MSFT Data Science Interview

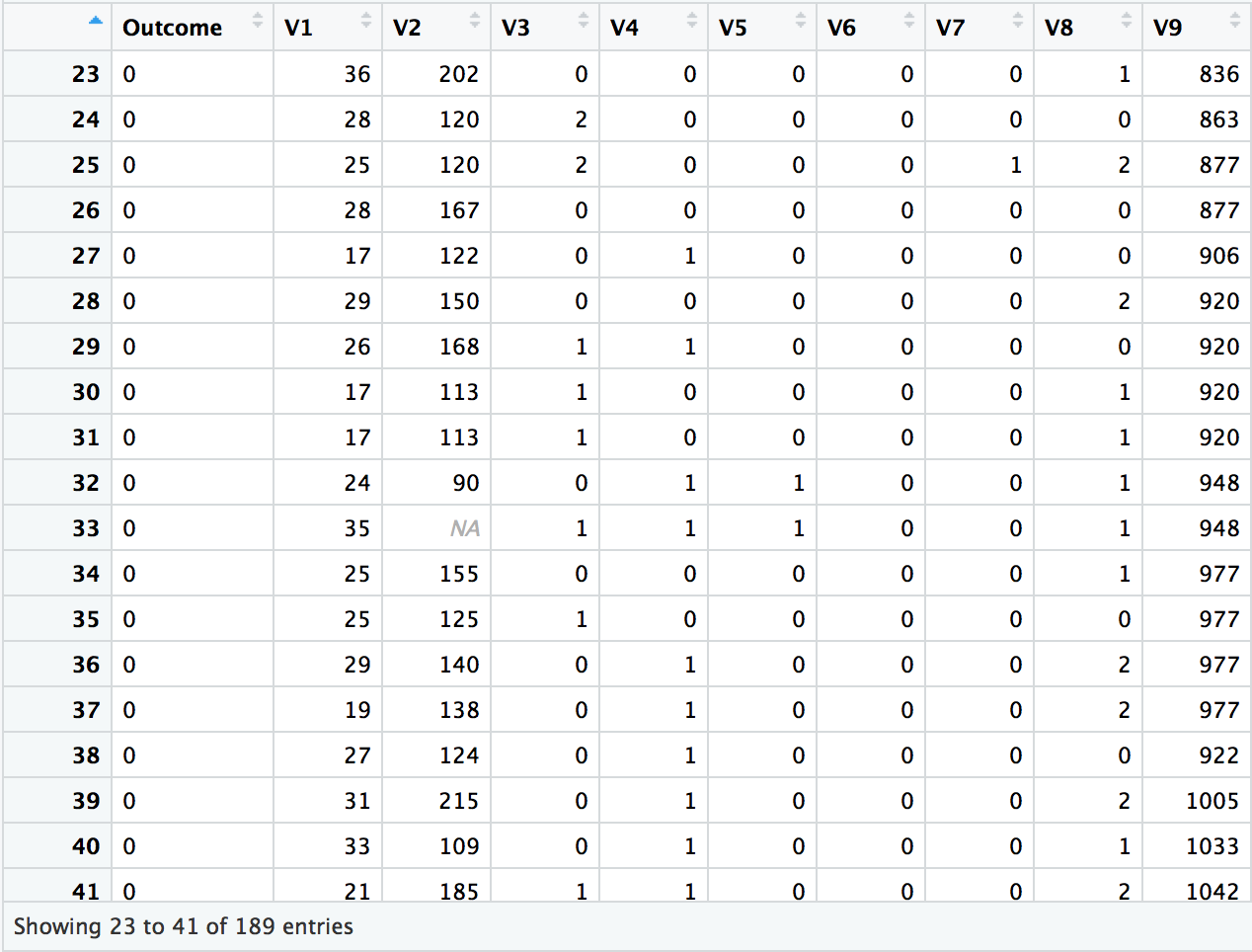
Eugene’s Modeling Test

William Constantine, Ph.D.

**E-Mail**: [wlbconstan@gmail.com](mailto:wlbconstan@gmail.com)

**Phone**: (253) 324-8738

**GitHub Project URL**: https://github.com/wconstan/msft-office-sample.git



# Objective

Dr. Chudin supplied a small data set with an identified dependent variable (Outcome) and nine predictor variables (V1, …, V9). The objective was to build a model for predicting the Outcome variable using the predictors and to discuss conclusions.

