Predictive Modeling

In people analytics, inferential models like those covered in chapters "Linear Regression", "Linear Model Extensions", and "Logistic Regression" are generally warranted by the research objectives. However, there are times when we need to go beyond interpreting coefficients to understand relative influences of predictors on an outcome and leverage the models to estimate or *predict* the most likely future values. This type of modeling is often referred to as **predictive analytics** and is the subject of this chapter.

A branch of **Artificial Intelligence (AI)** known as **Machine Learning (ML)** is often associated with predictive modeling. ML is a set of methods that aim to improve performance on a set of tasks by learning from data (Mitchell, 1997). ML applications can be found in medical diagnostics, autonomous vehicles, speech recognition, automated securities trading, lending decisions, marketing, and many other domains. The difference between statistics and ML is largely philosophical. Logistic regression, for example, is covered in both statistics and ML textbooks. While statistics focuses more on modeling and ML is more algorithmic, both can be used for prediction. The broader field of **data science** often further confounds distinctions between these disciplines, though data science represents the entire end-to-end process—from data extraction and engineering to modeling and analysis.

A good use case for a predictive model in people analytics is data restatement to adjust for reorganizations over time. In this case, accuracy may be more important than the explainability of the model. A predictive model may be used to predict and assign a current functional executive to historical records to support leader-wise trending analyses. For example, consider a scenario in which the current VP of Product Marketing was hired six months ago to replace the former VP of Product Marketing who was in the role for the prior five-year period. If the new VP wants to see monthly termination counts for their organization over the past three years, term records prior to the VP's start date need to be associated with the *current*—rather than *former*—VP of Product Marketing to accomplish this. A model can be trained to learn from patterns in the combinations of current workers' department, leader,

https://doi.org/10.1007/978-3-031-28674-2_13

and job attributes that can be used to assign current executives to past data (e.g., historical termination events, month-end worker snapshots).

It is important to note that while there are AI/ML applications for people analytics, feeding data to black box models without an understanding of how the underlying algorithms work is generally a bad idea. Despite the glamour predictive analytics has seen in recent years due to the allure of a magical elixir that can portend the future, more times than not inferential statistical approaches are more appropriate in a people analytics setting (Fig. 1). Unfortunately, the hype has given rise to AI snake oil to justify a premium price point for modern HR tech solutions, which are often little more than the descriptive statistics covered in chapter "Descriptive Statistics".

It is both a blessing and a curse that a predictive model can be built with a single line of code in R, and it is dangerous to blindly make recommendations on the basis of the output. A simpler model that you can understand and explain is generally a better option than a more complex one that you cannot. There is a high probability that stakeholders will ask deeper questions about *why* a particular segment of the workforce is predicted to exit at higher rates, for example, and answering these questions requires a deeper understanding of the factors that led to them being classified as such.

People data are messy, and understanding why people vary in attitudes, perceptions, and behaviors is an inherently difficult endeavor. Spoiler alert: there is no crystal ball that will soften this reality.

Cross-Validation

Predictive modeling involves training a model on a data set referred to as a **training set** and using the model to make predictions for observations on a separate data set known as the **test set** or **validation set** to evaluate model performance.

A model is blind to data in the test set, since only the data in the training set are used to *train* the model. Therefore, the test set provides a convenient way to compare the actual known values to the predicted values and estimate how well the model will generalize to other data. Evaluating model performance on the basis of

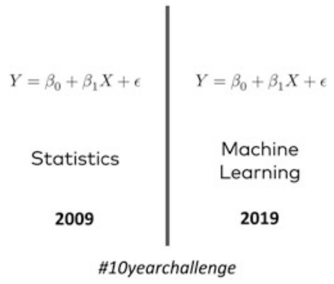

$Y = \beta_0 + \beta_1 X + \epsilon$
Statistics
2009
$Y = \beta_0 + \beta_1 X + \epsilon$
Machine Learning
2019
#10yearchallenge

Fig. 1 Satirical illustration based on the viral meme of 2019 which depicts the enduring popularity and utility of linear regression, even in light of the billions companies invest in ML each year

training data would be akin to having students take an exam after providing them the answers. Strong performance on the training data is almost a certainty, so the value of a model rests on its performance on data not used to build it.

This partitioning procedure is known as **cross-validation (CV)**. While there are many methods of splitting data into training and test sets, a common feature among all is that the partitioning strategy is random. Without randomization, the model may learn patterns characteristic of the training set that result in inaccurate predictions for other data which do not feature consistent patterning.

This section will explore a few of the most common CV methods.

Validation Set Approach

The **validation set approach** is the most basic form of CV. This approach involves defining proportions by which to partition data into training and test sets—usually 2/3 and 1/3, respectively. The model is trained on the training set and then differences between actual y and predicted $\hat{y}$ values are calculated on the test set to evaluate model performance.

Leave-One-Out

Leave-one-out CV fits a model using $n - 1$ observations n times. The test error is then evaluated by calculating differences between actual y and predicted $\hat{y}$ values for all omitted observations.

k-Fold

k-fold CV randomly partitions observations into k groups, or *folds*, that are approximately equal in size. The first fold is treated as the test set, and the model is trained on the remaining $k - 1$ folds. This procedure is repeated k times, each with a different set of observations (*fold*) as the test set. The test error is then evaluated by calculating differences between actual y and predicted $\hat{y}$ values across all test sets.

