Chapter 5

# Evaluating Predictive Performance

In this chapter, we discuss how the predictive performance of data mining methods can be assessed. We point out the danger of overfitting to the training data, and the need to test model performance on data that were not used in the training step. We discuss popular performance metrics. For prediction, metrics include Average Error, MAPE, and RMSE (based on the validation data). For classification tasks, metrics based on the confusion matrix include overall accuracy, specificity and sensitivity, and metrics that account for misclassification costs. We also show the relation between the choice of cutoff value and classification performance, and present the ROC curve, which is a popular chart for assessing method performance at different cutoff values. When the goal is to accurately classify the most interesting or important records, called *ranking*, rather than accurately classify the entire sample (e.g., the 10% of customers most likely to respond to an offer, or the 5% of claims most likely to be fraudulent), lift charts are used to assess performance. We also discuss the need for oversampling rare classes and how to adjust performance metrics for the oversampling. Finally, we mention the usefulness of comparing metrics based on the validation data to those based on the training data for the purpose of detecting overfitting. While some differences are expected, extreme differences can be indicative of overfitting.

## Python

In this chapter, we will use `pandas` for data handling, `statsmodels` for regression models, `scikit-learn` for performance metrics, and `matplotlib` for visualization. We will also make use of the utility functions from the Python Utilities Functions Appendix.

*Data Mining for Business Analytics: Concepts, Techniques, and Applications in Python,* First Edition.
Galit Shmueli, Peter C. Bruce, Peter Gedeck, and Nitin R. Patel
© 2020 John Wiley & Sons, Inc. Published 2020 by John Wiley & Sons, Inc.

import required functionality for this chapter

```
import math
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.metrics import accuracy_score, roc_curve, auc
import matplotlib.pylab as plt

from dmba import regressionSummary, classificationSummary
from dmba import liftChart, gainsChart
```

## 5.1 Introduction

In supervised learning, we are interested in predicting the outcome variable for new records. Three main types of outcomes of interest are:

**Predicted numerical value**: when the outcome variable is numerical (e.g., house price)

**Predicted class membership**: when the outcome variable is categorical (e.g., buyer/nonbuyer)

**Propensity**: the probability of class membership, when the outcome variable is categorical (e.g., the propensity to default)

Prediction methods are used for generating numerical predictions, while classification methods ("classifiers") are used for generating propensities and, using a cutoff value on the propensities, we can generate predicted class memberships.

A subtle distinction to keep in mind is the two distinct predictive uses of classifiers: one use, *classification*, is aimed at predicting class membership for new records. The other, *ranking*, is detecting among a set of new records the ones most likely to belong to a class of interest.

Let's now examine the approach for judging the usefulness of a prediction method used for generating numerical predictions (Section 5.2), a classifier used for classification (Section 5.3), and a classifier used for ranking (Section 5.4). In Section 5.5, we'll look at evaluating performance under the scenario of oversampling.

## 5.2 Evaluating Predictive Performance

First, let us emphasize that predictive accuracy is not the same as goodness-of-fit. Classical statistical measures of performance are aimed at finding a model that fits well to the data on which the model was trained. In data mining, we are interested in models that have high predictive accuracy when applied to *new* records. Measures such as $R^2$ and standard error of estimate are common metrics in classical regression modeling, and residual analysis is used to gauge goodness-of-fit in that situation. However, these measures do not tell us much about the ability of the model to predict new records.

For assessing prediction performance, several measures are used. In all cases, the measures are based on the validation set, which serves as a more objective ground than the training set to assess predictive accuracy. This is because records in the validation set are more similar to the future records to be predicted, in the sense that they are not used to select predictors or to estimate the model parameters. Models are trained on the training set, applied to the validation set, and measures of accuracy then use the prediction errors on that validation set.

## Naive Benchmark: The Average

The benchmark criterion in prediction is using the average outcome value (thereby ignoring all predictor information). In other words, the prediction for a new record is simply the average across the outcome values of the records in the training set ($\bar{y}$). This is sometimes called a naive benchmark. A good predictive model should outperform the benchmark criterion in terms of predictive accuracy.

## Prediction Accuracy Measures

The prediction error for record $i$ is defined as the difference between its actual outcome value and its predicted outcome value: $e_i = y_i - \hat{y}_i$. A few popular numerical measures of predictive accuracy are:

**MAE** (mean absolute error/deviation) = $\frac{1}{n}\sum_{i=1}^{n}|e_i|$. This gives the magnitude of the average absolute error.

**Mean Error** = $\frac{1}{n}\sum_{i=1}^{n}e_i$. This measure is similar to MAE except that it retains the sign of the errors, so that negative errors cancel out positive errors of the same magnitude. It therefore gives an indication of whether the predictions are on average over- or underpredicting the outcome variable.

**MPE** (mean percentage error) = $100 \times \frac{1}{n}\sum_{i=1}^{n}e_i/y_i$. This gives the percentage score of how predictions deviate from the actual values (on average), taking into account the direction of the error.

**MAPE** (mean absolute percentage error) = $100 \times \frac{1}{n}\sum_{i=1}^{n}|e_i/y_i|$. This measure gives a percentage score of how predictions deviate (on average) from the actual values.

**RMSE** (root mean squared error) = $\sqrt{\frac{1}{n}\sum_{i=1}^{n}e_i^2}$. This is similar to the standard error of estimate in linear regression, except that it is computed on the validation data rather than on the training data. It has the same units as the outcome variable.

Such measures can be used to compare models and to assess their degree of prediction accuracy. Note that all these measures are influenced by outliers. To check outlier influence, we can compute median-based measures (and compare to the above mean-based measures) or simply plot a histogram or boxplot of the errors. Plotting the prediction errors' distribution is in fact very useful and can highlight more information than the metrics alone.

To illustrate the use of predictive accuracy measures and charts of prediction error distribution, consider the error metrics and charts shown in Table 5.1 and Figure 5.1. These are the result of fitting a certain predictive model to prices of used Toyota Corolla cars. The training set includes 861 cars and the validation set includes 575 cars. Results are displayed separately for the training and validation sets. We can see from the histogram and boxplot corresponding to the validation set that most errors are in the [−2000, 2000] range.

### Comparing Training and Validation Performance

Errors that are based on the training set tell us about model fit, whereas those that are based on the validation set (called "prediction errors") measure the model's ability to predict new data (predictive performance). We expect training errors to be smaller than the validation errors (because the model was fitted using the training set), and the more complex the model, the greater the likelihood that it will *overfit* the training data (indicated by a greater difference between the training and validation errors). In an extreme case of overfitting, the training errors would be zero (perfect fit of the model to the training data), and the validation errors would be non-zero and non-negligible. For this reason, it is important to compare the error plots and metrics (RMSE, MAE, etc.) of the training and validation sets. Table 5.1 illustrates this comparison: the training set performance measures appear slightly lower (better) than those for the validation set and the charts in Figure 5.1, which reveal more than the metrics alone, show a similar distribution of errors in the the training and validation sets.

**TABLE 5.1** **PREDICTION ERROR METRICS FROM A MODEL FOR TOYOTA CAR PRICES. TRAINING AND VALIDATION**

code for accuracy measure

```
# Reduce data frame to the top 1000 rows and select columns for regression analysis
car_df = pd.read_csv('ToyotaCorolla.csv')

# create a list of predictor variables by remvoing output variables and text columns
excludeColumns = ('Price', 'Id', 'Model', 'Fuel_Type', 'Color')
predictors = [s for s in car_df.columns if s not in excludeColumns]
outcome = 'Price'

# partition data
X = car_df[predictors]
y = car_df[outcome]
train_X, valid_X, train_y, valid_y = train_test_split(X, y, test_size=0.4, random_state=1)

# train linear regression model
reg = LinearRegression()
reg.fit(train_X, train_y)

# evaluate performance
# training
regressionSummary(train_y, reg.predict(train_X))
# validation
regressionSummary(valid_y, reg.predict(valid_X))
```

**Partial output**

```
# training
Regression statistics

                      Mean Error (ME) : 0.0000
       Root Mean Squared Error (RMSE) : 1121.0606
            Mean Absolute Error (MAE) : 811.6770
          Mean Percentage Error (MPE) : -0.8630
Mean Absolute Percentage Error (MAPE) : 8.0054

# validation
Regression statistics

                      Mean Error (ME) : 97.1891
       Root Mean Squared Error (RMSE) : 1382.0352
            Mean Absolute Error (MAE) : 880.1396
          Mean Percentage Error (MPE) : 0.0138
Mean Absolute Percentage Error (MAPE) : 8.8744
```

