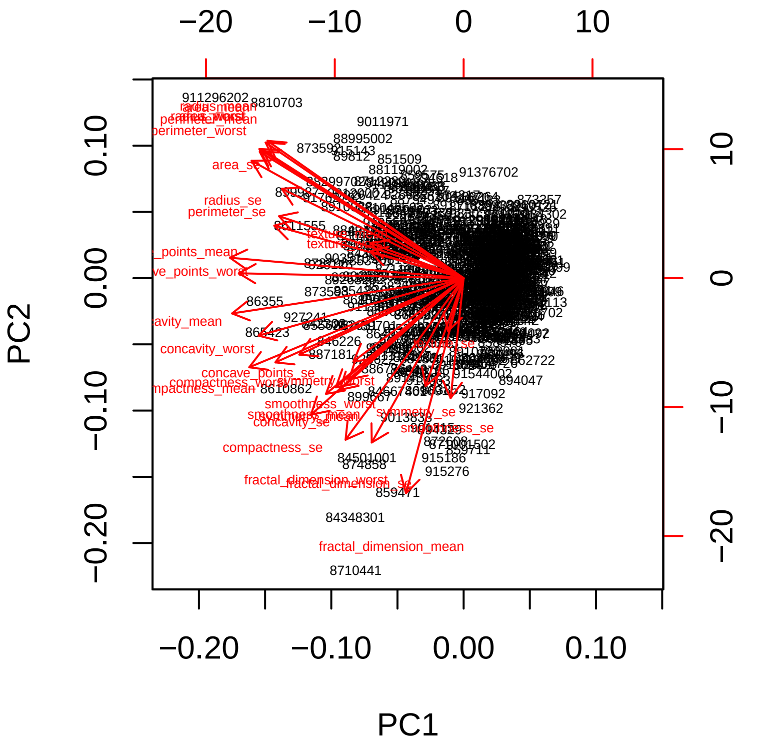
[16] "Principal component analysis", *Wikipedia* @ https://en.wikipedia.org/wiki/Principal_component_analysis

But because this is a experiment in and a report on model testing, we will include *PCA* as a data preprocessing routine.

PCA Biplot

For EDA purposes, running `wdbc_PCA <- prcomp(wdbc_mx, center = TRUE, scale = TRUE)` returns the following biplot:

```r
biplot(wdbc_PCA, cex = 0.45)
```



The closer the lines are, the more correlated the variables. To no surprise, `fractal_dimension_mean` (bottom-center) and `radius_mean` (top-left) are at a near 90 degrees angle: no meaningful collinearity. In contrast, `radius_mean` and `area_mean` (both top-left) are so correlated that the lines and labels almost merge.

The table below samples the first 5 principal components, showing how each has transformed the first eight predictor variables in the original data set.

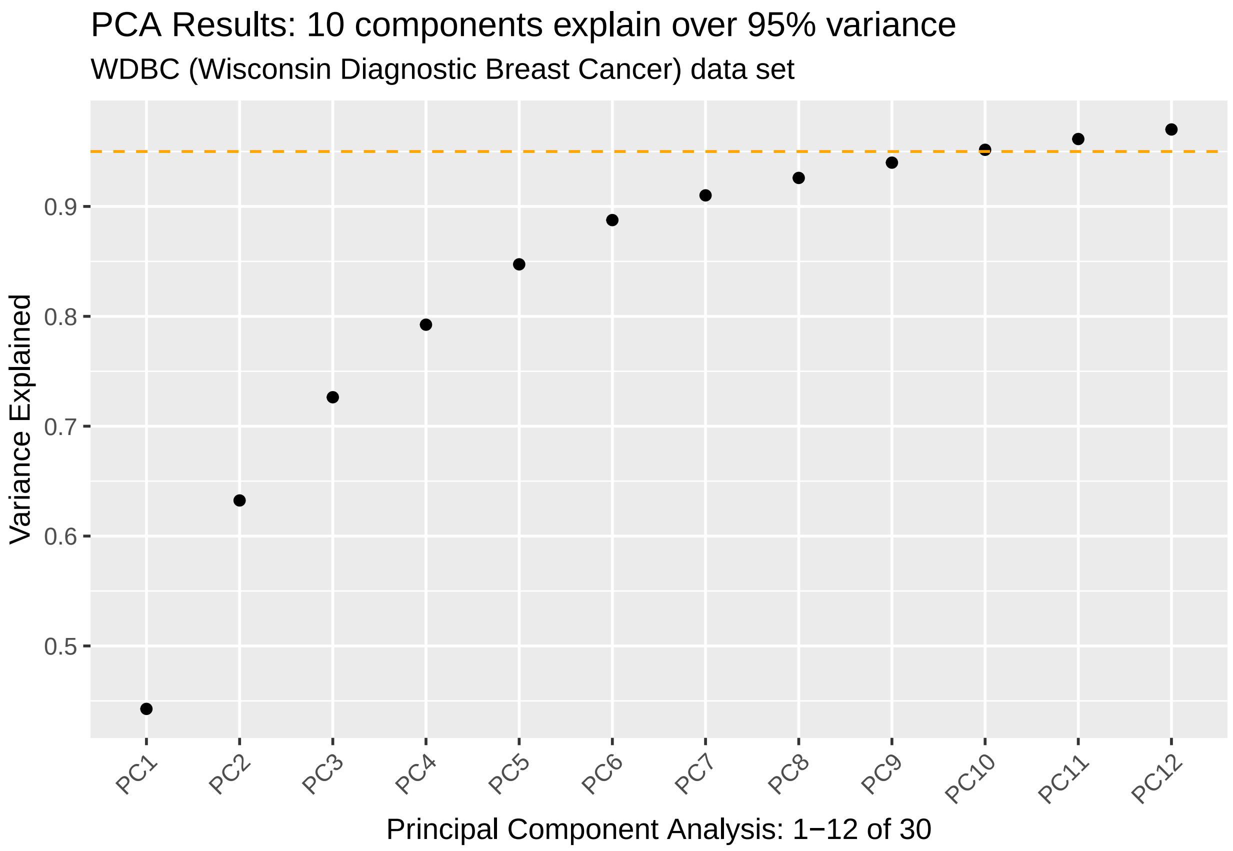
Table 8: PCA Matrix: First 8 Variables for First 5 PC

Variable	PC1	PC2	PC3	PC4	PC5
radius_mean	-0.2189024	0.2338571	-0.0085312	0.0414090	-0.0377864
texture_mean	-0.1037246	0.0597061	0.0645499	-0.6030500	0.0494689
perimeter_mean	-0.2275373	0.2151814	-0.0093142	0.0419831	-0.0373747
area_mean	-0.2209950	0.2310767	0.0286995	0.0534338	-0.0103313
smoothness_mean	-0.1425897	-0.1861130	-0.1042919	0.1593828	0.3650885
compactness_mean	-0.2392854	-0.1518916	-0.0740916	0.0317946	-0.0117040
concavity_mean	-0.2584005	-0.0601654	0.0027338	0.0191228	-0.0863754
concave_points_mean	-0.2608538	0.0347675	-0.0255635	0.0653359	0.0438610

Variable	PC1	PC2	PC3	PC4	PC5

Cummulative Variance Explained

If we plot the PCA summary results, we see that these same 5 principal components explain nearly 85% of the data variance, and the first 10 principal components explain over 95%.