Model Performance

There are several methods of quantifying how well models perform on test data in order to assess the extent to which the model will generalize. Predictive modeling applications will be categorized as either *classification* or *forecasting* to reflect the families of use cases germane to people analytics.

Classification

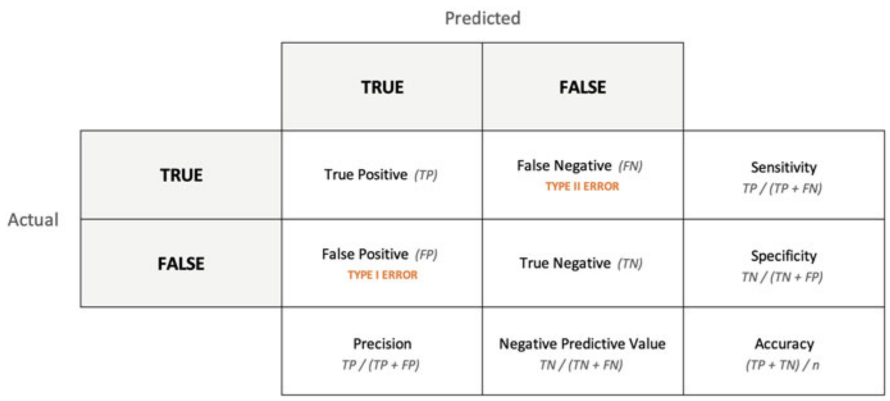
In a classification setting, predictions are either right or wrong. Therefore, calculating the overall error rate across test data is straightforward:

$$\frac{1}{n}\sum_{i=1}^{n} I\left(y_i \neq \hat{y}_i\right),$$

where I is an indicator variable equal to 1 if $y_i \neq \hat{y}_i$ and 0 if $y_i = \hat{y}_i$.

A **confusion matrix** is often used in classification to parse the overall model accuracy rate into component parts and understand whether the model performs at a level appropriate to a defined tolerance level per the research objective. In an attrition project, it may be more important to correctly predict high performers who leave than to correctly predict those who stay, as prediction errors for the former are likely far more costly. Several performance metrics are provided by the confusion matrix to aid in a more granular understanding of model performance, which are represented in Fig. 2:

- **True Positive**: Number of correct true predictions
- **True Negative**: Number of correct false predictions
- **False Positive**: Number of incorrect true predictions (Type 1 error)
- **False Negative**: Number of incorrect false predictions (Type 2 error)
- **Accuracy**: Rate of correct predictions overall
- **Sensitivity**: Rate of actual true cases predicted correctly (also known as **Recall**)



		Predicted		
		TRUE	FALSE	
Actual	TRUE	True Positive *(TP)*	False Negative *(FN)* TYPE II ERROR	Sensitivity *TP / (TP + FN)*
	FALSE	False Positive *(FP)* TYPE I ERROR	True Negative *(TN)*	Specificity *TN / (TN + FP)*
		Precision *TP / (TP + FP)*	Negative Predictive Value *TN / (TN + FN)*	Accuracy *(TP + TN) / n*

Fig. 2 Confusion matrix

- **Specificity**: Rate of actual false cases predicted correctly
- **Precision**: Rate of correct predictions among all cases predicted true
- **Negative Predictive Value**: Rate of correct predictions among all cases predicted false

Forecasting

While predictions are either right or wrong in a classification context, evaluating model performance in a forecasting context involves assessing the *magnitude* of differences between actual y and predicted $\hat{y}$ values—usually across time periods. There are many methods for assessing forecasting model performance, and we will focus on some of the most common.

- Mean absolute deviation (MAD): average absolute difference between actual y and predicted $\hat{y}$ values

$$MAD = \frac{\sum |y_i - \hat{y}_i|}{n}$$

- Mean square error (MSE): average squared difference between actual y and predicted $\hat{y}$ values

$$MSE = \frac{\sum (y_i - \hat{y}_i)^2}{n}$$

  Squaring differences accomplishes two key objectives: (1) converts negative differences to positive (consistent with the MAD approach) and (2) imposes a greater penalty on larger differences, which causes error rates to increase at an exponential rather than linear rate (e.g., $2^2 = 4$, $3^2 = 9$, $4^2 = 16$). MSE is perhaps the most pervasive model performance measure in predictive modeling.

- Mean absolute percentage error (MAPE): average absolute difference expressed as a percentage

$$MAPE = \left(\frac{100}{n}\right) \sum \left| \frac{y_i - \hat{y}_i}{y_i} \right|$$

Bias–Variance Tradeoff

Bias–variance tradeoff refers to the important endeavor of minimizing two sources of error that prevent models from generalizing beyond their training data: *bias* and *variance*.

