Report on Naive Bayes Classification of Hand Written Letter Data

Saswot Nayak

14th November 2020

1 Introduction

This report discusses the implementation of Naive Bayes classifier on the given data, as a project of the CS561 course. We will go through the workings of the Naive Bayes technique, discuss the workings of the implemented model in relation to the data, performance of the model under different parameters and comparing against **scikit-learn** model, and further possible improvement for accuracy of the model.

2 Problem Statement

One has to implement the Naive Bayes to solve a classification problem without using any libraries or modules.

2.1 Model

The Naive Bayes model can be either Multinomial or Gaussian in nature. But here only Gaussian Naive Bayes implemented.

2.2 Data

The given data is a **CSV** file containing 372450 rows and 785 columns, Where each full row represents one image of handwritten letter and each column starting from 1st columns to 785th column represents pixel values of image. So every image is basically of size (28 * 28). i.e. 784 pixel. And the 0th row denotes the label corresponding to the given image. The labels are encoded such that all English letters A to Z, encoded as 0 to 25 respectively.

3 Naive Bayes

It is a classification technique based on Bayes' Theorem with an assumption of independence among predictors. In simple terms, a Naive Bayes classifier

assumes that the presence of a particular feature in a class is unrelated to the presence of any other feature. It is very much suitable for our data, as each pixel value are not co-related among themselves.
The Bayes' theorem given as ,

$$P(A|B) = \frac{P(B|A) \cdot P(A)}{P(B)}$$

Where,
P(A | B) = Probability of A given B
P(B | A) = Probability of B given A
P(A), P(B) = Independent probability of A and B

More specifically we can write the above formula as,

$$\begin{aligned} P(y^t = k|D) &= \frac{P(y^t = k, D)}{P(D)}, \text{where } D = (X_1 = x_1^t, X_2 = x_2^t .....X_{784} = x_{784}^t) \\ &= \frac{P(y^t = k) \cdot P(X_1 = x_1^t, X_2 = x_2^t ....|y^t = k)}{\sum_{k=0}^{25} P(y^t = k, D)} \\ &= \frac{P(y^t = k) \cdot \prod_{i=1}^{785} P(X_i = x_i^t|y^t = k)}{\sum_{k=0}^{25} P(y^t = k, D)} \qquad \text{Each features (pixels) are independent.} \\ &= \frac{P(y^t = k) \cdot \prod_{i=1}^{785} P(X_i = x_i^t|y^t = k)}{\sum_{k=0}^{25} P(y^t = k) \prod_{i=1}^{785} P(X_i = x_i^t|y^t = k)} \end{aligned} \tag{1}$$

$Where, P(y^t = k|D) = Posterior$
$P(y^t = k) = Prior$
$\prod_{i=1}^{785} P(X_i = x_i^t|y^t = k) = Likelihood$
$\sum_{k=0}^{25} P(y^t = k) \prod_{i=1}^{785} P(X_i = x_i^t|y^t = k) = Marginal/Evidence$

In above equation probability is very small quantity, i.e. $0 < p < 1$. So we can take log across numerator and denominator. As our desired probability is proportional to numerator. So dropping Marginal probability won't affect our result as must. So our modified equation will be,

$$P(y^t = k|D) \propto \log P(y^t = k) + \sum_{k=1}^{785} P(X_i = x_i^t|y^t = k) \tag{2}$$

As I have implemented Gaussian Naive Bayes as opposed to Multinomial one. The above equation can be written as

$$\begin{aligned} P(y^t = k|D) &\propto \log P(y^t = k) + \sum_{k=1}^{785} P(X_i = x_i^t|y^t = k) \\ \text{Where, } P(X_i = x_i^t|y^t = k) &= e^{-\frac{(x-\mu_k)^2}{2\sigma_k^2}} \cdot \frac{1}{\sqrt[2]{2\pi\sigma_k^2}} \end{aligned} \tag{3}$$

4 Implementation

This section gives insights to how the implementation of Naive Bayes Classifier done and what sort of measures taken in-order to make efficient implementation.

4.1 Data Preparation

The given CSV file loaded into Pandas DataFrame called "data". After looking inside the data briefly, some of the images get printed to get the visual idea of the data. Before prepping data i.e. the main DataFrame, data shuffled and split into train and test sets such that 90% data goes to train set and remaining 10% data goes into test set. Further test and train set got divided into four sets.

X_train = This set contains all the features from train set in form of Pandas DataFrame.

Y_train = This set contains all the corresponding label from train set in form of Pandas Series.