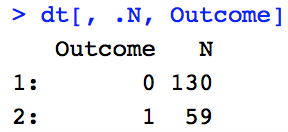
# Scripting Language

I chose the R language to develop the models for this exercise as there are many advanced statistical and machine learning packages freely available and it is easy to generate good looking plots to explain the results. The R script I created is attached to this report. You can also clone a GitHub repository that I created for this project to obtain the source: https://github.com/wconstan/msft-office-sample.git.

# Data Considerations

## Class Imbalance

The data has a sizable class imbalance with Outcome=0 having more than twice the number of observations labeled as Outcome=1:



Most machine learning classifiers are sensitive to an imbalance in the predictors used for training the model. Generally, an unbalanced dataset will bias the prediction model towards the more common class. There are two main techniques used for balancing data:

* **under-sampling**: a random subset of observations is taken from the class with more samples to match the lesser number of samples available in other classes. The main disadvantage of under-sampling is that we lose potentially relevant information from the omitted samples.
* **oversampling**: samples are duplicated from the class with fewer instances or are based on the data that we have, so as to match the number of samples in each class. The risk in this approach is contaminating both the test and training data sets with oversampled values, which may lead to overfitting and an overestimation of our model’s performance and generalizability.

There are also hybrid approaches that involve a mix of over- and under-sampling. For the current study, I compare the classification results for the following approaches:

1. do nothing (not appealing)
2. under-sampling
3. over-sampling
4. [ROSE](https://journal.r-project.org/archive/2014/RJ-2014-008/RJ-2014-008.pdf): A Package for Binary Imbalanced Learning
5. [SMOTE](https://arxiv.org/pdf/1106.1813.pdf): Synthetic Minority Over-sampling Technique

## Missing Data

There is only a single value that is missing in the data: observation 33 of the V2 predictor. I used rfImpute(Outcome ~ ., dt) to fill in the missing point, where dt is the data frame containing all supplied observations. rfImpute works by first replacing NA values with corresponding column medians and then feeding the resulting data set into a random forest model. The proximity matrix from the random forest is used to update the imputation of the NA values, which is taken as the weighted average of the non-missing observations where the weights are the proximities. As V2 was originally comprised of integer values, I coerced the resulting imputation to type integer as well.

# Classification Models

## Variable Correlations

The first step in the modeling process began with and visual examination of variable correlation through a pairs plot with color-coded classes (Fig. 1). The data was standardized in this plot for easier visual comparison. I observed in the final row of the pairs plot that the classes seemed *perfectly* divided over *all* paired variables with an appropriate value of V9 to split the data. A visualization of one of these pairs and the resulting perfect split is shown in Fig. 2. Based on this observation, I manually split the (standardized) data into two sets using V9 >= -0.578 as a rule and assessed the classification error using the **majority** class as a prediction for all observations. This emulates what would be done by a decision tree at the first split/node. In this case, the parent node (original data) had a classification error of 0.31 while the classification error of the two children (after the split) was **zero**! Seemingly, our work is done! However, if the V9 split provides a *perfect* model for the supplied data, the natural question arises: what might be wrong with such a model when applied to out-of-sample data? Could V9 be noisy, for example? Or could the distribution of the V9 observations be significantly different in the out-of-sample data than that supplied to us in the training data? These issues need to be addressed in order to make our model more generalizable.



Figure 1: Pairs plot of imputed data



Figure 2: Split on standardized data at V9=-0.578 illustrates a perfect division of the classes in the V1-V9 space

## Additive Noise

To make our model more generalizable, I generated noisy versions of by adding to it Gaussian white noise ()

such that had a specified signal-to-noise ratio:

where is the (sample) standard deviation of x. Figures 3 shows a comparison of the final row of the pairs plot for three different levels of additive noise: none, SNR=2, and SNR=1. As the noise level increases (SNR decreases), you can see that the classes become more mixed, making a simple single cut of the data at a particular threshold less effective.