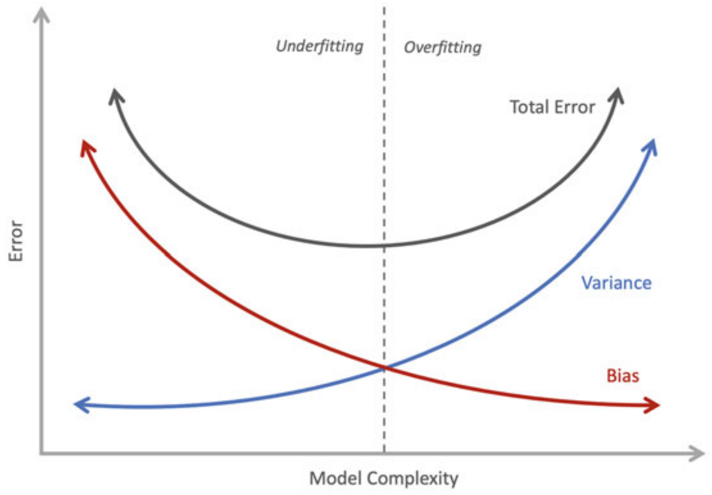


Fig. 3 Bias–variance tradeoff. Dashed line represents optimal model performance

- **Bias**: Error from erroneous assumptions in the model. High bias results from models that are too simplistic to accurately capture the relationships between predictors and the outcome; this is known as **underfitting**.
- **Variance**: Error from sensitivity to small fluctuations in training data. High variance results from models that capture random noise rather than the significant patterns in the training data; this is known as **overfitting**.

As a general rule, the more flexible the model, the more variance and less bias. As shown in Fig. 3, minimizing test error by achieving a model with optimal fit to the data requires limiting both bias and variance.

Tree-Based Algorithms

While there are many flexible ML algorithms, such as **Extreme Gradient Boosting (XGBoost)**, **Artificial Neural Networks (ANN)**, and **Support Vector Machines (SVM)**, that tend to perform well across a range of prediction problems, these will not be covered as we will focus on more *interpretable* tree-based algorithms that have more applications to people analytics.

Decision Trees

In addition to the inferential models covered in the previous chapters, **decision trees** are also excellent tools that lend to simple and effective narratives about factors

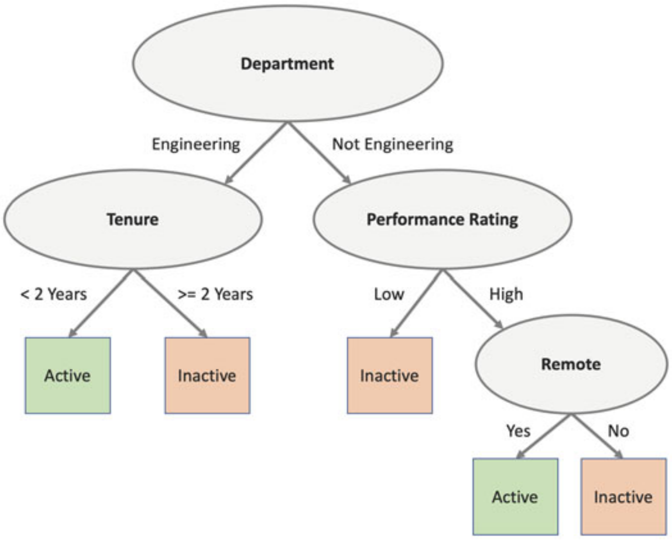


Fig. 4 Conceptual decision tree for employee attrition prediction

influencing outcomes in either a regression or a classification setting. As illustrated in Fig. 4, decision trees resemble a tree that depicts a set of decisions as well as consequences of those decisions. The top-level `Department` variable is known as a root node, and the remaining `Tenure`, `Performance Rating`, and `Remote` nodes are known as interior nodes. Decisions represented in `Active` and `Inactive` boxes are referred to as leaf, terminal, or end nodes.

As evidenced by the inactive status prediction in the leaf node, this decision tree shows that employees in the Engineering department are unlikely to stick around for two or more years. In addition, employees in other departments terminate if they are low performers or if they are high performers who do not work remotely. It is important to note that in practice, it is rare to achieve complete purity in leaf nodes, as there is usually a mix of results in a given node—though a more frequent class or range of values is expected in the presence of meaningful variables. If leaf nodes for a classification problem are comprised of a single class, it may be evidence of overfitting, especially if the n-count is small.

Predictions based on a deep tree with an excessive number of partitions and few observations will likely be highly inaccurate beyond the training data. The goal of decision trees is to arrive at a set of decisions that best delineate one class or range of values from others by identifying patterns and natural cutpoints among a reasonably large subset of the data at each level. There must be signal in the features used to partition the data such that predictions are an improvement over random guessing (e.g., 50/50 chance in a binary classification setting).

Random Forests

A **random forest (RF)** is a natural extension of the decision tree. As the name implies, a random forest is a large number (forest) of individual decision trees that operate as an ensemble. The process of fitting multiple models on different subsets of training data and then combining predictions across all models is referred to as **bagging**. This is a case of *wisdom of the crowd* decision-making in which a large number of uncorrelated trees (models) functioning as a committee should outperform individual trees.

To understand the mechanics of a random forest, consider an investment strategy in which you diversify an investment portfolio by spreading investments across different assets. By investing in assets that are uncorrelated, there is a lower likelihood that the portfolio's value will be negatively impacted by a negative event impacting a single holding. In the same way, a random forest constructs an ensemble of trees that are each based on different randomized subsets of data and combinations of features to amalgamate the information and arrive at more accurate predictions. The potential for poor performance from a single tree is mitigated by the many trees working in concert with one another.

Though random forests combine information from many decision trees, there are still intuitive ways of understanding which features are most important in segmenting employees to understand drivers of various outcomes.

Predictive Modeling

We will now integrate these concepts into attrition classification and forecasting examples. The high-level prediction workflow will follow four steps:

- **Step 1**: Partition data into training and test sets for cross-validation.
- **Step 2**: Build models using training data.
- **Step 3**: Use models to make predictions on test data.
- **Step 4**: Evaluate model performance.

