Combining Methods: Ensembles and Uplift Modeling

In this chapter, we look at two useful approaches that combine methods for improving predictive power: *ensembles* and *uplift modeling*. An ensemble combines multiple supervised models into a "super-model." The previous chapters in this part of the book introduced different supervised methods for prediction and classification. Earlier, in Chapter 5, we learned about evaluating predictive performance, which can be used to compare several models and choose the best one. An ensemble is based on the powerful notion of *combining models*. Instead of choosing a single predictive model, we can combine several models to achieve improved predictive accuracy. In this chapter, we explain the underlying logic of why ensembles can improve predictive accuracy and introduce popular approaches for combining models, including simple averaging, bagging, and boosting.

In *uplift modeling*, we combine supervised modeling with A-B testing, which is a simple type of a randomized experiment. We describe the basics of A-B testing and how it is used along with predictive models in persuasion messaging not to predict outcomes but to predict who should receive which message or treatment.

Python

In this chapter, we will use `pandas` for data handling and `scikit-learn` for the models. We will also make use of the utility functions from the Python Utilities Functions Appendix. Use the following import statements for the Python code in this chapter.

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 pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import AdaBoostClassifier
from sklearn.ensemble import BaggingClassifier
from sklearn.ensemble import RandomForestClassifier
from dmba import classificationSummary
```

13.1 Ensembles[1]

Ensembles played a major role in the million-dollar Netflix Prize contest that started in 2006. At the time, Netflix, the largest DVD rental service in the United States, wanted to improve their movie recommendation system (from www.netflixprize.com):

> Netflix is all about connecting people to the movies they love. To help customers find those movies, we've developed our world-class movie recommendation system: Cinematch^{SM}...And while Cinematch is doing pretty well, it can always be made better.

In a bold move, the company decided to share a large amount of data on movie ratings by their users, and set up a contest, open to the public, aimed at improving their recommendation system:

> We provide you with a lot of anonymous rating data, and a prediction accuracy bar that is 10% better than what Cinematch can do on the same training data set.

During the contest, an active leader-board showed the results of the competing teams. An interesting behavior started appearing: Different teams joined forces to create combined, or *ensemble* predictions, which proved more accurate than the individual predictions. The winning team, called "BellKor's Pragmatic Chaos" combined results from the "BellKor" and "Big Chaos" teams alongside additional members. In a 2010 article in *Chance* magazine, the Netflix Prize winners described the power of their ensemble approach:

[1]This and subsequent sections in this chapter copyright © 2019 Datastats, LLC, and Galit Shmueli. Used by permission.

An early lesson of the competition was the value of combining sets of predictions from multiple models or algorithms. If two prediction sets achieved similar RMSEs, it was quicker and more effective to simply average the two sets than to try to develop a new model that incorporated the best of each method. Even if the RMSE for one set was much worse than the other, there was almost certainly a linear combination that improved on the better set.

Why Ensembles Can Improve Predictive Power

The principle of combining methods is popular for reducing risk. For example, in finance, portfolios are created for reducing investment risk. The return from a portfolio is typically less risky, because the variation is smaller than each of the individual components.

In predictive modeling, "risk" is equivalent to variation in prediction error. The more our prediction errors vary, the more volatile our predictive model. Consider predictions from two different models for a set of n records. $e_{1,i}$ is the prediction error for the ith record by method 1 and $e_{2,i}$ is the prediction error for the same record by method 2.

Suppose that each model produces prediction errors that are, on average, zero (for some records the model over-predicts and for some it under-predicts, but on average the error is zero):

$$E(e_{1,i}) = E(e_{2,i}) = 0.$$

If, for each record, we take an average of the two predictions: $\overline{y}_i = \frac{\hat{y}_{1,i}+\hat{y}_{2,i}}{2}$, then the expected mean error will also be zero:

$$\begin{aligned} E\left(y_i - \overline{y_i}\right) &= E\left(y_i - \frac{\hat{y}_{1,i}+\hat{y}_{2,i}}{2}\right) \qquad (13.1)\\ &= E\left(\frac{y_i - \hat{y}_{1,i}}{2} + \frac{y_i - \hat{y}_{2,i}}{2}\right) = E\left(\frac{e_{1,i}+e_{2,i}}{2}\right) = 0. \end{aligned}$$

This means that the ensemble has the same mean error as the individual models. Now let us examine the variance of the ensemble's prediction errors:

$$\mathrm{V}ar\left(\frac{e_{1,i}+e_{2,i}}{2}\right) = \frac{1}{4}\left(\mathrm{V}ar(e_{1,i}) + \mathrm{V}ar(e_{2,i})\right) + \frac{1}{4} \times 2\mathrm{C}ov\left(e_{1,i}, e_{2,i}\right). \qquad (13.2)$$

This variance can be lower than each of the individual variances $\mathrm{V}ar(e_{1,i})$ and $\mathrm{V}ar(e_{2,i})$ under some circumstances. A key component is the covariance (or equivalently, correlation) between the two prediction errors. The case of no correlation leaves us with a quantity that can be smaller than each of the individual variances. The variance of the average prediction error will be even smaller when the two prediction errors are negatively correlated.

In summary, using an average of two predictions can potentially lead to smaller error variance, and therefore better predictive power. These results generalize to more than two methods; you can combine results from multiple prediction methods or classifiers.

THE WISDOM OF CROWDS

In his book *The Wisdom of Crowds*, James Surowiecki recounts how Francis Galton, a prominent statistician from the 19th century, watched a contest at a county fair in England. The contest's objective was to guess the weight of an ox. Individual contest entries were highly variable, but the mean of all the estimates was surprisingly accurate—within 1% of the true weight of the ox. On balance, the errors from multiple guesses tended to cancel one another out. You can think of the output of a predictive model as a more informed version of these guesses. Averaging together multiple guesses will yield a more precise answer than the vast majority of the individual guesses. Note that in Galton's story, there were a few (lucky) individuals who scored better than the average. An ensemble estimate will not always be more accurate than all the individual estimates in all cases, but it will be more accurate most of the time.