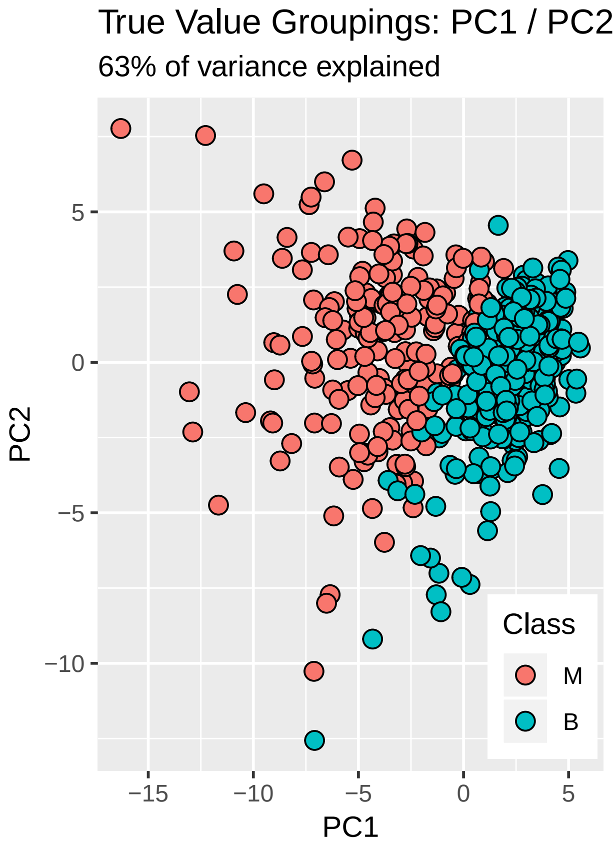
The table below provides the detailed summary results for the same first 5 principal components.

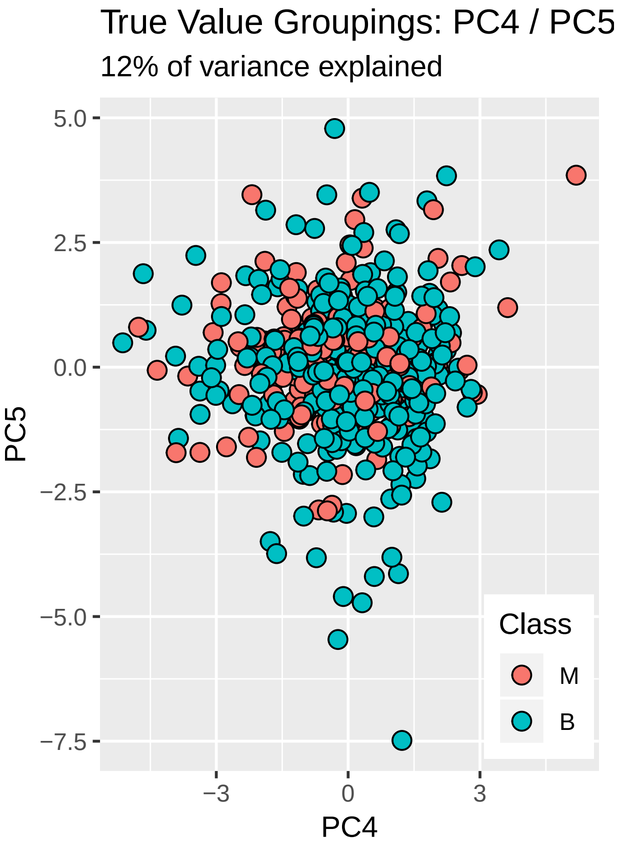
Table 9: PCA Summary: First 5 Components

Results	PC1	PC2	PC3	PC4	PC5
Standard deviation	3.644394	2.385656	1.678675	1.407352	1.284029
Proportion of Variance	0.442720	0.189710	0.093930	0.066020	0.054960
Cumulative Proportion	0.442720	0.632430	0.726360	0.792390	0.847340

PC1 & PC2 vs. PC4 & PC5

Between them, PC1 and PC2 explain 63% of the data variance; PC4 and PC5, 12%. We can plot the data directly against both pairs, as follows:





To no surprise, for PC1 and PC2, which are orthogonally opposed to each other, we have two fairly distinct groupings; but for PC4 and PC5, which explain considerably less variance and by necessity are closer, display no such clear groupings.

PCA Summary

The law of diminishing returns sets in quickly with *PCA*. It breaks the `wdbc_data` down into 30 principal components, but by PC17, `0.9911300` of the variance is explained: for all practical purposes, PC18 to PC30 add nothing and could be safely dropped from modelling.

In principle, this data set (if larger) should strongly benefit from PCA preprocessing.

Modelling

As discussed above in the **Overview**, this project uses an ensemble of twenty-one ML alogorithms (models), on two different training and test splits and with three different data preprocesing routines. So each model is run six times total, and the results are evaluated for **Accuary** (correctly predicting "M" or malignant) and **F Measure**.

Models and Preparations

The data preproccsing commands for `caret`, and the models for the ensemble as below:

```
##
# Experiement with preprocessing: one NULL, two typical
##
prep_0 <- NULL
prep_1 <- c("center", "scale")
```

```r
prep_2 <- c("nzv", "center", "scale", "pca")

###
## Select Models for Ensemble: 21
###
models <- c("adaboost", "avNNet", "gamboost", "gamLoess", "glm", "gbm",
            "knn", "kknn", "lda", "mlp", "monmlp", "naive_bayes", "qda",
            "ranger", "Rborist", "rf", "rpart", "svmLinear", "svmRadial",
            "svmRadialCost", "svmRadialSigma")

mod_names <- enframe(models, value = "Model", name = NULL)
```

Each model for each run, split, and prep, shares the same `trainControl` as discussed in the **Overview**, with cv (cross validation) set to "10".

The generic function for running the ensemble is as follows:

```r
set.seed(2019)
fits_1 <- lapply(models, function(model){
  print(model)
  train(diagnosis ~ ., data = train_set,  method = model,
        trControl = myControl, preProcess = prep_1)
})
```

`fits_1`, in this case, indicates Run One on the 50/50 split, using `prep_1`: the data centered and scaled. The model results are then used to make predictions against the test set, and the results are saved as a dataframe to generate a `confustionMatrix` per model saved as a large list.

```r
#  Predictions
predictions_1 <- sapply(fits_1, function(object)
  predict(object, newdata = test_set))

# Predictions for CFM & F Measure
pred_ft_1 <- predictions_1 %>% as.data.frame() %>%
  mutate_if(., is.character, as.factor) %>%
  mutate_all(., factor, levels = c("M" , "B"))

# Confusion Matrix List for Prep_1
CFM_Prep_1 <- sapply(pred_ft_1 , function(object) {
  CFM <- confusionMatrix(data = object,  reference = test_set$diagnosis)
  list(CFM)
})
```

The *Accuracy* and *F Measure* scores per model, and other results as needed, are then pulled from the `confusionMatrix` list: in the example above, `CFM_Prep_1`.