Classification

To demonstrate the prediction workflow steps for classification, we will leverage our `employees` data set:

```
# Load library
library(peopleanalytics)

# Load data
```

```r
data("employees")

# Load employee data
prediction_dat <- employees

# One-hot encode active outcome variable, setting inactives to
↪  1 and actives to 0
prediction_dat$active <- ifelse(prediction_dat$active == 'No',
↪  1, 0)
```

Step 1: Partition Data into Training and Test Sets for Cross-Validation
For this example, we will implement the validation set approach for CV.

```r
# Load library
library(dplyr)

# Set seed for reproducible training and test sets
set.seed(9876)

# Randomly select 2/3 of employees for the training set
training_ids <- sample(prediction_dat$employee_id, size =
↪  nrow(prediction_dat) * 2/3, replace = FALSE)

# Create training data
training_dat <- prediction_dat |> dplyr::filter(employee_id
↪  %in% training_ids)

# Create test data using remaining 1/3 of observations
test_dat <- prediction_dat |> dplyr::filter(!employee_id %in%
↪  training_ids)

# Return Boolean to validate that all observations are
↪  accounted for
nrow(training_dat) + nrow(test_dat) == nrow(prediction_dat)
```

```
## [1] TRUE
```

Step 2: Build Models Using Training Data
Machine learning (ML) algorithms are sensitive to imbalanced classes. That is, when there is not an equal representation of each class we wish to predict in the data (e.g., employees who left and employees who did not), it can adversely impact our results. In the case of our `employees` data set, there are 1233 observations for active employees but only 237 observations for inactive employees. Therefore, we will introduce a popular technique to address this known as **Synthetic Minority**

Oversampling Technique (SMOTE), which will be important for the RF model we will train. This technique takes random samples with replacement (i.e., each observation may be chosen more than once) from the minority class to augment the class's representation in the data set and achieve balanced classes.

While functions exist for implementing SMOTE, such as the `SMOTE()` function from the `DMwR` library, in the spirit of demystifying black box ML approaches we will step through this procedure without the use of an available function:

```r
# Calculate class representation delta in training data
training_class_delta <- nrow(training_dat |>
↪  dplyr::filter(active == 0)) - nrow(training_dat |>
↪  dplyr::filter(active == 1))

# Copy training data to separate data frame for oversampling
training_dat_os <- training_dat

# Set seed for reproducible results
set.seed(9876)

# Oversample the underrepresented inactive class by
↪  training_class_delta to align observation counts with
↪  active class
# Note: A loop is not the most efficient -- especially with
↪  large data sets -- but it is leveraged here to simplify
↪  instruction on SMOTE mechanics
for (i in 1:training_class_delta){

  # Sample employee id from underrepresented class
  oversampled_id <-
↪  sample(training_dat_os[training_dat_os$active == 1,
↪  'employee_id'], size = 1, replace = TRUE)

  # Store observation for sampled employee id
  new_obs <- unique(training_dat_os |>
↪  dplyr::filter(employee_id == oversampled_id))

  # Append sampled observation to training data frame
  training_dat_os <- rbind(training_dat_os, new_obs)
}

# Return Boolean to validate that classes are equal in the
↪  training data
nrow(training_dat_os |> dplyr::filter(active == 0)) ==
↪  nrow(training_dat_os |> dplyr::filter(active == 1))
```

```
## [1] TRUE
```

Next, we will fit a binomial logistic regression model using a subset of predictors from the oversampled training data. For comparison, let us summarize a model using original and oversampled data.

```
##
## Call:
## glm(formula = active ~ overtime + job_lvl + engagement + interview_rating,
##     family = binomial, data = training_dat)
##
## Deviance Residuals:
##      Min        1Q    Median        3Q       Max
## -2.78408  -0.00157   0.00000   0.00000   2.54959
##
## Coefficients:
##                  Estimate Std. Error z value Pr(>|z|)
## (Intercept)       85.7594    16.4594   5.210 1.88e-07 ***
## overtimeYes        2.5156     0.8766   2.870  0.00411 **
## job_lvl           -0.1372     0.4805  -0.285  0.77528
## engagement        -0.6845     0.6304  -1.086  0.27755
## interview_rating -24.9181     4.7673  -5.227 1.72e-07 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 869.013  on 979  degrees of freedom
## Residual deviance:  44.466  on 975  degrees of freedom
## AIC: 54.466
##
## Number of Fisher Scoring iterations: 11
```

```
##
## Call:
## glm(formula = active ~ overtime + job_lvl + engagement + interview_rating,
##     family = binomial, data = training_dat_os)
##
## Deviance Residuals:
##    Min      1Q  Median      3Q     Max
## -3.801   0.000   0.000   0.000   2.704
##
## Coefficients:
##                   Estimate Std. Error z value Pr(>|z|)
## (Intercept)      131.12905   19.56650   6.702 2.06e-11 ***
## overtimeYes        2.27109    0.67614   3.359 0.000782 ***
## job_lvl            0.04793    0.36730   0.130 0.896172
## engagement        -0.41182    0.42738  -0.964 0.335252
## interview_rating -37.87539    5.64636  -6.708 1.97e-11 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 2276.295  on 1641  degrees of freedom
## Residual deviance:   88.655  on 1637  degrees of freedom
## AIC: 98.655
##
## Number of Fisher Scoring iterations: 12
```

As we can see, results of the two binomial logistic regression models are consistent in the sense that only `overtime` and `interview_rating` emerge as significant in classifying employees into active and inactive classes. Since the oversampled data have a larger n-count, there is greater power to detect effects, and we see this reflected in the larger coefficients and lower standard errors.

