Naive Bayes

Ian Kessler IAN.KESSLER@STUDENT.MONTANA.EDU
Department of Computer Science
Montana State University
Bozeman, MT 59717, USA

Ethan Skelton ETHAN.SKELTON@STUDENT.MONTANA.EDU
Department of Computer Science
Montana State University
Bozeman, MT 59717, USA

Editor: Ian Kessler and Ethan Skelton

Abstract

In this project, we implemented a classification algorithm similar to the Naive Bayes algorithm that trains a model from any of our given real world data sets to predict the class of an example from the respective data set. Training sets and test sets were assigned using 10-fold cross validation. Our data sets comprise of breast cancer data, glass data, iris data, soy bean data, and vote data, where each data set has its own defined labels for its features and class.

Most of our data sets had numerical data, which needed to discretized by a chosen number of bins. Furthermore, we implemented smoothing parameters to prevent zero conditional probabilities stored in our model. We tuned both the bin number hyperparameter and one of the smoothing hyperparameters to see which values were optimal for them based off of the average of our two evaluation measures, the zero one loss and the p macro. We hypothesized that the optimal bin number is 7 for numerical data, and the optimal 'm' value–thought of as the number of pseudo-examples–is 1 when computing conditional probabilities.

Through tuning our data sets we found that a bin number of 11 and a 'm' value of 23 was optimal for our data sets. We ran our algorithm multiple times to find the best hyperparameters, starting from the best hyperparameters of the previous test. Through this method we found that the algorithm settled on these values, which we are assuming are the best hyperparameters for the data that we have been presented with.

Keywords: Naive Bayes, Classification

1. Problem Statement

For any of the five given data sets, we could train our model to predict the class of any example from the respective data set. Our model can be trained on identifying the existence of breast cancer, the type of glass, the type of iris, the type of soybean, and the political party from their respective data sets.

Our model is based off of Naive Bayes. Naive Bayes predicts the class of a sample by a conditional probability with the condition of the sample having its features based off of the training set. The conditional probabilities are computed with the naive assumption that the features are conditionally independent given the class.

Our algorithm that we use is different from, though similar to, Naive Bayes. In a given training set, for each class, we compute,

$$Q(C = c_i) = \frac{\#\{\mathbf{x} \in c_i\}}{N}$$

where N is the number of examples in the training set. Then, for each attribute and each class, we calculate,

$$F(A_j = a_k, C = c_i) = \frac{\#\{(\mathbf{x}_{A_j} = a_k) \wedge (\mathbf{x} \in c_i)\} + 1}{N_{c_i} + d}$$

Then, to classify an example from the test set, we compute

$$C(\mathbf{x}) = Q(C = c_i) \times \prod_{j=1}^{d} F(A_j = a_k, C = c_i)$$

Then, we return

$$class(\mathbf{x}) = argmax_{c_i \in C} C(x),$$

which returns the class with the highest value for C($\mathbf{x}$).

We work with numerical data, which needs to be discretized in some way. Thus, we have to make a decision on the number of bins to describe our features with. We believe if we use too few bins, then, in general, our datasets will be underfittted, whereas if we use too many bins, then, in general, our datasets will be over-fitted. We hypothesize that the best number of bins is seven.

With this model, there is a possibility of having no data for an attribute-class combination, resulting in a zero probability in our trained model. To handle this issue, we introduce smoothing parameters to our conditional probabilities, which changed a probability from

$$P(A_j = a_k, C = c_i) = \frac{\#\{(\mathbf{x}_{A_j} = a_k) \wedge (\mathbf{x} \in c_i)\} + 1}{N_{c_i} + d}$$

to

$$P(A_j = a_k, C = c_i) = \frac{\#\{(\mathbf{x}_{A_j} = a_k) \wedge (\mathbf{x} \in c_i)\} + 1 + mp}{N_{c_i} + d + m},$$

where "m" can be thought of as a number of pseudo-examples, and "p" is a very low probability. We hypothesize that "m" should be 1 and "p" should be 0.0001.

2. Experimental Approach and Program Design

2.1 Data Preprocessing

2.1.1 Data Retrieval and Data Description

We downloaded the data sets as csv files from UCI Machine Learning repository at `https://archive.ics.uci.edu/ml/`. These data sets did not have the attribute names or the class name labeled to the columns, so we had to look at the data description text file to know what attribute column labels and class column label to append to each csv file. For

each data set, we had to consider where the class column of the data was located in order to properly assign the class label to each data set. The 'Vote' data set had the class column at the beginning of the data set, whereas the rest of the data sets had the class column at the end of the data set.

