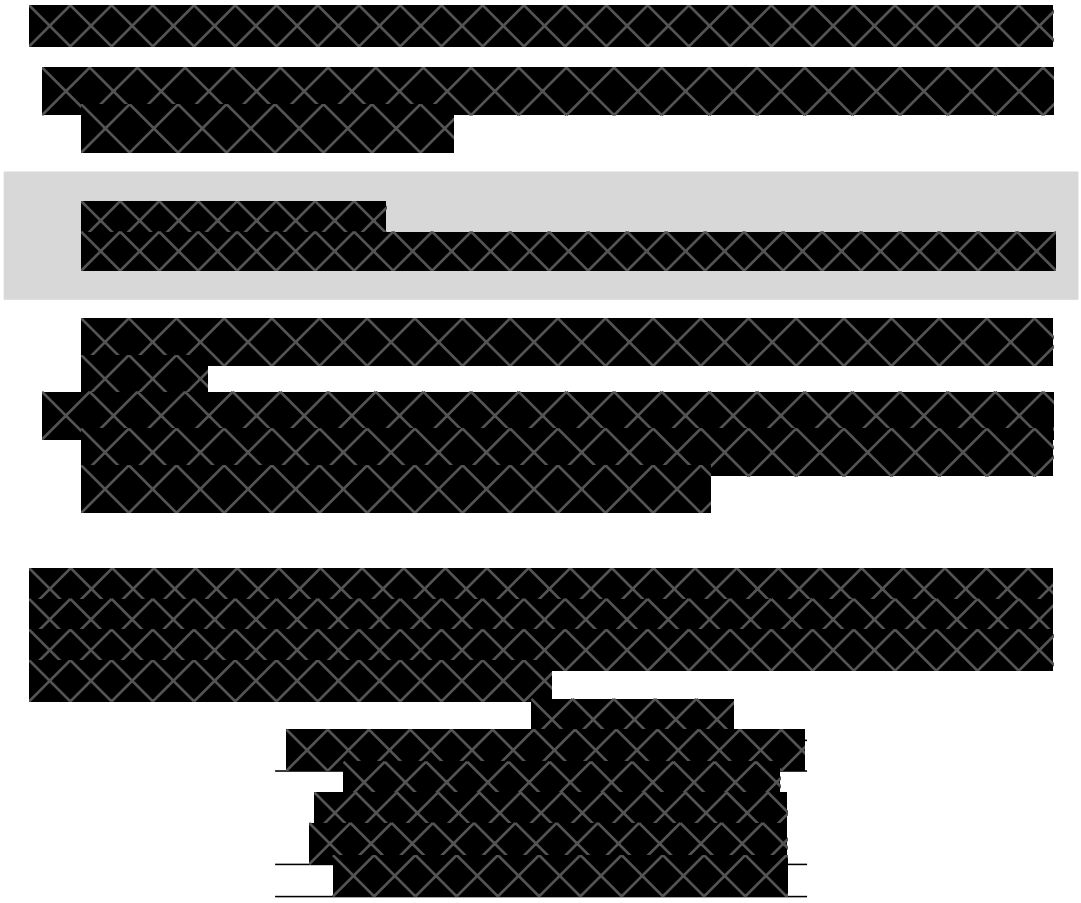

10.5.1 Remark

Our results disagree with the results of Mosteller and Wallace [41] as we find that six of the disputed papers were more likely authored by Hamilton as he claimed.

[redacted] Customer Segmentation

A recurring challenge in business marketing is to identify customers that are alike with regard to purchasing habits. If distinct groups or *segments* of customers can be identified, then a business may characterize their customers with respect to demographics or behavior. Tailoring their communications to specific segments allows for more relevant messages and improvements in customer experience. From the standpoint of the business, monitoring activity by customer segment promotes the understanding of their customers and allows for timely changes and improvements to business practices. A group

of customers that are alike with respect to purchasing habits is referred to as a *customer segment*. The process of partitioning a set of customers into segments is known as *customer segmentation*.

We return to the grocery store data set  The investigation was loosely organized about understanding shopping behaviors by customer segment. There are clear differences between segments with respect to several variables.

