Assignment 2

Machine Learning: BITS F464

Submitted To

Prof. Bhanu Murthy

BITS Pilani Hyderabad Campus

Submitted By

Name	BITS ID
Aashutosh A V	2021A7PS0056H
Saurabh K Atreya	2021A7PS0190H
Tarimala Vignesh Reddy	2021A7PS0234H

Birla Institute of Technology and Science, Hyderabad Campus, India

Acknowledgements

We are grateful to have been given the opportunity to work on this project. We had a great opportunity to work together as a team, and collectively this was a great learning experience in enhancing our Machine Learning skills. Applying theoretical knowledge from class into code and getting positive results has been an enriching experience, and learning to work with new libraries has been very enjoyable to say the least.

Aashutosh
Saurabh
Vignesh

Contents

Building a Naive-Bayes Classifier to predict income on the Adult Dataset

Methodology

The first step to build a classifier for this dataset was to heavily pre-process the data, considering the number of categorical columns present in this dataset.

We converted all the categorical columns to numerical data, and then ran our Naive Bayes algorithm on it.

The approach towards building the Naive Bayes Classifier was to use conditional probabilities, and posterior probabilities with the presumption that the data columns are mostly independent of each other.

This was necessary because the product of generative probabilities based on conditionality is very dependent on the fact that the data columns are independent of each other or less dependent enough that they can be presumed independent.

We first calculated the prior probabilities of each class, assuming a standard normal distribution on the dataset.

Then we calculate the posterior probability by multiplying the prior probability with the conditional probability of the feature.

Importantly, we also applied the Laplace Smoothing technique on it as follows:

For solving the problem of zero probability, we used the following formula:

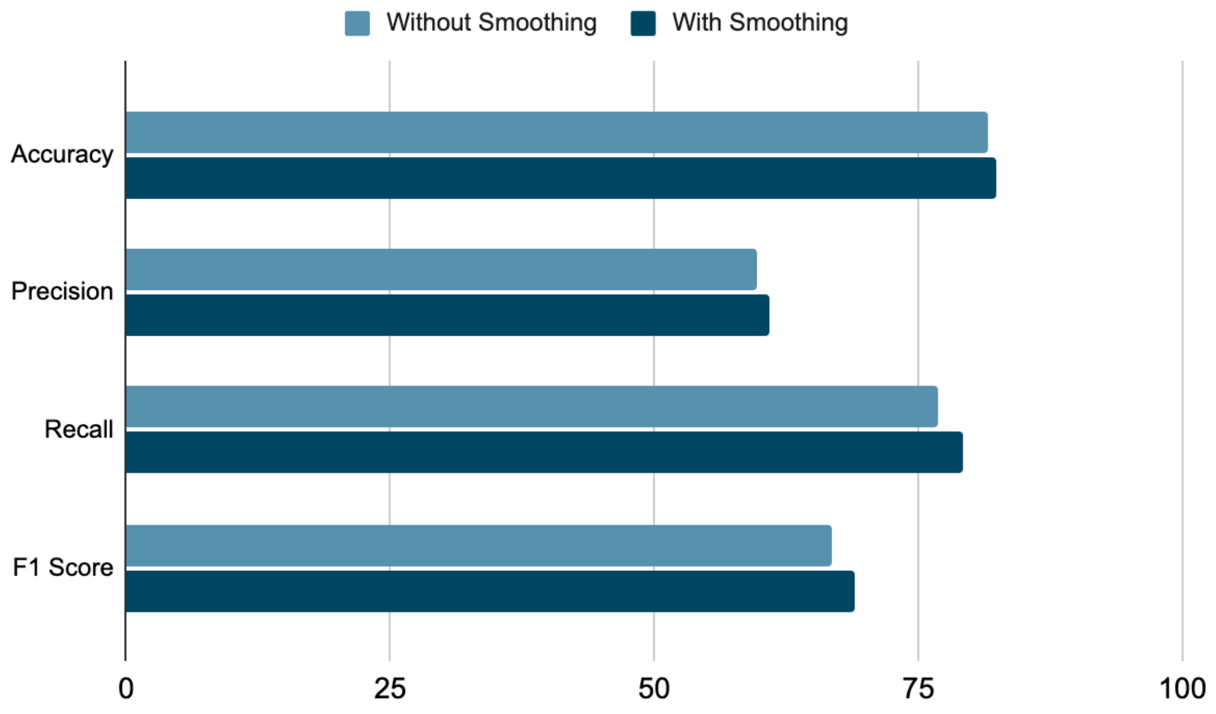
$$P(w'|positive) = \frac{\text{number of reviews with w' and y = positive} + \alpha}{\mathrm{N} + \alpha * K}$$