Simple Averaging

The simplest approach for creating an ensemble is to combine the predictions, classifications, or propensities from multiple models. For example, we might have a linear regression model, a regression tree, and a k-NN algorithm. We use each of the three methods to score, say, a test set. We then combine the three sets of results.

The three models can also be variations that use the same algorithm. For example, we might have three linear regression models, each using a different set of predictors.

Combining Predictions In prediction tasks, where the outcome variable is numerical, we can combine the predictions from the different methods simply by taking an average. In the above example, for each record in the test set, we have three predictions (one from each model). The ensemble prediction is then the average of the three values.

One alternative to a simple average is taking the median prediction, which would be less affected by extreme predictions. Another possibility is computing a weighted average, where weights are proportional to a quantity of interest. For instance, weights can be proportional to the accuracy of the model, or if different data sources are used, the weights can be proportional to the quality of the data.

Ensembles for prediction are useful not only in cross-sectional prediction, but also in time series forecasting (see Chapters 16–18). In forecasting, the same approach of combining future forecasts from multiple methods can lead to more precise predictions. One example is the weather forecasting application Forecast.io (www.forecast.io), which describes their algorithm as follows:

> Forecast.io is backed by a wide range of data sources, which are aggregated together statistically to provide the most accurate forecast possible for a given location.

Combining Classifications In the case of classification, combining the results from multiple classifiers can be done using "voting": For each record, we have multiple classifications. A simple rule would be to choose the most popular class among these classifications. For example, we might use a classification tree, a naive Bayes classifier, and discriminant analysis for classifying a binary outcome. For each record we then generate three predicted classes. Simple voting would choose the most common class among the three.

As in prediction, we can assign heavier weights to scores from some models, based on considerations such as model accuracy or data quality. This would be done by setting a "majority rule" that is different from 50%.

Combining Propensities Similar to predictions, propensities can be combined by taking a simple (or weighted) average. Recall that some algorithms, such as naive Bayes (see Chapter 8), produce biased propensities and should therefore not be simply averaged with propensities from other methods.

Bagging

Another form of ensembles is based on averaging across multiple random data samples. *Bagging*, short for "bootstrap aggregating," comprises two steps:

1. Generate multiple random samples (by sampling with replacement from the original data)—this method is called "bootstrap sampling."
2. Running an algorithm on each sample and producing scores.

Bagging improves the performance stability of a model and helps avoid overfitting by separately modeling different data samples and then combining the results. It is therefore especially useful for algorithms such as trees and neural networks.

Boosting

Boosting is a slightly different approach to creating ensembles. Here the goal is to directly improve areas in the data where our model makes errors, by forcing the model to pay more attention to those records. The steps in boosting are:

1. Fit a model to the data.
2. Draw a sample from the data so that misclassified records (or records with large prediction errors) have higher probabilities of selection.
3. Fit the model to the new sample.
4. Repeat Steps 2–3 multiple times.

Bagging and Boosting in Python

Although bagging and boosting can be applied to any data mining method, most applications are for trees, where they have proved extremely effective. In Chapter 9, we described random forests, an ensemble based on bagged trees. We illustrated a random forest implementation for the personal loan example. `scikit-learn` has a variety of methods to combine classifiers (models for predicting categorical outcomes) and regressors (models for predicting numerical outcomes) using bagging or boosting. Here, we will demonstrate this for decision tree classifiers. Table 13.1 shows the Python code and output producing a bagged tree and a boosted tree for the personal loan data, and how they are used to generate classifications for the validation set. In this example we see that bagging and boosting both produce better validation accuracy than the single tree.

Advantages and Weaknesses of Ensembles

Combining scores from multiple models is aimed at generating more precise predictions (lowering the prediction error variance). The ensemble approach is most useful when the combined models generate prediction errors that are negatively associated, but it can also be useful when the correlation is low. Ensembles can use simple averaging, weighted averaging, voting, medians, etc. Models can be based on the same algorithm or on different algorithms, using the same sample or different samples. Ensembles have become a major strategy for participants in data mining contests, where the goal is to optimize some predictive measure. In that sense, ensembles also provide an operational way to obtain solutions with high predictive power in a fast way, by engaging multiple teams of "data crunchers" working in parallel and combining their results.

Ensembles that are based on different data samples help avoid overfitting. However, remember that you can also overfit the data with an ensemble if you tweak it (e.g., choosing the "best" weights when using a weighted average).

The major disadvantage of an ensemble is the resources that it requires: computationally, as well as in terms of software availability and the analyst's skill and time investment. Ensembles that combine results from different algorithms require developing each of the models and evaluating them. Boosting-type ensembles and bagging-type ensembles do not require such effort, but they do

TABLE 13.1 **EXAMPLE OF BAGGING AND BOOSTING CLASSIFICATION TREES (PERSONAL LOAN DATA)**

code for bagging and boosing trees

