K-Nearest Neighbours, Naive Bayes and Decision Trees for Binary Classification of Low Birth Weight

Anagha Molkalmur, Parth Shah, Rajath S[1]

***Abstract*— This paper details the forecasting methods K-Nearest Neighbours, Naive Bayes and Decision Trees for predicting if a baby will be of low birth weight or not.**

I. Problem Statement

Low birth weight is used to refer to babies born with weight less than 2.5 Kg and this acts as an indicator of sickness in newborn babies. It has been identified as a serious public health problem the world over. Studies show correlations between maternal health during pregnancy and the child's birth weight. This can be considered as a binary classification problem of LBW or NOT-LBW for the newborn.[1] This forecasting method can be done using K-Nearest Neighbours, Naive Bayes and Decision Trees

II. Implementation

The data set provided consisted of 101 rows, with 9 attributes and 1 column for label. The label is discrete, with either 0's or 1's. The age, weight, BP, Hb columns are continuous, while the community, history, education, IFA, res columns are all discrete. The data cleaning was done in such a way that the average for the continuous variables and mode for discrete values, was used to fill in the missing values with respect to the labels, that is, average for age, weight, BP and Hb and mode for community, history, education, IFA and res were taken separately for label=0 and separately for label=1. The education column was dropped as all the values in the column were 5. The above was implemented using Excel VBA scripting.

A. K-Nearest Neighbours

K-Nearest Neighbours (KNN) is a lazy algorithm which means that it calculates the distances only when a query instance is provided to it. The data is first normalized for each row to a value between 0 and 1 and stored. Upon receiving a query, it calculates the distance against all existing instances. It then sorts these instances in the ascending order and picks the 'k' nearest ones. The class of these nearest instances are inspected and the class with the maximum count is picked as the classification of the query. Distances can be calculated using multiple techniques but we stick to classical geometrical methods since our data is a combination of discrete and continuous. The two distances employed are Manhattan Distance and Euclidean Distance. They are given respectively by:

[1]The authors are in alphabetical order as the contribution was equal. Machine Learning Assignment as part of UE17CS303 for PES University.

$$d = \sum_{i}^{n} abs(x_{query} - x_i)$$

$$d = \sum_{i}^{n} (x_{query} - x_i)^2$$

Where n are the number of attributes, i iterates through all the attributes. The difference is between the query instance and all the given instances.

To determine the optimal value of 'k', we shuffled the data set and ran the KNN classifier with a random 10% of the data as test data. Results in $k = 7$ for Euclidean and $k = 5$ for Manhattan. With a 100 instances, each classification takes about 0.00079 seconds. 1.73E13 are the number of possible shuffles. Running our classifier on 1E6 covers approximately 0.00001% of this sample space. These runs performed with Manhattan Distance outperforms Euclidean Distance in terms of accuracy by approximately 1.44%. These results are empirical with no rationale.

B. Gaussian Naive Bayes

Naive Bayes is a conditional probability model: given a problem instance to be classified, represented by a vector x=(x_1,…,x_n) representing some n features (independent variables), it assigns to this instance probabilities

$$p(C_k \mid x_1, \dots, x_n)$$

for each k classes.
Using Bayes Theorem the conditional probability can be decomposed as follows

$$p(C_k \mid \mathbf{x}) = \frac{p(C_k)\ p(\mathbf{x} \mid C_k)}{p(\mathbf{x})}$$

In plain English, the Bayesian Probability terminology can be expressed as:

$$posterior = \frac{prior \times likelihood}{evidence}$$

We calculate likelihood with the following formula

$$p(x = v \mid C_k) = \frac{1}{\sqrt{2\pi\sigma_k^2}}\, e^{-\frac{(v-\mu_k)^2}{2\sigma_k^2}}$$

where μ, σ stand for mean and variance respectively. We further assume that all features in a particular instance

X is mutually independent. Therefore we obtain the joint probability as follows

$$p(C_k \mid x_1, \ldots, x_n) = p(C_k) \prod_{i=1}^{n} p(x_i \mid C_k)\,,$$

Therefore, we finally obtain probability of a class given the features as

$$p(C_k \mid x_1, \ldots, x_n) = \frac{1}{Z} p(C_k) \prod_{i=1}^{n} p(x_i \mid C_k)$$

where Z is a scaling factor. We could also take log on both sides to obtain log probabilities which makes the equation stable
The corresponding classifier, a Bayes classifier, is the function that assigns a class label $\mathbf{\hat{y} = C_k}$ for some k. Thereby obtaining the following decision rule:

$$\hat{y} = argmax\ p(C_k) \prod_{i=1}^{n} p(x_i \mid C_k).$$

where $\mathbf{k} \in \{\mathbf{1}, \ldots, \mathbf{K}\}$
In the algorithm implemented by us, here are the following steps:

1) Split the data set into train and test
2) Calculate the prior probabilities for each class
3) Calculate mean and variance for every attribute of each class
4) If variance was observed to be 0 for any attribute, examine the data set and drop the corresponding columns.
5) Once we have all means and variances computed for every attribute of each class, we calculate the likelihood of every attribute for each class.
6) Calculate the posterior probability of each class and choose the class which has the maximum probability.Note: Log probabilities can be used for stability.