As alpha increases, the likelihood probability moves towards uniform distribution (0.5). Theoretical estimates say that using alpha = 1 in this formula is generally the best way to tackle the issue of zero probability, which is what we decided to do.

We also performed a 10-split on the data, in an attempt to maximize the utility of the model, for any test input.

Compiled Results

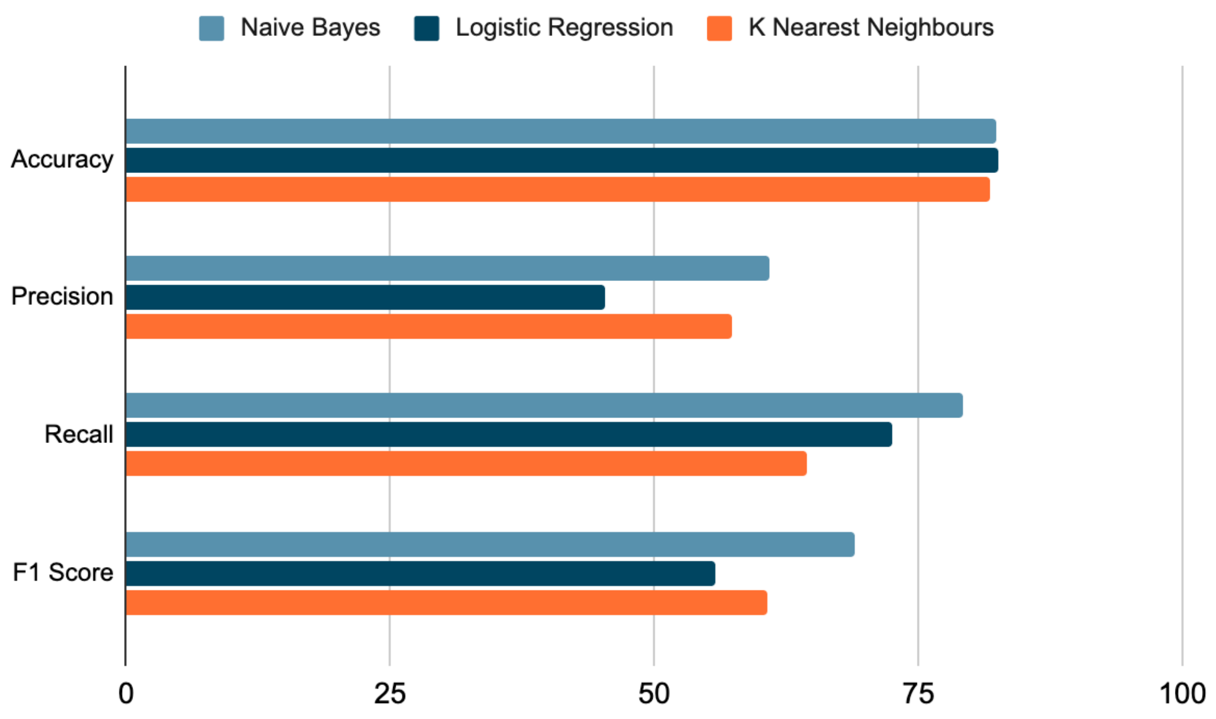
We initially decided to compare the Naive Bayes classifier's outputs before and after smoothing:



	Without Smoothing	With Smoothing
Accuracy	81.63	82.4
Precision	59.64	60.93
Recall	76.9	79.27
F1 Score	66.87	68.9

The fact that the metric values after smoothing aren't significantly better than the ones before smoothing indicates that the dataset is already quite well balanced and the feature values are not too sparse, leading to zero probabilities not being a significant issue.

We then compared the model post smoothing to the scores of Logistic Regression and the K-Nearest Neighbours classifier, and found the following results.