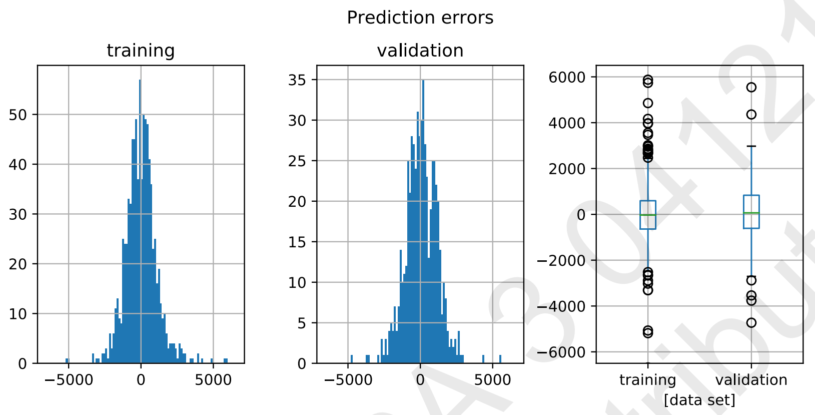


FIGURE 5.1 HISTOGRAMS AND BOXPLOTS OF TOYOTA PRICE PREDICTION ERRORS, FOR TRAINING AND VALIDATION SETS

code for creating Figure 5.1

```
pred_error_train = pd.DataFrame({
    'residual': train_y - reg.predict(train_X),
    'data set': 'training'
})
pred_error_valid = pd.DataFrame({
    'residual': valid_y - reg.predict(valid_X),
    'data set': 'validation'
})
boxdata_df = pred_error_train.append(pred_error_valid, ignore_index=True)

fig, axes = plt.subplots(nrows=1, ncols=3)
fig.set_size_inches(9, 4)
common = {'bins': 100, 'range': [-6500, 6500]}
pred_error_train.hist(ax=axes[0], **common)
pred_error_valid.hist(ax=axes[1], **common)
boxdata_df.boxplot(ax=axes[2], by='data set')

axes[0].set_title('training')
axes[1].set_title('validation')
axes[2].set_title(' ')
axes[2].set_ylim(-6500, 6500)
plt.suptitle('Prediction errors')
plt.subplots_adjust(bottom=0.1, top=0.85, wspace=0.35)
plt.show()
```

### Cumulative Gains and Lift Charts

In some applications, the goal is to search, among a set of new records, for a subset of records that gives the highest cumulative predicted values. In such cases, a graphical way to assess predictive performance is through the *cumulative gains chart* and *lift chart*. This compares the model's predictive performance to a baseline model that has no predictors[1]. Cumulative gains and lift charts for a continuous response are relevant only when we are searching for a set of records that gives the highest cumulative predicted values. Such charts are not relevant if we are interested in predicting the outcome value for *each* new record.

To illustrate this type of goal, called *ranking*, consider a car rental firm that renews its fleet regularly so that customers drive late-model cars. This entails disposing of a large quantity of used vehicles on a continuing basis. Since the firm is not primarily in the used car sales business, it tries to dispose of as much of its fleet as possible through volume sales to used car dealers. However, it is profitable to sell a limited number of cars through its own channels. Its volume deals with the used car dealers allow it flexibility to pick and choose which cars to sell in this fashion, so it would like to have a model for selecting cars for resale through its own channels. Since all cars were purchased some time ago and the deals with the used car dealers are for fixed prices (specifying a given number of cars of a certain make and model class), the cars' costs are now irrelevant and the dealer is interested only in maximizing revenue. This is done by selecting for its own resale, the cars likely to generate the most revenue. The lift chart in this case gives the predicted lift for revenue.

The cumulative gains and lift charts are based on ordering the set of records of interest (typically validation data) by their predicted value, from high to low. Then, we accumulate the actual values and plot their cumulative value (=gains) on the $y$-axis as a function of the number of records accumulated (the $x$-axis value). This is the *cumulative gains* curve. This curve is compared to assigning a naive prediction ($\bar{y}$) to each record and accumulating these average values, which results in a diagonal line. The further away the cumulative gains curve from the diagonal benchmark line, the better the model is doing in separating records with high value outcomes from those with low value outcomes. The same information can be presented in a decile lift chart, where the ordered records are grouped into ten deciles, and for each decile, the chart presents the ratio of model gains to naive benchmark gains, which is called *lift*.

[1]The terms *lift* and *gains* are occasionally used interchangeably; to avoid confusion we use the prevalent definitions in which *cumulative gains* refers to the sum of outcome values for the top cases identified by a model, and *lift* to the ratio of that sum to the sum obtained through random selection.

Figure 5.2 shows a cumulative gains chart and decile lift chart based on fitting a linear regression model to the Toyota data. The charts are based on the validation data of 575 cars. It can be seen that the model's predictive performance in terms of gains is better than the baseline model, since its cumulative gains curve is higher than that of the baseline model. The charts in Figure 5.2 would be useful in the following scenario: choosing the top 10% of the cars that gave the highest predicted sales, for example, we would gain 1.75 times the amount of revenue, compared to choosing 10% of the cars at random. This lift number can also be computed from the cumulative gains chart by comparing the sales for 57 random cars (the value of the baseline curve at $x = 57$), which is \$607,703 (= the sum of the actual sales for the 575 validation set cars divided by 10) with the actual sales of the 40 cars that have the highest predicted values (the value of the cumulative gains curve at $x = 57$), \$1,073,830. The ratio between these numbers is 1.76.

## 5.3 Judging Classifier Performance

The need for performance measures arises from the wide choice of classifiers and predictive methods. Not only do we have several different methods, but even within a single method there are usually many options that can lead to completely different results. A simple example is the choice of predictors used within a particular predictive algorithm. Before we study these various algorithms in detail and face decisions on how to set these options, we need to know how we will measure success.

A natural criterion for judging the performance of a classifier is the probability of making a *misclassification error*. Misclassification means that the record belongs to one class but the model classifies it as a member of a different class. A classifier that makes no errors would be perfect, but we do not expect to be able to construct such classifiers in the real world due to "noise" and not having all the information needed to classify records precisely. Is there a minimal probability of misclassification that we should require of a classifier?

### Benchmark: The Naive Rule

A very simple rule for classifying a record into one of $m$ classes, ignoring all predictor information $(x_1, x_2, \ldots, x_p)$ that we may have, is to classify the record as a member of the majority class. In other words, "classify as belonging to the most prevalent class." The *naive rule* is used mainly as a baseline or benchmark for evaluating the performance of more complicated classifiers. Clearly, a classifier that uses external predictor information (on top of the class membership allocation) should outperform the naive rule. There are various performance

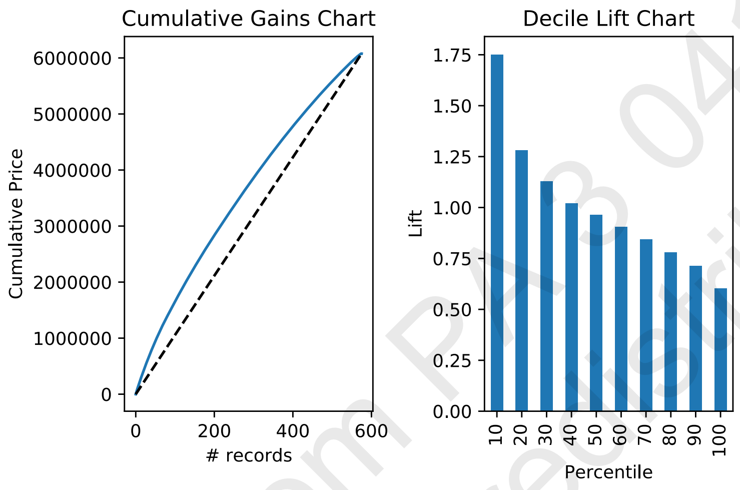


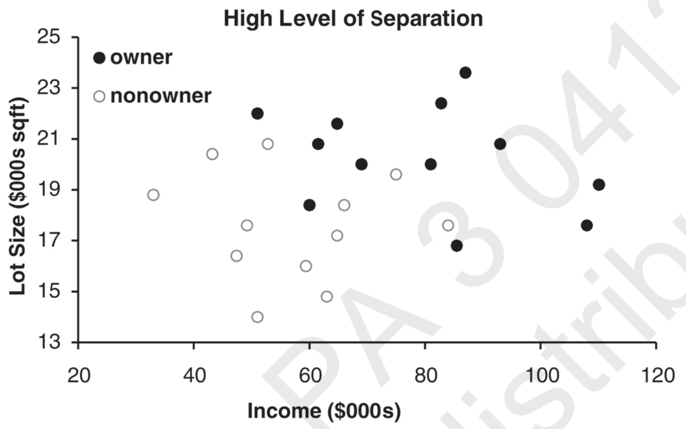
FIGURE 5.2 CUMULATIVE GAINS CHART (LEFT) AND DECILE LIFT CHART (RIGHT) FOR CONTINUOUS OUTCOME VARIABLE (SALES OF TOYOTA CARS)

code for generating cumulative gains and decile lift charts in Figure 5.2

```
pred_v = pd.Series(reg.predict(valid_X))
pred_v = pred_v.sort_values(ascending=False)

fig, axes = plt.subplots(nrows=1, ncols=2)
ax = gainsChart(pred_v, ax=axes[0])
ax.set_ylabel('Cumulative Price')
ax.set_title('Cumulative Gains Chart')

ax = liftChart(pred_v, ax=axes[1], labelBars=False)
ax.set_ylabel('Lift')

plt.tight_layout()
plt.show()
```

measures based on the naive rule that measure how much better than the naive rule a certain classifier performs.

Similar to using the sample mean ($\bar{y}$) as the naive benchmark in the numerical outcome case, the naive rule for classification relies solely on the $y$ information and excludes any additional predictor information.