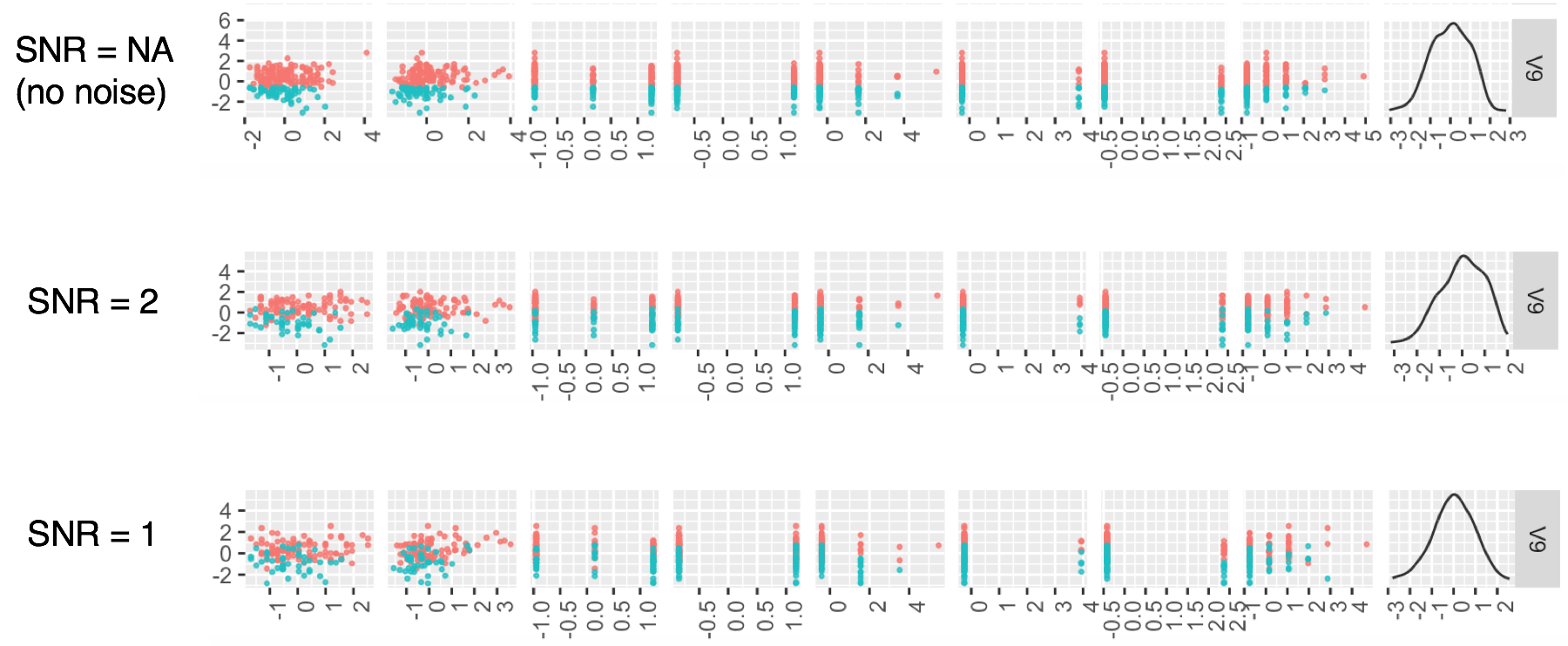


Figure 3: Pairs plots for V9 with different levels of additive noise

This experiment seems a bit restrictive: why not add noise to other variables? I chose to manipulate only the V9 variable as it is dominates the other variables by importance, at least according to a random forest model using the original data:



## Modeling Approach

### Classifiers

I developed both *random forest* and *logistic regression* classification models using the **caret** package:

* Logistic regression was chosen as it is a staple classifier for binary outcomes, modeling the log-odds as a weighted sum of the explanatory variables.
* Decision trees partition the data in such a way as to optimize the homogeneity of classes in the resulting sub-nodes. We have demonstrated this very effect in splitting the V9 predictor at just the right value! So, using a decision tree as a classifier for our data seems like a natural choice. However, decision trees are prone to overfitting and a random forest is usually preferred to increase generalizability by training each tree using a subsample of observations and a subset of the features. Random forests output the class that is the mode of the classes (classification) or mean prediction (regression) over the ensemble.

### Cross Validation

We have a very small data set and so the traditional split of the data into *training*, *validation*, and *test* sets is not recommended. We always should create a *test* set and I have done so using a 70/30 split of the original data. Instead of additionally splitting the remaining 70% into both training and validation sets, however, we invoke the use of k-fold cross validation, which is illustrated in Fig. 4 (taken from <https://www.kaggle.com/dansbecker/cross-validation>). The basic idea is to take advantage of *all* of the remaining data by running experiments with different holdout sets for validation and keeping the rest for training. Running all of the experiments ensures that you have used all of the data at some point to validate your models. Generally, cross-validation gives a more accurate measure of model quality but comes at the cost of larger/high computational burden. Since our data set is quite small, the computational burden is low and we can ignore this detriment.

