# BITS F464 : Machine Learning Assignment – 2 Report

## Team Members

1. Kushagra Verma - 2020A7PS0225H
2. Sai Panda - 2020A7PS0080H
3. Khooshrin Aspi Pithawalla - 2020A7PS2067H

# Part A: Naive Bayes Classifier to predict income

## Task 1

### The Dataset

The dataset provided had 32561 examples, with each example containing 14 features and a label indicating whether the annual income was 50K. Out of the 14 features, 6 features had continuous values (namely, "age", "fnlwgt", "education-num", "capital-gain", "capital-loss" and "hours-per-week") and the remaining 8 features had discrete values (namely, "workclass", "education", "marital-status", "occupation", "relationship", "race", "sex" and "native-country"). There were 24720 examples with income 50K and 7841 examples with income 50K.

#### Fill Missing Values

There were 2399 examples which had one or more missing values. We filled the missing values by going over each column and replacing the missing values with the most commonly occurring non-missing value in case of discrete features and with the mean of all the non-missing values in case of continuous features.

#### Create Training and Testing Sets

We split the dataset and used 67% of the dataset for training and 33% for testing. Our testing split contained 21816 examples and training split contained 10745 examples.

## Task 2

### Calculating Prior Probability

We calculated prior probability of each of two classes using maximum likelihood estimator.

### Calculating Conditional Probability

We calculated conditional probabilities of every feature assuming that features are independent of each other given the income class. We binned the continuous features into 10 bins of equal size for calculating likelihood.

### Predicting Class

We made predictions for the testing split using prior and likelihood that we had calculated earlier. We ignored the denominator since it is common to both income classes and directly compared the numerators to make our prediction.

## Task 3

### Evaluation

We evaluated our predictions by calculating the accuracy, precision, recall and F1-score. We calculated them using the following formulae:

### Smoothing

We applied Laplacian smoothing with values as 1, 10, 50 and 100. We used the following formula for smoothing:

where is the number of discrete values that education can take.

### Results

Here are the results averaged over 10 different test-train splits:

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| α | TP | FP | TN | FN | Accuracy | Precision | Recall | F1-Score |
| N/A | 6821.0 | 587.8 | 1992.0 | 1344.2 | 0.689511 | 0.920660 | 0.835379 | 0.875944 |
| 1.0 | 6821.0 | 591.2 | 1988.6 | 1344.2 | 0.689828 | 0.920238 | 0.835378 | 0.875753 |
| 10.0 | 6853.1 | 621.8 | 1958.0 | 1312.1 | 0.695663 | 0.916818 | 0.839310 | 0.876347 |
| 50.0 | 7036.6 | 772.9 | 1806.9 | 1128.6 | 0.726803 | 0.901034 | 0.861786 | 0.880967 |
| 100.0 | 7243.1 | 948.4 | 1631.4 | 922.1 | 0.762355 | 0.884225 | 0.887078 | 0.885641 |

