Chapter 10
Case Study: Compressive Strength of Concrete Mixtures

Thus far, the focus has been on observational data sets where the values of the predictors were not pre-specified. For example, the QSAR data used in the previous chapters involved a collection of diverse compounds that captured a sufficient amount of the "chemical space." This particular data set was not created by specifying exact values for the chemical properties (such as molecular weight). Instead compounds were sampled from an appropriate population for use in the model.

Designed experiments are created by planning the exact values of the predictors (referred to as the factors in this context) using some sort of strategic methodology. The configurations of predictor settings are created so that they have good mathematical and experimental properties. One such property is *balance*. A balanced design is one where no one experimental factor (i.e., the predictors) has more focus than the others. In most cases, this means that each predictor has the same number of possible levels and that the frequencies of the levels are equivalent for each factor. The properties used to choose the best experimental design are driven by the stage of experimentation.

Box et al. (1978) popularized the concept of *sequential experimentation* where a large number of possible experimental factors are screened with low resolution (i.e., "casting a wide net") to determine the *active* or important factors that relate to the outcome. Once the importance of the predictors are quantified, more focused experiments are created with the subset of important factors. In subsequent experiments, the nature of the relationship between the important factors can be further elucidated. The last step in the sequence of experiments is to fine-tune a small number of important factors. *Response surface experiments* (Myers and Montgomery 2009) use a smaller set of predictor values. Here, the primary goal is to optimize the experimental settings based on a nonlinear model of the experimental predictors.

DOI 10.1007/978-1-4614-6849-3_10,

Designed experiments and predictive models have several differences[1]:

- A sequence of studies is preferred over a single, comprehensive data set that attempts to include all possible predictors (i.e., experimental factors) with many values per predictor. The iterative paradigm of planning, designing, and then analyzing an experiment is, on the surface, different than most predictive modeling problems.
- Until the final stages of sequential experimentation, the focus is on understanding which predictors affect the outcome and how. Once response surface experiments are utilized, the focus of the activities is solely about prediction.

This case study will focus on the prediction of optimal formulations of concrete mixture-based data from designed experiments.

Concrete is an integral part of most industrialized societies. It is used to some extent in nearly all structures and in many roads. One of the main properties of interest (beside cost) is the compressive strength of the hardened concrete. The composition of many concretes includes a number of dry ingredients which are mixed with water and then are allowed to dry and harden. Given its abundance and critical role in infrastructure, the composition is important and has been widely studied. In this chapter, models will be created to help find potential recipes to maximize compressive strength.

Yeh (2006) describes a standard type of experimental setup for this scenario called a *mixture design* (Cornell 2002; Myers and Montgomery 2009). Here, boundaries on the upper and lower limits on the mixture *proportion* for each ingredient are used to create multiple mixtures that methodically fill the space within the boundaries. For a specific type of mixture design, there is a corresponding linear regression model that is typically used to model the relationship between the ingredients and the outcome. These linear models can include interaction effects and higher-order terms for the ingredients. The ingredients used in Yeh (2006) were:

- Cement (kg/m^3)
- Fly ash (kg/m^3), small particles produced by burning coal
- Blast furnace slag (kg/m^3)
- Water (kg/m^3)

[1] There are cases where specialized types of experimental designs are utilized with predictive models. In the field of chemometrics, an orthogonal array-type design followed by the sequential elimination of level combination algorithm has been shown to improve QSAR models (Mandal et al. 2006, 2007). Also, the field of *active learning* sequentially added samples based on the training set using the predictive model results (Cohn et al. 1994; Saar-Tsechansky and Provost 2007a).

- Superplasticizer (kg/m^3), an additive that reduces particle aggregation
- Coarse aggregate (kg/m^3)
- Fine aggregate (kg/m^3)

Yeh (2006) also describes an additional non-mixture factor related to compressive strength: the age of the mixture (at testing). Since this is not an ingredient, it is usually referred to as a *process factor*. Specific experimental designs (and linear model forms) exist for experiments that combine mixture and process variables (see Cornell (2002) for more details).

Yeh (1998) takes a different approach to modeling concrete mixture experiments. Here, separate experiments from 17 sources with common experimental factors were combined into one "meta-experiment" and the author used neural networks to create predictive models across the whole mixture space. Age was also included in the model. The public version of the data set includes 1030 data points across the different experiments, although Yeh (1998) states that some mixtures were removed from his analysis due to nonstandard conditions. There is no information regarding exactly which mixtures were removed, so the analyses here will use all available data points. Table 10.1 shows a summary of the predictor data (in amounts) and the outcome.

Figure 10.1 shows scatter plots of each predictor versus the compressive strength. Age shows a strong nonlinear relationship with the predictor, and the cement amount has a linear relationship. Note that several of the ingredients have a large frequency of a single amount, such as zero for the superplasticizer and the amount of fly ash. In these cases, the compressive strength varies widely for those values of the predictors. This might indicate that some of the partitioning methods, such as trees or MARS, may be able to isolate these mixtures within the model and more effectively predict the compressive strength. For example, there are 53 mixtures with no superplasticizer or fly ash but with exactly 228 kg/m^3 of water. This may represent an important sub-population of mixtures that may benefit from a model that is specific to these types of mixtures. A tree- or rule-based model has the ability to model such a sub-group while classical regression models would not.

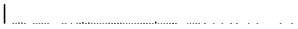
Although the available data do not denote which formulations came from each source, there are 19 distinct mixtures with replicate data points. The majority of these mixtures had only two or three duplicate conditions, although some conditions have as many as four replicates. When modeling these data, the replicate results should not be treated as though they are independent observations. For example, having replicate mixtures in both the training and test sets can result in overly optimistic assessments of how well the model works. A common approach here is to average the outcomes within each unique mixture. Consequentially, the number of mixtures available for modeling drops from 1030 to 992.