```
bank_df = pd.read_csv('UniversalBank.csv')
bank_df.drop(columns=['ID', 'ZIP Code'], inplace=True)

# split into training and validation
X = bank_df.drop(columns=['Personal Loan'])
y = bank_df['Personal Loan']
X_train, X_valid, y_train, y_valid = train_test_split(X, y, test_size=0.40, random_state=3)

# single tree
defaultTree = DecisionTreeClassifier(random_state=1)
defaultTree.fit(X_train, y_train)

classes = defaultTree.classes_
classificationSummary(y_valid, defaultTree.predict(X_valid), class_names=classes)

# bagging
bagging = BaggingClassifier(DecisionTreeClassifier(random_state=1),
                            n_estimators=100, random_state=1)
bagging.fit(X_train, y_train)

classificationSummary(y_valid, bagging.predict(X_valid), class_names=classes)

# boosting
boost = AdaBoostClassifier(DecisionTreeClassifier(random_state=1),
                            n_estimators=100, random_state=1)
boost.fit(X_train, y_train)

classificationSummary(y_valid, boost.predict(X_valid), class_names=classes)
```

Output

```
> # single tree
Confusion Matrix (Accuracy 0.9825)

       Prediction
Actual    0    1
     0 1778   15
     1   20  187

> # bagging
Confusion Matrix (Accuracy 0.9855)

       Prediction
Actual    0    1
     0 1781   12
     1   17  190

> # boosting
Confusion Matrix (Accuracy 0.9840)