The following customer segments have been identified:

Primary These members appear to utilize the co-op as the primary place they grocery shop.

Secondary These members shop regularly at the co-op but are likely to shop at other grocery stores.

Light These members have joined the co-op but shop very seldom and tend to purchase few items.

Niche This is a collection of 17 segments containing relatively few members. These customers shop one department almost exclusively. Examples of the niche segments are *produce*, *packaged grocery*, and *cheese*.

Both members and non-members buy at the co-op. Non-members are not identifiable by segment at the time of a transaction. However, the co-op would like to provide incentives and services at the time of sale to the non-members based on their predicted customer segment. With this goal in mind, we ask the following question: based on the receipt, that is, a list of purchased items, can we identify a customer segment that the non-member customer most resembles?[6] In analytical terms, the objective is to estimate the relative likelihood that a non-member belongs to each of the 20 customer segments based on their point-of-sale receipt. The last step is to label the non-member according to the most likely segment.

We return to a problem already encountered in the Federalist Papers analysis before tackling the prediction problem.

 *Additive Smoothing*

The incidence of non-stop words that were not used by all three authors presented a complication in the Federalist Papers problem. Some of the non-stop words were not used by all three authors, and so the frequency of occurrence of such a word was zero for at least one author. For these words, the estimated probability of use for one or more authors was zero. Here lies the problem. The prediction function computes a linear combination of the frequencies of use and the logarithms of the estimated probability of use. If one such word appears in a paper, then an estimate of zero for author A eliminates A from

[6] If so, then the non-members may be offered segment-specific information and incentives.

being the author of the paper, no matter how similar the rest of the paper's word distribution is to the word distribution of author A. We had the luxury with the Federalist Papers of ignoring those zero-frequency words since there were more than 1100 words that were used by all three. With only three authors, enough common words were left to build an accurate prediction function. If there had been a larger number of authors, then ignoring words not common to all authors may have been costly with respect to the discriminative information carried by the words.

In the customer segmentation analysis, a group is a customer segment. There are thousands of items for sale (at least 12,890), and most of the segments are niche segments. Members of the niche segments are very exclusive in their selection of store items and some niche segments may not purchase a significant number of items. Our casual approach to dismissing items with zero frequency of purchase may negatively affect the prediction function.

Let's consider a more conservative approach to handling categories (a word or a store item) for which there are no observations on use (or purchase) for one or more the groups. The solution is to smooth the probability estimates. We recommend a technique known as additive smoothing (also known as Laplace smoothing). The idea is to add terms to both the numerator and denominator of a sample proportion so that if the numerator is zero, then the estimate will not be zero, but instead a small positive fraction.

Let d denote the number of groups or classes, and n denote the number of categories. In the Federalist Papers analysis, there were $d = 3$ groups of papers, each belonging to one of the authors, and $n = 1103$ categories, as there were 1103 non-stop words used at least once by all three authors. The probability of drawing item j from among the items bought by a customer belonging to segment k is denoted by $\pi_{k,j}$. The sample proportion estimator of $\pi_{k,j}$ is

$$p_{k,j} = \frac{x_{k,j}}{\sum_{i=1}^{n} x_{k,i}},$$

where $x_{k,j}$ is the frequency of purchases of item j by members of segment k. The sum of $p_{k,j}$ over items (indexed by j) is one since every purchase must be one of the n items. We found 12,890 unique items among the purchases of the members. The smoothed estimator of $\pi_{k,j}$ is a modification of the sample proportion given by

$$\widehat{\pi}_{k,j} = \frac{x_{k,j} + \alpha}{\sum_{i=1}^{n} x_{k,i} + d\alpha}, \tag{10.9}$$

where α is the smoothing parameter. A common choice is $\alpha = 1$ though smaller values are sometimes preferred. To illustrate, suppose that $\alpha = 1$ and $x_{k,j} = 0$ for some segment and item. Then, $\widehat{\pi}_{k,j} = 1/(\sum_{i=1}^{n} x_{k,i} + d)$. On the other hand, if every purchase from a segment k was of item j, then $x_{k,j} = \sum_{i=1}^{n} x_{k,i}$ and $\widehat{\pi}_{k,j} = (x_{k,j} + 1)/(x_{k,j} + d)$.

