BITS F464 - Machine Learning
Assignment – 2

Under Prof : NL Bhanumurthy

Done By:

1.	**Rahul Karna L K**	**2020AAPS0437H**
2.	**Diyya Kalyan Sai**	**2020A4PS1522H**
3.	**Kunchala Sri Vatsav Reddy**	**2020A7PS0274H**

Part A – Naive Bayes Classifier to Predict Income

1.Description of Algorithm/Model:-

Naïve Bayes Classification relies on conditional probability to help classify unknown samples. In our case of income filtering we want to classify unseen incomes/samples as more than $50k per year, and less than $50k per year.

For this, we use the equation given below: □(income≥ $50k |□1, □2 , . . , □□) ∝ □(income≥ $50k) × ∏ □(□□ |income≥ $50) [Pi running from i =1 to n]

Here, w1, w2,… wn represents all the words present in the data to be classified. The denominator would be the same in both cases, and hence relevant to the final decision being made.

In our model income are classified as more than or equal to $50k per year if:

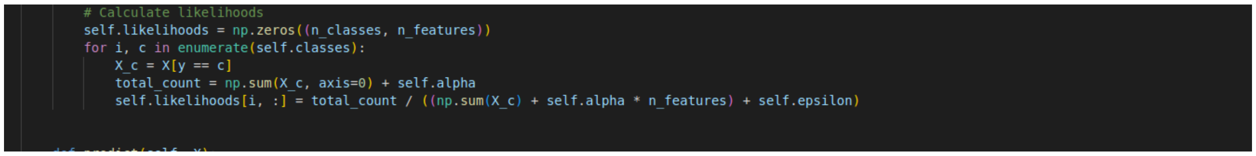
□(income≥ $50k) ×∏□(□□ |income≥ $50k) ≥ □(income<$50k) ×∏□(□□ |income<$50k) [In both Pi running from i =1 to n]

And as more than $50k per year otherwise.

One case that needs to be taken care of is when there is a word in the email to be tested, which is not present in the training data. We are presented with two trivial solutions to this dilemma:

- Ignore the word. But this would mean we are considering P(w|income≥ $50k) and P(w|income< $50k) both as the same which would be illogical.
- Take probability as 0 as it is not occurring in either dataset. But this would mean we are considering P(w|income≥ $50k) and P(w|income< $50k) both as 0 which is also incorrect.

As neither of these would be appropriate, we resort to ***Laplace smoothing*** to take care of the zero probability in Naïve Bayes. The formula is for Laplace smoothing is given as:



```
    # Calculate likelihoods
    self.likelihoods = np.zeros((n_classes, n_features))
    for i, c in enumerate(self.classes):
        X_c = X[y == c]
        total_count = np.sum(X_c, axis=0) + self.alpha
        self.likelihoods[i, :] = total_count / ((np.sum(X_c) + self.alpha * n_features) + self.epsilon)
```

P(w|income≥ $50k) = □□□□□□ □□ income≥ $50k data □□□*h* □*h*□ □□□□ □ + □ □□□□□□ □□ income≥ $50k data + (□□□□□□ □□ □□□□□□□□□) × □.

Here □ is the smoothing parameter, which is usually taken as 1. Similar formulas can be written for income<$50 case as well.

2.Tasks-

Task 1: Data Preprocessing
We first pre process the data by normalizing the data (Normal z-normalization) and also filling missing data using the average/mode of the data. We then create 67:33 split of the dataset for training and testing using pandas DataFrame.

Task 2: Naive Bayes Classifier Implementation
Instead of implementing functions separately we create a Naive Bayes class, this is used to perform the following functions:-

- def fit: Calculate Prior Probability and likelihood of each Class in the training dataset
- def predict: Calculates posterior for each class and predict the class with highest posterior
- def prior(self, c), def posterior(self, x, c), def likelihood(self, x, c): These three are used to call/access the calculated prior, posterior and the likelihood probabilities respectively.