       Prediction
Actual    0    1
     0 1779   14
     1   18  189
```

have a computational cost (although boosting can be parallelized easily). Ensembles that rely on multiple data sources require collecting and maintaining multiple data sources. And finally, ensembles are "blackbox" methods, in that the relationship between the predictors and the outcome variable usually becomes nontransparent.

13.2 Uplift (Persuasion) Modeling

Long before the advent of the Internet, sending messages directly to individuals (i.e., direct mail) held a big share of the advertising market. Direct marketing affords the marketer the ability to invite and monitor direct responses from consumers. This, in turn, allows the marketer to learn whether the messaging is paying off. A message can be tested with a small section of a large list and, if it pays off, the message can be rolled out to the entire list. With predictive modeling, we have seen that the rollout can be targeted to that portion of the list that is most likely to respond or behave in a certain way. None of this was possible with traditional media advertising (television, radio, newspaper, magazine).

Direct response also made it possible to test one message against another and find out which does better.

A-B Testing

A-B testing is the marketing industry's term for a standard scientific experiment in which results can be tracked for each individual. The idea is to test one treatment against another, or a treatment against a control. "Treatment" is simply the term for the intervention you are testing: In a medical trial it is typically a drug, device, or other therapy; in marketing it is typically an offering to a consumer—for example, an e-mail, or a web page shown to a consumer. A general display ad in a magazine would not generally qualify, unless it had a specific call to action that allowed the marketer to trace the action (e.g., purchase) to a given ad, plus the ability to split the magazine distribution randomly and provide a different offer to each segment.

An important element of A-B testing is random allocation—the treatments are assigned or delivered to individuals randomly. That way, any difference between treatment A and treatment B can be attributed to the treatment (unless it is due to chance).

Uplift

An A-B test tells you which treatment does better on average, but says nothing about which treatment does better for which individual. A classic example is in political campaigns. Consider the following scenario: The campaign director for Smith, a Democratic Congressional candidate, would like to know which voters should be called to encourage to support Smith. Voters that tend to vote Democratic but are not activists might be more inclined to vote for Smith if they got a call. Active Democrats are probably already supportive of him, and therefore a call to them would be wasted. Calls to Republicans are not only wasteful, but they could be harmful.

Campaigns now maintain extensive data on voters to help guide decisions about outreach to individual voters. Prior to the 2008 Obama campaign, the practice was to make rule-based decisions based on expert political judgment. Since 2008, it has increasingly been recognized that, rather than relying on judgment or supposition to determine whether an individual should be called, it is best to use the data to develop a model that can predict whether a voter will respond positively to outreach.

Gathering the Data

US states maintain publicly available files of voters, as part of the transparent oversight process for elections. The voter file contains data such as name, address, and date of birth. Political parties have "poll-watchers" at elections to record who votes, so they have additional data on which elections voters voted in. Census data for neighborhoods can be appended, based on voter address. Finally, commercial demographic data can be purchased and matched to the voter data. Table 13.2 shows a small extract of data derived from the voter file for the US state of Delaware.[2] The actual data used in this problem are in the file *Voter-Persuasion.csv* and contain 10,000 records and many additional variables beyond those shown in Table 13.2.

First, the campaign director conducts a survey of 10,000 voters to determine their inclination to vote Democratic. Then she conducts an experiment, randomly splitting the sample of 10,000 voters in half and mailing a message promoting Smith to half the list (treatment A), and nothing to the other half (treatment B). The control group that gets no message is essential, since other campaigns or news events might cause a shift in opinion. The goal is to measure the change in opinion after the message is sent out, relative to the no-message control group.

The next step is conducting a post-message survey of the same sample of 10,000 voters, to measure whether each voter's opinion of Smith has shifted in a positive direction. A binary variable, Moved_AD, will be added to the above data, indicating whether opinion has moved in a Democratic direction (1) or not (0).

[2]Thanks to Ken Strasma, founder of the microtargeting firm HaystaqDNA and director of targeting for the 2004 Kerry campaign and the 2008 Obama campaign, for these data.

TABLE 13.2 DATA ON VOTERS (SMALL SUBSET OF VARIABLES AND RECORDS) AND DATA DICTIONARY

Voter	Age	NH_White	Comm_PT	H_F1	Reg_Days	PR_Pelig	E_Elig	Political_C
1	28	70	0	0	3997	0	20	1
2	23	67	3	0	300	0	0	1
3	57	64	4	0	2967	0	0	0
4	70	53	2	1	16620	100	90	1
5	37	76	2	0	3786	0	20	0

Data Dictionary

Age	Voter age in years
NH_White	Neighborhood average of % non-Hispanic white in household
Comm_PT	Neighborhood % of workers who take public transit
H_F1	Single female household (1 = yes)
Reg_Days	Days since voter registered at current address
PR_Pelig	Voted in what % of non-presidential primaries
E_Pelig	Voted in what % of any primaries
Political_C	Is there a political contributor in the home? (1 = yes)

Table 13.3 summarizes the results of the survey, by comparing the movement in a Democratic direction for each of the treatments. Overall, the message (Message = 1) is modestly effective.

Movement in a Democratic direction among those who got no message is 34.4%. This probably reflects the approach of the election, the heightening campaign activity, and the reduction in the "no opinion" category. It also illustrates the need for a control group. Among those who did get the message, the movement in a Democratic direction is 40.2%. So, overall, the lift from the message is 5.8%.

TABLE 13.3 RESULTS OF SENDING A PRO-DEMOCRATIC MESSAGE TO VOTERS

	#Voters	# Moved Dem.	% Moved Dem.
Message = 1 (message sent)	5000	2012	40.2%
Message = 0 (no message sent)	5000	1722	34.4%