Table 10.1: Data for the concrete mixtures

```
                    9 Variables     1030  Observations
```

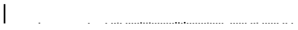
```
Cement
     n missing unique Mean   0.05  0.10  0.25  0.50  0.75  0.90  0.95
 1,030       0    278 281.2 143.7 153.5 192.4 272.9 350.0 425.0 480.0

lowest : 102.0 108.3 116.0 122.6 132.0
highest: 522.0 525.0 528.0 531.3 540.0
```



```
BlastFurnaceSlag
     n missing unique Mean 0.05 0.10 0.25 0.50  0.75  0.90  0.95
 1,030       0    185  73.9  0.0  0.0  0.0 22.0 142.9 192.0 236.0

lowest :   0.0  11.0  13.6  15.0  17.2
highest: 290.2 305.3 316.1 342.1 359.4
```

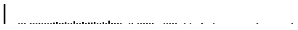


```
FlyAsh
     n missing unique Mean 0.05 0.10 0.25 0.50  0.75  0.90  0.95
 1,030       0    156 54.19  0.0  0.0  0.0  0.0 118.3 141.1 167.0

lowest :   0.0  24.5  59.0  60.0  71.0
highest: 194.0 194.9 195.0 200.0 200.1
```

```
Water
    n missing unique Mean  0.05  0.10  0.25  0.50  0.75  0.90  0.95
 1030       0    195 181.6 146.1 154.6 164.9 185.0 192.0 203.5 228.0

lowest : 121.8 126.6 127.0 127.3 137.8
highest: 228.0 236.7 237.0 246.9 247.0
```



```
Superplasticizer
     n missing unique Mean 0.05 0.10 0.25 0.50  0.75  0.90  0.95
 1,030       0    111 6.205 0.00 0.00 0.00 6.40 10.20 12.21 16.05

lowest :  0.0  1.7  1.9  2.0  2.2,
highest: 22.0 22.1 23.4 28.2 32.2
```

```
CoarseAggregate
     n missing unique Mean   0.05  0.10  0.25  0.50   0.75   0.90   0.95
 1,030       0    284 972.9 842.0 852.1 932.0 968.0 1029.4 1076.5 1104.0

lowest :  801.0  801.1  801.4  811.0  814.0
highest: 1124.4 1125.0 1130.0 1134.3 1145.0
```

```
FineAggregate
    n missing unique Mean  0.05  0.10  0.25   .50  0.75   .90  0.95
 1030       0    302 773.6 613.0 664.1 730.9 779.5 824.0 880.8 898.1

lowest : 594.0 605.0 611.8 612.0 613.0
highest: 925.7 942.0 943.1 945.0 992.6
```

```
Age
     n missing unique Mean 0.05 0.10 0.25 0.50 0.75 0.90 0.95
 1,030       0     14 45.66    3    3    7   28   56  100  180

            1   3   7 14  28 56 90 91 100 120 180 270 360 365
Frequency   2 134 126 62 425 91 54 22  52   3  26  13   6  14
%           0  13  12  6  41  9  5  2   5   0   3   1   1   1
```

```
CompressiveStrength
     n missing unique Mean  0.05  0.10  0.25  0.50  0.75  0.90  0.95
 1,030       0    845 35.82 10.96 14.20 23.71 34.45 46.14 58.82 66.80

lowest :  2.33  3.32  4.57  4.78  4.83
highest: 79.40 79.99 80.20 81.75 82.60
```

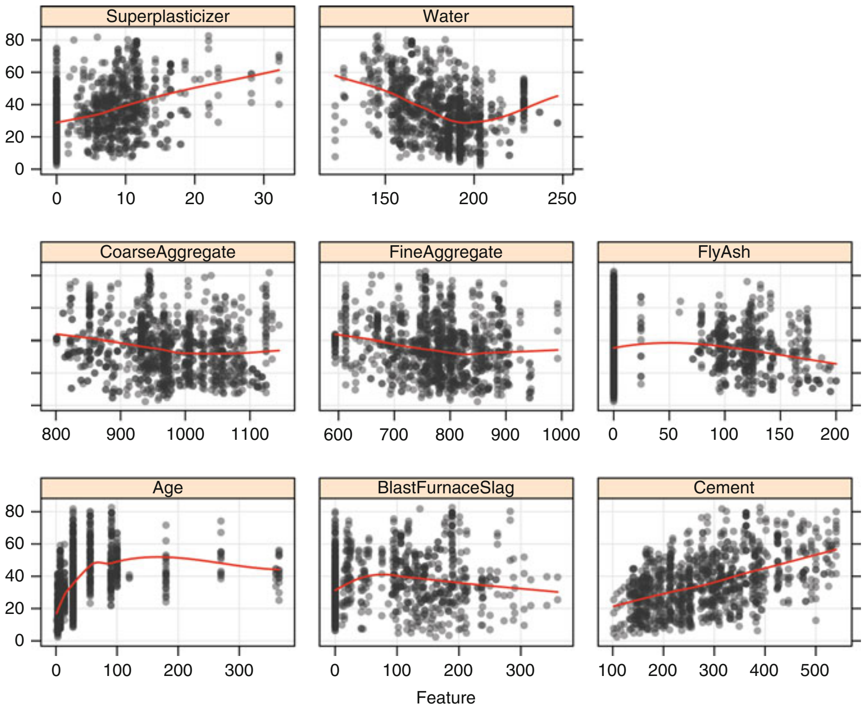


Fig. 10.1: Scatter plots of the concrete predictors versus the compressive strength

10.1 Model Building Strategy

The neural network models used in Yeh (1998) were single-layer networks with eight hidden units. Multiple data splitting approaches were used by the original author. Four models were fit with different training sets such that all the data from a single source were held out each time. These models resulted in test set R^2 values ranging from 0.814 to 0.895. They also used a random sample of 25 % of the data for holdout test sets. This was repeated four times to produce test set R^2 values between 0.908 and 0.922.