	Naive Bayes	Logistic Regression	K Nearest Neighbours
Accuracy	82.4	82.5	81.88
Precision	60.93	45.26	57.34
Recall	79.27	72.62	64.39
F1 Score	68.9	55.76	60.66

Conclusion

The observed values of the metrics don't seem to differ much, but just out of magnitude of the values, we can say that the Naive Bayes and Logistic Regression models seem to perform better than the K Nearest Neighbours model in accuracy, though the Naive Bayes Model clearly beats the other two in terms of Precision and Recall Score.

Building a Basic Neural Network for Image Classification

Methodology

We used the PyTorch Framework for building this Neural Network. Considering the options given to us regarding the possible architectures, we used the following combinations:

1. Either 2 or 3 Layers
2. Hidden Layers sized 100 or 150
3. Activation Functions among ‘tanh’, ‘relu’ and ‘sigmoid’
4. Optimizers between ‘Adam’ and ‘SGD’
5. Loss Function: Categorical Cross Entropy
6. Output Function: Softmax

In processing the data, we have one-hot-encoded the output layer to categorical outputs ranging from 0-9, which are the outputs for the MNIST dataset.

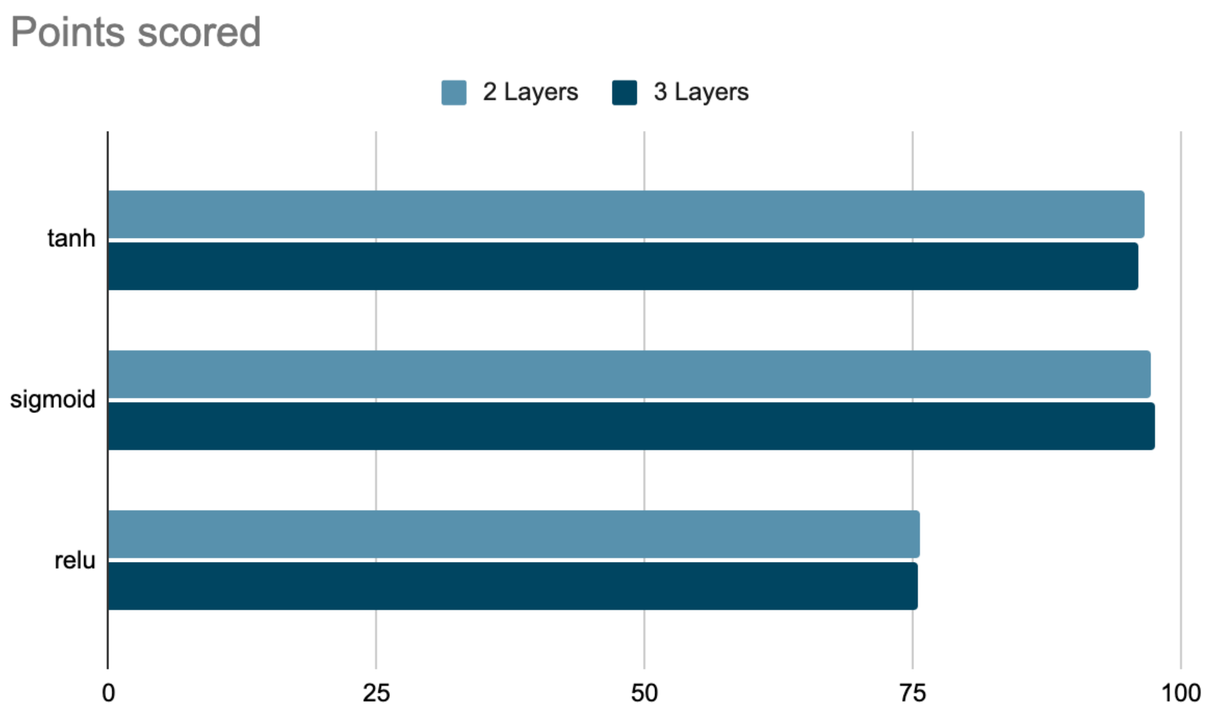
In training the models for the combinations mentioned above, we used 10 training epochs with batch size 32 on each model with a learning rate of 0.001 on the optimizer function. The corresponding test results were recorded for the trained models.