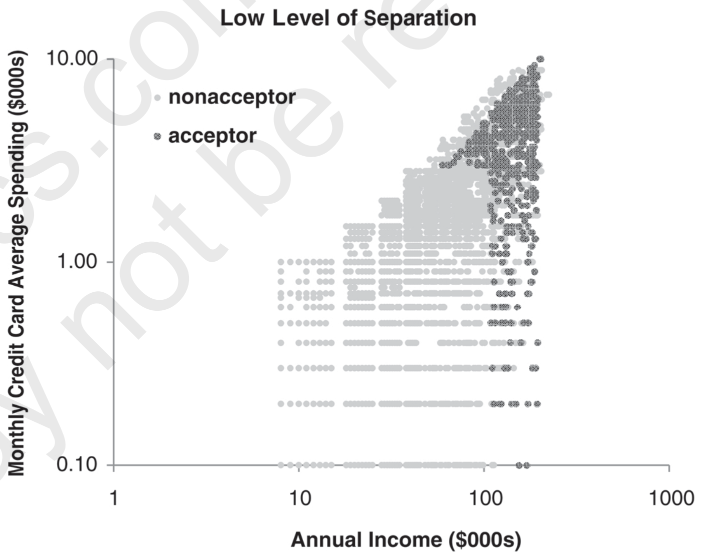


FIGURE 5.3 HIGH (TOP) AND LOW (BOTTOM) LEVELS OF SEPARATION BETWEEN TWO CLASSES, USING TWO PREDICTORS

## Class Separation

If the classes are well separated by the predictor information, even a small dataset will suffice in finding a good classifier, whereas if the classes are not separated at all by the predictors, even a very large dataset will not help. Figure 5.3 illustrates this for a two-class case. The top panel includes a small dataset ($n$ = 24 records) where two predictors (income and lot size) are used for separating owners from nonowners [we thank Dean Wichern for this example, described in Johnson and Wichern (2002)]. Here, the predictor information seems useful in that it separates the two classes (owners/nonowners). The bottom panel shows a much larger dataset ($n$ = 5000 records) where the two predictors (income and monthly average credit card spending) do not separate the two classes well in most of the higher ranges (loan acceptors/nonacceptors).

## The Confusion (Classification) Matrix

In practice, most accuracy measures are derived from the *confusion matrix*, also called *classification matrix*. This matrix summarizes the correct and incorrect classifications that a classifier produced for a certain dataset. Rows and columns of the confusion matrix correspond to the predicted and true (actual) classes, respectively. Table 5.2 shows an example of a classification (confusion) matrix for a two-class (0/1) problem resulting from applying a certain classifier to 3000 records. The two diagonal cells (upper left, lower right) give the number of correct classifications, where the predicted class coincides with the actual class of the record. The off-diagonal cells give counts of misclassification. The lower left cell gives the number of class 1 members that were misclassified as 0's (in this example, there were 85 such misclassifications). Similarly, the upper right cell gives the number of class 0 members that were misclassified as 1's (25 such records). In Python, we can obtain a confusion matrix using the function *confusion_matrix()* in the `scikit-learn` package. This function creates the cross-tabulation of actual and predicted classes. The utility *ClassificationSummary* (see Appendix) produces a more readable confusion matrix. We will see an example later in this chapter.

TABLE 5.2 CONFUSION MATRIX BASED ON 3000 RECORDS AND TWO CLASSES

| | Predict Class 0 | Predict Class 1 |
|---|---|---|
| **Actual 0** | 2689 | 25 |
| **Actual 1** | 85 | 201 |

The confusion matrix gives estimates of the true classification and misclassification rates. Of course, these are estimates and they can be incorrect, but if we have a large enough dataset and neither class is very rare, our estimates will be reliable. Sometimes, we may be able to use public data such as US Census data to estimate these proportions. However, in most business settings, we will not know them.

### Using the Validation Data

To obtain an honest estimate of future classification error, we use the confusion matrix that is computed from the *validation data*. In other words, we first partition the data into training and validation sets by random selection of records. We then construct a classifier using the training data, and then apply it to the validation data. This will yield the predicted classifications for records in the validation set (see Figure 2.4 in Chapter 2). We then summarize these classifications in a confusion matrix. Although we can summarize our results in a confusion matrix for training data as well, the resulting confusion matrix is not useful for getting an honest estimate of the misclassification rate for new data due to the danger of overfitting.

In addition to examining the validation data confusion matrix to assess the classification performance on new data, we compare the training data confusion matrix to the validation data confusion matrix, in order to detect overfitting: although we expect somewhat inferior results on the validation data, a large discrepancy in training and validation performance might be indicative of overfitting.

### Accuracy Measures

Different accuracy measures can be derived from the classification matrix. Consider a two-class case with classes $C_1$ and $C_2$ (e.g., buyer/non-buyer). The schematic confusion matrix in Table 5.3 uses the notation $n_{i,j}$ to denote the number of records that are class $C_i$ members and were classified as $C_j$ members. Of course, if $i \neq j$, these are counts of misclassifications. The total number of records is $n = n_{1,1} + n_{1,2} + n_{2,1} + n_{2,2}$.

TABLE 5.3 CONFUSION MATRIX: MEANING OF EACH CELL

| | | Predicted Class | |
|---|---|---|---|
| | | $C_1$ | $C_2$ |
| Actual Class | $C_1$ | $n_{1,1}$ = number of $C_1$ records classified correctly | $n_{1,2}$ = number of $C_1$ records classified incorrectly as $C_2$ |
| | $C_2$ | $n_{2,1}$ = number of $C_2$ records classified incorrectly as $C_1$ | $n_{2,2}$ = number of $C_2$ records classified correctly |

A main accuracy measure is the *estimated misclassification rate*, also called the *overall error rate*. It is given by

$$\text{err} = \frac{n_{1,2} + n_{2,1}}{n},$$

where $n$ is the total number of records in the validation dataset. In the example in Table 5.2, we get err $= (25 + 85)/3000 = 3.67\%$.

We can measure accuracy by looking at the correct classifications—the full half of the cup—instead of the misclassifications. The *overall accuracy* of a classifier is estimated by

$$\text{accuracy} = 1 - \text{err} = \frac{n_{1,1} + n_{2,2}}{n}.$$

In the example, we have $(201 + 2689)/3000 = 96.33\%$.

## Propensities and Cutoff for Classification

The first step in most classification algorithms is to estimate the probability that a record belongs to each of the classes. These probabilities are also called *propensities*. Propensities are typically used either as an interim step for generating predicted class membership (classification), or for rank-ordering the records by their probability of belonging to a class of interest. Let us consider their first use in this section. The second use is discussed in Section 5.4.

If overall classification accuracy (involving all the classes) is of interest, the record can be assigned to the class with the highest probability. In many records, a single class is of special interest, so we will focus on that particular class and compare the propensity of belonging to that class to a *cutoff value* set by the analyst. This approach can be used with two classes or more than two classes, though it may make sense in such cases to consolidate classes so that you end up with two: the class of interest and all other classes. If the probability of belonging to the class of interest is above the cutoff, the record is assigned to that class.

**CUTOFF VALUES FOR TRIAGE**

In some cases, it is useful to have two cutoffs, and allow a "cannot say" option for the classifier. In a two-class situation, this means that for a record, we can make one of three predictions: The record belongs to $C_1$, or the record belongs to $C_2$, or we cannot make a prediction because there is not enough information to pick $C_1$ or $C_2$ confidently. Records that the classifier cannot classify are subjected to closer scrutiny either by using expert judgment or by enriching the set of predictor variables by gathering additional information that is perhaps more difficult or expensive to obtain. An example is classification of documents found during legal discovery (reciprocal forced document disclosure in a legal proceeding). Under traditional human-review systems, qualified legal personnel are needed to review what might be tens of thousands of documents to determine their relevance to a case. Using a classifier and a triage outcome, documents could be sorted into clearly relevant, clearly not relevant, and the gray area documents requiring human review. This substantially reduces the costs of discovery.

The default cutoff value in two-class classifiers is 0.5. Thus, if the probability of a record being a class $C_1$ member is greater than 0.5, that record is classified as a $C_1$. Any record with an estimated probability of less than 0.5 would be classified as a $C_2$. It is possible, however, to use a cutoff that is either higher or lower than 0.5. A cutoff greater than 0.5 will end up classifying fewer records as $C_1$'s, whereas a cutoff less than 0.5 will end up classifying more records as $C_1$. Typically, the misclassification rate will rise in either case.

Consider the data in Table 5.4, showing the actual class for 24 records, sorted by the probability that the record is an "owner" (as estimated by a data mining algorithm). If we adopt the standard 0.5 as the cutoff, our misclassification rate is 3/24, whereas if we instead adopt a cutoff of 0.25, we classify more records as owners and the misclassification rate goes up (comprising more nonowners misclassified as owners) to 5/24. Conversely, if we adopt a cutoff of 0.75, we classify fewer records as owners. The misclassification rate goes up (comprising more owners misclassified as nonowners) to 6/24. All this can be seen in the classification tables in Table 5.5.

To see the entire range of cutoff values and how the accuracy or misclassification rates change as a function of the cutoff, we can plot the performance measure of interest vs. the cutoff. The results for the riding mowers example are shown in Figure 5.4. We can see that the accuracy level is pretty stable around 0.8 for cutoff values between 0.2 and 0.8.