X_test = This set contains all the features from test set in form of Pandas DataFrame.

Y_test = This set contains all the corresponding label from test set in form of Pandas Series.

Then the index of four data sets reset as shuffling makes the order random. All the features in test and train set i.e. X_train, X_test are normalized by min-max normalizing method, in our data simply dividing each features by 255 (Pixel value varies (0,255)). Normalized data gives ease of calculation afterwards. Then all four data sets converted into Numpy Ndarray.

4.2 Model building

The **Gaussian Naive Bayes** implemented as a Python class which has four methods to fit the training data set and to predict on testing data set.

Training set got fit into model by calculating prior for each unique label class and mean, variance for each feature per unique class label. Smoothing parameter is added to variance in order to avoid **"divide by zero"** error and smoothing parameter gives in accuracy boost as it solves the conflict for corner most pixel. Model ran with different **smoothing factor** to check the behaviour of the model.

Then all features of a image from test set passed through predict method to calculate posterior using the **Gaussian probability Mass Function** with mean and variance from earlier. posterior for all unique label calculated for each set of features (one image). Then only that label is selected which has highest probability among other labels.

Above procedure followed for each image in the test set one by one and the predicted label got recorded into Y_pred Ndarray.

4.3 Evaluation of model

In order to evaluate the model the predicted label got compared to the original label of test set. Accuracy metric has chosen to evaluate the performance of the model.

Accuracy given by, $accuracy = \frac{\text{No of correct predicted label}}{\text{Total label}}$

Confusion matrix has drawn for all the labels, to see where exactly model did wrong. It also tell us the balance of the given data.

5 Observation

- My model gave 69.47% accuracy when fit with smoothing factor $10e-3$.It took 56.85 sec to predict results on the test set. If looked closely into confusion matrix ,its evident that model poorly recognize letters like 'O','S','U','X'. Though in case of other letters model did not do immensely great, but model performed very poor for above letters.

- Model also ran with different smoothing factors like $\{10e-1, 10e-2, 10e-3, 10e-4\}$. Model got accuracy improved With each smoothing factor, but after smoothing factor $10e-3$, It dipped in accuracy. So best possible smoothing factor taken into consideration i.e. $10e-3$.

  My model gave,
  accuracy 62.48% with smoothing factor $10e-1$.
  accuracy 67.26% with smoothing factor $10e-2$.
  accuracy 69.46% with smoothing factor $10e-3$.
  accuracy 68.14% with smoothing factor $10e-4$.

- I have also tried without smoothing factor, Model raised "divides by zero" error. So I have added a small bias value to the numerator in**PDF** calculation. Though model ran correctly, but the accuracy got was poor i.e. 33.45%.

- I have also tried other solution before finalizing smoothing factor to variance. I tried changing pixel value to 1 those have 0. Essentially i tried to avoid the "divide by zero" error. But then the strategy failed as mean and variance of those pixel values (corner most pixels) got zero. Again it gave '0' probability from **Gaussian PDF** and then 'log(0)' gave error.

- At last I have also compared the performance of own Gaussian Naive Bayes with **Scikit-Learn Gaussian and Multinomial Naive Bayes**

implementation. My model performed fairly good compared to **Scikit-Learn** implementation.

My model gave accuracy 69.46% and takes 57.68 sec to predict.
Scikit-Learn Gaussian model gave 55.68% and takes only 4.04 sec to predict. **Scikit-Learn** Multinomial model gave 70.95% and takes only 0.22 sec to predict.

Though my model performed okay, But it fails miserably in run time. My honest estimation is it has to do with poor data processing technique..

6 Conclusion

Though my model fairly did with hand written letter data set against **scikit-learn model**. But there can be much more improvement in accuracy of the model. Given time frame of the project it was not possible for me to try more different things. But given chance i would like to try some of the below techniques to improve the performance of the model.

6.1 Data improvement

If we look at data, we can sense that data is not balanced properly. Some letters has more data and some has less. It severely affects the accuracy of the model. I would like to curate more data for those letters.

6.2 Model improvement

I have tried my model with smoothing factor $10e-3$ and then $10e-4$, where accuracy dropped. I would like try the in between values for smoothing factor.

6.3 Run-time improvement

Though my model gives good accuracy, it did poorly in run-time. I would try with different other data structures for my data other than Ndarray,and select that suits the best. I would try to structure my program in different way so that at a time large number of data need not be processed, that may reduced the run time.

7 References

- **Scikit-Learn Implementation:** Click Here
- **Naive Bayes on MNIST:** Click Here
- **Data Normalization:** Click Here