Although an apples-to-apples comparison cannot be made with the analyses of Yeh (1998), a similar data splitting approach will be taken for this case study. A random holdout set of 25 % ($n = 247$) will be used as a test set and five repeats of 10-fold cross-validation will be used to tune the various models.

In this case study, a series of models will be created and evaluated. Once a final model is selected, the model will be used to predict mixtures with optimal compressive strength within practical limitations.

How should the predictors be used to model the outcome? Yeh (1998) discusses traditional approaches, such as relying on the water-to-cement ratio, but suggests that the existing experimental data are not consistent with historical strategies. In this chapter, the predictors will enter the models as the proportion of the total amount. Because of this, there is a built-in dependency in the predictor values (any predictor can be automatically determined by knowing the values of the other seven). Despite this, the pairwise correlations are not large, and, therefore, we would not expect methods that are designed to deal with collinearity (e.g., PLS, ridge regression) to have performance that is superior to other models.

A suite of models were tested:

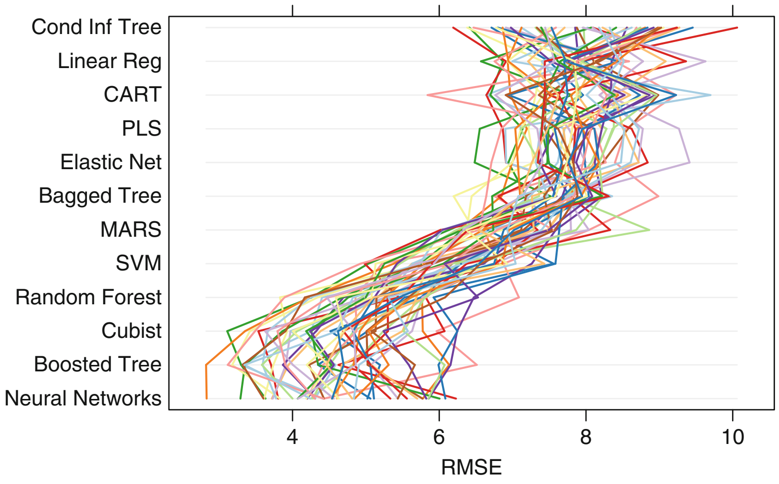
- Linear regression, partial least squares, and the elastic net. Each model used an expanded set of predictors that included all two-factor interactions (e.g., age $\times$ water) and quadratic terms.
- Radial basis function support vector machines (SVMs).
- Neural network models.
- MARS models.
- Regression trees (both CART and conditional inference trees), model trees (with and without rules), and Cubist (with and without committees and neighbor-based adjustments).
- Bagged and boosted regression trees, along with random forest models.

The details of how the models were tuned are given in the Computing section at the end of the chapter.

10.2 Model Performance

The same cross-validation folds were used for each model. Figure 10.2 shows *parallel-coordinate plots* for the resampling results across the models. Each line corresponds to a common cross-validation holdout. From this, the top performing models were tree ensembles (random forest and boosting), rule ensembles (Cubist), and neural networks. Linear models and simple trees did not perform well. Bagged trees, SVMs, and MARS showed modest results but are clearly worse than the top cluster of models. The averaged R^2 statistics ranged from 0.76 to 0.92 across the models. The top three models (as ranked by resampling) were applied to the test set. The RMSE values are roughly consistent with the cross-validation rankings: 3.9 (boosted tree), 4.2 (neural networks), and 4.5 (cubist).

Figure 10.3 shows plots of the raw data, predictions, and residuals for the three models. The plots for each model are fairly similar; each shows good concordance between the observed and predicted values with a slight “fanning out” at the high end of compressive strength. The majority of the residuals



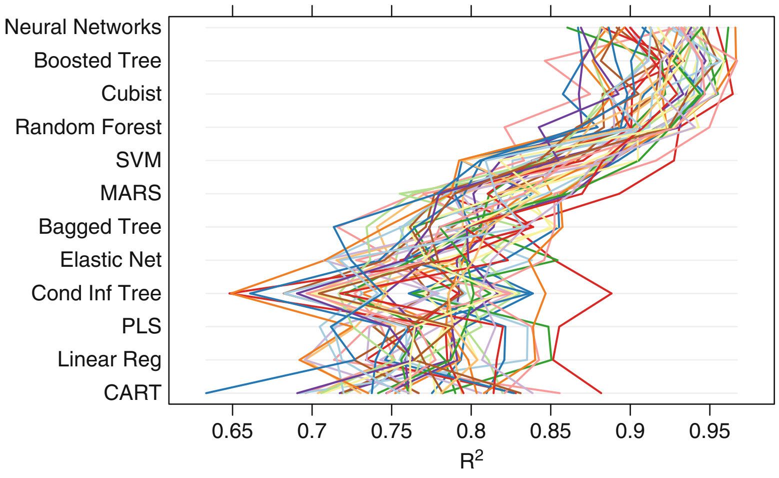


Fig. 10.2: Parallel coordinate plots for the cross-validated RMSE and R^2 across different models. Each line represents the results for a common cross-validation holdout set

are within ±2.8 MPa with the largest errors slightly more than 15 MPa. There is no clear winner or loser in the models based on these plots.