Why would we want to use cutoff values different from 0.5 if they increase the misclassification rate? The answer is that it might be more important to classify owners properly than nonowners, and we would tolerate a greater misclassification of the latter. Or the reverse might be true; in other words, the costs of misclassification might be asymmetric. We can adjust the cutoff value in such a case to classify more records as the high-value class, that is, accept more misclassifications where the misclassification cost is low. Keep in mind

TABLE 5.4 24 RECORDS WITH THEIR ACTUAL CLASS AND THE PROBABILITY (PROPENSITY) OF THEM BEING CLASS "OWNER" MEMBERS, AS ESTIMATED BY A CLASSIFIER

| Actual Class | Probability of Class "owner" | Actual Class | Probability of Class "owner" |
|---|---|---|---|
| owner | 0.9959 | owner | 0.5055 |
| owner | 0.9875 | nonowner | 0.4713 |
| owner | 0.9844 | nonowner | 0.3371 |
| owner | 0.9804 | owner | 0.2179 |
| owner | 0.9481 | nonowner | 0.1992 |
| owner | 0.8892 | nonowner | 0.1494 |
| owner | 0.8476 | nonowner | 0.0479 |
| nonowner | 0.7628 | nonowner | 0.0383 |
| owner | 0.7069 | nonowner | 0.0248 |
| owner | 0.6807 | nonowner | 0.0218 |
| owner | 0.6563 | nonowner | 0.0161 |
| nonowner | 0.6224 | nonowner | 0.0031 |

that we are doing so after the data mining model has already been selected—we are not changing that model. It is also possible to incorporate costs into the picture before deriving the model. These subjects are discussed in greater detail below.

### Performance in Case of Unequal Importance of Classes

Suppose that it is more important to predict membership correctly in class $C_1$ than in class $C_2$. An example is predicting the financial status (bankrupt/solvent) of firms. It may be more important to predict correctly a firm that is going bankrupt than to predict correctly a firm that is going to remain solvent. The classifier is essentially used as a system for detecting or signaling bankruptcy. In such a case, the overall accuracy is not a good measure for evaluating the classifier.

TABLE 5.5 **CONFUSION MATRICES BASED ON CUTOFFS OF 0.5, 0.25, AND 0.75 (RIDING MOWERS EXAMPLE)**

```
owner_df = pd.read_csv('ownerExample.csv')

## cutoff = 0.5
predicted = ['owner' if p > 0.5 else 'nonowner' for p in owner_df.Probability]
classificationSummary(owner_df.Class, predicted, class_names=['nonowner', 'owner'])

Confusion Matrix (Accuracy 0.8750)

         Prediction
  Actual nonowner    owner
nonowner       10        2
   owner        1       11

## cutoff = 0.25
predicted = ['owner' if p > 0.25 else 'nonowner' for p in owner_df.Probability]
classificationSummary(owner_df.Class, predicted, class_names=['nonowner', 'owner'])

Confusion Matrix (Accuracy 0.7917)

         Prediction
  Actual nonowner    owner
nonowner        8        4
   owner        1       11

## cutoff = 0.75
predicted = ['owner' if p > 0.75 else 'nonowner' for p in owner_df.Probability]
classificationSummary(owner_df.Class, predicted, class_names=['nonowner', 'owner'])

Confusion Matrix (Accuracy 0.7500)

         Prediction
  Actual nonowner    owner
nonowner       11        1
   owner        5        7
```

function *classificationSummary* can be found in the Python Utilities Functions Appendix.

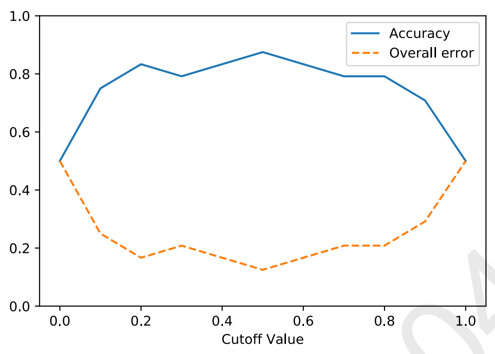


FIGURE 5.4 PLOTTING ACCURACY AND OVERALL ERROR AS A FUNCTION OF THE CUTOFF VALUE (RIDING MOWERS EXAMPLE)

code for creating Figure 5.4

```
df = pd.read_csv('liftExample.csv')

cutoffs = [i * 0.1 for i in range(0, 11)]
accT = []
for cutoff in cutoffs:
    predicted = [1 if p > cutoff else 0 for p in df.prob]
    accT.append(accuracy_score(df.actual, predicted))

line_accuracy = plt.plot(cutoffs, accT, '-', label='Accuracy')[0]
line_error = plt.plot(cutoffs, [1 - acc for acc in accT], '--', label='Overall error')[0]
plt.ylim([0,1])
plt.xlabel('Cutoff Value')
plt.legend(handles=[line_accuracy, line_error])
plt.show()
```

Suppose that the important class is $C_1$. The following pair of accuracy measures are the most popular:

**The sensitivity** (also termed recall) of a classifier is its ability to detect the important class members correctly. This is measured by $n_{1,1}/(n_{1,1} + n_{1,2})$, the percentage of $C_1$ members classified correctly.

**The specificity** of a classifier is its ability to rule out $C_2$ members correctly. This is measured by $n_{2,2}/(n_{2,1} + n_{2,2})$, the percentage of $C_2$ members classified correctly.

It can be useful to plot these measures against the cutoff value in order to find a cutoff value that balances these measures.

**ROC Curve** A more popular method for plotting the two measures is through *ROC* (Receiver Operating Characteristic) *curves*. Starting from the lower left, the ROC curve plots the pairs {sensitivity, specificity} as the cutoff value descends from 1 to 0. (A typical alternative presentation is to plot 1-specificity on the $x$-axis, which allows 0 to be placed on the left end of the axis, and 1 on the right.) Better performance is reflected by curves that are closer to the top-left corner. The comparison curve is the diagonal, which reflects the average performance of a guessing classifier that has no information about the predictors or outcome variable. This guessing classifier guesses that a proportion $\alpha$ of the records is 1's and therefore assigns each record an equal probability $P(Y = 1) = \alpha$. In this case, on average, a proportion $\alpha$ of the 1s will be correctly classified (Sensitivity=$\alpha$), and a proportion $\alpha$ of the 0s will be correctly classified (1-Specificity=$\alpha$). As we increase the cutoff value $\alpha$ from 0 to 1, we get the diagonal line Sensitivity = 1-Specificity. Note that the naive rule is one point on this diagonal line, where $\alpha$ = proportion of actual 1's.

A common metric to summarize an ROC curve is "area under the curve (AUC)," which ranges from 1 (perfect discrimination between classes) to 0.5 (no better than random guessing). The ROC curve for the owner/nonowner example and its corresponding AUC are shown in Figure 5.5.

**COMPUTING RATES: FROM WHOSE POINT OF VIEW?**

Sensitivity and specificity measure the performance of a classifier from the point of view of the "classifying agency" (e.g., a company classifying customers or a hospital classifying patients). They answer the question "how well does the classifier segregate the important class members?". It is also possible to measure accuracy from the perspective of the entity being classified (e.g., the customer or the patient), who asks "given my predicted class, what is my chance of actually belonging to that class?", although this question is usually less relevant in a data mining application. The terms "false discovery rate" and "false omission rate" are measures of performance from the perspective of the individual entity. If $C_1$ is the important (positive) class, then they are defined as

*The false discovery rate (FDR)* is the proportion of $C_1$ predictions that are wrong, equal to $n_{2,1}/(n_{1,1} + n_{2,1})$. Note that this is a ratio within the row of $C_1$ predictions (i.e., it uses only records that were classified as $C_1$).

*The false omission rate (FOR)* is the proportion of $C_2$ predictions that are wrong, equal to $n_{1,2}/(n_{1,2} + n_{2,2})$. Note that this is a ratio within the row of $C_2$ predictions (i.e., it uses only records that were classified as $C_2$).

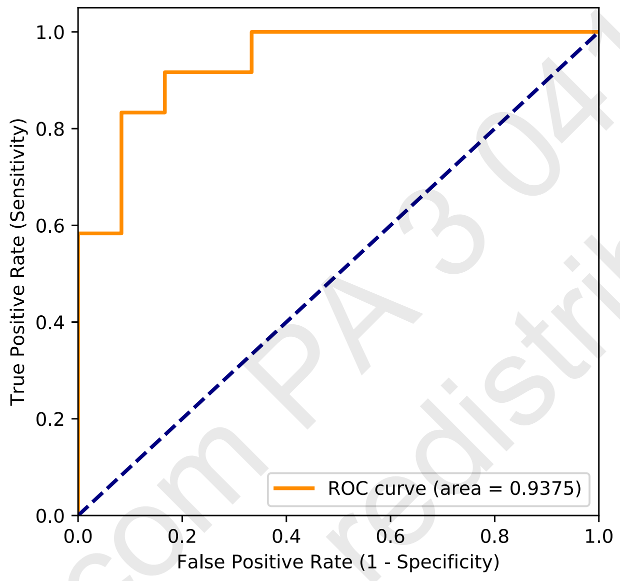


FIGURE 5.5 ROC CURVE FOR RIDING MOWERS EXAMPLE

code for generating ROC curve and computing AUC

```
from sklearn.metrics import roc_curve, auc

# compute ROC curve and AUC
fpr, tpr, _ = roc_curve(df.actual, df.prob)
roc_auc = auc(fpr, tpr)

plt.figure(figsize=[5, 5])
plt.plot(fpr, tpr, color='darkorange',
         lw=2, label='ROC curve (area = %0.4f)' % roc_auc)
plt.plot([0, 1], [0, 1], color='navy', lw=2, linestyle='--')
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('False Positive Rate (1 - Specificity)')
plt.ylabel('True Positive Rate (Sensitivity)')
plt.legend(loc="lower right")
```

### Asymmetric Misclassification Costs

Implicit in our discussion of the lift curve, which measures how effective we are in identifying the members of one particular class, is the assumption that the error of misclassifying a record belonging to one class is more serious than for the other class. For example, misclassifying a household as unlikely to respond to a sales offer when it belongs to the class that would respond incurs a greater cost (the opportunity cost of the foregone sale) than the converse error. In the former case, you are missing out on a sale worth perhaps tens or hundreds of dollars. In the latter, you are incurring the costs of contacting someone who will not purchase. In such a scenario, using the misclassification rate as a criterion can be misleading.