Train and Test Splitting

The first train/test split is an arbitrary 50/50, intended to examine how well particular models work with limited data. The second train/test split, meant to examine how well the models learn on more data, follows a debatable rule of thumb that the final validation set should "be inversely proportional to the square root of the number of free adjustable parameters."[17] Since `1/sqrt(30)` rounds out to 18%, leaving 104 observations in the final test set, 82/18 seemed good enough.

[17] "Is there a rule-of-thumb for how to divide a dataset into training and validation sets?", *StackOverFlow* (Kiril: answered November 28, 2012) @ https://stackoverflow.com/questions/13610074/is-there-a-rule-of-thumb-for-how-to-divide-a-dataset-into-training-and-validatio/13612921#13612921

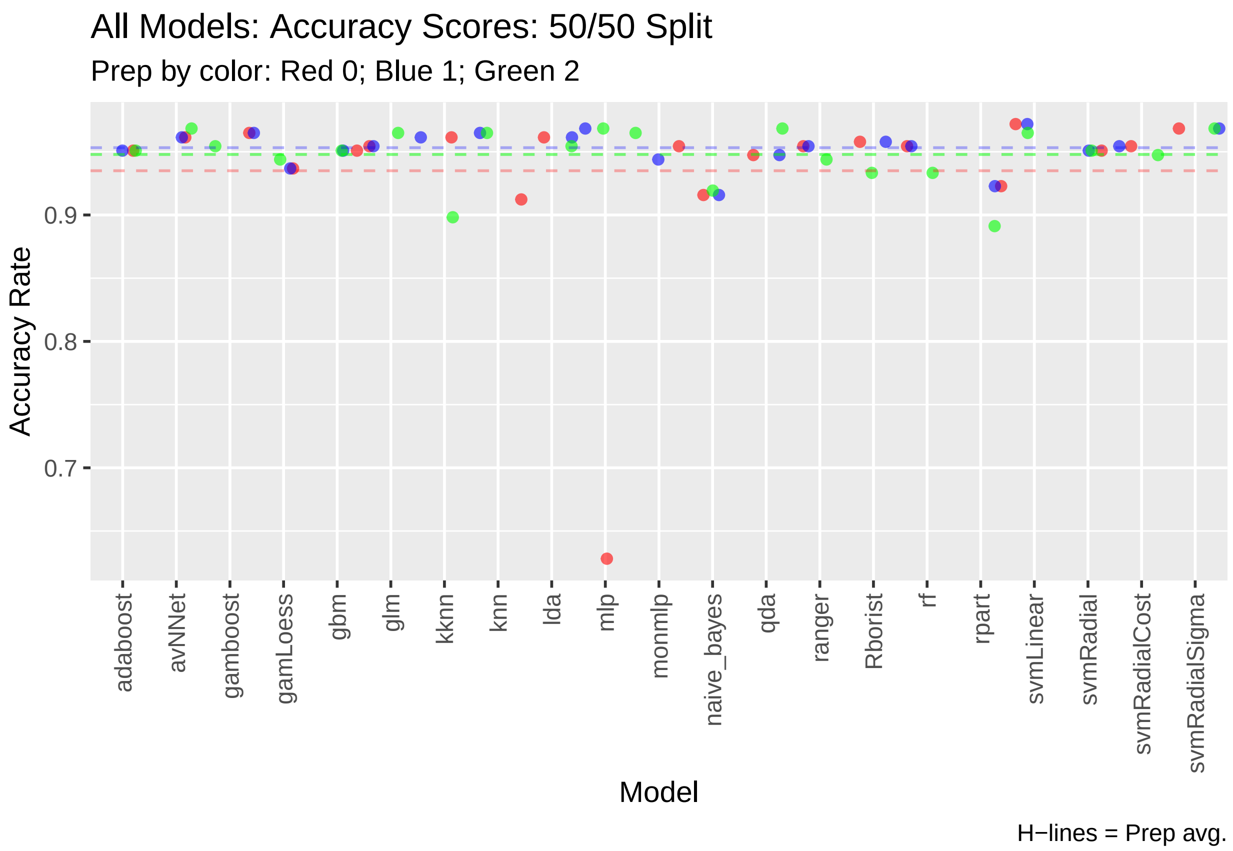
```
#  Create first stage train and test sets:  50% for model training; 50% for testing
#  Second stage: 82% train; 18 % test
#############################   50/50 train/test split
# which models do well with limited data?
Y <- wdbc_data$diagnosis

set.seed(2019)
test_index <- createDataPartition(Y, times = 1, p = 0.5, list = FALSE)

# Apply index
test_set_id <- wdbc_data[test_index, ] # will use ID later
train_set_id <- wdbc_data[-test_index, ]

# Remove id for testing
test_set <- test_set_id %>% select(-id)
train_set <- train_set_id %>% select(-id)

#############################   82/18 train/test split  test_ratio <- 1/sqrt(30)
#  Which models have an ML advantage?

set.seed(2019)
test_index2 <- createDataPartition(Y, times = 1, p = 0.18, list = FALSE)
# Apply index
test2_id <- wdbc_data[test_index2, ]
train2_id <- wdbc_data[-test_index2, ]
# Remove id variable
test_2 <- test2_id %>% select(-id)
train_2 <- train2_id %>% select(-id)
```

A basic assumption of ML is that as the amount of training data increases, the model improves in terms of accuracy as it *learns* from the data. As we will see, this does NOT hold true in every case: or, at least, models improve at disparate rates depending on the data characteristics, the preprocessing (if any), and other conditions and constraints.

Results

Run One: 50/50 Train/Test

We will start with the big picture for Run One: graphing the accuracy scores for the 50/50 training test split, all data preprocessing routines. Then, I will break down Run One by prep: the preprocessing routine. Finally, I will offer the overall summary statistics.



High Performers and One Outlier

Right away, we see wildly different results per model according to the data preparation. In fact, using the *multi-layer perceptron* neural network model[18], `mlp`, on the raw data (`Prep_0` or no preprocessing) returns an accuracy score equivalent to guessing: to the *No Information Rate*. The confusion matrix results below:

```
CFM_Prep_0$mlp
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction   M   B
##          M   0   0
##          B 106 179
##
##                Accuracy : 0.6281
##                  95% CI : (0.5691, 0.6843)
##     No Information Rate : 0.6281
##     P-Value [Acc > NIR] : 0.5265
##
##                   Kappa : 0
##  Mcnemar's Test P-Value : <0.0000000000000002
##
##             Sensitivity : 0.0000
##             Specificity : 1.0000
```

[18] "mlp: Create and train a multi-layer perceptron (MLP)", *CRAN: RSNNS* @ https://rdrr.io/cran/RSNNS/man/mlp.html

```
##          Pos Pred Value :    NaN
##          Neg Pred Value : 0.6281
##              Prevalence : 0.3719
##          Detection Rate : 0.0000
##    Detection Prevalence : 0.0000
##       Balanced Accuracy : 0.5000
##
##        'Positive' Class : M
##
```

Importantly, MLP_0 failed to correctly identify any malignant cells. If `trainControl` were not set to `summaryFunction = twoClassSummary`, the accuracy rate would be 0 (zero). In contrast, MLP_1 (center, scale) and MLP_2 (nzv, center, scale, pca) show significant improvement.