2.1.2 Replace Missing Values

Some of the data sets had missing or unknown values denoted by '?'. For each of these data sets, we replaced each '?' with the same value that appeared frequently within the data set. In particular, in the "Breast Cancer" data set, we replaced each '?' with '3', and in the "Vote" data set, we replaced each '?' with 'n'.

2.1.3 Discretize Numerical Values

We dealt with numerical data by discretizing them into a set number of bins. We used python pandas quantile-based discretizing function, 'qcut'. We evaluated our classification algorithm on various bin numbers to see what bin number was optimal for each data set.

2.2 Experimental Design

2.2.1 Two Different Versions of the Data

For each data set, we tested our classification algorithm on two different versions of the data. The first version is the original data set from the repository without change. The second version is altered from the original by randomly selecting 10 percent of the features–rounded up to the nearest integer–and shuffling the values within each of those selected features, producing noise in the second version.

2.2.2 10-Fold Cross Validation

To perform 10-Fold Cross Validation on a data set, we first partitioned the data set into 10 subsets of almost equal size. Using that partition, we tested our classification algorithm on each part of the partition by a model trained on the rest of the data.

2.2.3 Bin Size

As described earlier, we needed to bin our numerical data. The number of bins that we used is a hyperparameter that we tuned. The only data set that we did not use binning was the 'Vote' data set, which only had values 'y'–meaning 'yes'–and 'n'–meaning 'no'–for any feature column. The bin numbers that we that we tested on were all integers from 1 to 20.

2.2.4 The 'm' Integer and The 'p' Probability For Smoothing

For the smoothing out process, we used an 'm' value–an positive integer–and a 'p' value–a probability–for each classification. These were hyperparameters that we tuned. Rather than tuning them independently, we had 'p' be dependent on 'm' by the following formula:

$$p = \frac{1}{100m}.$$

The 'm' values that we tested for were all integers from 20 to 29.

2.2.5 Putting It All Together

For each of the original five data sets, the choices of added noise, assignment of the training and test data from our 10-fold partition, the bin size, and the 'm' value were all independent of each other. Thus, we ran a four-layer nested for loop to account for each of those independent choices and trained and tested our classification algorithm on the respective transformed and assigned data sets, which computed a predicted class for each value of the respective test data set.

3. Results

3.1 Data Analysis

For each of the data set we calculated a zero loss function and a p macro loss function that we eventually average together for each combination of bin number, M value and prob value. We then combine all of the different tables together to come up with a merged set of data.

	Data	Bin_Number	M_Value	Prob_Value	Zero_One_Loss_Avg	P_Macro_Avg	Average
0	Iris	11	20	0.000500	0.930000	0.811111	0.870556
1	Iris	11	21	0.000476	0.930000	0.811111	0.870556
2	Iris	11	22	0.000455	0.930000	0.811111	0.870556
3	Iris	11	23	0.000435	0.930000	0.811111	0.870556
4	Iris	11	24	0.000417	0.930000	0.811111	0.870556
5	Iris	11	25	0.000400	0.930000	0.811111	0.870556
6	Iris	11	26	0.000385	0.930000	0.811111	0.870556
7	Iris	11	27	0.000370	0.930000	0.811111	0.870556
8	Iris	11	28	0.000357	0.930000	0.811111	0.870556
9	Iris	11	29	0.000345	0.930000	0.811111	0.870556
10	Iris	12	20	0.000500	0.926667	0.738333	0.832500
...							
91	Iris	20	21	0.000476	0.923333	0.763333	0.843333
92	Iris	20	22	0.000455	0.923333	0.763333	0.843333
93	Iris	20	23	0.000435	0.923333	0.763333	0.843333
94	Iris	20	24	0.000417	0.923333	0.763333	0.843333
95	Iris	20	25	0.000400	0.923333	0.763333	0.843333
96	Iris	20	26	0.000385	0.923333	0.763333	0.843333
97	Iris	20	27	0.000370	0.920000	0.763333	0.841667
98	Iris	20	28	0.000357	0.916667	0.763333	0.840000
99	Iris	20	29	0.000345	0.916667	0.763333	0.840000

3.2 Merging Data

The merged set of data combines all of the analysis data sets together. From this we then find the average for each combination of bin number, m value, and prob value and use that average to find which combination of hyperparameters is the best.