Note that we are assuming that the cost (or benefit) of making correct classifications is zero. At first glance, this may seem incomplete. After all, the benefit (negative cost) of classifying a buyer correctly as a buyer would seem substantial. And in other circumstances (e.g., scoring our classification algorithm to fresh data to implement our decisions), it will be appropriate to consider the actual net dollar impact of each possible classification (or misclassification). Here, however, we are attempting to assess the value of a classifier in terms of classification error, so it greatly simplifies matters if we can capture all cost/benefit information in the misclassification cells. So, instead of recording the benefit of classifying a respondent household correctly, we record the cost of failing to classify it as a respondent household. It amounts to the same thing and our goal becomes the minimization of costs, whether the costs are actual costs or missed benefits (opportunity costs).

Consider the situation where the sales offer is mailed to a random sample of people for the purpose of constructing a good classifier. Suppose that the offer is accepted by 1% of those households. For these data, if a classifier simply classifies every household as a non-responder, it will have an error rate of only 1% but it will be useless in practice. A classifier that misclassifies 2% of buying households as nonbuyers and 20% of the nonbuyers as buyers would have a higher error rate but would be better if the profit from a sale is substantially higher than the cost of sending out an offer. In these situations, if we have estimates of the cost of both types of misclassification, we can use the confusion matrix to compute the expected cost of misclassification for each record in the validation data. This enables us to compare different classifiers using overall expected costs (or profits) as the criterion.

Suppose that we are considering sending an offer to 1000 more people, where on average 1% of whom respond (1). Naively classifying everyone as a 0 has an error rate of only 1%. Using a data mining routine, suppose that we can produce these classifications:

| | Predict Class 0 | Predict Class 1 |
|---|---|---|
| Actual 0 | 970 | 20 |
| Actual 1 | 2 | 8 |

These classifications have an error rate of $100 \times (20 + 2)/1000 = 2.2\%$—higher than the naive rate.

Now suppose that the profit from a responder is \$10 and the cost of sending the offer is \$1. Classifying everyone as a 0 still has a misclassification rate of only 1%, but yields a profit of \$0. Using the data mining routine, despite the higher misclassification rate, yields a profit of \$60.

The matrix of profit is as follows (nothing is sent to the predicted 0's so there are no costs or sales in that column):

| **Profit** | Predict Class 0 | Predict Class 1 |
|---|---|---|
| Actual 0 | 0 | − \$20 |
| Actual 1 | 0 | \$80 |

Looked at purely in terms of costs, when everyone is classified as a 0, there are no costs of sending the offer; the only costs are the opportunity costs of failing to make sales to the ten 1's = \$100. The cost (actual costs of sending the offer, plus the opportunity costs of missed sales) of using the data mining routine to select people to send the offer to is only \$48, as follows:

| **Costs** | Predict Class 0 | Predict Class 1 |
|---|---|---|
| Actual 0 | 0 | \$20 |
| Actual 1 | \$20 | \$8 |

However, this does not improve the actual classifications themselves. A better method is to change the classification rules (and hence the misclassification rates) as discussed in the preceding section, to reflect the asymmetric costs.

A popular performance measure that includes costs is the *average misclassification cost*, which measures the average cost of misclassification per classified record. Denote by $q_1$ the cost of misclassifying a class $C_1$ record (as belonging to class $C_2$) and by $q_2$ the cost of misclassifying a class $C_2$ record (as belonging to class $C_1$). The average misclassification cost is

$$\frac{q_1 n_{1,2} + q_2 n_{2,1}}{n}.$$

Thus, we are looking for a classifier that minimizes this quantity. This can be computed, for instance, for different cutoff values.

It turns out that the optimal parameters are affected by the misclassification costs only through the ratio of these costs. This can be seen if we write the foregoing measure slightly differently:

$$\frac{q_1 n_{1,2} + q_2 n_{2,1}}{n} = \frac{n_{1,2}}{n_{1,1}+n_{1,2}} \frac{n_{1,1}+n_{1,2}}{n} q_1 + \frac{n_{2,1}}{n_{2,1}+n_{2,2}} \frac{n_{2,1}+n_{2,2}}{n} q_2.$$

Minimizing this expression is equivalent to minimizing the same expression divided by a constant. If we divide by $q_1$, it can be seen clearly that the minimization depends only on $q_2/q_1$ and not on their individual values. This is very practical, because in many cases it is difficult to assess the costs associated with misclassifying a $C_1$ member and a $C_2$ member, but estimating the ratio is easier.

This expression is a reasonable estimate of future misclassification cost if the proportions of classes $C_1$ and $C_2$ in the sample data are similar to the proportions of classes $C_1$ and $C_2$ that are expected in the future. If instead of a random sample, we draw a sample such that one class is oversampled (as described in the next section), then the sample proportions of $C_1$'s and $C_2$'s will be distorted compared to the future or population. We can then correct the average misclassification cost measure for the distorted sample proportions by incorporating estimates of the true proportions (from external data or domain knowledge), denoted by $p(C_1)$ and $p(C_2)$, into the formula:

$$\frac{n_{1,2}}{n_{1,1}+n_{1,2}} p(C_1)\, q_1 + \frac{n_{2,1}}{n_{2,1}+n_{2,2}} p(C_2)\, q_2.$$

Using the same logic as above, it can be shown that optimizing this quantity depends on the costs only through their ratio ($q_2/q_1$) and on the prior probabilities only through their ratio [$p(C_2)/p(C_1)$]. This is why software packages that incorporate costs and prior probabilities might prompt the user for ratios rather than actual costs and probabilities.

## Generalization to More Than Two Classes

All the comments made above about two-class classifiers extend readily to classification into more than two classes. Let us suppose that we have $m$ classes $C_1, C_2, \ldots, C_m$. The confusion matrix has $m$ rows and $m$ columns. The misclassification cost associated with the diagonal cells is, of course, always zero. Incorporating prior probabilities of the various classes (where now we have $m$ such numbers) is still done in the same manner. However, evaluating misclassification costs becomes much more complicated: For an $m$-class case we have $m(m-1)$ types of misclassifications. Constructing a matrix of misclassification costs thus becomes prohibitively complicated.

## 5.4 Judging Ranking Performance

We now turn to the predictive goal of detecting, among a set of new records, the ones most likely to belong to a class of interest. Recall that this differs from the goal of predicting class membership for each new record.

### Gains and Lift Charts for Binary Data

We already introduced cumulative gains charts and lift charts in the context of a numerical outcome (Section 5.2). We now describe these charts for a binary outcome. This is a more common usage than for predicted continuous outcomes. The gains and lift help us determine how effectively we can "skim the cream" by selecting a relatively small number of records and getting a relatively large portion of the responders.[2] The input required to construct these charts is a validation dataset that has been "scored" by appending to each record the propensity that it will belong to a given class.

Let's continue with the case in which a particular class is relatively rare and of much more interest than the other class: tax cheats, debt defaulters, or responders to a mailing. We would like our classification model to sift through the records and sort them according to which ones are most likely to be tax cheats, responders to the mailing, and so on. We can then make more informed decisions. For example, we can decide how many and which tax returns to examine if looking for tax cheats. The model will give us an estimate of the extent to which we will encounter more and more non-cheaters as we proceed through the sorted data starting with the records most likely to be tax cheats. Or we can use the sorted data to decide to which potential customers a limited-budget mailing should be targeted. In other words, we are describing the case when our goal is to obtain a rank ordering among the records according to their class membership propensities.

**Sorting by Propensity** To construct a cumulative gains chart, we sort the set of records by propensity, in descending order. This is the propensity to belong to the important class, say $C_1$. Then, in each row, we compute the cumulative number of $C_1$ members (Actual Class = $C_1$). For example, Table 5.6 shows the 24 records ordered in descending class "1" propensity. The right-most column accumulates the number of actual 1's. The cumulative gains chart then plots this cumulative column against the number of records.

It is straightforward to create a cumulative gains chart in Python – see Figure 5.6.

[2]The terms *lift* and *gains* are occasionally used interchangeably; to avoid confusion we use the prevalent definitions in which *gains* refers to total numbers or percentages of cases of interest identified by a model, and *lift* to the ratio of such cases to those found through random selection.