Table 10: mlp Results: Run One

Model	Acc_0	F1_0	Acc_1	F1_1	Acc_2	F1_2	PreProcess_Acc	Overall_Acc
mlp	0.6281	NA	0.9684	0.9581	0.9684	0.9581	0.9684	0.8549667

In contrast to `mlp` on the raw data, `svmLinear`, `svmRadialCost`, and `svmRadialSigma`, three of the *Support vector-machine*[19] learning models natively supported by `caret`[20], do surprising well with no data pre-processing and prove generally robust for all preps tested.

Other Neural Network Models

But the issue with `mlp` seems not to apply generally to all *neural network* models: two other neural network models, also did well across all preps: `avNNet` (*Neural Networks Using Model Averaging*)[21] and `monmlp` (*Multi-Layer Perceptron Neural Network with Optional Monotonicity Constraints*).[22]

Table 11: avNNet Results: Run One

Model	Acc_0	F1_0	Acc_1	F1_1	Acc_2	F1_2	PreProcess_Acc	Overall_Acc
avNNet	0.9614	0.9479	0.9614	0.9484	0.9684	0.9581	0.9649	0.9637333

Table 12: monmlp Results: Run One

Model	Acc_0	F1_0	Acc_1	F1_1	Acc_2	F1_2	PreProcess_Acc	Overall_Acc
monmlp	0.9544	0.9384	0.9439	0.9259	0.9649	0.9528	0.9544	0.9544

The above graph will be replotted, removing the `mlp prep_0` outlier, so that we can see the other results clearer, but first we will examine the results by `prep` (data preprocessing routine).

Run One: Results by Preprocessing Routine

[19] "Support-vector machine", *Wikipedia* @ https://en.wikipedia.org/wiki/Support-vector_machine

[20] "models: A List of Available Models in train", *CRAN: caret* @ https://rdrr.io/cran/caret/man/models.html

[21] "avNNet: Neural Networks Using Model Averaging", *RDocumentation* @ https://www.rdocumentation.org/packages/caret/versions/6.0-81/topics/avNNet

[22] "monmlp: Multi-Layer Perceptron Neural Network with Optional Monotonicity Constraints", *CRAN: monmlp* @ https://rdrr.io/cran/monmlp/

Table 13: Run One: NULL prep: Top Seven Models

Model	Accuracy	F_Measure	Total
svmLinear	0.9719	0.9623	1.9342
svmRadialSigma	0.9684	0.9577	1.9261
gamboost	0.9649	0.9524	1.9173
avNNet	0.9614	0.9479	1.9093
kknn	0.9614	0.9474	1.9088
lda	0.9614	0.9463	1.9077
Rborist	0.9579	0.9434	1.9013

Table 14: Run One: Prep_1: Top Seven Models

Model	Accuracy	F_Measure	Total
svmLinear	0.9719	0.9623	1.9342
mlp	0.9684	0.9581	1.9265
svmRadialSigma	0.9684	0.9577	1.9261
gamboost	0.9649	0.9524	1.9173
knn	0.9649	0.9515	1.9164
avNNet	0.9614	0.9484	1.9098
kknn	0.9614	0.9474	1.9088

Table 15: Run One: Prep_2: Top Seven Models

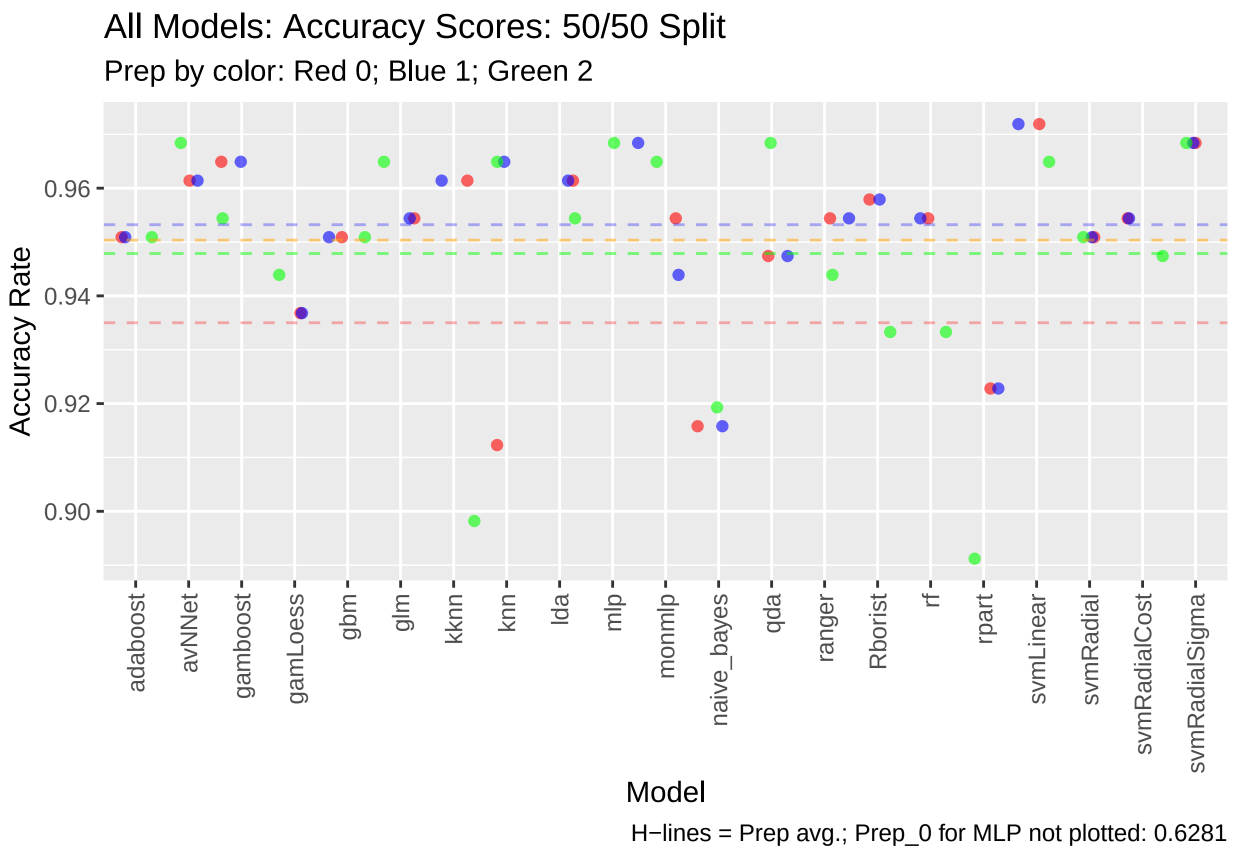
Model	Accuracy	F_Measure	Total
avNNet	0.9684	0.9581	1.9265
mlp	0.9684	0.9581	1.9265
qda	0.9684	0.9577	1.9261
svmRadialSigma	0.9684	0.9577	1.9261
glm	0.9649	0.9537	1.9186
monmlp	0.9649	0.9528	1.9177
knn	0.9649	0.9519	1.9168

The results for Run One are mixed. The two highest overall accuracy scores occurred on the raw data (`prep_0`) and the centered and scaled data (`prep_1`), the model `svmLinear` in both cases

But for the top seven models overall, 1/3 of the models tested, Prep_2 (pca) provided the best average scores, with `svmLinear` not even finishing in the top seven.