Using a similar syntax, we can fit an RF using the `randomForest()` function from the package by the same name:

```r
# Load library
library(randomForest)

# Train RF model using original data
rf.fit <- randomForest::randomForest(active ~ overtime +
↪  job_lvl + engagement + interview_rating, data =
↪  training_dat)

# Train RF model using oversampled data
rf.os.fit <- randomForest::randomForest(active ~ overtime +
↪  job_lvl + engagement + interview_rating, data =
↪  training_dat_os)
```

While RFs generally offer a significant lift in performance beyond a single decision tree, we could also apply *tuning wrappers* around the `randomForest()` function to tune the model's hyperparameters. Experimenting with a range of values for parameters, such as `mtry` for the number of variables to randomly sample and `ntree` for the number of trees to grow, may further improve model performance. Hyperparameter tuning is beyond the scope of this book, but Kuhn & Johnson (2013) is an excellent resource for a more exhaustive treatment on ML models.

Step 3: Use Models to Make Predictions on Test Data

While there are packages in R which provide performance metrics for predictive models, we will create a function for greater visibility into how each metric is calculated:

```r
# Develop function that returns a data frame of classification
↪  model performance statistics
classifier.perf <- function(actual, predicted){

  # Check for missing values; metrics will be computed on
  ↪  non-missing values only
  predicted <- predicted[!is.na(actual)]
  actual <- actual[!is.na(actual)]
  actual <- actual[!is.na(predicted)]
  # Produce counts for model performance metrics
  TP <- sum(actual == 1 & predicted == 1) # true positives
  TN <- sum(actual == 0 & predicted == 0) # true negatives
  FP <- sum(actual == 0 & predicted == 1) # false positives
  FN <- sum(actual == 1 & predicted == 0) # false negatives
  P <- TP + FN # total positives
  N <- FP + TN # total negatives

  # Store rates to variables
  accuracy <- signif(100 * (sum(actual == predicted) /
↪   length(actual)), 3)
  sensitivity <- signif(100 * (TP / (TP + FN)), 3)
  specificity <- signif(100 * (TN / (TN + FP)), 3)
  precision <- signif(100 * (TP / (TP + FP)), 3)
  neg_pred_val <- signif(100 * (TN / (TN + FN)), 3)

  # Format output
  stat_nm <- c("accuracy", "sensitivity", "specificity",
↪   "precision", "neg_pred_val")
  stat_vl <- c(accuracy, sensitivity, specificity, precision,
↪   neg_pred_val)

  # Return model performance statistics in a data frame
  return(data.frame(stat_nm, stat_vl))

}
```

We can use the `predict()` function in conjunction with the object holding the trained model to predict class values for our test data. For classification, we need to define a probability threshold for classifying observations into classes. This is an important consideration since we want to avoid investing in retention strategies for employees who are not actually going to leave (minimizing false positives), while ensuring employees who are truly at risk are flagged as such (maximizing true positives).

We will predict the class using both binomial logistic regression and RF models, trained on both balanced (SMOTE) and imbalanced class data, and store performance metrics in a single data frame for easy comparison:

```
# Initialize empty data frame for model performance stats
class.perf.metrics <- NULL

# Set probability threshold for classification
prob_threshold <- .7
```

```
# Predict with logistic regression model
class.perf.metrics <- rbind(class.perf.metrics,
↪  cbind.data.frame(
                           model = rep("GLM", nrow(test_dat)),
                           classifier.perf(
                           actual = test_dat$active,
                           predicted = ifelse(predict(glm.fit,
                            ↪  test_dat, type = "response") >=
                            ↪  prob_threshold, 1, 0))))

# Predict with logistic regression model (SMOTE)
class.perf.metrics <- rbind(class.perf.metrics,
↪  cbind.data.frame(
                           model = rep("GLM (SMOTE)",
                            ↪  nrow(test_dat)),
                           classifier.perf(
                           actual = test_dat$active,
                           predicted = ifelse(predict(glm.os.fit,
                            ↪  test_dat, type = "response") >=
                            ↪  prob_threshold, 1, 0))))

# Predict with RF model
class.perf.metrics <- rbind(class.perf.metrics,
↪  cbind.data.frame(
                           model = rep("RF", nrow(test_dat)),
                           classifier.perf(
                           actual = test_dat$active,
                           predicted = ifelse(predict(rf.fit,
                            ↪  test_dat, type = "response") >=
                            ↪  prob_threshold, 1, 0))))

# Predict with RF model (SMOTE)
class.perf.metrics <- rbind(class.perf.metrics,
↪  cbind.data.frame(
                           model = rep("RF (SMOTE)",
                            ↪  nrow(test_dat)),
                           classifier.perf(
                           actual = test_dat$active,
                           predicted = ifelse(predict(rf.os.fit,
                            ↪  test_dat, type = "response") >=
                            ↪  prob_threshold, 1, 0))))
```