Task 3: Evaluation and Improvement

- def precision(self, X, y), def recall(self, X, y), ef f1_score(self, X, y) def accuracy(self, X, y). These 4 are used to calculate the test statistics to figure out how efficient/good the model is.

Results obtained:-
Experiment with different smoothing techniques to improve the performance of your classifier. You need to study and understand the different smoothing techniques on your own.

```
Accuracy: 0.778079164155113
Precision: 0.919023
Recall: 0.288772
F1 score: 0.219730
```

For Various Random split of data	**ACCURACY**
1.	0.778079164155113
2.	0.802342342354234
3.	0.767876123124213
4.	0.772758176231233
5.	0.771213984298231
6.	0.812868761212123
7.	0.792312312312313
8.	0.779758751231233
9.	0.745459879823433

10.	`0.854766253123334`

3. Comparing the performance of our Naive Bayes classifier with other classification algorithms like logistic regression and k-nearest neighbors.

LR: 0.8207755676110107
KNN 0.8272051436608399
NB 0.78986871521386928

We observe that LR and KNN are giving better results than NB, this could be because of the following limitations of Naïve Bayes:-

4. Major limitations of the Naive Bayes classifier:
Naïve Bayes classifier gets "naïve" in its name because it assumes that the conditional probabilities are independent of one another and do not affect one another. This is known as ***naïve conditional independence assumption***. In other words, it assumes that the predictor features are all mutually independent. However, this is seldom the case in real life situations. Another drawback is that for the income classifier, this model gives ***no importance to the order of words***. While calculating probabilities, we just rely on the frequency of words occurring, like we are drawing a word from a bag of words. Every language has grammar rules and common phrases but Naïve Bayes ignores all of that.

KNN has the highest accuracy because it doesn't make any assumptions where as LR makes assumption of linear relationship between the features and outcome variable/class. KNN and LR being more than NB could indicate that the independent assumption of Naïve Bayes might not be exactly true.

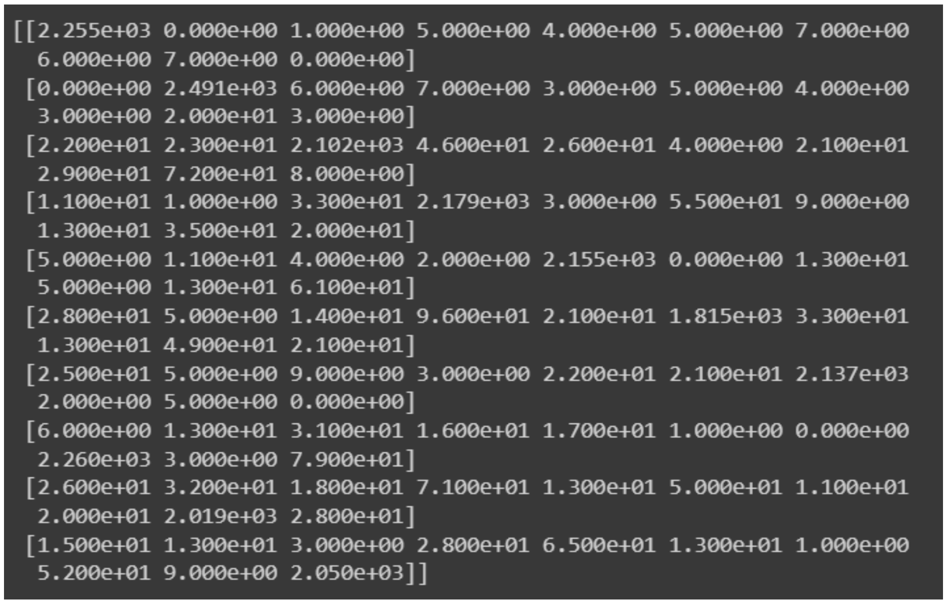
-Part B: Building a Basic Neural Network for Image Classification

The most basic neural network architecture which was built here for image classification consists of an input layer, one or more hidden layers, and an output layer. Each layer contains a number of neurons, and each neuron is connected to neurons in the previous and next layers. The input layer receives the input data, which is passed through the hidden layers, where computations are performed on the data, and finally, the output layer produces the final output.