The neural network model used 27 hidden units with a weight decay value of 0.1. The performance profile for this model (not shown, but can be reproduced using syntax provided in the Computing section below) showed that weight decay had very little impact on the effectiveness of the model. The final Cubist model used 100 committees and adjusted the predictions with 3-nearest neighbors. Similar to the Cubist profiles shown for the

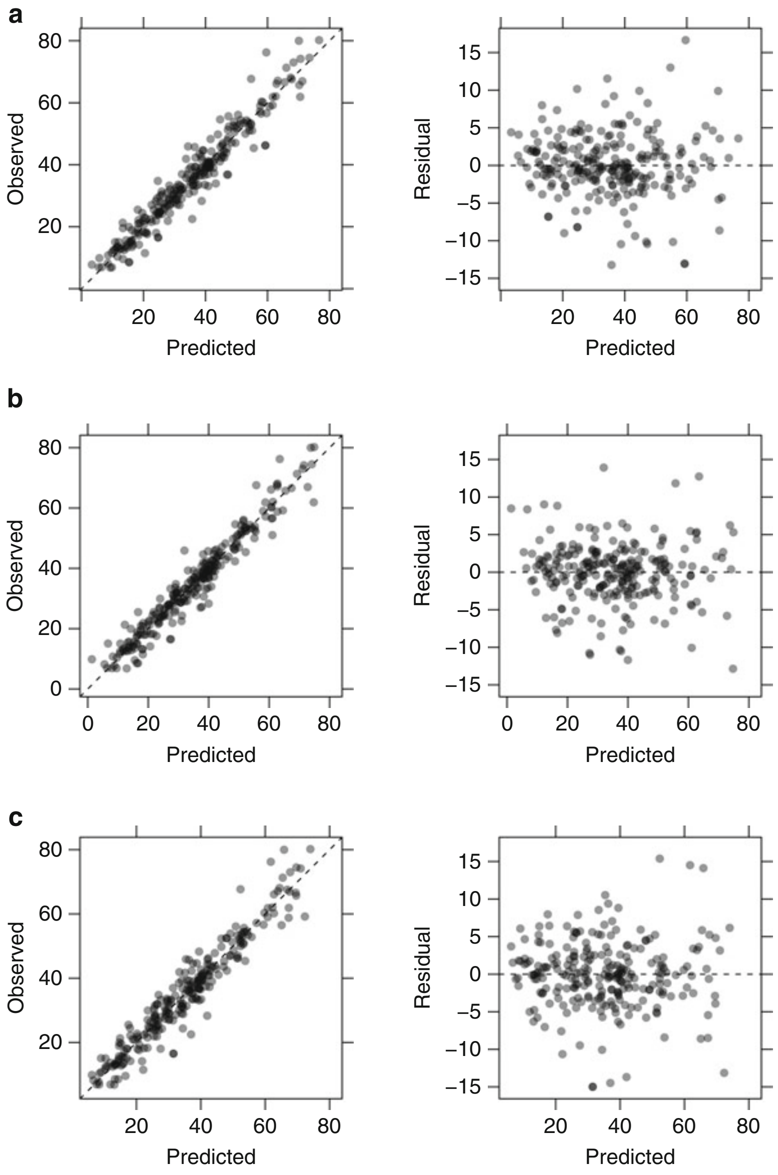


Fig. 10.3: Diagnostic plots of the test set results for three models. (**a**) Neural network (**b**) Boosted trees (**c**) Cubist

computational chemistry data (see the figure on page 211), performance suffered when the number of neighbors was either too low or too high. The boosted tree preferred a fast learning rate and deep trees.

10.3 Optimizing Compressive Strength

The neural network and Cubist models were used to determine possible mixtures with improved compressive strength. To do this, a numerical search routine can be used to find formulations with high compressive strength (as predicted by the model). Once a candidate set of mixtures is found, additional experiments would then be executed for the mixtures to verify that the strength has indeed improved. For illustrative purposes, the age of the formulation was fixed to a value of 28 days (there are a large number of data points in the training set with this value) and only the mixture ingredients will be optimized.

How, exactly, should the search be conducted? There are numerous numerical optimization routines that can search seven-dimensional space. Many rely on determining the gradient (i.e., first derivative) of the prediction equation. Several of the models have smooth prediction equations (e.g., neural networks and SVMs). However, others have many discontinuities (such as tree- and rule-based models and multivariate adaptive regression splines) that are not conducive to gradient-based search methods.

An alternative is to use a class of optimizers called *direct methods* that would not use derivatives to find the settings with optimal compressive strength and evaluate the prediction equation many more times than derivative-based optimizers. Two such search procedures are the Nelder–Mead simplex method (Nelder and Mead 1965; Olsson and Nelson 1975) and simulated annealing (Bohachevsky et al. 1986). Of these, the simplex search procedure had the best results for these data.[2] The Nelder–Mead method has the potential to get "stuck" in a sub-optimal region of the search space, which would generate poor mixtures. To counter-act this issue, it is common to repeat the search using different starting points and choosing the searches that are associated with the best results. To do this, 15–28-day-old mixtures were selected from the training set. The first of the 15 was selected at random and the remaining starting points were selected using the maximum dissimilarity sampling procedure discussed in Sect. 4.3.

Before beginning the search, constraints were used to avoid searching parts of the formulation space that were impractical or impossible. For example, the amount of water ranged from 5.1 % to 11.2 %. The search procedure was set to only consider mixtures with at least 5 % water.