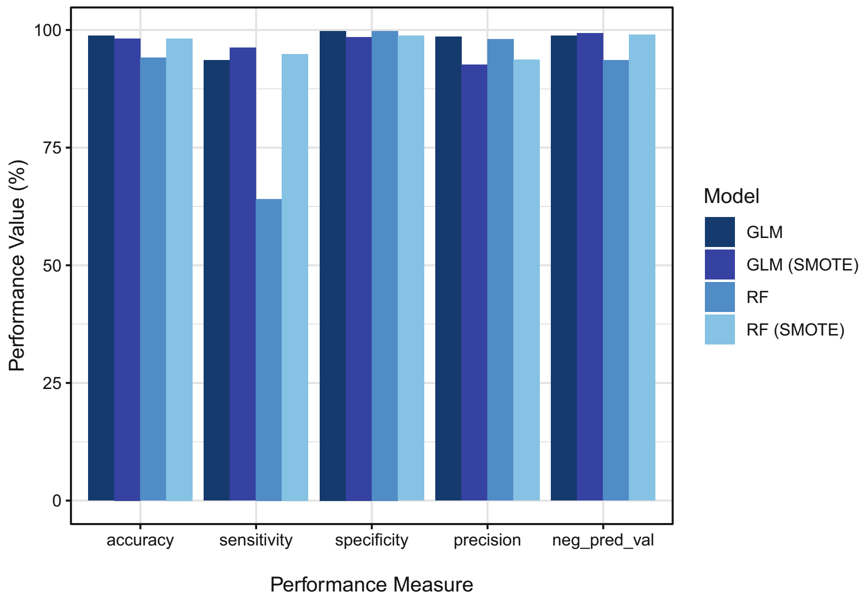


Fig. 5 Classification performance for binomial logistic regression and RF models using balanced (SMOTE) and imbalanced classes

Step 4: Evaluate Model Performance

As we can see in Fig. 5, with our stringent probability threshold set at `.7` for class delineation, all models had perfect specificity (correctly predicting those who stay) and precision (all employees who were predicted to attrit did). However, we see notable differences in sensitivity across the model types, and this is generally a very important performance measure in a predictive attrition project since the cost of not flagging employees who attrit can be costly. Sensitivity for the RF model trained on balanced classes (SMOTE) performed much better than its imbalanced RF counterpart (`95.6%` vs. `71.6%`), reinforcing that ML models are sensitive (no pun intended) to imbalanced classes.

The results also show that there is likely no benefit to compromising model interpretability by using a flexible ML model like RF since our trusty binomial logistic regression model performs just as well on these data. Nevertheless, we can construct what is known as a **Variable Importance Plot** on RF output using the `varImpPlot()` function from the `randomForest` library to understand the relative importance of each predictor in the model.

Variable importance plots are based on the mean decrease in **Gini importance**. Gini importance measures the average gain of purity by splits of a given variable. In other words, if a variable is useful, it tends to split mixed labeled nodes (nodes with both employees who separated and employees who stayed) into pure single class nodes. The most important variables are at the top of the variable importance

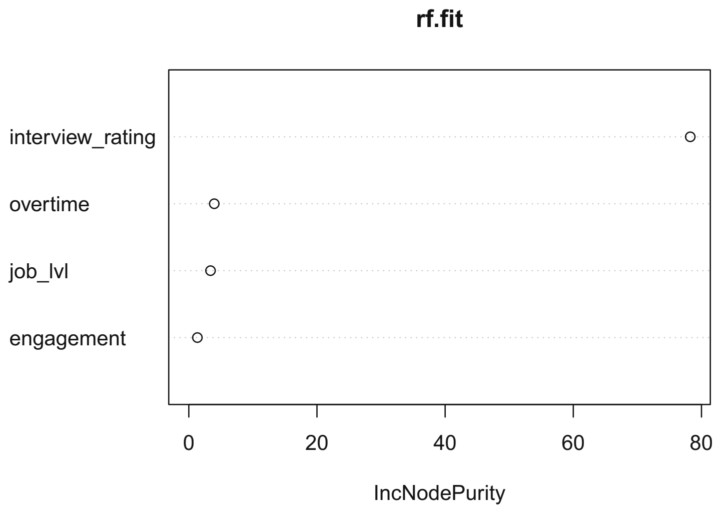


Fig. 6 Variable importance plot for Random Forest model

plot, indicating that without these variables nodes will not be as pure and as a result, classification performance will not be as strong.

Figure 6 shows that `interview_rating` is far more important than the second most important predictor, `overtime`. This is consistent with what we observed in the results of the binomial logistic regression models too.

Let us compare the information from the RF's Variable Importance Plot to a single decision tree built on training data with balanced classes. We can build and visualize a decision tree in R using the `rpart()` and `rpart.plot()` functions from libraries by the same names. `rpart` is an acronym for *recursive partitioning and regression trees*:

```
# Load libraries
library(rpart)
library(rpart.plot)

# Construct decision tree on balanced training data
tree <- rpart::rpart(active ~ overtime + job_lvl +  engagement
↪   + interview_rating, data = training_dat_os, method =
↪   "class")

# Visualize decision tree
rpart.plot::rpart.plot(tree)
```

As shown in Fig. 7, the most important predictor for splitting the data is `interview_rating` when using a single decision tree. At the root node, there is a 50% chance of leaving and staying, which is expected given we balanced the classes