The Data

Returning to the data at hand, there are two data sets, one consisting of receipts from members and the other consisting of receipts from non-members. Both data sets were collected during the month of December 2015. The file named `member_transactions.txt` has contains 50,193 observations, each of which is best thought of as a receipt. Columns are tab-delimited and contain the following information:

1. **owner**: an anonymized identifier of the co-op member.
2. **transaction identifier**: a unique identifier for the transaction, or equivalently, the receipt.
3. **segment**: the segment membership of the owner.
4. **date**: the date of the transaction.
5. **hour**: the hour of the transaction.
6. **item list**: a variable-length list of the items that were purchased.

Table 10.3 shows a partial record. Note that one item, a bag of St. Paul bagels, appears twice in the list of items.

Table 10.3 A partial record from the data file. The entire receipt is not shown since 40 items were purchased

Variable	Value
Owner	11,144
Transaction identifier	m22
Segment	Primary
Date	2015-12-30
Hour	12
Item list	Tilapia `\t` St.Paul Bagels (Bagged) `\t` St.Paul Bagels (Bagged) `\t` Salmon Atlantic Fillet `\t` O.Broccoli 10oz CF `\t` ...

A second file, `nonmember_transactions.txt`, contains transaction information for non-member customers. The non-members data file has the same columns as the previous data set except that the owner and customer segment identifiers are absent.

1. We begin by importing modules and defining the paths to the two data sets.

```
import sys
import numpy as np
from collections import defaultdict

working_dir = '../Data/'
mem_file_name = "member_transactions.txt"
non_mem_file_name = "nonmember_transactions.txt"
```

2. Initialize a dictionary of dictionaries named `segmentDict` to store the items that were purchased by the members of each segment. Initialize two more dictionaries to store the numbers of items purchased by each segment and the number of purchases of each item.

```
segmentDict = defaultdict(lambda: defaultdict(int))
segmentTotals = defaultdict(int) # A count of purchases by segment
itemTotals = defaultdict(int) # A count of purchases by item
```

   The instruction `defaultdict(lambda: defaultdict(int))` creates a dictionary of dictionaries. The outer dictionary is named `segmentDict`. The keys for this dictionary will be the groups—the customer segments. The inner dictionaries are not named, but the keys of these dictionaries are the item names and the values are the frequencies of occurrence of each item in the segment stored as integers. Because the dictionaries are `defaultdict` dictionaries, we do not need to test for and create an entry for each item if it's not already in the inner dictionary. Likewise, the inner dictionaries (the segment dictionaries) are automatically created when a newly encountered segment is passed to `segmentDict`.
3. The member transactions data file is read in this code segment. The code builds `segmentDict`, the dictionary containing the frequencies of occurrence of each item in each segment. The total number of transactions for each segment is computed and stored in `segmentTotals`. The totals are used to estimate the prior probabilities. Lastly, the totals for each item are computed and stored in `itemTotals`.

```
path = working_dir + mem_file_name
print(path)
with open(path,'r') as f :
    next(f) # skip header
    for record in f :
        record = record.strip().split("\t")
        segment = record[2]
        items = record[5:]
        segmentTotals[segment] += 1

        for item in items :
            item = item.lower()
            itemTotals[item] += 1
            segmentDict[segment][item] += 1
print(len(segmentDict))
print(segmentDict.keys())
```

   As each record is read, leading and trailing blanks are removed using the `.strip()` function.

The instruction `segmentDict[segment][item] += 1` increments the frequency of occurrence of the `item` in the segment dictionary. The `item` is a key for a particular segment dictionary, namely, `segmentDict[segment]`.

4. After all of the data is processed, calculate the natural logarithms of the prior probability estimates (the $\widehat{\pi}_k$'s)

```
totalItems = sum([n for n in segmentTotals.values()])
logPriorDict = {seg : np.log(n/totalItems)
                    for seg, n in segmentTotals.items()}
nSegments = len(logPriorDict)
```

The calculation of `logPriorDict` makes use of dictionary comprehension. Segments are used as keys and the values are the natural logarithms of the prior probability estimates. Check that the number of segments is 20.