C. Decision Trees

The data is split into data for the algorithm to be trained on and data for the algorithm to be tested on. A decision Tree is constructed based on the training data. The construction of the decision tree involves:

1) checking for purity of data- Checking number of unique values in column after split, if only 1, pure data, classify the data points, return classification; else,
2) The list of potential splits for each column is obtained, based on average of 2 unique values.
3) The best split from the list of potential splits is obtained using the entropy:

$$E(s) = \sum_{i=1}^{c} -p_i \log_2(p_i)$$

Where $E(s)$ is the Entropy and p_i is the probability of that instance.

and overall entropy function:

$$H = P_{d\_b}(\sum_{i=1}^{c} -p_i \log_2(p_i)\,) + P_{d\_a}(\sum_{j=1}^{c} -p_j \log_2(p_j)\,)$$

Where H is the Overall Entropy, P_{d_b} is the Probability of data_below, P_{d_a} is the Probability of data_above and p_i and p_j are the probability of the respective instances.

4) Data is split at best split point, the data below split point is labelled 'data_below' and data above split point is labelled 'data_above'.
5) Feature names (column headers are extracted)
6) Sub-tree is constructed with the format: $(yes) < -question- > (no)$. Either answer can be another feature, hence a list data structure is preferred.

Pruning on the tree is implemented to achieve faster calculation times. Based on trial on various test cases, max depth of 7 and minimum number of 1 sample before proceeding with splitting the data is required, so as to reduce the computational time taken by the algorithm to classify. The test data is now run through the constructed decision tree and is classified. The accuracy of classification of test data is calculated taking the mean of correctly classified data. The algorithm is run over several test cases.

III. Visualizations

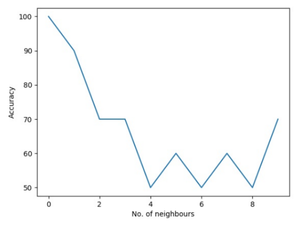


Fig1.0.0: Elbow Plot for 1 run

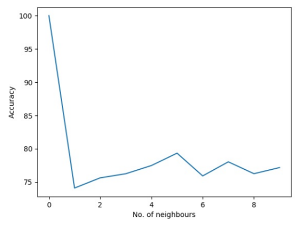


Fig1.0.1: Elbow Plot for 1E4 runs

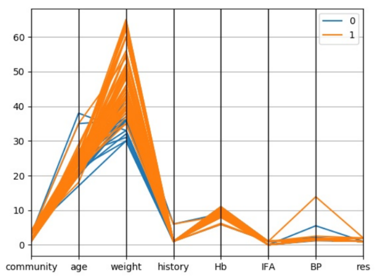


Fig1.1: Feature Distance Parallel Plot

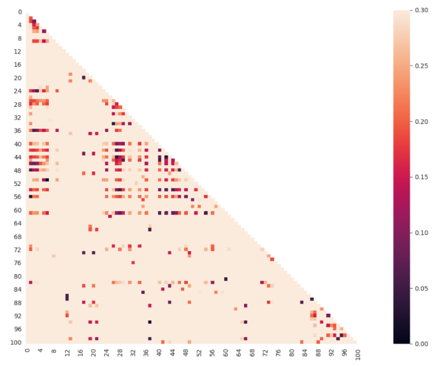


Fig1.2: Distance Heat Map

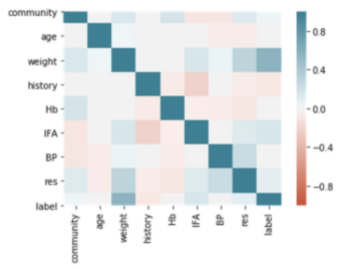
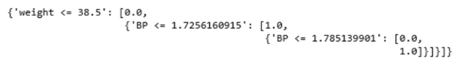


Fig 3.0: Attribute Correlation Heat Map



```
{'weight <= 38.5': [0.0,
                    {'BP <= 1.7256160915': [1.0,
                                            {'BP <= 1.785139901': [0.0,
                                                                   1.0]}]}]}
```

Fig 3.1: Decision Tree

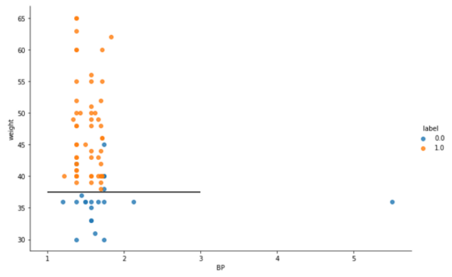


Fig 3.2: Scatterplot for weight and BP

IV. Results

A. K-Nearest Neighbours

For a neighbour count of 5 with Manhattan Distance, this model gives us the highest accuracy. The highest average accuracy is 79%. However with some shuffles, we get an accuracy of 90%.

B. Gaussian Naive Bayes

Gaussian Naive Bayes achieves a max accuracy of 93.75% and an average accuracy of 77.29% over 1000 train-test shuffles.The train test split had a ratio of 9:1.

C. Decision Trees

The highest average accuracy is 90.10% and the highest accuracy to classify test data, reached with a few shuffles, is 100% .

V. Conclusions

A. K-Nearest Neighbours

An average accuracy of 80% indicates that the distance between the data points of each class are not completely representative of their classes. This is confirmed by the plots.

B. Gaussian Naive Bayes

It is a simple and fast classifier and provides impressive accuracy.

C. Decision Trees

The average efficiency of about 92% indicates the inter dependence of the attributes.

References

[1] Early Prediction of LBW Cases via Minimum Error Rate Classifier: A Statistical Machine Learning Approach by Anisha R. Yarlapati; Sudeepa Roy Dey; Snehanshu Saha. Electronic ISBN: 978-1-5090-6517-2 Print on Demand(PoD) ISBN: 978-1-5090-6518-9