Visual Overview Revisited

Let's re-run the graph after removing MLP_0 as an outlier; following the revised graph, we will consult a summary table of the top seven models overall for Run One.



I've added one more line for this graph, in orange, showing the average accuracy score for `prep_0`, NULL, if we remove MLP_0. In fact, with the outlier removed, the average score for `prep_0` is now `0.949655`; this shows a performance improvement over the average score for `prep_2` (pca): `0.9480333`.

Again, the benefits of *PCA* are best realized with large data sets containing both many observartions and predictor variables. But that said, if we look at the average accuracy score for the top seven models for each prep, using *PCA* seems to be having some effect:

Run One: Top Averages Per Prep

Table 16: Run One: Mean Accuracy by Prep for Top Seven Models

Avg_Prep_0	Avg_Prep_1	Avg_Prep_2
0.9639	0.9659	0.9669

We'll see if using *PCA* results in any significant gains for the second split, when we increase the size of the training set. We'll also see if our list of top models changes.

It's also noting that popular models such as `naive_bayes`, `rpart`, and `Rborist` are among the worst performers. (This will hold true for the next run). Although these model would likely benefit from further tuning, one should be careful about choosing a model based on its current cachet.

Below, summary table for Run One: 50/50 test and train split.

Run One: Summary Table

Table 17: Run One: Top Seven Models by Accuracy per Prep

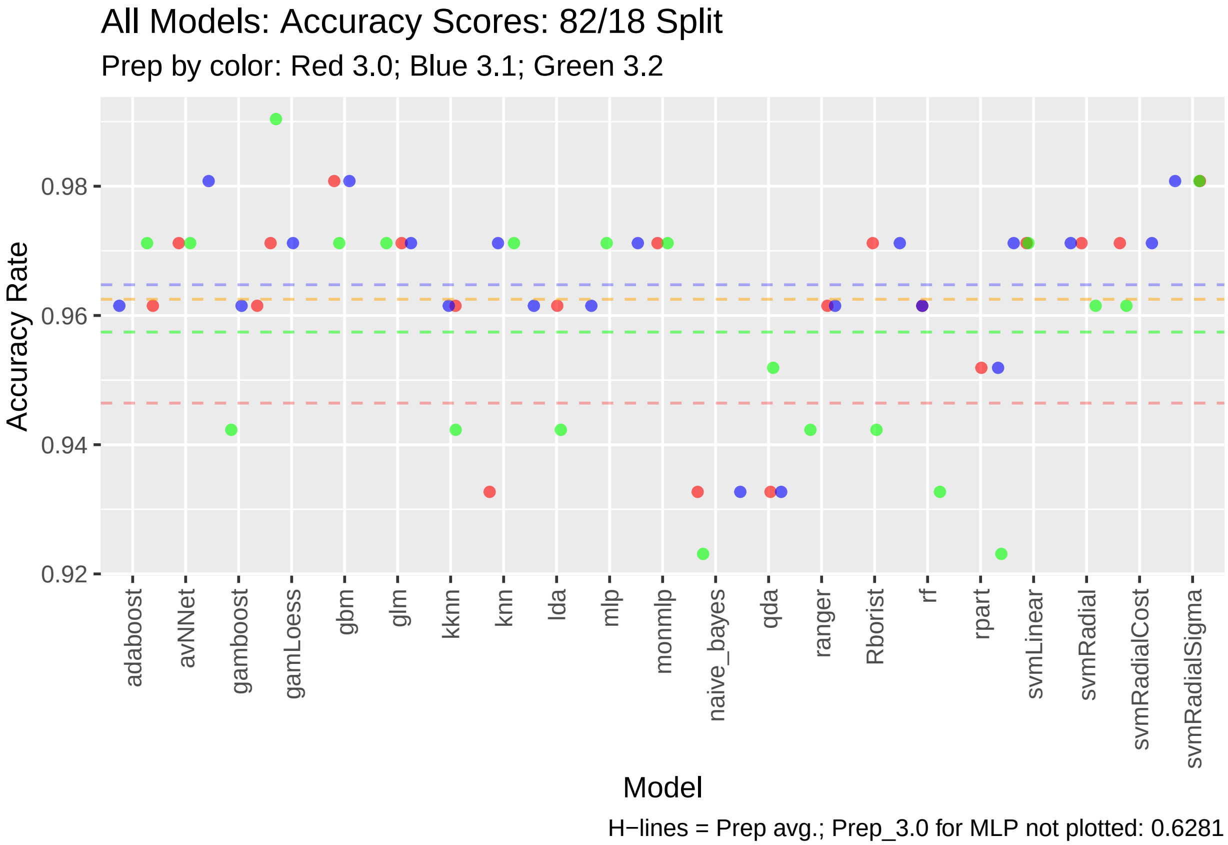
Model_0	Prep_Null	Model_1	Prep_1	Model_2	Prep_2	Model_Overall	Avg_Acc
svmLinear	0.9719	svmLinear	0.9719	avNNet	0.9684	svmLinear	0.9695667
svmRadialSigma	0.9684	mlp	0.9684	mlp	0.9684	svmRadialSigma	0.9684000
gamboost	0.9649	svmRadialSigma	0.9684	qda	0.9684	avNNet	0.9637333
avNNet	0.9614	gamboost	0.9649	svmRadialSigma	0.9684	gamboost	0.9614000
kknn	0.9614	knn	0.9649	glm	0.9649	lda	0.9590667
lda	0.9614	avNNet	0.9614	knn	0.9649	glm	0.9579000
Rborist	0.9579	kknn	0.9614	monmlp	0.9649	monmlp	0.9544000

Run Two: 82/18 Train/Test

Similar to Run One, We will start with the big picture for Run Two : graphing the accuracy scores for the 82/18 training test split, all data preprocessing routines. Then, I will break down Run Two by prep: the preprocessing routine. Finally, I will offer the overall summary statistics.

Since MLP_3.0, `mlp` on the raw data for the 82/18 split, also returns an *Accuracy* score equivalent to the *No Information Rate*, it has been removed from the following graph.

Run Two: Visual Overview



Some models clearly learned from the increased data set better than others. In particular, `gamLoess` finished with the highest accuracy score overall – and did generally well on all preps, in clear contrast to its performance for Run One. *Best learner*: `gamLoess`. The *Support-vector machine* models again proved robust, with `svmRadialSigma` as one of the top three models for all preps.