TABLE 5.6 RECORDS SORTED BY PROPENSITY OF OWNERSHIP (HIGH TO LOW) FOR THE MOWER EXAMPLE

| Obs | Propensity of 1 | Actual Class | Cumulative Actual Class |
|---|---|---|---|
| 1 | 0.995976726 | 1 | 1 |
| 2 | 0.987533139 | 1 | 2 |
| 3 | 0.984456382 | 1 | 3 |
| 4 | 0.980439587 | 1 | 4 |
| 5 | 0.948110638 | 1 | 5 |
| 6 | 0.889297203 | 1 | 6 |
| 7 | 0.847631864 | 1 | 7 |
| 8 | 0.762806287 | 0 | 7 |
| 9 | 0.706991915 | 1 | 8 |
| 10 | 0.680754087 | 1 | 9 |
| 11 | 0.656343749 | 1 | 10 |
| 12 | 0.622419543 | 0 | 10 |
| 13 | 0.505506928 | 1 | 11 |
| 14 | 0.471340450 | 0 | 11 |
| 15 | 0.337117362 | 0 | 11 |
| 16 | 0.217967810 | 1 | 12 |
| 17 | 0.199240432 | 0 | 12 |
| 18 | 0.149482655 | 0 | 12 |
| 19 | 0.047962588 | 0 | 12 |
| 20 | 0.038341401 | 0 | 12 |
| 21 | 0.024850999 | 0 | 12 |
| 22 | 0.021806029 | 0 | 12 |
| 23 | 0.016129906 | 0 | 12 |
| 24 | 0.003559986 | 0 | 12 |

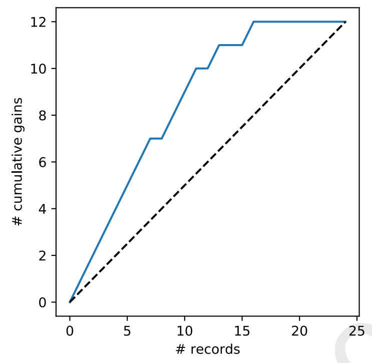


FIGURE 5.6 CUMULATIVE GAINS CHART FOR THE MOWER EXAMPLE

code for creating a cumulative gains chart

```
df = pd.read_csv('liftExample.csv')
df = df.sort_values(by=['prob'], ascending=False)
gainsChart(df.actual, figsize=(4, 4))
```

**Interpreting the cumulative gains chart** What is considered good or bad performance? The ideal ranking performance would place all the 1's at the beginning (the actual 1's would have the highest propensities and be at the top of the table), and all the 0's at the end. A curve corresponding to this ideal case would be a diagonal line with slope 1 which turns into a horizontal line (once all the 1's were accumulated). In the example, the curve for the best possible classifier—a classifier that makes no errors—would overlap the existing curve at the start, continue with a slope of 1 until it reached all the 12 1's, then continue horizontally to the right. See the red line in the graph on the right.

In contrast, a useless model would be one that randomly assigns propensities (shuffling the 1's and 0's randomly in the Actual Class column). Such behavior would increase the cumulative number of 1's, on average, by $\frac{\#1's}{n}$ in each row. And in fact, this is the diagonal dotted line joining the points (0,0) to (24,12) seen in Figure 5.6. This serves as a reference line. For any given number of records (the $x$-axis value), it represents the expected number of 1 classifications (=gains) if we did not have a model but simply selected records at random. It provides a benchmark against which we can evaluate the ranking performance of the model. In this example, although our model is not perfect, it seems to perform much better than the random benchmark.

How do we read a cumulative gains chart? For a given number of records ($x$-axis), the gains curve value on the $y$-axis tells us how much better we gain, and we can compare it to the gains from random assignment. For example, looking at Figure 5.6, if we use our model to choose the top 10 records, the curve tells us that we would be right for about nine of them. If we simply select 10 records at random, we expect to be right for $10 \times 12/24 = 5$ records. In terms of lift, the model gives us a "lift" in detecting class 1 members of $9/5 = 1.8$. The lift will vary with the number of records we choose to act on. A good classifier will give us a high lift when we act on only a few records. As we include more records, the lift will typically decrease.

### Decile Lift Charts

The information from the cumulative gains chart can be portrayed as a *decile lift chart*, as shown in Figure 5.7, which is widely used in direct marketing predictive modeling. The decile lift chart aggregates all the lift information into 10 buckets. The bars show, on the $y$-axis, the factor by which our model outperforms a random assignment of 0's and 1's, taking one decile at a time. Reading the bar on the left, we see that taking 8% of the records that are ranked by the model as "the most probable 1's" (having the highest propensities) yields twice as many 1's as would a random selection of 8% of the records. In this example, the decile lift chart indicates that we can even use the model to select the top 40% records with the highest propensities and still perform almost twice as well as random.

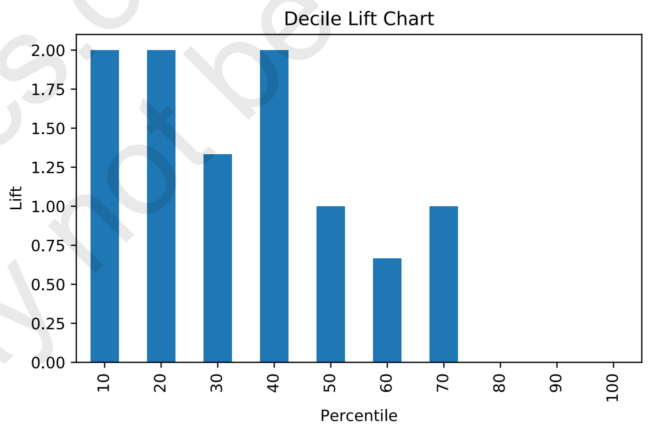


FIGURE 5.7 **DECILE LIFT CHART**

code for creating a decile lift chart

```
# use liftChart method from utilities
liftChart(df.actual, labelBars=False)
```

### Beyond Two Classes

Gains and lift charts cannot be used with a multiclass classifier unless a single "important class" is defined and the classifications are reduced to "important" and "unimportant" classes.

### Gains and Lift Charts Incorporating Costs and Benefits

When the benefits and costs of correct and incorrect classification are known or can be estimated, the gains and lift charts are still a useful presentation and decision tool. As before, we need a classifier that assigns to each record a propensity that it belongs to a particular class. The procedure is then as follows:

1. Sort the records in descending order of predicted probability of success (where *success* = belonging to the class of interest).
2. For each record, record the cost (benefit) associated with the actual outcome.
3. For the highest propensity (i.e., first) record, its $x$-axis value is 1 and its $y$-axis value is its cost or benefit (computed in Step 2) on the cumulative gains curve.
4. For the next record, again calculate the cost (benefit) associated with the actual outcome. Add this to the cost (benefit) for the previous record. This sum is the $y$-axis coordinate of the second point on the cumulative gains curve. Its $x$-axis value is 2.
5. Repeat Step 4 until all records have been examined. Connect all the points, and this is the cumulative gains curve.
6. The reference line is a straight line from the origin to the point $y$ = total net benefit and $x = n(n$ = number of records).

**Note:** It is entirely possible for a reference line that incorporates costs and benefits to have a negative slope if the net value for the entire dataset is negative. For example, if the cost of mailing to a person is \$0.65, the value of a responder is \$25, and the overall response rate is 2%, the expected net value of mailing to a list of 10,000 is $(0.02 \times \$25 \times 10,000) - (\$0.65 \times 10,000) = \$5000 - \$6500 = -\$1500$. Hence, the $y$-value at the far right of the lift curve ($x$ = 10,000) is $-1500$, and the slope of the reference line from the origin will be negative. The optimal point will be where the cumulative gains curve is at a maximum (i.e., mailing to about 3000 people) in Figure 5.8.

### Cumulative Gains as a Function of Cutoff

We could also plot the cumulative gains as a function of the cutoff value. The only difference is the scale on the $x$-axis. When the goal is to select the top

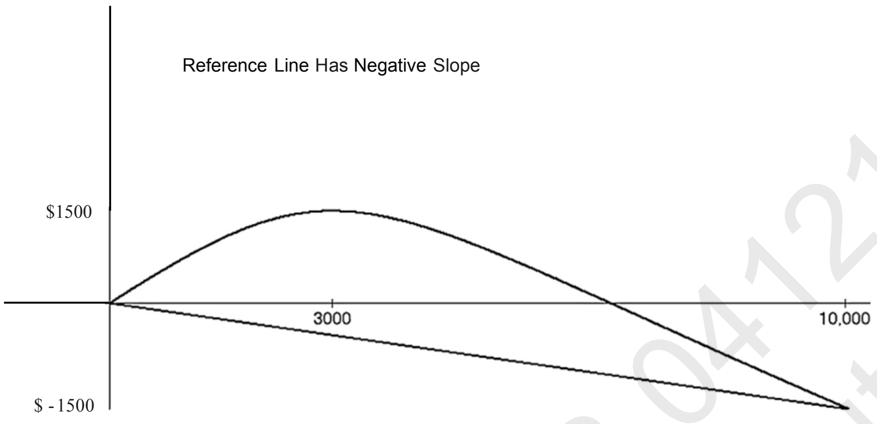


FIGURE 5.8 CUMULATIVE GAINS CURVE INCORPORATING COSTS

records based on a certain budget, the cumulative gains vs. number of records is preferable. In contrast, when the goal is to find a cutoff that distinguishes well between the two classes, the cumulative gains vs. cutoff value is more useful.

## 5.5 Oversampling

As we saw briefly in Chapter 2, when classes are present in very unequal proportions, simple random sampling may produce too few of the rare class to yield useful information about what distinguishes them from the dominant class. In such cases, stratified sampling is often used to oversample the records from the rarer class and improve the performance of classifiers. It is often the case that the rarer events are the more interesting or important ones: responders to a mailing, those who commit fraud, defaulters on debt, and the like. This same stratified sampling procedure is sometimes called *weighted sampling* or *undersampling*, the latter referring to the fact that the more plentiful class is undersampled, relative to the rare class. We shall stick to the term *oversampling*.

In all discussions of *oversampling*, we assume the common situation in which there are two classes, one of much greater interest than the other. Data with more than two classes do not lend themselves to this procedure.