	Data	Bin_Number	M_Value	Prob_Value	Zero_One_Loss_Avg	P_Macro_Avg	Average
0	Iris	11	20	0.000500	0.930000	0.811111	0.870556
1	Iris	11	21	0.000476	0.930000	0.811111	0.870556
2	Iris	11	22	0.000455	0.930000	0.811111	0.870556
3	Iris	11	23	0.000435	0.930000	0.811111	0.870556
4	Iris	11	24	0.000417	0.930000	0.811111	0.870556
5	Iris	11	25	0.000400	0.930000	0.811111	0.870556
6	Iris	11	26	0.000385	0.930000	0.811111	0.870556
7	Iris	11	27	0.000370	0.930000	0.811111	0.870556
8	Iris	11	28	0.000357	0.930000	0.811111	0.870556
9	Iris	11	29	0.000345	0.930000	0.811111	0.870556
10	Iris	12	20	0.000500	0.926667	0.738333	0.832500
...							
491	Vote	20	21	0.000476	0.898811	0.896118	0.897464
492	Vote	20	22	0.000455	0.898811	0.896118	0.897464
493	Vote	20	23	0.000435	0.898811	0.896118	0.897464
494	Vote	20	24	0.000417	0.898811	0.896118	0.897464
495	Vote	20	25	0.000400	0.898811	0.896118	0.897464
496	Vote	20	26	0.000385	0.898811	0.896118	0.897464
497	Vote	20	27	0.000370	0.899947	0.897607	0.898777
498	Vote	20	28	0.000357	0.899947	0.897607	0.898777
499	Vote	20	29	0.000345	0.899947	0.897607	0.898777

3.3 Summary Data

Then we average to find an average for each combination of hyperparameters.

Bin_Number	M_Value	Prob_Value	Average
11	24	0.000417	0.847443
	25	0.000400	0.847668
	26	0.000385	0.843631
	27	0.000370	0.840560
	28	0.000357	0.840072
	29	0.000345	0.835787
	30	0.000333	0.834195
	31	0.000323	0.834421
	32	0.000313	0.834183
	33	0.000303	0.833718
12	24	0.000417	0.830309
	25	0.000400	0.829582
	26	0.000385	0.830654
	27	0.000370	0.830904
	28	0.000357	0.829350
	29	0.000345	0.827584
	30	0.000333	0.826380
	31	0.000323	0.825791
	32	0.000313	0.830259
	33	0.000303	0.829964
...			
19	24	0.000417	0.821701
	25	0.000400	0.821351
	26	0.000385	0.818495
	27	0.000370	0.814423
	28	0.000357	0.813279
	29	0.000345	0.813127
	30	0.000333	0.814232
	31	0.000323	0.813714
	32	0.000313	0.814169
	33	0.000303	0.814948
20	24	0.000417	0.810603
	25	0.000400	0.808124
	26	0.000385	0.809645
	27	0.000370	0.809228
	28	0.000357	0.809346
	29	0.000345	0.808810
	30	0.000333	0.806560
	31	0.000323	0.806270
	32	0.000313	0.806458
	33	0.000303	0.805967

3.4 Tuning Data

From averaging everything we are able to find the best combination of hyperparameters which we can then start the next set of testing off with.

Bin_Number	M_Value	Prob_Value	Average
11	23	0.000435	0.862202

4. Conclusions

We found that our algorithm moved towards a bin value that was neither too high or too low. We estimated that this would be the case since we have some data sets with a wide range of values and others with a much more limited range. We concluded that if the algorithm picked too low of a bin number then underfitting would occur and the datasets that needed a wide array of values would lose their accuracy; whereas, the same would occur on the opposite side. If there is too high of a bin number then the datasets that did not need a wide array of values would be overfit, so through tuning we found that a bin size of 11 allowed the datasets be predicted the most often.

With the m value we found that a higher m value was preferred by the algorithm. We hypothesized that an m value of 1 would be more than enough and that raising the value would not change the prediction too much; however, the algorithm seemed to prefer a higher m value which eventually ended up being a value of 23.

5. Summary

In summary we found that a bin value of 11 and an m value of 23 allowed the algorithm to predict the right class most often in all of the datasets.