Compiled Results

The following is a compilation of the average test accuracies obtained from the corresponding neural networks, for 10 random 67%-33% splits performed on the entire MNIST dataset.

Number of Layers/ Hidden Layer Size		**Optimizer**	**Activation Function**		
			tanh	**Sigmoid**	**ReLU**
2	100	SGD	88.89%	59.36%	72.77%
2	100	Adam	96.49%	96.73%	68.56%
2	150	SGD	88.83%	60.97%	75.58%
2	150	Adam	96.66%	97.27%	61.89%
3	100	SGD	88.29%	11.30%	71.38%
3	100	Adam	95.94%	97.52%	73.75%
3	150	SGD	88.52%	11.29%	75.76%
3	150	Adam	96.01%	97.64%	61.43%

The following are the best output accuracies based on activation function & number of layers



It is important to note that all of these best accuracies for both 2 Layer networks and 3 layer networks are much better using the Adam optimizer than the SGD optimizer for tanh and sigmoid, although the result was otherwise for Relu.

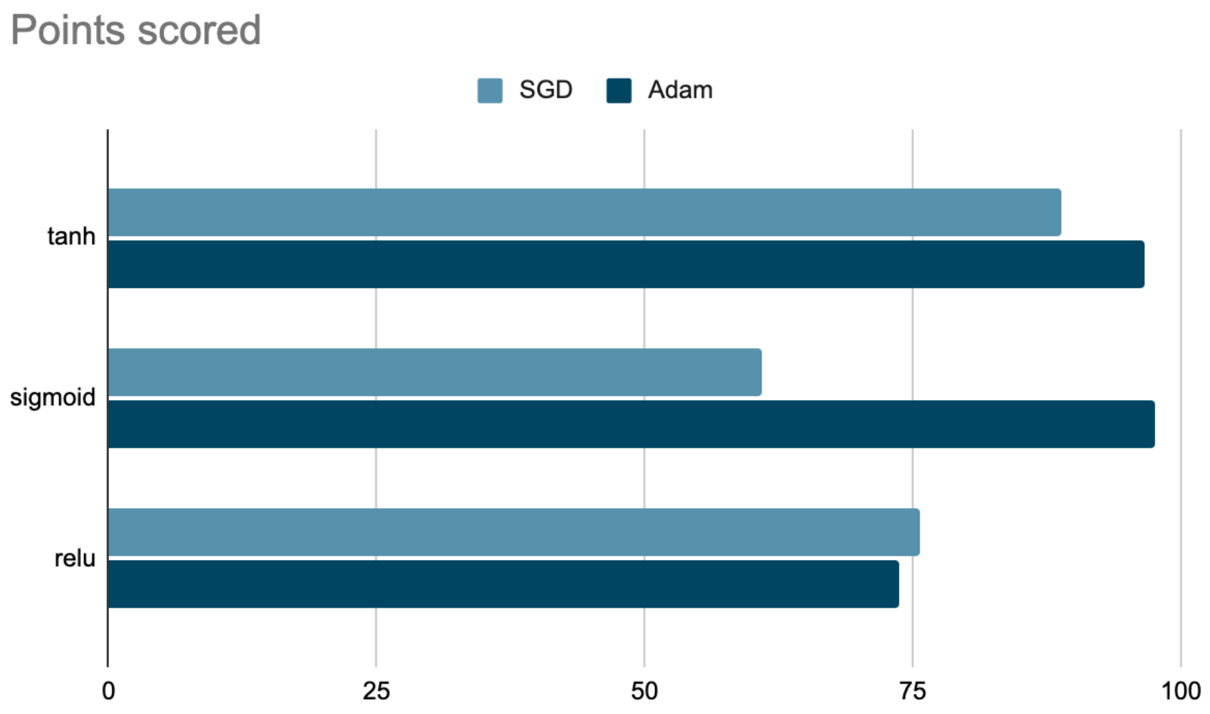
	2 Layers	3 Layers
tanh	96.66	96.01
sigmoid	97.27	97.64
relu	75.58	75.56

It is also interesting to observe that the accuracies for the 2-Layer-Networks are better than the 3-Layer networks. This is in accordance with Ockham’s Razor principle as the simpler model is performing better here.