Consider the data in Figure 5.9, where × represents non-responders, and ○, responders. The two axes correspond to two predictors. The dashed vertical line does the best job of classification under the assumption of equal costs: It results in just one misclassification (one ○ is misclassified as an ×). If we incorporate more realistic misclassification costs—let's say that failing to catch a ○ is five times

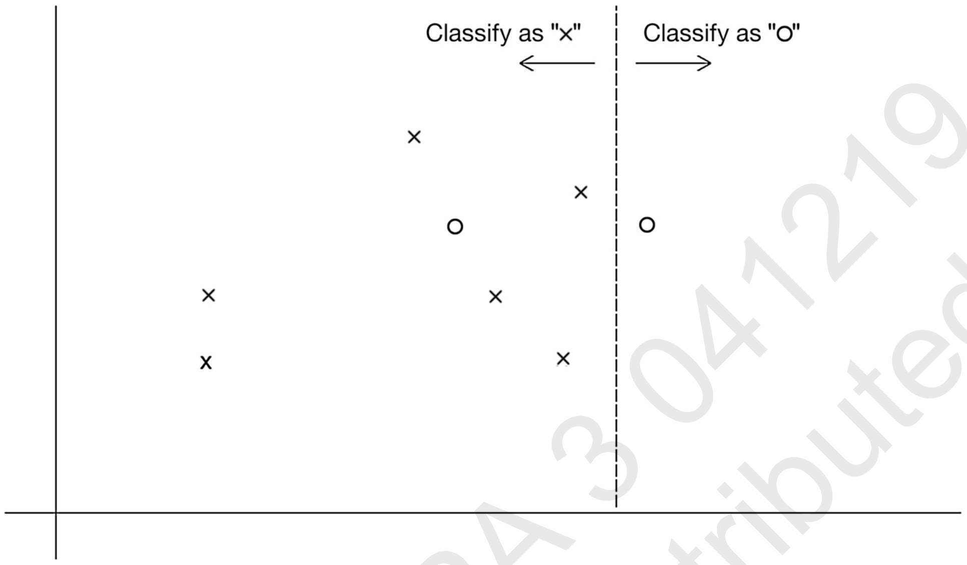


FIGURE 5.9 CLASSIFICATION ASSUMING EQUAL COSTS OF MISCLASSIFICATION

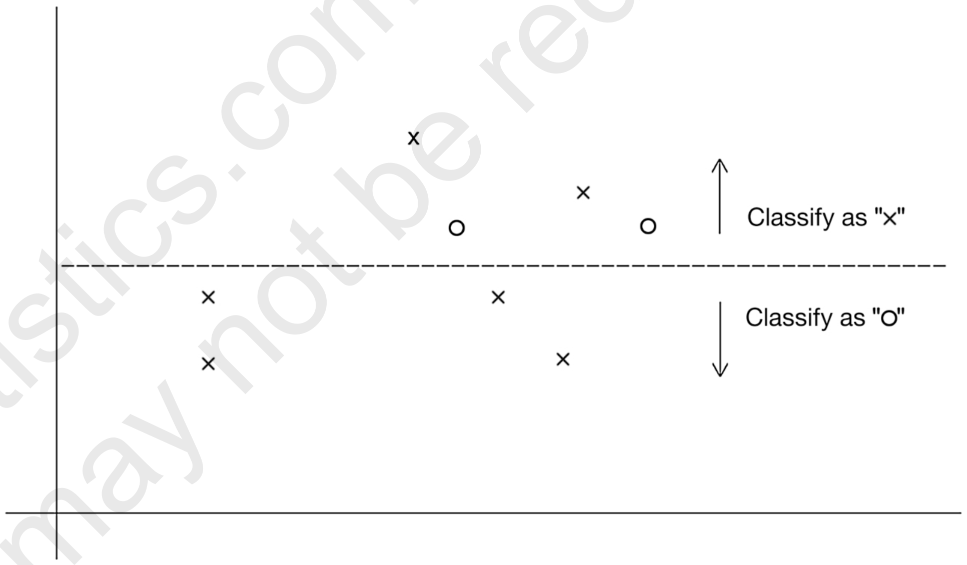


FIGURE 5.10 CLASSIFICATION ASSUMING UNEQUAL COSTS OF MISCLASSIFICATION

as costly as failing to catch an ×—the costs of misclassification jump to 5. In such a case, a horizontal line as shown in Figure 5.10, does a better job: It results in misclassification costs of just 2.

Oversampling is one way of incorporating these costs into the training process. In Figure 5.11, we can see that classification algorithms would automatically

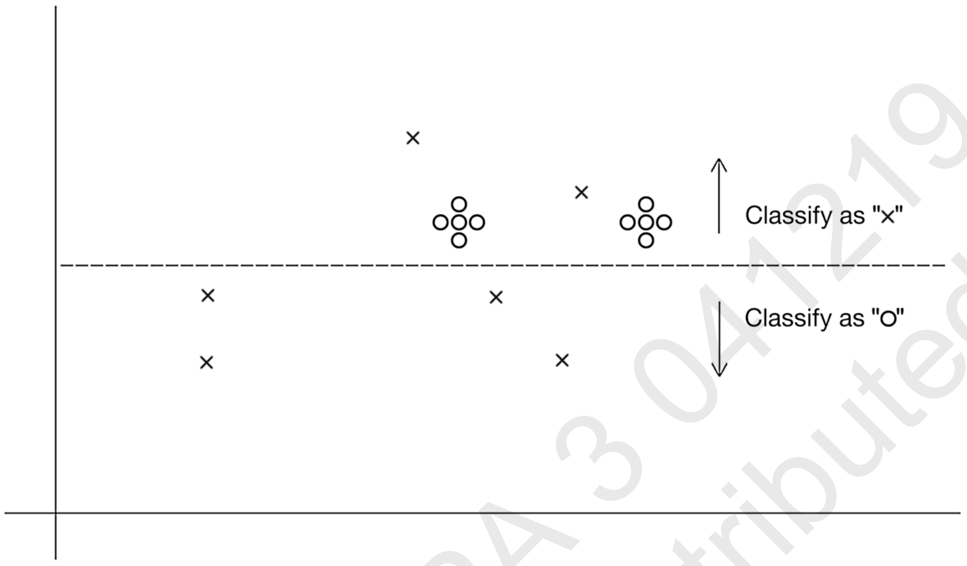


FIGURE 5.11 CLASSIFICATION USING OVERSAMPLING TO ACCOUNT FOR UNEQUAL COSTS

determine the appropriate classification line if four additional ◦'s were present at each existing ◦. We can achieve appropriate results either by taking five times as many ◦'s as we would get from simple random sampling (by sampling with replacement if necessary), or by replicating the existing ◦'s fourfold.

Oversampling without replacement in accord with the ratio of costs (the first option above) is the optimal solution, but may not always be practical. There may not be an adequate number of responders to assure that there will be enough of them to fit a model if they constitute only a small proportion of the total. Also, it is often the case that our interest in discovering responders is known to be much greater than our interest in discovering non-responders, but the exact ratio of costs is difficult to determine. When faced with very low response rates in a classification problem, practitioners often sample equal numbers of responders and non-responders as a relatively effective and convenient approach. Whatever approach is used, when it comes time to assess and predict model performance, we will need to adjust for the oversampling in one of two ways:

1. score the model to a validation set that has been selected without oversampling (i.e., via simple random sampling), or

2. score the model to an oversampled validation set, and reweight the results to remove the effects of oversampling.

The first method is more straightforward and easier to implement. We describe how to oversample and how to evaluate performance for each of the two methods.

When classifying data with very low response rates, practitioners typically:

- train models on data that are 50% responder, 50% non-responder
- validate the models with an unweighted (simple random) sample from the original data

## Oversampling the Training Set

How is weighted sampling done? When responders are sufficiently scarce that you will want to use all of them, one common procedure is as follows:

1. First, the response and non-response data are separated into two distinct sets, or *strata*.
2. Records are then randomly selected for the training set from each stratum. Typically, one might select half the (scarce) responders for the training set, then an equal number of non-responders.
3. The remaining responders are put in the validation set.
4. Non-responders are randomly selected for the validation set in sufficient numbers to maintain the original ratio of responders to non-responders.
5. If a test set is required, it can be taken randomly from the validation set.

## Evaluating Model Performance Using a Non-oversampled Validation Set

Although the oversampled data can be used to train models, they are often not suitable for evaluating model performance, because the number of responders will (of course) be exaggerated. The most straightforward way of gaining an unbiased estimate of model performance is to apply the model to regular data (i.e., data not oversampled). In short, train the model on oversampled data, but validate it with regular data.

## Evaluating Model Performance if Only Oversampled Validation Set Exists

In some cases, very low response rates may make it more practical to use oversampled data not only for the training data, but also for the validation data. This might happen, for example, if an analyst is given a dataset for exploration and prototyping that is already oversampled to boost the proportion with the rare response of interest (perhaps because it is more convenient to transfer and work with a smaller dataset). In such cases, it is still possible to assess how well the model will do with real data, but this requires the oversampled validation set to be reweighted, in order to restore the class of records that were underrepresented in the sampling process. This adjustment should be made to the confusion matrix and to the lift chart in order to derive good accuracy measures. These adjustments are described next.

**I. Adjusting the Confusion Matrix for Oversampling** Suppose the response rate in the data as a whole is 2%, and that the data were oversampled, yielding a sample in which the response rate is 25 times higher (50% responders). The relationship is as follows:

**Responders:** 2% of the whole data; 50% of the sample

**Non-responders:** 98% of the whole data, 50% of the sample

Each responder in the whole data is worth 25 responders in the sample (50/2). Each non-responder in the whole data is worth 0.5102 non-responders in the sample (50/98). We call these values *oversampling weights*.