[2] The reader can also try simulated annealing using the code at the end of the chapter.

Table 10.2: The top three optimal mixtures predicted from two models where the age was fixed at a value of 28. In the training set, matching on age, the strongest mixtures had compressive strengths of 81.75, 79.99, and 78.8

Model	Cement	Slag	Ash	Plast.	C. Agg.	F. Agg.	Water	Prediction
Cubist								
New mix 1	12.7	14.9	6.8	0.5	34.0	25.7	5.4	89.1
New mix 2	21.7	3.4	5.7	0.3	33.7	29.9	5.3	88.4
New mix 3	14.6	13.7	0.4	2.0	35.8	27.5	6.0	88.2
Neural network								
New mix 4	34.4	7.9	0.2	0.3	31.1	21.1	5.1	88.7
New mix 5	21.2	11.6	0.1	1.1	32.4	27.8	5.8	85.7
New mix 6	40.8	4.9	6.7	0.7	20.3	20.5	6.1	83.9

In the training set, there were 416 formulations that were tested at 28 days. Of these, the top three mixtures had compressive strengths of 81.75, 79.99, and 78.8. Table 10.2 shows the top three predicted mixtures for a smooth and non-smooth model (neural networks and Cubist, respectively). The models are able to find formulations that are predicted to have better strength than those seen in the data.

The Cubist mixtures were predicted to have similar compressive strengths. Their formulations were differentiated by the cement, slag, ash, and plasticizer components. The neural network mixtures were in a nearby region of mixture space and had predicted values that were lower than the Cubist model predictions but larger than the best-observed mixtures. In each of the six cases, the mixtures have very low proportions of water. Principal component analysis was used to represent the training set mixture (in seven-dimensional space) using two components. A PCA plot of the 28-day data is shown in Fig. 10.4. The principal component values for the 15 mixtures used as starting points for the search procedure are shown (as $\times$ symbols) as are the other 401 time-matched data points in the training set (shown as small grey dots). The top three predictions from the two models are also shown. Many of the predicted mixtures are near the outskirts of the mixture space and are likely to suffer some model inaccuracy due to extrapolation. Given this, it is extremely important to validate these new formulations scientifically and experimentally.

More complex approaches to finding optimal mixtures can also be used. For example, it may be important to incorporate the cost of the mixture (or other factors) into the search. Such a *multivariate* or *multiparameter* optimization can be executed a number of ways. One simple approach is *desirability functions* (Derringer and Suich 1980; Costa et al. 2011). Here, the impor-

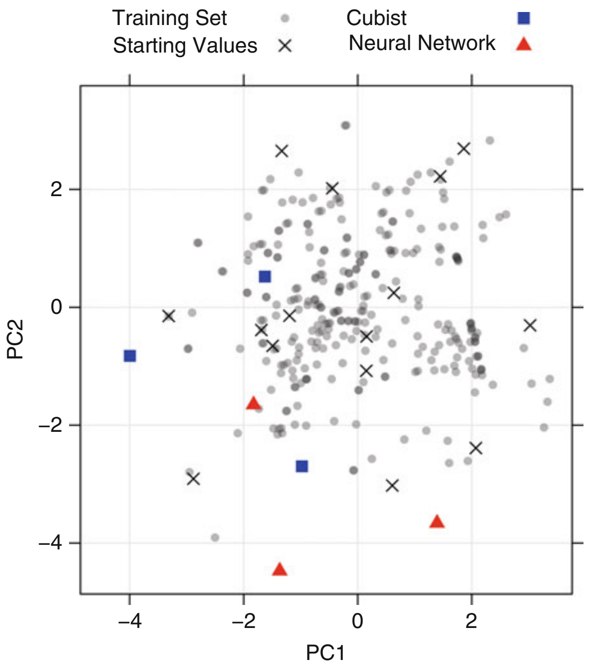


Fig. 10.4: A PCA plot of the training set data where the mixtures were aged 28 days. The search algorithm was executed across 15 different training set mixtures (shown as × in the plot). The top three optimal mixtures predicted from two models are also shown

tant characteristics of a mixture (e.g., strength and cost) are mapped to a common desirability scale between 0 and 1, where one is most desirable and zero is completely undesirable. For example, mixtures above a certain cost may be unacceptable. Mixtures associated with costs at or above this value would have zero desirability (literally). As the cost decreases the relationship between cost and desirability might be specified to be linearly decreasing. Figure 10.5 shows two hypothetical examples of desirability function for cost and strength. Here, formulations with costs greater than 20 and strength less than 70 are considered completely unacceptable. Once desirability functions are created by the user for every characteristic to be optimized, the overall desirability is combined, usually using a geometric mean. Note that, since the geometric mean multiplies values, if any one desirability function has a score of 0, all other characteristics would be considered irrelevant (since the overall value is also 0). The overall desirability would be optimized by a search procedure to find a solution that takes all the characteristics into account. Wager et al. (2010) and Cruz-Monteagudo et al. (2011) show examples of this approach.