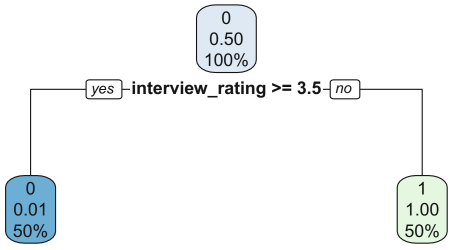


Fig. 7 Decision tree for employee attrition prediction using training data with balanced classes (SMOTE)

using SMOTE. A prediction that is no better than a fair coin toss is of course not helpful. Walking down to the leaf nodes, we can see that for the 50% of employees with `interview_rating >= 3.5`, the algorithm predicts they will stay (`status = 0`); for the 50% of employees with `interview_rating < 3.5`, the algorithm predicts they will leave (`status = 1`). Given what we observed in the results of the binomial logistic regression output, additional partitioning by `overtime` may further increase node purity on the test data since classes are mixed for those with interview ratings between `3.3` and `3.6`. However, given the strong performance splitting data only on `interview_rating`, further partitioning will likely result in modeling noise and overfitting.

Forecasting

To demonstrate the prediction workflow steps for forecasting, we will leverage our `turnover_trends` data set:

```
# Store forecasting data
forecasting_dat <- turnover_trends
```

Step 1: Partition Data into Training and Test Sets for Cross-Validation
Since we have 60 months of data for each combination of values for the `job`, `level`, and `remote` variables, we will select a combination for which `turnover_rate` can be projected for future months. To simplify, we will train a model using data for the first 48 months and test using the final 12 months.

```
# Create training data
train_dat <- forecasting_dat |> dplyr::filter(job == 'People
↪  Scientist' & level == 1 & year %in% 1:4)

# Create test data
test_dat <- forecasting_dat |> dplyr::filter(job == 'People
↪  Scientist' & level == 1 & remote == 'Yes' & year == 5)
```

Step 2: Build Models Using Training Data

Given the significant quadratic and cubic terms identified in chapter "Linear Model Extensions", we will fit a cubic regression model on the training data.

```
# Fit cubic model
train.cube.fit <- lm(turnover_rate ~ year + month + I(month^2)
↪  + I(month^3) + remote, data = train_dat)

# Produce model summary
summary(train.cube.fit)
```

```
##
## Call:
## lm(formula = turnover_rate ~ year + month + I(month^2) + I(month^3) +
##     remote, data = train_dat)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -0.3360 -0.1605  0.0075  0.1680  0.3210
##
## Coefficients:
##               Estimate Std. Error t value Pr(>|t|)
## (Intercept)  2.7650000  0.1594281  17.343  < 2e-16 ***
## year        -0.1130000  0.0230955  -4.893 4.33e-06 ***
## month        2.4100000  0.0935806  25.753  < 2e-16 ***
## I(month^2)  -0.4100000  0.0163907 -25.014  < 2e-16 ***
## I(month^3)   0.0200000  0.0008311  24.064  < 2e-16 ***
## remoteYes   -1.6400000  0.0516430 -31.756  < 2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.253 on 90 degrees of freedom
## Multiple R-squared:  0.9499, Adjusted R-squared:  0.9471
## F-statistic: 341.4 on 5 and 90 DF,  p-value: < 2.2e-16
```

All linear, quadratic, and cubic terms on `month` are statistically significant at the $p < 0.001$ level.

Step 3: Use Models to Make Predictions on Test Data

We will again build a function to evaluate the performance of the fitted models applied to test data rather than using delivered functions. The following function will return *mean absolute deviation (MAD)*, *mean squared error (MSE)*, and *mean absolute percentage error (MAPE)*:

```
# Develop function that returns a data frame of forecasting
↪  model performance statistics
forecast.perf <- function(actual, predicted){

  # Check for missing values; metrics will be computed on
  ↪  non-missing values only
  predicted <- predicted[!is.na(actual)]
  actual <- actual[!is.na(actual)]
  actual <- actual[!is.na(predicted)]

  # Store rates to variables
  mad <- round(mean(abs(actual - predicted)), 2)
  mse <- round(mean((actual - predicted)^2), 2)
  mape <- round(mean(abs((actual - predicted) / actual)) *
↪  100, 2)

  # Return model performance statistics in a data frame
  return(data.frame(mad, mse, mape))

}
```

For this forecast, we will also produce a prediction interval. A **prediction interval** is a range of values that is likely to contain the outcome value for a single new observation given a set of predictors. For example, a 95% prediction interval of `[10 15]` indicates that we can be 95% confident that the observation for which a prediction is being made will have an actual outcome value between `10` and `15`. Note that this is very different from a confidence interval in inferential statistics, which is a range of values that likely contains the value of an unknown population parameter.