Similar to Run One, the highest average accuracy score for all models resulted from centering and scaling the data (Prep_3.1); the lowest, from no data pre-processing (Prep_3.0); but the lowest corrected score (MLP_3.0 removed) resulted from using *PCA*. Valuable as *PCA* is, not all ML projects–depending on the data or model or desired outcomes–benefit from using it.

Similar also to Run One, `naive_bayes`, `rpart`, and `Rborist` performed (relatively) poorly; and `qda` took a step backwards. Again, no doubt all these models would benefit from tuning: but as tested under these conditions, they qualify as the *worst learners*.

Run Two: Results by Preprocessing Routine

We will now look at the top seven results for each prep in turn:

Table 18: Run Two: NULL prep: Top Seven Models

Model	Accuracy	F_Measure	Total
gbm	0.9808	0.9737	1.9545
svmRadialSigma	0.9808	0.9737	1.9545
Rborist	0.9712	0.9610	1.9322
svmRadial	0.9712	0.9610	1.9322
svmRadialCost	0.9712	0.9610	1.9322
avNNet	0.9712	0.9600	1.9312
gamLoess	0.9712	0.9600	1.9312

Table 19: Run Two: Prep_1: Top Seven Models

Model	Accuracy	F_Measure	Total
avNNet	0.9808	0.9737	1.9545
gbm	0.9808	0.9737	1.9545
svmRadialSigma	0.9808	0.9737	1.9545
Rborist	0.9712	0.9610	1.9322
svmRadial	0.9712	0.9610	1.9322
svmRadialCost	0.9712	0.9610	1.9322
gamLoess	0.9712	0.9600	1.9312

Table 20: Run Two: Prep_2: Top Seven Models

Model	Accuracy	F_Measure	Total
gamLoess	0.9904	0.9870	1.9774
svmRadialSigma	0.9808	0.9737	1.9545
adaboost	0.9712	0.9610	1.9322
gbm	0.9712	0.9610	1.9322
avNNet	0.9712	0.9600	1.9312
glm	0.9712	0.9600	1.9312
knn	0.9712	0.9600	1.9312

Four interesting results emerge from Prep_0 and Prep_1. First, the *Gradient boosting machine*[23] model `gbm` shows up as a top performer. It clearly benefits from having more data, and learns well (at least in this

[23] "Gradient boosting", *Wikipedia* @ https://en.wikipedia.org/wiki/Gradient_boosting

instance). Second, the *Adaptive Boosting*[24] model adaboost, once the data is centered and scaled (and later, with PCA) also learns well. Third, the *General Additive Model* with *LOESS* (locally weighted polynomial regression)[25], gamloess, not only cracks the top seven unlike Run One, but when using Prep_2 turns in the highest accuracy score of all models tested. Fourth and finally, the old school *Generalized Linear Model*[26] glm also makes the top seven.

Run Two: Top Averages Per Prep

If we average the accuracy scores for the top seven models per prep for Run Two, we find a slight advantage to preprocessing the data first rather than directly modeling the raw data:

Table 21: Run Two: Mean Accuracy by Prep for Top Seven Models

Avg_3.0	Avg_3.1	Avg_3.2
0.9739429	0.9753143	0.9753143

Likewise, the best overall score does belong to gamLoess using prep_2 *PCA*. So the case for data preprocessing is not conclusive, given the data set under consideration, but some models (such as mlp) clearly either require or benefit strongly (e.g., gamLoess, adaboost) from data preprocessing.

Run Two: Summary Table

The table for the top seven per prep as follows:

Table 22: Run Two: Top Seven Models by Accuracy per Prep

Model_3.0	Prep_Null	Model_3.1	Prep_1	Model_3.2	Prep_2	Model_Overall	Avg_Acc
gbm	0.9808	avNNet	0.9808	gamLoess	0.9904	svmRadialSigma	0.9808
svmRadialSigma	0.9808	gbm	0.9808	svmRadialSigma	0.9808	gamLoess	0.9776
avNNet	0.9712	svmRadialSigma	0.9808	adaboost	0.9712	gbm	0.9776
gamLoess	0.9712	gamLoess	0.9712	avNNet	0.9712	avNNet	0.9744
glm	0.9712	glm	0.9712	glm	0.9712	glm	0.9712
monmlp	0.9712	knn	0.9712	gbm	0.9712	monmlp	0.9712
Rborist	0.9712	monmlp	0.9712	knn	0.9712	svmLinear	0.9712

For Prep_2, *PCA*, the model avNNet shows up again as a top performer with a higher accuracy score it had on the raw data or Prep_1 (the centered and scaled data). The advanced models gamLoess, adaboost and gbm now also show as top performers, presumably now because they had more data to learn from. And rounding it up overall, the versatile and robust svmRadialSigma remains a top performer; glm again sets the baseline performance to beat; and the other ML workhorse model knn, which was a top performer for two Preps in Run One, cracks the top seven for the first time in Run Two Prep_1.

For model development and evaluation purposes, glm and knn remain baseline performance standards for good reasons.

Top 10 Models: Both Runs All Preps

We can now consider model performance for both runs and all preps. The table below records the accuaracy average first for prep_1 and prep_2, "CS_to PCA", and second, including prep_0 or NULL, for "All_Preps".

[24] "Adaboost", *Wikipedia* @ https://en.wikipedia.org/wiki/AdaBoost

[25] "LOESS" (redirects to "Local regression"), *Wikipedia* @ https://en.wikipedia.org/wiki/Local_regression

[26] "Generalized Linear Model", *Wikipedia* @ https://en.wikipedia.org/wiki/Generalized_linear_model

Table 23: All Runs/Preps: Top 10 Models for Accuracy

Model	Acc_0	Acc_1	Acc_2	Acc_3.0	Acc_3.1	Acc_3.2	CS_to_PCA	All_Preps
svmRadialSigma	0.9684	0.9684	0.9684	0.9808	0.9808	0.9808	0.974600	0.9746000
svmLinear	0.9719	0.9719	0.9649	0.9712	0.9712	0.9712	0.969800	0.9703833
avNNet	0.9614	0.9614	0.9684	0.9712	0.9808	0.9712	0.970450	0.9690667
glm	0.9544	0.9544	0.9649	0.9712	0.9712	0.9712	0.965425	0.9645500
gbm	0.9509	0.9509	0.9509	0.9808	0.9808	0.9712	0.963450	0.9642500
monmlp	0.9544	0.9439	0.9649	0.9712	0.9712	0.9712	0.962800	0.9628000
svmRadialCost	0.9544	0.9544	0.9474	0.9712	0.9712	0.9615	0.958625	0.9600167
svmRadial	0.9509	0.9509	0.9509	0.9712	0.9712	0.9615	0.958625	0.9594333
gamLoess	0.9368	0.9368	0.9439	0.9712	0.9712	0.9904	0.960575	0.9583833
gamboost	0.9649	0.9649	0.9544	0.9615	0.9615	0.9423	0.955775	0.9582500

First, the *Support-vector machine* models `svm~` proved accurate and robust for both runs and all preps: this an impressive family of models, worth including in future ML projects if only for the development and testing stage.

Second, the *Generalized additive models*, excepting the base `gam` model itself which was eliminated during development, also performed well overall and seemed to learn well when the training data was increased.