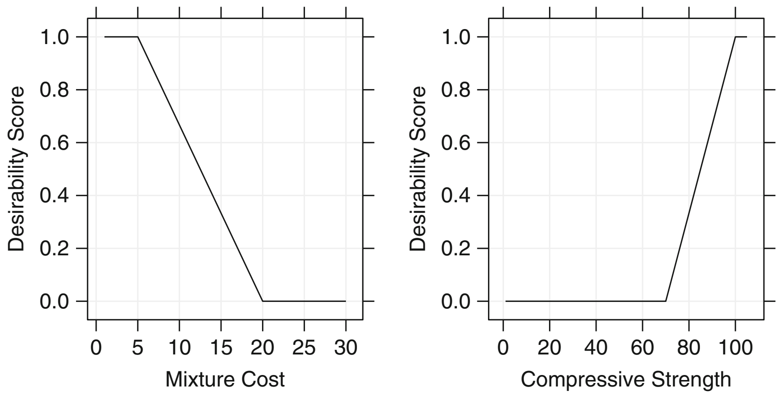


Fig. 10.5: Examples of individual desirability functions for mixture cost and compression strength. The geometric mean of these scores can be optimized for the purpose of finding strong, low-cost mixtures

10.4 Computing

This section uses functions from the caret, desirability, Hmisc, and plyr packages.

The concrete data can be found in the UCI Machine Learning repository. The AppliedPredictiveModeling package contains the original data (in amounts) and an alternate version that has the mixture proportions:

```
> library(AppliedPredictiveModeling)
> data(concrete)
> str(concrete)

  'data.frame':   1030 obs. of  9 variables:
   $ Cement              : num  540 540 332 332 199 ...
   $ BlastFurnaceSlag    : num  0 0 142 142 132 ...
   $ FlyAsh              : num  0 0 0 0 0 0 0 0 0 0 ...
   $ Water               : num  162 162 228 228 192 228 228 228 228 228 ...
   $ Superplasticizer    : num  2.5 2.5 0 0 0 0 0 0 0 0 ...
   $ CoarseAggregate     : num  1040 1055 932 932 978 ...
   $ FineAggregate       : num  676 676 594 594 826 ...
   $ Age                 : int  28 28 270 365 360 90 365 28 28 28 ...
   $ CompressiveStrength: num  80 61.9 40.3 41 44.3 ...

> str(mixtures)

  'data.frame':   1030 obs. of  9 variables:
   $ Cement              : num  0.2231 0.2217 0.1492 0.1492 0.0853 ...
   $ BlastFurnaceSlag    : num  0 0 0.0639 0.0639 0.0569 ...
   $ FlyAsh              : num  0 0 0 0 0 0 0 0 0 0 ...
   $ Water               : num  0.0669 0.0665 0.1023 0.1023 0.0825 ...
   $ Superplasticizer    : num  0.00103 0.00103 0 0 0 ...
   $ CoarseAggregate     : num  0.43 0.433 0.418 0.418 0.42 ...
```

```
 $ FineAggregate      : num  0.279 0.278 0.266 0.266 0.355 ...
 $ Age                : int  28 28 270 365 360 90 365 28 28 28 ...
 $ CompressiveStrength: num  80 61.9 40.3 41 44.3 ...
```

Table 10.1 was created using the `describe` function in the Hmisc package, and Fig. 10.1 was create using the `featurePlot` function in caret:

```
> featurePlot(x = concrete[, -9],
+             y = concrete$CompressiveStrength,
+             ## Add some space between the panels
+             between = list(x = 1, y = 1),
+             ## Add a background grid ('g') and a smoother ('smooth')
+             type = c("g", "p", "smooth"))
```

The code for averaging the replicated mixtures and splitting the data into training and test sets is

```
> averaged <- ddply(mixtures,
+                   .(Cement, BlastFurnaceSlag, FlyAsh, Water,
+                     Superplasticizer, CoarseAggregate,
+                     FineAggregate, Age),
+                   function(x) c(CompressiveStrength =
+                                 mean(x$CompressiveStrength)))
> set.seed(975)
> forTraining <- createDataPartition(averaged$CompressiveStrength,
+                                    p = 3/4)[[1]]
> trainingSet <- averaged[ forTraining,]
> testSet  <- averaged[-forTraining,]
```

To fit the linear models with the expanded set of predictors, such as interactions, a specific model formula was created. The dot in the formula below is shorthand for all predictors and `(.)^2` expands into a model with all the linear terms and all two-factor interactions. The quadratic terms are created manually and are encapsulated inside the `I()` function. This "as-is" function tells R that the squaring of the predictors should be done arithmetically (and not symbolically).

The formula is first created as a character string using the `paste` command, then is converted to a bona fide R formula.

```
> modFormula <- paste("CompressiveStrength ~ (.)^2 + I(Cement^2) + ",
+                     "I(BlastFurnaceSlag^2) + I(FlyAsh^2) + I(Water^2) +",
+                     " I(Superplasticizer^2) + I(CoarseAggregate^2) + ",
+                     "I(FineAggregate^2) + I(Age^2)")
> modFormula <- as.formula(modFormula)
```

Each model used repeated 10-fold cross-validation and is specified with the `trainControl` function:

```
> controlObject <- trainControl(method = "repeatedcv",
+                               repeats = 5,
+                               number = 10)
```

To create the exact same folds, the random number generator is reset to a common seed prior to running `train`. For example, to fit the linear regression model:

```
> set.seed(669)
> linearReg <- train(modFormula,
+                    data = trainingSet,
+                    method = "lm",
+                    trControl = controlObject)
> linearReg
  745 samples
   44 predictors

  No pre-processing
  Resampling: Cross-Validation (10-fold, repeated 5 times)

  Summary of sample sizes: 671, 671, 672, 670, 669, 669, ...

  Resampling results

    RMSE  Rsquared  RMSE SD  Rsquared SD
    7.85  0.771     0.647    0.0398
```

The output shows that 44 predictors were used, indicating the expanded model formula was used.