The task at hand was to build a neural network which would correctly identify the handwritten digits from the MNIST dataset with as much accuracy as possible.

Implementation of code:
First the data set was loaded into the notebook, and the 28*28 matrix was normalized and flattened into a 2D array. Then the training and testing data was split into 70/30 or 67/33 wherever necessary and then the neural network was built.
We used keras to implement the neural network. First a basic neural network with no hidden layers was built to check the correctness of the algorithm. Keras was used in implementing the neural network.
Then slowly we built our way up towards ⅔ hidden layers with varying no. of neurons in each hidden layer (100/150) and each layer had a different activation function. This was to help us understand which model performs best subject to different conditions. Adam optimizer and sparse_categorical_crossentropy were used as optimization and loss algorithms. Finally the accuracy of the testing data, and confusion matrix for all the 12 models was printed out.

The confusion matrix for 2 hidden layers and sigmoid as activation function looked as follows:



```
[[2.255e+03 0.000e+00 1.000e+00 5.000e+00 4.000e+00 5.000e+00 7.000e+00
  6.000e+00 7.000e+00 0.000e+00]
 [0.000e+00 2.491e+03 6.000e+00 7.000e+00 3.000e+00 5.000e+00 4.000e+00
  3.000e+00 2.000e+01 3.000e+00]
 [2.200e+01 2.300e+01 2.102e+03 4.600e+01 2.600e+01 4.000e+00 2.100e+01
  2.900e+01 7.200e+01 8.000e+00]
 [1.100e+01 1.000e+00 3.300e+01 2.179e+03 3.000e+00 5.500e+01 9.000e+00
  1.300e+01 3.500e+01 2.000e+01]
 [5.000e+00 1.100e+01 4.000e+00 2.000e+00 2.155e+03 0.000e+00 1.300e+01
  5.000e+00 1.300e+01 6.100e+01]
 [2.800e+01 5.000e+00 1.400e+01 9.600e+01 2.100e+01 1.815e+03 3.300e+01
  1.300e+01 4.900e+01 2.100e+01]
 [2.500e+01 5.000e+00 9.000e+00 3.000e+00 2.200e+01 2.100e+01 2.137e+03
  2.000e+00 5.000e+00 0.000e+00]
 [6.000e+00 1.300e+01 3.100e+01 1.600e+01 1.700e+01 1.000e+00 0.000e+00
  2.260e+03 3.000e+00 7.900e+01]
 [2.600e+01 3.200e+01 1.800e+01 7.100e+01 1.300e+01 5.000e+01 1.100e+01
  2.000e+01 2.019e+03 2.800e+01]
 [1.500e+01 1.300e+01 3.000e+00 2.800e+01 6.500e+01 1.300e+01 1.000e+00
  5.200e+01 9.000e+00 2.050e+03]]
```

MODEL	ACCURACY

2 hidden layers, 100 neurons, sigmoid	0.9751
2 hidden layers, 100 neurons, tanh	0.9850
2 hidden layers, 100 neurons, relu	0.9857
2 hidden layers, 150 neurons, sigmoid	0.9803
2 hidden layers, 150 neurons, tanh	0.9869
2 hidden layers, 150 neurons, relu	0.9873
3 hidden layers, 100 neurons, sigmoid	0.9788
3 hidden layers, 100 neurons, tanh	0.9833
3 hidden layers, 100 neurons, relu	0.9844
3 hidden layers, 150 neurons, sigmoid	0.9803
3 hidden layers, 150 neurons, tanh	0.9852
3 hidden layers, 150 neurons, relu	0.9859

According to the accuracy scores, the best model is *2 layers, 150 neurons, relu* with an accuracy of 98.73%

Since, the accuracy values are similar (+-2%) the models are statistically insignificant.