5. The next step is to build the prediction function, $f(\mathbf{x}_0|D)$. The function will consume a predictor vector $\mathbf{x}_0$ consisting of a receipt listing each item that was purchased by a customer. The prediction of y_0 is computed from the posterior probability estimates (the $\widehat{\pi}_{k,j}$'s) which in turn are computed from `segmentTotals`. We also use the prior probability estimates $\widehat{\pi}_k, k = 1, 2, \ldots, d$. The logarithms of these estimates are passed in as well.

   The function declaration is

```
def predictFunction(x0, segmentTotals, logPriorDict, segmentDict, alpha):
    ''' Predicts segment from x0
        Input: x0: a transaction list to classify
               segmentTotals: the counts of transactions by segment
               logPriorDict: the log-priors for each segment
               segmentDict: transaction by segment by item.
               alpha: the Laplace smoothing parameter.
    '''
```

6. Continuing with the `predictFunction`, the next code segment computes the logarithm of the posterior probability estimates of segment membership for the target $\mathbf{x}_0$. Recall that the prediction of group (or segment) membership is determined by the function

$$f(\mathbf{x}_0|D) = \arg\max_k \left\{ \log(\widehat{\pi}_k) + \sum_{j=1}^{n} x_{0,j} \log(\widehat{\pi}_{k,j}) \right\},$$

where k indexes customer segments, $\widehat{\pi}_k$ is the estimated prior probability of membership in the kth segment, and $\widehat{\pi}_{k,j}$ is the estimated probability of purchasing the jth item by a customer belonging to segment k. Also, the logarithm of $\widehat{\pi}_{k,j}$

$$\log(\widehat{\pi}_{k,j}) = \log(x_{k,j} + \alpha) - \log\left(\sum_{i=1}^{n} x_{k,i} + d\alpha\right), \tag{10.10}$$

where $x_{k,j}$ is the total number of purchases of the jth item by members of customer segment k. The denominator $\sum_{i=1}^{n} x_{k,i} + d\alpha$ of the estimator $\widehat{\pi}_{k,i}$ is a constant—it doesn't depend on the item (indexed by j) so it can be computed once for each segment (indexed by k). Since the prediction function operates on the natural logarithm scale, we compute the logarithm of the denominator.

Be aware that customers often purchase several of the same items, in which case one might expect the count would be greater than one in $\mathbf{x}_0$. However, our item list is different: the item will have multiple occurrences in the list if the item was scanned more than once (see Table 10.3 for an example of a receipt).

We use nested `for` loops. The outer `for` loop iterates over the segments and the inner iterates over items in the transaction vector $\mathbf{x}_0$.

```
logProbs = defaultdict(float)
for segment in segmentDict :
    denominator = sum([itemTotals[item] for item in itemTotals])
                      + alpha * nSegments
    logDen = np.log(denominator)
    for item in x0 :
        logProbs[segment] += np.log(segmentDict[segment][item] + alpha)
                           - logDen
```

Because `segmentDict[segment]` is a `defaultdict`, it returns zero if there is no entry for `item` in `segmentDict[segment]` rather than a `keyError`.

Even if we pass $\alpha = 0$ and `segmentDict[segment][item]` is zero, an exception is not created. Instead, the `np.log` function returns `-inf`, mathematically, $-\infty$. The practical effect is that the segment without any purchases of the item cannot yield the maximum probability of segment membership since the estimated log-posterior probability of membership in the segment will be $-\infty$. This strategy is reasonable if the number of observations in the data set is large, as it is in this case with 50,192 transactions.

7. The last code segment in the `predictFunction` determines the segment associated with the maximum estimated posterior probability (on the logarithmic scale). The first line of code creates a list of two-element lists from the `logProbs` dictionary. The `Numpy` function `argmax` is used to extract the index of the largest element.[7]

```
    lstLogProbs = [[seg, logProb] for seg, logProb in logProbs.items()]
    index = np.argmax([logProb for _,logProb in lstLogProbs])
    prediction, maxProb = lstLogProbs[index]
    return(prediction)
```

The calculation of `index` uses list comprehension to build a list containing only the log-probability estimates. The underscore character `_` instructs the `Python` interpreter to ignore the first element of each pair in building the list.