The other two linear models were created with:

```
> set.seed(669)
> plsModel <- train(modForm, data = trainingSet,
+                   method = "pls",
+                   preProc = c("center", "scale"),
+                   tuneLength = 15,
+                   trControl = controlObject)
> enetGrid <- expand.grid(.lambda = c(0, .001, .01, .1),
+                         .fraction = seq(0.05, 1, length = 20))
> set.seed(669)
> enetModel <- train(modForm, data = trainingSet,
+                    method = "enet",
+                    preProc = c("center", "scale"),
+                    tuneGrid = enetGrid,
+                    trControl = controlObject)
```

MARS, neural networks, and SVMs were created as follows:

```
> set.seed(669)
> earthModel <- train(CompressiveStrength ~ ., data = trainingSet,
+                     method = "earth",
+                     tuneGrid = expand.grid(.degree = 1,
+                                            .nprune = 2:25),
+                     trControl = controlObject)
> set.seed(669)
> svmRModel <- train(CompressiveStrength ~ ., data = trainingSet,
+                    method = "svmRadial",
+                    tuneLength = 15,
+                    preProc = c("center", "scale"),
+                    trControl = controlObject)
> nnetGrid <- expand.grid(.decay = c(0.001, .01, .1),
+                         .size = seq(1, 27, by = 2),
+                         .bag = FALSE)
> set.seed(669)
> nnetModel <- train(CompressiveStrength ~ .,
+                    data = trainingSet,
+                    method = "avNNet",
+                    tuneGrid = nnetGrid,
+                    preProc = c("center", "scale"),
+                    linout = TRUE,
+                    trace = FALSE,
+                    maxit = 1000,
+                    trControl = controlObject)
```

The regression and model trees were similarly created:

```
> set.seed(669)
> rpartModel <- train(CompressiveStrength ~ .,
+                     data = trainingSet,
+                     method = "rpart",
+                     tuneLength = 30,
+                     trControl = controlObject)


> set.seed(669)
> ctreeModel <- train(CompressiveStrength ~ .,
+                     data = trainingSet,
+                     method = "ctree",
+                     tuneLength = 10,
+                     trControl = controlObject)


> set.seed(669)
> mtModel <- train(CompressiveStrength ~ .,
+                  data = trainingSet,
+                  method = "M5",
+                  trControl = controlObject)
```

The following code creates the remaining model objects:

```
> set.seed(669)
> treebagModel <- train(CompressiveStrength ~ .,
+                       data = trainingSet,
+                       method = "treebag",
+                       trControl = controlObject)
> set.seed(669)
> rfModel <- train(CompressiveStrength ~ .,
+                  data = trainingSet,
+                  method = "rf",
+                  tuneLength = 10,
+                  ntrees = 1000,
+                  importance = TRUE,
+                  trControl = controlObject)
> gbmGrid <- expand.grid(.interaction.depth = seq(1, 7, by = 2),
+                        .n.trees = seq(100, 1000, by = 50),
+                        .shrinkage = c(0.01, 0.1))
> set.seed(669)
> gbmModel <- train(CompressiveStrength ~ .,
+                   data = trainingSet,
+                   method = "gbm",
+                   tuneGrid = gbmGrid,
+                   verbose = FALSE,
+                   trControl = controlObject)
> cubistGrid <- expand.grid(.committees = c(1, 5, 10, 50, 75, 100),
+                           .neighbors = c(0, 1, 3, 5, 7, 9))
> set.seed(669)
> cbModel <- train(CompressiveStrength ~ .,
+                  data = trainingSet,
+                  method = "cubist",
+                  tuneGrid = cubistGrid,
+                  trControl = controlObject)
```

The resampling results for these models were collected into a single object using caret's `resamples` function. This object can then be used for visualizations or to make formal comparisons between the models.

```
> allResamples <- resamples(list("Linear Reg" = lmModel,
+                                "PLS" = plsModel,
+                                "Elastic Net" = enetModel,
+                                MARS = earthModel,
+                                SVM = svmRModel,
+                                "Neural Networks" = nnetModel,
+                                CART = rpartModel,
+                                "Cond Inf Tree" = ctreeModel,
+                                "Bagged Tree" = treebagModel,
+                                "Boosted Tree" = gbmModel,
+                                "Random Forest" = rfModel,
+                                Cubist = cbModel))
```

Figure 10.2 was created from this object as

```
> ## Plot the RMSE values
> parallelPlot(allResamples)
> ## Using R-squared:
> parallelplot(allResamples, metric = "Rsquared")
```

Other visualizations of the resampling results can also be created (see `?xyplot.resamples` for other options).

The test set predictions are achieved using a simple application of the `predict` function:

```
> nnetPredictions <- predict(nnetModel, testData)
> gbmPredictions <- predict(gbmModel, testData)
> cbPredictions <- predict(cbModel, testData)
```

To predict optimal mixtures, we first use the 28-day data to generate a set of random starting points from the training set.