Third, two of the neural network models,`avNNet` and `monmlp`, also showed promise.

Fourth and finally, for this sort of data and modelling, the classic *Generalized linear model* `glm` finished in the top ten overall and remains the baseline performance standard to beat.

Failure Tables

Evaluating model performance as above should also encourage us to take a more detailed look at the data. Failure case here refers to a particular observation row as identifed by the unique ID number.

The relevant questions are: Did the better performing models have similar or different failure cases? Since no model performed perfectly, did all models fail on a certain case or set? Did different preparations lead to common fail cases in modelling? Do high rate failure cases have certain features in common? In contrast, were some cases uncontroversial for all or mostly all models?

Finally, we can also use *Failure Tables* to track the performance of individual models in detail.

The variable `Percent` in the tables below refers on a scale of 0 to 1, with 1 as 100% of the number of correct predictions per case. So an observation with a score of 0.5 would mean that 50% of the models got it right; 0, 0%; and 1, 100%.

Overall Results

For Run One, the 50/50 train/test split, running `mean(All_Predictions_Run_One$Percent == 1)` results in `0.477193`: 48% of the observations were uncontroversial. All models correctly classified these cases according to their true value diagnosis. We could examine these cases in more detail to understand why, and perhaps even remove the majority of them for the purposes of future modelling.

For Run Two, the 82/18 train/test split, `mean(All_Predictions_Run_Two$Percent == 1)` results in `0.5480769` or 55%, which suggests that several of the models have learned from expanded training data.

By combining the total number of such cases by ID number, and running `table(Obvious_Cases$diagnosis)` we find that there are 168 unproblematic cases total: all of them class "B" or dianogsis benign.

We can also isolate the most problematic cases. Below, the first eight most problematic cases for Run One and Run Two.

Table 24: 50/50 Split: Failures Common to All Preps: First 8 rows/cols

id	diagnosis	Percent	adaboost_0	avNNet_0	gamboost_0	gamLoess_0	glm_0
892189	M	0.0000000	B	B	B	B	B
921386	B	0.1428571	B	M	M	M	M
889403	M	0.2222222	B	M	B	B	M
91594602	M	0.2380952	B	M	B	M	M
859711	B	0.3809524	B	B	B	B	B
855133	M	0.3968254	B	B	B	B	B
857438	M	0.4126984	M	M	M	M	M
8810987	M	0.4444444	M	B	M	B	M

Table 25: 82/18 Split: Failures Common to All Preps: First 8 rows/cols

id	diagnosis	Percent	adaboost_3.0	avNNet_3.0	gamboost_3.0	gamLoess_3.0	glm_3.0
892189	M	0.0000000	B	B	B	B	B
889403	M	0.1428571	B	M	B	B	B
906564	B	0.4285714	B	B	B	B	B
846381	M	0.4603175	M	B	B	M	M
905680	M	0.4761905	B	M	B	M	M
845636	M	0.5714286	B	M	M	M	M
881094802	M	0.6031746	M	B	M	B	B
8710441	B	0.6031746	B	B	B	B	B

(In the above tables, only the first 6 model results for Prep_0 are shown. 115 more model results are included in the entire table).

Since the both test sets as randomly generated by `caret` share some common cases, it should not surprise us that we have common failures to all run and preps, with one failure case common to all models:

Table 26: The 8 Failures Common to All Runs-Prep Combinations

id	diagnosis	Percent	adaboost_0	avNNet_0	gamboost_0	gamLoess_0	glm_0
892189	M	0.0000000	B	B	B	B	B
889403	M	0.1825397	B	M	B	B	M
905680	M	0.5476190	B	M	B	M	M
8710441	B	0.5476190	B	B	B	M	M
906564	B	0.5396825	M	B	B	B	B
862717	M	0.6031746	M	M	M	M	M
881094802	M	0.5079365	M	M	M	M	M
846381	M	0.6428571	M	B	M	M	M

Data Details for Common Failure Cases

We can examine the data itself in more detail to understand why it was so difficult to diagnose correctly:

Table 27: Common Failures: Data Details [First 5 Vars]

id	Percent	diagnosis	radius_mean	texture_mean	perimeter_mean	area_mean	smoothness_mean
892189	0.0000	M	11.760	18.14	75.00	431.1	0.0997
889403	0.1825	M	15.610	19.38	100.00	758.6	0.0784
881094802	0.5079	M	17.420	25.56	114.50	948.0	0.1006
906564	0.5397	B	14.690	13.98	98.22	656.1	0.1031
905680	0.5476	M	15.130	29.81	96.71	719.5	0.0832
8710441	0.5476	B	9.731	15.34	63.78	300.2	0.1072
862717	0.6032	M	13.610	24.98	88.05	582.7	0.0949
846381	0.6429	M	15.850	23.95	103.70	782.7	0.0840

Importantly, 6 of 8 failure cases, including the 4 with the highest rate of incorrect classifications, were malignant cells mistakenly identified as benign.

Failures Shared Across Data Preps

We can also examine the failure cases for each type of prep, as the three tables below show:

Table 28: Null Prep Fails: Both Splits: First 8 rows/cols

id	diagnosis	Null_percent	adaboost_0	avNNet_0	gamboost_0	gamLoess_0	glm_0
892189	M	0.0000000	B	B	B	B	B
889403	M	0.2619048	B	M	B	B	M
905680	M	0.4761905	B	M	B	M	M
8710441	B	0.6190476	B	B	B	M	M
906564	B	0.7142857	M	B	B	B	B
846381	M	0.7857143	M	B	M	M	M
881094802	M	0.8333333	M	M	M	M	M
9013838	M	0.8333333	B	M	M	M	M

Table 29: Prep_1 Fails: Both Splits: First 8 rows/cols

id	diagnosis	Prep1_percent	adaboost_1	avNNet_1	gamboost_1	gamLoess_1	glm_1
892189	M	0.0000000	B	B	B	B	B
889403	M	0.1904762	B	B	B	B	M
905680	M	0.4761905	B	M	B	M	M
8710441	B	0.6428571	B	B	B	M	M
906564	B	0.7142857	M	B	B	B	B
881094802	M	0.7619048	M	B	M	M	M
862717	M	0.8571429	M	M	M	M	M
915452	B	0.8809524	B	B	B	M	B

Table 30: Prep_2 Fails: Both Splits: First 8 rows/cols

id	diagnosis	Prep2_percent	adaboost_2	avNNet_2	gamboost_2	gamLoess_2	glm_2
892189	M	0.0000000	B	B	B	B	B
889403	M	0.0952381	B	M	B	B	M
881094802	M	0.4761905	M	B	B	B	B

id	diagnosis	Prep2_percent	adaboost_2	avNNet_2	gamboost_2	gamLoess_2	glm_2
8710441	B	0.5952381	M	B	B	B	B
905680	M	0.6190476	B	M	M	M	M
845636	M	0.6666667	M	M	B	M	M
906564	B	0.6904762	M	B	M	B	B
862717	M	0.7857143	M	M	M	M	M

A cursory examination of these tables strongly suggests the case data itself, and not the preprocessing or lack thereof, largely determined the failure rate.

