CS165A MP1 Report

Matthew Ho, msho@ucsb.edu, 8893265

February 2022

1 Code Architecture

1.1 `nb_classifier.py`

My NBClassifier class is defined in this module. Here is a description of each method:

1. **constructor**: takes no arguments and just declares member variables to store distribution parameters (I assume all continuous variables follow a Gaussian distribution) for continuous variables and MLE estimates for discrete variables.

2. **train**: takes the training dataset as input and gathers parameters and counts used at inference time. This method has an optional boolean argument that prints the training time.

3. **normpdf**: computes the probability of a Gaussian variable taking a particular value using the mean and standard deviation set by the train method. I implement this directly (instead of using scipy's implementations) to take advantage of numpy's vectorized operations.

4. **test**: takes the test set, a list of parameters to use, their corresponding weights, and optional flags for printing the predictions to stdout and printing the accuracy and time taken. This method adds two new columns to the test data 'p0' and 'p1' representing the log posterior probability of the corresponding hypothesis (Y = 0, 1) for each example (row). These columns are initialized to have all 0s and the inference process steps through each specified parameter, adding the log likelihoods from that parameter to each posterior column. After adding the contribution from each parameter, the model makes a prediction by taking the hypothesis with the higher posterior probability. Depending on the boolean flags supplied, the method prints predictions and/or the overall accuracy.

1.2 `custom_features.py`

I do minor feature engineering by computing bmi and categorizing the bmi and blood pressure columns based on online health charts.

1.3 `mcmc2.py`

I follow the Monte Carlo Markov Chain method of learning weights outlined in Zhang and Sheng 2004. In short, I start with weights of 1, pick a random direction to step in and keep walking in this direction so long as it doesn't decrease the accuracy. I incorporate the random restart search method of taking a random step when the accuracy remains unchanged for a number of iterations.

H. Zhang and Shengli Sheng, "Learning weighted naive Bayes with accurate ranking," Fourth IEEE International Conference on Data Mining (ICDM'04), 2004, pp. 567-570, doi: 10.1109/ICDM.2004.10030.

1.4 `nb_driver.py`

This is the driver program that reads in the training and test sets, initializes and trains the classifier, and finally outputs the predictions.

2 Preprocessing

I use the pandas library's DataFrame to store the datasets. Each row represents one example (each column is a feature).

3 Model Building

For each label (0 or 1), I take the subset of examples with that label and compute the mean and standard deviation for each continuous variable and use filtering and counting (MLE) to store the likelihoods for each value of discrete variables conditioned on the label. For example, the train method would take the "sit-up count" column (continuous) and find the mean and standard deviation separately for the label=0 and label=1 subsets. As for a discrete variable like gender, the train method would count how many of each discrete value (male or female) under each label's subset. The discerete variable counts and continous variable parameters fully describe the likelihoods needed for inference.

4 Results

Using all of the default parameters unweighted, the classifier gets .7797 accuracy. I ran all combinations of default parameters and found the best performing set achieves accuracy=.8026 using only parameters gender, weight, sit and bend forward, sit up count, and diastolic. I tried using the Monte Carlo method to learn weights, and my best performing weights (corresponding to all default parameters + my custom parameters) achieves 0.8468 accuracy. (All of the accuracy scores reported in the section is on the public test set)

5 Challenges

My first challenge was learning numpy and pandas. I previously had minimal experience with either library, but knowing their prevalence in both industry and academia, I definitely wanted to use them for this project. I got over this hump by spending time reading documentation and web tutorials. The more significant challenge was improving my model's accuracy. I wanted to do my best to avoid any manual labor (feature engineering, reworking architecture to model a more complicated Bayes net, etc.) and only use machine learning techniques to improve my classifier. I ended up spending far too long on this project, experimenting with a variety of approaches, only to improve accuracy by a few percents. The first thing I did after implementing the classifier was choosing parameters. I ran all parameter combinations against the public test set and used the six parameters listed in the previous section for the longest time (apparently a mistake since, all the parameters + weights ended up performing better). The next thing I tried was learning weights. Searching Naive Bayes literature led me to the Monte Carlo random direction method that proved to be most helpful. Even so, I was stuck on 0.81 accuracy for a whole week as I tried to tune some "hyperparameters" such as feature choice and step sizes. I even implemented a kind of greedy grid search for the next step (evaluating all combinations of $\{+\lambda, 0, -\lambda\}$ for each dimension). The final two things I tried were mild feature engineering (I added bmi and blood pressure categories), and discretizing the continuous variables into bins based on Gini impurity. The custom features ended up in my final submission's set of features, and the discretization was scrapped (though I suspect it definitely could have led to improvements if I spent more time experimenting). At times, this project was frustrating because it seemed many of the approaches I took yielded no benefit and therefore was wasted effort. Still, I had a fun time experimenting, and more importantly, I learned a great deal.

6 Weakness

The biggest performance jump I saw was with the Monte Carlo weight "learning." Despite helping my accuracy, I consider the method a weakness in itself because there's absolutely no guarantee that this performance is at a global or even a local max. There could be a huge jump one step away from the current weight vector, but the random direction never goes that way before getting stuck and making a suboptimal step in a different direction. Another weakness is not addressing the Naive assumption. A major issue with Naive Bayes is the strong assumption of conditional independence, and my classifier only slightly addresses this with the weighting. Given more time, I would have tried a semi-Naive approach that models a more elaborate Bayes net.