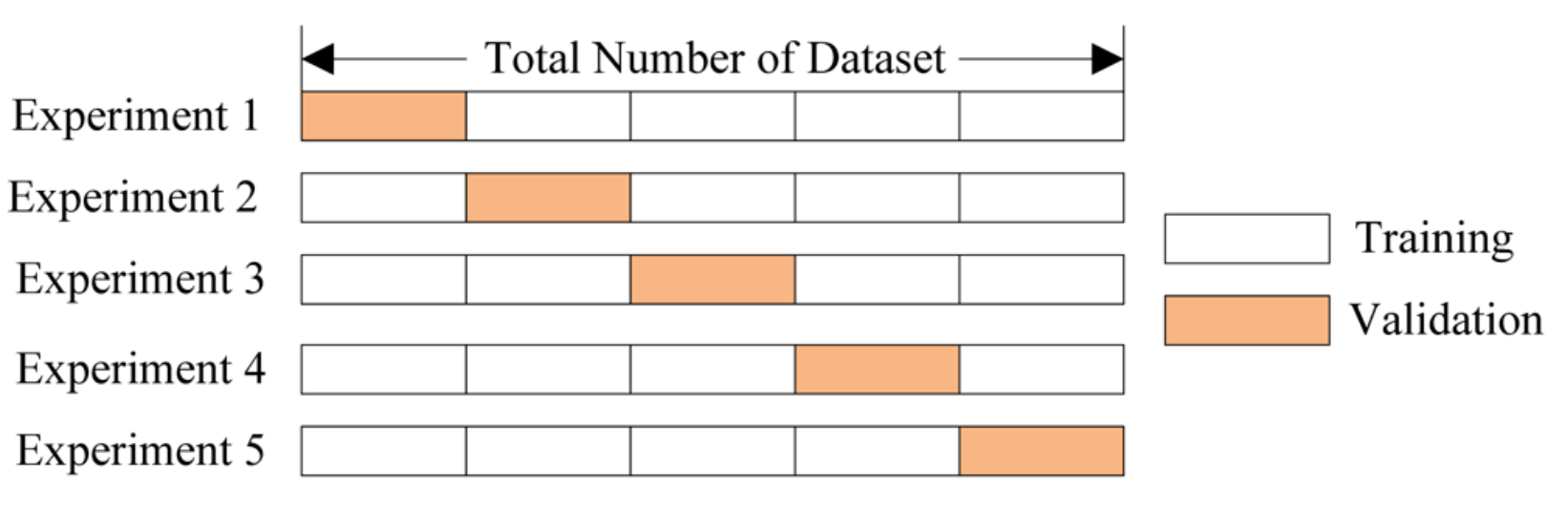


Figure 4: k-fold partitioning of data for cross validation

Note that we still have a class imbalance to consider given that we our training and test sets are stratified by class:



### Class Imbalance

The imbalance in the training data was addressed using the five methods: *none*, *under-sampling*, *oversampling*, *ROSE*, and *SMOTE*. These are easily configured in the **caret** package via an optional **sampling** parameter in the trainControl functional. Each model was run using a 10-fold cross validation repeated 5 times and the following performance metrics gathered for each model: *sensitivity*, *specificity*, *precision*, *recall*, *F1*, and *balanced* *accuracy*. The repeats ensure that the data is split in different ways.

## Model Performance

Figure 4 shows a summary of the model performances faceted by sampling mode (used to counter class imbalance) and V9 SNR value (intended to make the models more robust to noise). While the results of the SNR=NA (original data) row look stellar, I would suggest that they should not to be trusted given the likely lack of model generalizability. I have more confidence in the generalizability of models where we added noise. Of particular interest is the {rose, SNR=1} result, which demonstrates an *overall* better performance relative to other combinations at the same noise level. The random forest model does quite a bit better than logistic regression in the {rose, SNR=1} group and would be the model I would choose as the “optimal” model for scoring new data, relative to the other models tested.



Figure 5: Model performance summary

# Conclusions

In this exercise, I addressed many potential pitfalls with the following solutions:

* **Missing data**: used imputation to fill missing gaps
* **Class imbalance**: explored the efficacy of various sampling techniques to offset class imbalance
* **Small data set**: invoked k-fold cross validation to utilize all of the remaining non-test data
* **Generalizability**: added noise to V9 data so that the models are exposed to such effects potentially seen in out-of-sample data.
* **Overfitting**: models rarely (if ever) are capable of obtaining *perfect* performance in the real-world. Using random forest and adding noise to features helps to offset overfitting.