We can now append two variables to the voter data shown earlier in Table 13.2: *message* [whether they received the message (1) or not (0)] and *Moved_AD* [whether they moved in a Democratic direction (1) or not (0)]. The augmented data are shown in Table 13.4.

A Simple Model

We can develop a predictive model with Moved_AD as the outcome variable, and various predictor variables, *including the treatment Message*. Any classification

TABLE 13.4 OUTCOME VARIABLE (MOVED_AD) AND TREATMENT VARIABLE (MESSAGE) ADDED TO VOTER DATA

Voter	Age	NH_White	Comm_PT	H_F1	Reg_Days	PR_Pelig	E_Elig	Political_C	Message	Moved_AD
1	28	70	0	0	3997	0	20	1	0	1
2	23	67	3	0	300	0	0	1	1	1
3	57	64	4	0	2967	0	0	0	0	0
4	70	53	2	1	16620	100	90	1	0	0
5	37	76	2	0	3786	0	20	0	1	0

TABLE 13.5 CLASSIFICATIONS AND PROPENSITIES FROM PREDICTIVE MODEL (SMALL EXTRACT)

Voter	Message	Actual Moved_AD	Predicted Moved_AD	Predicted Prob.
1	0	1	1	0.5975
2	1	1	1	0.5005
3	0	0	0	0.2235
4	0	0	0	0.3052
5	1	0	0	0.4140

method can be used; Table 13.5 shows the first few lines from the output of some predictive model used to predict Moved_AD.

However, our interest is not just how the message did overall, nor is it whether we can predict the probability that a voter's opinion will move in a favorable direction. Rather our goal is to predict how much (positive) impact the message will have on a specific voter. That way the campaign can direct its limited resources toward the voters who are the most persuadable—those for whom sending the message will have the greatest positive effect.

Modeling Individual Uplift

To answer the question about the message's impact on each voter, we need to model the effect of the message at the individual voter level. For each voter, uplift is defined as follows:

Uplift = increase in propensity of favorable opinion after receiving message

To build an uplift model, we follow the following steps to estimate the change in probability of "success" (propensity) that comes from receiving the treatment (the message):

1. Randomly split a data sample into treatment and control groups, conduct an A-B test, and record the outcome (in our example: Moved_AD)
2. Recombining the data sample, partition it into training and validation sets; build a predictive model with this outcome variable and include a predictor variable that denotes treatment status (in our example: Message). If logistic regression is used, additional interaction terms between treatment status and other predictors can be added as predictors to allow

TABLE 13.6 CLASSIFICATIONS AND PROPENSITIES FROM PREDICTIVE MODEL (SMALL EXTRACT) WITH MESSAGE VALUES REVERSED

Voter	Message	Actual Moved_AD	Predicted Moved_AD	Predicted Prob.
1	1	1	1	0.6908
2	0	1	1	0.3996
3	1	0	0	0.3022
4	1	0	0	0.3980
5	0	0	0	0.3194

the treatment effect to vary across records (in data-driven methods such as trees and KNN this happens automatically).

3. Score this predictive model to a partition of the data; you can use the validation partition. This will yield, for each validation record, its propensity of success given its treatment.
4. Reverse the value of the treatment variable and re-score the same model to that partition. This will yield for each validation record its propensity of success had it received the other treatment.
5. Uplift is estimated for each individual by P(Success | Treatment = 1) − P(Success | Treatment = 0)
6. For new data where no experiment has been performed, simply add a synthetic predictor variable for treatment and assign first a '1,' score the model, then a '0,' and score the model again. Estimate uplift for the new record(s) as above.

Continuing with the small voter example, the results from Step 3 were shown in Table 13.5—the right column shows the propensities from the model. Next, we re-train the predictive model, but with the values of the treatment variable Message reversed for each row. Table 13.6 shows the propensities with variable Message reversed (you can see the reversed values in column Message). Finally, in Step 5, we calculate the uplift for each voter.

Table 13.7 shows the uplift for each voter—the success (Moved_AD = 1) propensity given Message = 1 minus the success propensity given Message = 0.

TABLE 13.7 UPLIFT: CHANGE IN PROPENSITIES FROM SENDING MESSAGE VS. NOT SENDING MESSAGE

Voter	Prob. if Message = 1	Prob. if Message = 0	Uplift
1	0.6908	0.5975	0.0933
2	0.5005	0.3996	0.1009
3	0.3022	0.2235	0.0787
4	0.3980	0.3052	0.0928
5	0.4140	0.3194	0.0946

Computing Uplift with Python

This entire process that we showed manually can be done using `scikit-learn`. One difference to our manual computation is that we used a logistic regression whereas here we will use a random forest classifier. Table 13.8 shows the result of implementing the uplift analysis in `scikit-learn` using a random forest. The output shows the two conditional probabilities, P(Success | Treatment = 1) and P(Success | Treatment = 0), estimated for each record. The difference between the values in Tables 13.7 and 13.8 are due to the use of two different predictive algorithms (logistic regression vs. random forests).

Using the Results of an Uplift Model

Once we have estimated the uplift for each individual, the results can be ordered by uplift. The message could then be sent to all those voters with a positive uplift, or, if resources are limited, only to a subset—those with the greatest uplift.

Uplift modeling is used mainly in marketing and, more recently, in political campaigns. It has two main purposes:

- To determine whether to send someone a persuasion message, or just leave them alone.
- When a message is definitely going to be sent, to determine which message, among several possibilities, to send.

Technically this amounts to the same thing—"send no message" is simply another category of treatment, and an experiment can be constructed with multiple treatments, for example, no message, message A, and message B. However, practitioners tend to think of the two purposes as distinct, and tend to focus on the first. Marketers want to avoid sending discount offers to customers who would make a purchase anyway, or renew a subscription anyway. Political campaigns, likewise, want to avoid calling voters who would vote for their candidate in any case. And both parties especially want to avoid sending messages or offers where the effect might be antagonistic—where the uplift is negative.

13.3 Summary

In practice, the methods discussed in this book are often used not in isolation, but as building blocks in an analytic process whose goal is always to inform and provide insight.

In this chapter, we looked at two ways that multiple models are deployed. In ensembles, multiple models are weighted and combined to produce improved predictions. In uplift modeling, the results of A-B testing are folded into the

TABLE 13.8 **UPLIFT IN PYTHON APPLIED TO THE VOTERS DATA**

code for uplift