One more point to consider is the number of neurons used in the hidden layers.

Our outputs indicate that we get better accuracies using 100 neurons in the hidden layers for the tanh and Relu activation functions in both 2-Layer & 3-Layer networks, although 150 neurons seems to be the better choice when using the Sigmoid activation function.

For comparison, the following chart illustrates the difference between the accuracies obtained using the 2 optimizers:

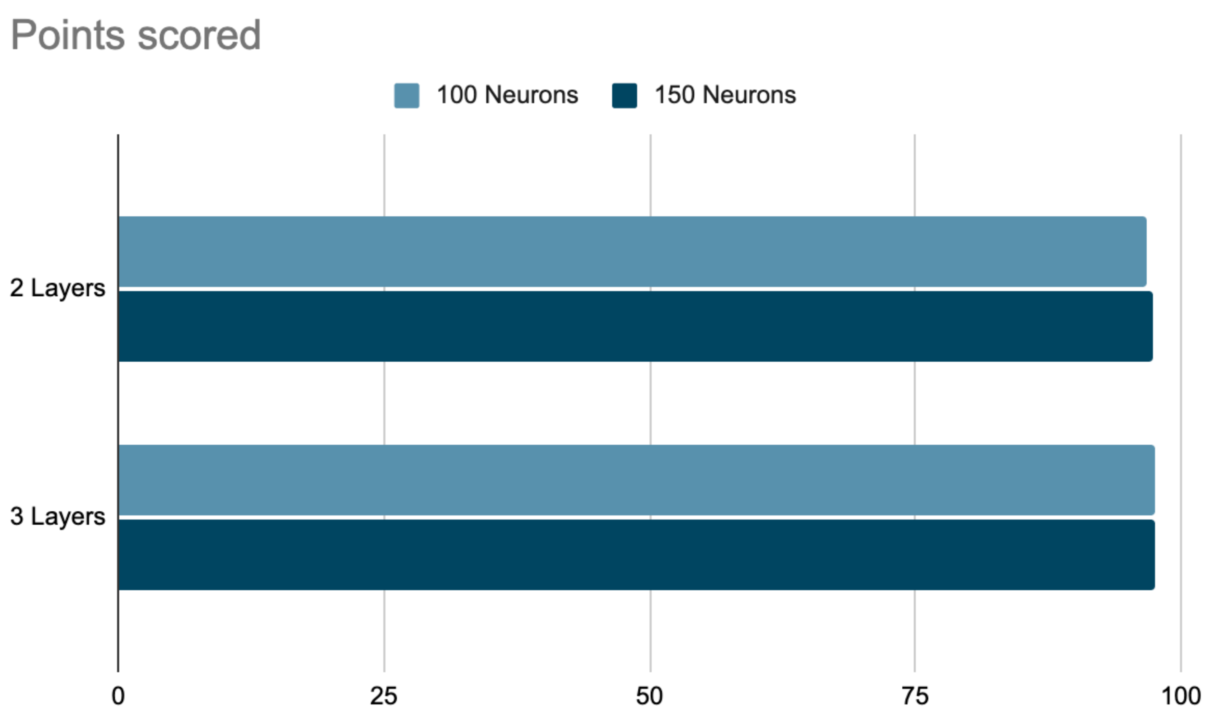


Although the Adam optimizer clearly beats the SGD optimizer, we can surely draw a conclusion that the SGC optimizer does work better in combination with the Relu activation function than the others.

Looking at the following data gives a clearer picture on what activation-function-optimizer pair seems to yield the best results

	SGD	Adam
tanh	88.89	96.66
sigmoid	60.97	97.64
relu	75.76	73.75

We also decided to check for any observations only based on the number of neurons used, and these were the results:



	100 Neurons	150 Neurons
2 Layers	96.73	97.27
3 Layers	97.52	97.64

Note: All these best accuracies came with using the Sigmoid function for activation.

Conclusion

We also observed that for the given dataset and the framework we used, 150 neurons seemed to be the better performer in terms of best accuracy. However, 3 layer neural networks with 100 neurons and 2 layer models with 150 neurons are not statistically significant from the best model as their accuracies are similar.