8. Now that the prediction function has been programmed, we need to verify that it produces reasonable results. The first check is to apply the function to the membership data and determine the proportion of training observations that are assigned to the correct segment. We'll re-use the code from the beginning of the tutorial to process the data file. We create an empty list named `outcomes` to store the pairs $(y_i, \widehat{y}_i)$ as the `with open` loop iterates over records and computes the predictions.

```
outcomes = []
alpha = 0
with open(working_dir + mem_file_name,'r') as f :
    next(f) # skip headers
    for idx, record in enumerate(f) :
        record = record.strip().split("\t")
        segment, item_list = record[2], record[5:]
        x0 = [item.lower() for item in item_list]
```

9. Call `predictFunction` and save the results by appending to the `outcomes` list:

```
prediction = predictFunction(x0, segmentTotals, logPriorDict,
                                segmentDict, alpha)
outcomes.append([segment,prediction])
```

This code segment immediately follows the translation of `item_list` to $\mathbf{x}_0$.

10. Every 100 records, compute an estimate of the overall accuracy of the function.

```
if idx % 100 == 0 :
    acc = np.mean([int(segment==prediction )
                   for segment, prediction in outcomes])
    print(str(idx),acc)
```

List comprehension is used to construct a list of boolean values (true or false) by the comparison `segment==prediction`. The conversion of a boolean to integer produces 1 if the boolean variable is true and 0 if the value is false. The mean of the binary vector is the proportion of observations that were correctly classified. With our choice of $\alpha = 0$, we obtained an estimated accuracy rate of .711.

11. The nonmember data set contains 23,251 receipts. We will compute predictions for each receipt and track the number of receipts that are predicted to belong to each of the 20 market segments.

    a. Change the input file name to read the non-members data set.
    b. Change the instruction that extracts the `item_list` and the record identifier. The new instruction should look like this:

```
recordID, item_list = record[0], record[3:]
```

    c. Initialize a `defaultdict` for which the values are sets before the nonmembers data file is processed. The keys of the dictionary will be segments, and the values will be a set of record identifiers that were predicted to belong to the segment.

```
segCounts = defaultdict(set)
```

    d. After `predictionFunction` returns `prediction`, add the record identifier to the dictionary. The key is the predicted segment.

```
segCounts[prediction].add(recordID)
```

    The function `add` adds a record identifier to a set.
    e. Run the script and print the contents of `segCounts`. Our results are shown in Table 10.4.

```
for seg in segCounts :
    total += len(segCounts[seg])
    print(" & ".join([seg,str(round(len(segCounts[seg])/idx,3))] ) )
```

Remarks

We have some concerns with the results of the prediction function. First, the accuracy estimate is not a great deal larger than the proportion of members that have been identified as members of the primary segment (.649). If we had

Table 10.4 Predicted customer segments for non-members. The tabled values are the proportion of predictions by customer segment. $N = 23{,}251$ receipts were classified to segment

Segment	Proportion
Primary	.873
Secondary	.077
Light	.0
Niche-juice bar	.003
Niche-frozen	.0
Niche-supplements	.0
Niche-meat	.001
Niche-bread	.0
Niche-personal care	.001
Niche-herbs & spices	.001
Niche-general merchandise	.0
Niche-beer & wine	.0
Niche-packaged grocery	.014
Niche-produce	.007
Niche-bulk	.001
Niche-cheese	.0
Niche-refrigerated grocery	.002
Niche-deli	.02

assigned every member to the primary segment, the accuracy rate estimate would be .649. We are not out-performing a simple baseline prediction function by a lot. Another item of some concern is the rate at which non-members are assigned to the primary segment—.873. This rate is substantially larger than the proportion of primary segment customers (.649). This observation raises a second concern that the prediction function is biased and assigns too many receipts to the primary segment. The problem is difficult to address and it may be that non-members tend to purchase items similar to those preferred by the primary segment customers in which case, the prediction function is not necessarily biased.