Assume that the validation confusion matrix looks like this:

**CONFUSION MATRIX, OVERSAMPLED DATA (VALIDATION)**

| | Predicted 0 | Predicted 1 | Total |
|---|---|---|---|
| **Actual 0** | 390 | 110 | 500 |
| **Actual 1** | 80 | 420 | 500 |
| **Total** | 470 | 530 | 1000 |

At this point, the misclassification rate appears to be $(80 + 110)/1000 =$ 19%, and the model ends up classifying 53% of the records as 1's. However, this reflects the performance on a sample where 50% are responders.

To estimate predictive performance when this model is used to score the original population (with 2% responders), we need to undo the effects of the oversampling. The actual number of responders must be divided by 25, and the actual number of non-responders divided by 0.5102.

The revised confusion matrix is as follows:

**CONFUSION MATRIX, REWEIGHTED**

| | Predicted 0 | Predicted 1 | Total |
|---|---|---|---|
| **Actual 0** | $390/0.5102 = 764.4$ | $110/0.5102 = 215.6$ | 980 |
| **Actual 1** | $80/25 = 3.2$ | $420/25 = 16.8$ | 20 |
| **Total** | | | 1,000 |

The adjusted misclassification rate is $(3.2 + 215.6)/1000 = 21.9\%$. The model ends up classifying $(215.6 + 16.8)/1000 = 23.24\%$ of the records as 1's, when we assume 2% responders.

**II. Adjusting the Cumulative Gains Curve for Oversampling** Lift and the cumulative gains curve are likely to be more useful measures in low-response situations, where our interest lies not so much in classifying all the records correctly as in finding a model that guides us toward those records most likely to contain the response of interest (under the assumption that scarce resources preclude examining or contacting all the records). Typically, our interest in such a case is in maximizing value or minimizing cost, so we will show the adjustment process incorporating the benefit/cost element. The following procedure can be used:

1. Sort the validation records in order of the predicted probability of success (where success = belonging to the class of interest).
2. For each record, record the cost (benefit) associated with the actual outcome.
3. Divide that value by the oversampling rate. For example, if responders are overweighted by a factor of 25, divide by 25.
4. For the highest probability (i.e., first) record, the value above is the $y$-coordinate of the first point on the cumulative gains curve. The $x$-coordinate is index number 1.
5. For the next record, again calculate the adjusted value associated with the actual outcome. Add this to the adjusted cost (benefit) for the previous record. This sum is the $y$-coordinate of the second point on the cumulative gains curve. The $x$-coordinate is index number 2.
6. Repeat Step 5 until all records have been examined. Connect all the points, and this is the cumulative gains curve.
7. The reference line is a straight line from the origin to the point $y$ = total net benefit and $x = n$ ($n$ = number of records).

## PROBLEMS

**5.1** A data mining routine has been applied to a transaction dataset and has classified 88 records as fraudulent (30 correctly so) and 952 as non-fraudulent (920 correctly so). Construct the confusion matrix and calculate the overall error rate.

**5.2** Suppose that this routine has an adjustable cutoff (threshold) mechanism by which you can alter the proportion of records classified as fraudulent. Describe how moving the cutoff up or down would affect

**a.** the classification error rate for records that are truly fraudulent

**b.** the classification error rate for records that are truly nonfraudulent

**5.3** FiscalNote is a startup founded by a Washington, DC entrepreneur and funded by a Singapore sovereign wealth fund, the Winklevoss twins of Facebook fame, and others. It uses machine learning and data mining techniques to predict for its clients whether legislation in the US Congress and in US state legislatures will pass or not. The company reports 94% accuracy. (*Washington Post*, November 21, 2014, "Capital Business")

Considering just bills introduced in the US Congress, do a bit of internet research to learn about numbers of bills introduced and passage rates. Identify the possible types of misclassifications, and comment on the use of overall accuracy as a metric. Include a discussion of other possible metrics and the potential role of propensities.

**5.4** Consider Figure 5.12, the decile lift chart for the transaction data model, applied to new data.

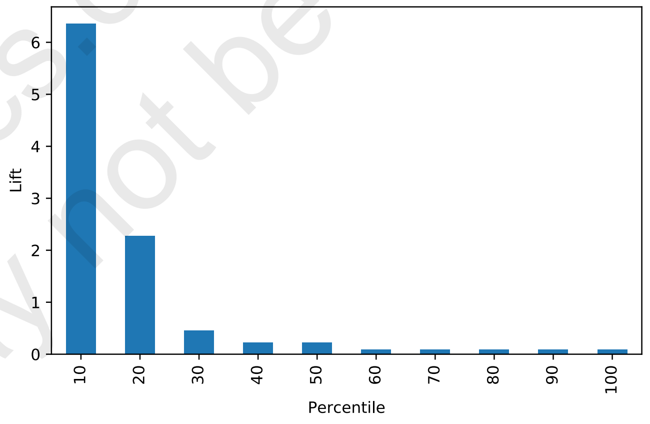


FIGURE 5.12 DECILE LIFT CHART FOR TRANSACTION DATA

**a.** Interpret the meaning of the first and second bars from the left.

**b.** Explain how you might use this information in practice.

**c.** Another analyst comments that you could improve the accuracy of the model by classifying everything as nonfraudulent. If you do that, what is the error rate?

**d.** Comment on the usefulness, in this situation, of these two metrics of model performance (error rate and lift).

**5.5** A large number of insurance records are to be examined to develop a model for predicting fraudulent claims. Of the claims in the historical database, 1% were judged to be fraudulent. A sample is taken to develop a model, and oversampling is used to provide a balanced sample in light of the very low response rate. When applied to this sample ($n = 800$), the model ends up correctly classifying 310 frauds, and 270 nonfrauds. It missed 90 frauds, and classified 130 records incorrectly as frauds when they were not.

**a.** Produce the confusion matrix for the sample as it stands.

**b.** Find the adjusted misclassification rate (adjusting for the oversampling).

**c.** What percentage of new records would you expect to be classified as fraudulent?

**5.6** A firm that sells software services has been piloting a new product and has records of 500 customers who have either bought the services or decided not to. The target value is the estimated profit from each sale (excluding sales costs). The global mean is about \$2500. However, the cost of the sales effort is not cheap—the company figures it comes to \$2500 for each of the 500 customers (whether they buy or not). The firm developed a predictive model in hopes of being able to identify the top spenders in the future. The cumulative gains and decile lift charts for the validation set are shown in Figure 5.13.

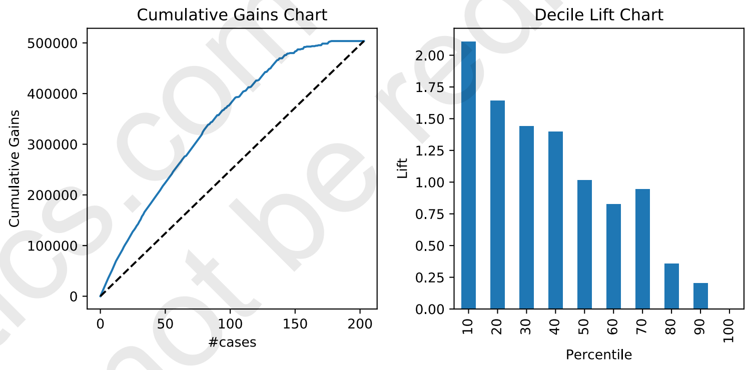


FIGURE 5.13 CUMULATIVE GAINS AND DECILE LIFT CHARTS FOR SOFTWARE SERVICES PRODUCT SALES (VALIDATION SET)

**a.** If the company begins working with a new set of 1000 leads to sell the same services, similar to the 500 in the pilot study, without any use of predictive modeling to target sales efforts, what is the estimated profit?

**b.** If the firm wants the average profit on each sale to roughly double the sales effort cost, and applies an appropriate cutoff with this predictive model to a new set of 1000 leads, how far down the new list of 1000 should it proceed (how many deciles)?

**c.** Still considering the new list of 1000 leads, if the company applies this predictive model with a lower cutoff of \$2500, how far should it proceed down the ranked leads, in terms of deciles?

**d.** Why use this two-stage process for predicting sales—why not simply develop a model for predicting profit for the 1000 new leads?

**5.7** Table 5.7 shows a small set of predictive model validation results for a classification model, with both actual values and propensities.

**a.** Calculate error rates, sensitivity, and specificity using cutoffs of 0.25, 0.5, and 0.75.

**b.** Create a decile lift chart.

TABLE 5.7 PROPENSITIES AND ACTUAL CLASS MEMBERSHIP FOR VALIDATION DATA

| Propensity of 1 | Actual |
|---|---|
| 0.03 | 0 |
| 0.52 | 0 |
| 0.38 | 0 |
| 0.82 | 1 |
| 0.33 | 0 |
| 0.42 | 0 |
| 0.55 | 1 |
| 0.59 | 0 |
| 0.09 | 0 |
| 0.21 | 0 |
| 0.43 | 0 |
| 0.04 | 0 |
| 0.08 | 0 |
| 0.13 | 0 |
| 0.01 | 0 |
| 0.79 | 1 |
| 0.42 | 0 |
| 0.29 | 0 |
| 0.08 | 0 |
| 0.02 | 0 |