We can produce a prediction interval by passing an additional `interval = 'predict'` argument into the `predict()` function:

```
# Initialize empty data frames for model predictions and performance
↪  stats
forecast.metrics <- NULL
forecast.err.rates <- NULL

# Predict on test_dat
forecast.metrics <- rbind(forecast.metrics, cbind.data.frame(
                          month = test_dat$month,
                          actual = test_dat$turnover_rate,
                          predicted = predict(train.cube.fit,
                          ↪  test_dat, type = "response"),
                          lwr_bound =
                          ↪  as.data.frame(predict(train.cube.fit,
                          ↪  test_dat, type = "response", interval =
                          ↪  "predict"))$lwr,
                          upr_bound =
                          ↪  as.data.frame(predict(train.cube.fit,
                          ↪  test_dat, type = "response", interval =
                          ↪  "predict"))$upr))
```

Step 4: Evaluate Model Performance

Next, we can pass a vector of actual and corresponding predicted values into the `forecast.perf()` function to return MAD, MSE, and MAPE performance metrics for the fitted model applied to year 5 data.

```
# Calculate error rates for year 5 forecast
forecast.perf(actual = forecast.metrics$actual, predicted =
↪  forecast.metrics$predicted)
```

```
##    mad   mse  mape
## 1 3.84 14.75 48.39
```

Given 95% of the variance in turnover rates was explained by the cubic regression model fitted to our training data ($R^2 = 0.95$), these error rates are surprisingly high for the test data.

Evaluating the average turnover rate by year will help in reconciling the high R^2 on the training data with the high error rates on the test data:

```
# Calculate year-wise turnover rate mean
yr1_avg <- train_dat |> dplyr::filter(remote == 'Yes' & year
↪  == 1) |> dplyr::summarize(Mean = mean(turnover_rate))
yr2_avg <- train_dat |> dplyr::filter(remote == 'Yes' & year
↪  == 2) |> dplyr::summarize(Mean = mean(turnover_rate))
yr3_avg <- train_dat |> dplyr::filter(remote == 'Yes' & year
↪  == 3) |> dplyr::summarize(Mean = mean(turnover_rate))
yr4_avg <- train_dat |> dplyr::filter(remote == 'Yes' & year
↪  == 4) |> dplyr::summarize(Mean = mean(turnover_rate))
yr5_avg <- test_dat |> dplyr::summarize(Mean =
↪  mean(turnover_rate))

# Display year-wise turnover rate mean
print(c(yr1_avg, yr2_avg, yr3_avg, yr4_avg, yr5_avg))
```

```
## $Mean
## [1] 4.506667
##
## $Mean
## [1] 4.816667
##
## $Mean
## [1] 4.046667
##
```

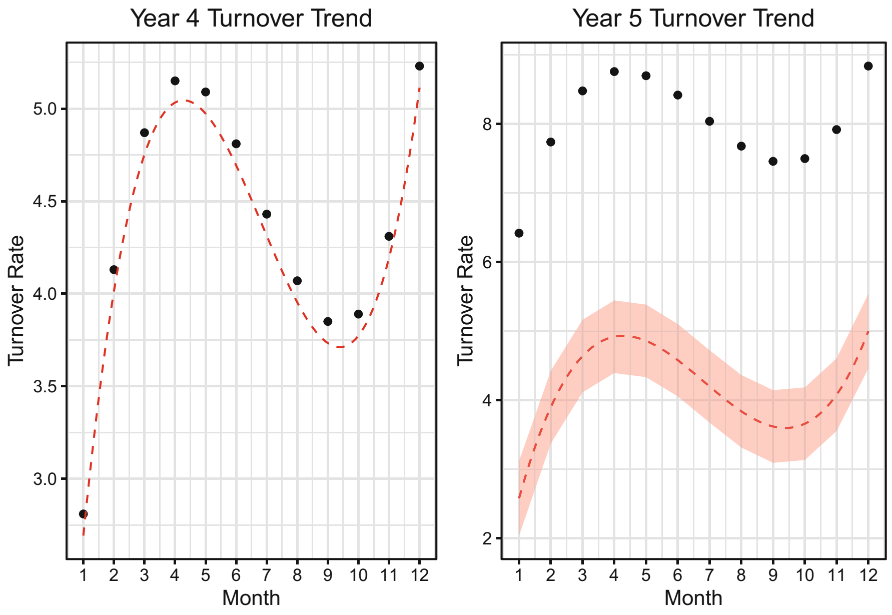


Fig. 8 Left: fitted model (red dashed line) with good fit to year 4 training data (black dots). Right: fitted model (red dashed line) with poor fit to year 5 test data (black dots). 95% prediction interval is represented by the red shaded area around the fit line

```
## $Mean
## [1] 4.386667
##
## $Mean
## [1] 7.996667
```

There is clearly a significant difference in average turnover for year 5 (test data) relative to years 1–4 (training data). Since the fitted model had no visibility into year 5 data, it did not account for the spike in turnover beyond year 4. Differences are further evidenced in Fig. 8, in which actual values for year 5 are far and away outside the 95% prediction interval.

This is an important lesson that highlights the centrality of cross-validation in evaluating whether predictive models will generalize beyond the available data. We can easily fit a model that performs well on training data and claim that the model has exceptional accuracy. However, what matters in predictive modeling is how well the model performs on data it has not seen as part of the training process.

Review Questions

1. What factors influence the balance between model interpretability and flexibility?
2. How does cross-validation (CV) help improve the performance of predictive models?
3. What is bias–variance tradeoff?
4. In a classification setting, what performance metrics are available in a confusion matrix?
5. What are some measures used to evaluate the performance of a forecast?
6. What is Synthetic Minority Oversampling Technique (SMOTE), and how does it help improve the performance of machine learning (ML) models?
7. How is the stack ranking of predictors in a variable importance plot determined?
8. How does a prediction interval differ from a confidence interval?
9. How can a prediction interval be calculated in R?
10. What does high R^2 on training data and high MSE on test data indicate about the utility of a predictive model?