```
voter_df = pd.read_csv('Voter-Persuasion.csv')

# Preprocess data frame
predictors = ['AGE', 'NH_WHITE', 'COMM_PT', 'H_F1', 'REG_DAYS',
            'PR_PELIG', 'E_PELIG', 'POLITICALC', 'MESSAGE_A']
outcome = 'MOVED_AD'

classes = list(voter_df.MOVED_AD.unique())

# Partition the data
X = voter_df[predictors]
y = voter_df[outcome]
X_train, X_valid, y_train, y_valid = train_test_split(X, y, test_size=0.40, random_state=1)

# Train a random forest classifier using the training set
rfModel = RandomForestClassifier(n_estimators=100, random_state=1)
rfModel.fit(X_train, y_train)

# Calculating the uplift
uplift_df = X_valid.copy()  # Need to create a copy to allow modifying data

uplift_df.MESSAGE_A = 1
predTreatment = rfModel.predict_proba(uplift_df)
uplift_df.MESSAGE_A = 0
predControl = rfModel.predict_proba(uplift_df)

upliftResult_df = pd.DataFrame({
    'probMessage': predTreatment[:,1],
    'probNoMessage': predControl[:,1],
    'uplift': predTreatment[:,1] - predControl[:,1],
    }, index=uplift_df.index)
upliftResult_df.head()
```

Output