Since distances between the formulations will be used as a measure of dissimilarity, the data are pre-processed to have the same mean and variance for each predictor. After this, a single random mixture is selected to initialize the maximum dissimilarity sampling process:

```
> age28Data <- subset(trainingData, Age == 28)
> ## Remove the age and compressive strength columns and
> ## then center and scale the predictor columns
> pp1 <- preProcess(age28Data[, -(8:9)], c("center", "scale"))
> scaledTrain <- predict(pp1, age28Data[, 1:7])
> set.seed(91)
> startMixture <- sample(1:nrow(age28Data), 1)
> starters <- scaledTrain[startMixture, 1:7]
```

After this, the maximum dissimilarity sampling method from Sect. 4.3 selects 14 more mixtures to complete a diverse set of starting points for the search algorithms:

```
> pool <- scaledTrain
> index <- maxDissim(starters, pool, 14)
> startPoints <- c(startMixture, index)
> starters <- age28Data[startPoints,1:7]
```

Since all seven mixture proportions should add to one, the search procedures will conduct the search without one ingredient (water), and the water proportion will be determined by the sum of the other six ingredient proportions. Without this step, the search procedures would pick candidate mixture values that would not add to one.

```
> ## Remove water
> startingValues <- starters[, -4]
```

To maximize the compressive strength, the R function `optim` searches the mixture space for optimal formulations. A custom R function is needed to translate a candidate mixture to a prediction. This function can find settings to *minimize* a function, so it will return the negative of the compressive strength. The function below checks to make sure that (a) the proportions are between 0 and 1 and (b) the proportion of water does not fall below 5 %. If these conditions are violated, the function returns a large positive number which the search procedure will avoid (as `optim` is for minimization).

```
> ## The inputs to the function are a vector of six mixture proportions
> ## (in argument 'x') and the model used for prediction ('mod')
> modelPrediction <- function(x, mod)
+ {
+   ## Check to make sure the mixture proportions are
+   ## in the correct range
+   if(x[1] < 0 | x[1] > 1) return(10^38)
+   if(x[2] < 0 | x[2] > 1) return(10^38)
+   if(x[3] < 0 | x[3] > 1) return(10^38)
+   if(x[4] < 0 | x[4] > 1) return(10^38)
+   if(x[5] < 0 | x[5] > 1) return(10^38)
+   if(x[6] < 0 | x[6] > 1) return(10^38)
+
+   ## Determine the water proportion
+   x <- c(x, 1 - sum(x))
+
+   ## Check the water range
+   if(x[7] < 0.05) return(10^38)
+
+   ## Convert the vector to a data frame, assign names
+   ## and fix age at 28 days
+   tmp <- as.data.frame(t(x))
+   names(tmp) <- c('Cement','BlastFurnaceSlag','FlyAsh',
+                   'Superplasticizer','CoarseAggregate',
+                   'FineAggregate', 'Water')
+   tmp$Age <- 28
+   ## Get the model prediction, square them to get back to the
+   ## original units, then return the negative of the result
+   -predict(mod, tmp)
+   }
```

First, the Cubist model is used:

```
> cbResults <- startingValues
> cbResults$Water <- NA
> cbResults$Prediction <- NA
> ## Loop over each starting point and conduct the search
> for(i in 1:nrow(cbResults))
+   {
+     results <- optim(unlist(cbResults[i,1:6]),
+                      modelPrediction,
+                      method = "Nelder-Mead",
+                      ## Use method = 'SANN' for simulated annealing
+                      control=list(maxit=5000),
+                      ## The next option is passed to the
+                      ## modelPrediction() function
+                      mod = cbModel)
+     ## Save the predicted compressive strength
+     cbResults$Prediction[i] <- -results$value
+     ## Also save the final mixture values
+     cbResults[i,1:6] <- results$par
+   }
> ## Calculate the water proportion
> cbResults$Water <- 1 - apply(cbResults[,1:6], 1, sum)
> ## Keep the top three mixtures
> cbResults <- cbResults[order(-cbResults$Prediction),][1:3,]
> cbResults$Model <- "Cubist"
```

We then employ the same process for the neural network model:

```
> nnetResults <- startingValues
> nnetResults$Water <- NA
> nnetResults$Prediction <- NA
> for(i in 1:nrow(nnetResults))
+   {
+     results <- optim(unlist(nnetResults[i, 1:6,]),
+                      modelPrediction,
+                      method = "Nelder-Mead",
+                      control=list(maxit=5000),
+                      mod = nnetModel)
+     nnetResults$Prediction[i] <- -results$value
+     nnetResults[i,1:6] <- results$par
+   }
> nnetResults$Water <- 1 - apply(nnetResults[,1:6], 1, sum)
> nnetResults <- nnetResults[order(-nnetResults$Prediction),][1:3,]
> nnetResults$Model <- "NNet"
```

To create Fig. 10.4, PCA was conducted on the 28-day-old mixtures and the six predicted mixtures were projected. The components are combined and plotted:

```
> ## Run PCA on the data at 28\,days
> pp2 <- preProcess(age28Data[, 1:7], "pca")
> ## Get the components for these mixtures
> pca1 <- predict(pp2, age28Data[, 1:7])
> pca1$Data <- "Training Set"
> ## Label which data points were used to start the searches
> pca1$Data[startPoints] <- "Starting Values"


> ## Project the new mixtures in the same way (making sure to
> ## re-order the columns to match the order of the age28Data object).
> pca3 <- predict(pp2, cbResults[, names(age28Data[, 1:7])])
> pca3$Data <- "Cubist"
> pca4 <- predict(pp2, nnetResults[, names(age28Data[, 1:7])])
> pca4$Data <- "Neural Network"
> ## Combine the data, determine the axis ranges and plot
> pcaData <- rbind(pca1, pca3, pca4)
> pcaData$Data <- factor(pcaData$Data,
+                        levels = c("Training Set","Starting Values",
+                                   "Cubist","Neural Network"))


> lim <- extendrange(pcaData[, 1:2])
> xyplot(PC2 ~ PC1, data = pcaData, groups = Data,
+        auto.key = list(columns = 2),
+        xlim = lim, ylim = lim,
+        type = c("g", "p"))
```

Desirability functions can be calculated with the desirability package. The functions `dMin` and `dMax` can be used to create desirability function curve definitions for minimization and maximization, respectively.