Finally, we can also drill down in detail on the performance of any given model or models. Since `gamLoess` provided the highest overall accuracy score, and emerged as one of the best learning models, we will use it as a case study.

gamLoess Failure Details

Table 31: gamLoess Accuracy & Fail Counts by Prep

gamLoess	Acc_0	Acc_1	Acc_2	Acc_3.0	Acc_3.1	Acc_3.2
Accuracy	0.9368	0.9368	0.9439	0.9712	0.9712	0.9904
Fail Count	18.0000	18.0000	16.0000	3.0000	3.0000	1.0000

Table 32: gamLoess Accuracy & Fail Counts by Prep

gamLoess	Acc_0	Acc_1	Acc_2	Acc_3.0	Acc_3.1	Acc_3.2
Accuracy	0.9368	0.9368	0.9439	0.9712	0.9712	0.9904
Fail Count	18.0000	18.0000	16.0000	3.0000	3.0000	1.0000

Table 33: gamLoess Fails: Run One: [First 8]

id	diagnosis	percent	gamLoess_0	gamLoess_1	gamLoess_2
855133	M	0	B	B	B
877159	M	0	B	B	B
889403	M	0	B	B	B
8915	B	0	M	M	M
892189	M	0	B	B	B
905557	B	0	M	M	M
9112085	B	0	M	M	M
91813701	B	0	M	M	M

Table 34: gamLoess Fails: Run Two: [3 total]

id	diagnosis	percent	gamLoess_3.0	gamLoess_3.1	gamLoess_3.2
892189	M	0.0000000	B	B	B
881094802	M	0.3333333	B	B	M
889403	M	0.3333333	B	B	M

As we saw earlier, `gamLoess` learns well – and likewise clearly benefits from data preprocessing using a standard *PCA* stack. In the final run/prep, for which it returned the highest overall accuracy score, it failed only on the case that every other model for all runs/preps failed on: `id 892189`.

Someone with expert knowledge could better explain why this cell, with a true value diagnosis as "M" or malignant, passed for "B" or benign in every classification test thus far.

In general terms, using the data details to better understand model failure can help us to improve existing models, or to develop new models that better deal with highly problematic cases.

Conclusion

Insofar as this project has resulted in any clear findings or recommendations, they are the following three:

1. Do extensive testing and development, including different data splits and preprocessing routines, before deciding upon a final model or approach.
2. Define your project goals before modelling: based on your data and desired outcomes, determine what would count as good result. Then benchmark accordingly.
3. Save your results data, and not just key scores or values. It provides insight to what worked and why, and as to what failed and why. If the ML project is worth doing in the first place, at some point it will likely be revisited for possible improvement.

Why these three?

Judging from what I read on *Medium: Towards Data Science*[27] and elsewhere, the trend right now is to use the most popular advanced approaches, with the assumption that the *neural network* or *nonparametric* alogrithim *de jour* with the right combination of *PCA* or other sophisticated data transformations will magically solve the problems for you.

Alternatively, and equally fashionable, just run `Keras` on a cloud account even if you have no idea otherwise what you are doing. My favorites are the bloggers who brag about being data scientists without having to understand basic statistics.

That notwithstanding, as the above experiment has shown, depending upon the data and test conditions, advanced models do not always deliver the best results, and not every model requires or even benefits from *PCA*.

Moreover, if I had went with the early best results, I would have eliminated some of the top performing models – the *best learners* – once the training data increased in size.

Only by experimenting, with goals in mind, was I able to identify the better ensemble models for this task of `twoClassSummary` classification.

One of the 20th century's greatest computer scientists, Donald Knuth[28], once infamously warned a fellow computer scientist in a letter:

"Beware of bugs in the above code; I have only proved it correct, not tried it."

Many of us currently doing ML seem to have the opposite dilemma: the code is bug-free, runs fine and fast, but that proves neither it nor the results correct for the task at hand.

Know the data.

Even if informally, consider some form of **Preregistration**:[29] "When you preregister your research, you're simply specifying to your plan in advance. . . . Preregistration separates hypothesis-generating (exploratory) from hypothesis-testing (confirmatory) research."

[27] *Medium: Toward Data Science* @ https://towardsdatascience.com/

[28] "Donald Knuth", *Wikipedia* @ https://en.wikipedia.org/wiki/Donald_Knuth

[29] "What is Preregistration?", *Center for Open Science* @ https://cos.io/prereg/

Instead of *p-hacking*[30], ensemble ML makes it very easy to engage in model-hacking: that is, grab the first or most impressive result from the ensemble and call it a day. But our longer term goals include better results on the next set of related data. So it might help not to quit too early even when it looks like success. Finding the failure curve, after the law of diminishing returns sets in, helps us better identify the area of success.

Models

The *Support-vector machine* family of models for `caret`, the `svm~` group, turned in an impressive overall performance.

The more advanced *neural network* and *Generalized additive models* also did well, particularly when the training data size was increased, and the data preprocessed. These models would likely scale well.

For this data set, the *random forest* models in general did not do as well; and other current favorites (rightfully so), such as `naive_bayes` and `rpart` also underperformed. But again, fine-tuning the parameters for these models would have likely improved performance.

Finally, for model development and evaluation purposes, the well-established `glm` and `knn` remain baseline performance standards for good reasons. Run these, and then try to do better.

Data Preprocessing

The overall results are mixed, but some models benefit significantly from basic data preprocessing such as centering and scaling (`prep_1`), and one model tested, `mlp`, required it. The models that worked well on the raw data (`prep_0`) also generally worked equally well or better on centered and scaled data (`prep_1`). So it seems there is little risk, but possible to considerable reward in centering and scaling the data.

Please note however that *random forest* models typically do not require data preprocessing – but also that the *random forest* models generally underperformed for this particular data set and classification task.

In theory and often enough in practice, *Principal Component Analysis* (PCA) yields huge benefits for data sets suffering from the "curse of dimensonality". But rather than assuming *PCA* will always bring about a better result, one should do a controlled experiment during the modelling testing and development stage. For this data set, the results from using *PCA* were mixed.

Failure Cases

The problem could be with the data. Mistakes get made. Even if not, the question as to why some cases are invariably diagnosed incorrectly is an interesting one. But unless we can pinpoint particular observations and their effects on models, we'll never know.

As stated earlier, even the most advanced current ML project could likely be improved upon in the near or intermediate future. By capturing the failure cases, we can use the data details to better understand existing model failure. This can help us to develop better models, which when used for analyzing medical data could prove life-saving.

Sources, References, and Inspiration

For all URLs listed in text or in notes, please assume the retrieval dates range from 2019-03-01 to 2019-03-12. As we all know, web-based content can change quickly or even vanish. All links were re-tested before the final version of the this report was submitted. Thank you.

```
# Capstone Project Two: 21 models tested on the WDBC data
# Submitted on 2019-03-12, by Thomas J. Haslam
# In partial fulfillment of the requirements for the Harvard edX:
#    Data Science Professional Certificate
```

[30] "P-hacking", *Psychology Wiki* @ http://psychology.wikia.com/wiki/P-hacking