```
      probMessage  probNoMessage  uplift
9953         0.77           0.62    0.15
3850         0.39           0.39    0.00
4962         0.20           0.14    0.06
3886         0.86           0.62    0.24
5437         0.10           0.28   -0.18
```

predictive modeling process as a predictor variable to guide choices not just about whether to send an offer or persuasion message, but also as to who to send it to.

PROBLEMS

13.1 **Acceptance of Consumer Loan** Universal Bank has begun a program to encourage its existing customers to borrow via a consumer loan program. The bank has promoted the loan to 5000 customers, of whom 480 accepted the offer. The data are available in file *UniversalBank.csv*. The bank now wants to develop a model to predict which customers have the greatest probability of accepting the loan, to reduce promotion costs and send the offer only to a subset of its customers.

We will develop several models, then combine them in an ensemble. The models we will use are (1) logistic regression, (2) k-nearest neighbors with $k = 3$, and (3) classification trees. Preprocess the data as follows:

- Bin the following variables so they can be used in Naive Bayes: Age (5 bins), Experience (10 bins), Income (5 bins), CC Average (6 bins), and Mortgage (10 bins).
- Education and Family can be used as is, without binning.
- Zip code can be ignored.
- Use one-hot-encoding to convert the categorical data into indicator variables
- Partition the data: 60% training, 40% validation.

a. Fit models to the data for (1) logistic regression, (2) k-nearest neighbors with $k = 3$, (3) classification trees, and (4) Naive Bayes. Use Personal Loan as the outcome variable. Report the validation confusion matrix for each of the models.

b. Create a data frame with the actual outcome, predicted outcome, and each of the models. Report the first 10 rows of this data frame.

c. Add two columns to this data frame for (1) a majority vote of predicted outcomes, and (2) the average of the predicted probabilities. Using the classifications generated by these two methods derive a confusion matrix for each method and report the overall accuracy.

d. Compare the error rates for the four individual methods and the two ensemble methods.

13.2 **eBay Auctions—Boosting and Bagging** Using the eBay auction data (file *eBayAuctions.csv*) with variable Competitive as the outcome variable, partition the data into training (60%) and validation (40%).

a. Run a classification tree, using the default settings of *DecisionTreeClassifier*. Looking at the validation set, what is the overall accuracy? What is the lift on the first decile?

b. Run a boosted tree with the same predictors (use *AdaBoostClassifier* with *DecisionTreeClassifier* as the base estimator). For the validation set, what is the overall accuracy? What is the lift on the first decile?

c. Run a bagged tree with the same predictors (use *BaggingClassifier*). For the validation set, what is the overall accuracy? What is the lift on the first decile?

d. Run a random forest (use *RandomForestClassifier*). Compare the bagged tree to the random forest in terms of validation accuracy and lift on first decile. How are the two methods conceptually different?

13.3 **Predicting Delayed Flights (Boosting).** The file *FlightDelays.csv* contains information on all commercial flights departing the Washington, DC area and arriving at New York during January 2004. For each flight there is information on the departure and arrival airports, the distance of the route, the scheduled time and date of the flight, and so on. The variable that we are trying to predict is whether or not a flight is delayed. A delay is defined as an arrival that is at least 15 minutes later than scheduled.

Data Preprocessing. Transform variable day of week info a categorical variable. Bin the scheduled departure time into eight bins (in Python use function *pd.cut()* from the `pandas` package). Partition the data into training (60%) and validation (40%).

Run a boosted classification tree for delay. With the exception of setting `n_estimators=500` and `random_state=1`, use default setting for the Decision-TreeClassifier and the AdaBoostClassifier.

a. Compared with the single tree, how does the boosted tree behave in terms of overall accuracy?

b. Compared with the single tree, how does the boosted tree behave in terms of accuracy in identifying delayed flights?

c. Explain why this model might have the best performance over the other models you fit.

13.4 **Hair Care Product—Uplift Modeling** This problem uses the data set in *Hair-Care-Product.csv*, courtesy of SAS. In this hypothetical case, a promotion for a hair care product was sent to some members of a buyers club. Purchases were then recorded for both the members who got the promotion and those who did not.

a. What is the purchase propensity

  i. among those who received the promotion?

  ii. among those who did not receive the promotion?

b. Partition the data into training (60%) and validation (40%) and fit:

  i. Uplift using a Random Forest.

  ii. Uplift using k-NN.

c. Report the two models' recommendations for the first three members.