### Comparison with other classification techniques

1. K-Nearest Neighbors (KNN)

The performance of the NB Classifier is compared against different permutations of the KNN classifier. To do this, we take different values of K from 1 to sqrt(N) where N is the amount of training data and run the KNN classifier for every K value ten times with different train-test splits and then take the average of the 10 iterations for every value of K. Here are the results of the 10 different train-test splits:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| K Value | Accuracy | Precision | Recall | F1-Score |
| 5 | 0.775589 | 0.808502 | 0.922748 | 0.861852 |
| 10 | 0.79449 | 0.797357 | 0.977548 | 0.878301 |
| 15 | 0.79665 | 0.798248 | 0.979541 | 0.879643 |
| 20 | 0.797431 | 0.794503 | 0.988726 | 0.881031 |
| 25 | 0.798008 | 0.794102 | 0.990605 | 0.881528 |
| 30 | 0.795896 | 0.790685 | 0.99415 | 0.880814 |
| 35 | 0.795793 | 0.790352 | 0.994689 | 0.880818 |
| 40 | 0.793923 | 0.788178 | 0.996062 | 0.880003 |
| 45 | 0.793541 | 0.787757 | 0.996294 | 0.879832 |
| 50 | 0.792108 | 0.786222 | 0.997092 | 0.879183 |
| 55 | 0.791922 | 0.785942 | 0.997374 | 0.879118 |
| 60 | 0.790889 | 0.78482 | 0.998 | 0.878658 |
| 65 | 0.79007 | 0.784181 | 0.997938 | 0.878234 |
| 70 | 0.788906 | 0.783052 | 0.998356 | 0.877687 |
| 75 | 0.788302 | 0.782536 | 0.998417 | 0.877386 |
| 80 | 0.787017 | 0.781385 | 0.998675 | 0.876762 |
| 85 | 0.786561 | 0.780991 | 0.998736 | 0.876537 |
| 90 | 0.785249 | 0.779878 | 0.998871 | 0.875887 |
| 95 | 0.784812 | 0.7795 | 0.998933 | 0.875672 |
| 100 | 0.783658 | 0.778486 | 0.999142 | 0.875112 |
| 105 | 0.783192 | 0.778071 | 0.999239 | 0.874887 |
| 110 | 0.782168 | 0.777202 | 0.999362 | 0.874385 |
| 115 | 0.781675 | 0.776798 | 0.999387 | 0.874139 |
| 120 | 0.780512 | 0.775858 | 0.999423 | 0.873557 |
| 125 | 0.779898 | 0.77536 | 0.999448 | 0.873251 |
| 130 | 0.778846 | 0.774507 | 0.999497 | 0.872728 |
| 135 | 0.778306 | 0.77407 | 0.999522 | 0.872459 |
| 140 | 0.777403 | 0.773352 | 0.999534 | 0.872008 |
| 145 | 0.776994 | 0.773023 | 0.999546 | 0.871804 |
| 150 | 0.776361 | 0.77251 | 0.999583 | 0.871491 |
| 155 | 0.77611 | 0.772302 | 0.999607 | 0.871368 |
| 160 | 0.775505 | 0.771816 | 0.999632 | 0.871068 |
| 165 | 0.77517 | 0.771542 | 0.999656 | 0.870903 |
| 170 | 0.774593 | 0.771085 | 0.999669 | 0.870616 |
| 175 | 0.774444 | 0.770947 | 0.999718 | 0.870547 |
| 180 | 0.774128 | 0.770684 | 0.999755 | 0.870393 |

We can observe experimentally that the best performance takes place when K=25 with an accuracy of 0.798008. We observe that the accuracies of KNN and NB are within comparable range. Through this we can conclude that KNN has an edge over NB as it compares the test data with the closest neighbors in the feature space and uses their classification labels to classify the test data instead of using the independent probabilistic approach. We can conclude that the features are nearly independent of each other but not completely independent which KNN is able to capture however NB is unable to capture the same. Nevertheless, the probabilistic conditional independence of the features captured by NB is a good approximation for the classification task.

1. Logistic Regression (LR)

The performance of the NB Classifier is compared against different permutations of the LR classifier. To do this, we take ten different train-test splits and then take the average of the 10 iterations to compare the performance metrics. Here are the results of the 10 different train-test splits:

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| TN | FP | FN | TP | Accuracy | Precision | Recall | F1-Score |
| 7896 | 281 | 1861 | 707 | 0.800651 | 0.809265 | 0.965635 | 0.880562 |
| 7941 | 259 | 1886 | 659 | 0.800372 | 0.80808 | 0.968415 | 0.881012 |
| 7843 | 276 | 1915 | 711 | 0.796091 | 0.803751 | 0.966006 | 0.87744 |
| 7836 | 245 | 1988 | 676 | 0.792182 | 0.797638 | 0.969682 | 0.875286 |
| 7870 | 281 | 1921 | 673 | 0.795067 | 0.803799 | 0.965526 | 0.877271 |
| 7843 | 293 | 1918 | 691 | 0.79423 | 0.803504 | 0.963987 | 0.87646 |
| 7916 | 267 | 1858 | 704 | 0.802234 | 0.809904 | 0.967371 | 0.881662 |
| 7804 | 272 | 1956 | 713 | 0.792648 | 0.79959 | 0.96632 | 0.875084 |
| 7929 | 267 | 1867 | 682 | 0.801396 | 0.809412 | 0.967423 | 0.881392 |
| 7926 | 274 | 1888 | 657 | 0.79879 | 0.807622 | 0.966585 | 0.879982 |

Average Performance Metrics:

* Accuracy: 0.7977261 - Precision: 0.8052625
* Recall: 0.966329 - F1-Score: 0.8784649

We can observe experimentally that LR gives an accuracy of 0.7977261. We observe that the accuracies of LR and NB are within comparable range. Through this we can conclude that LR has an edge over NB as it fits the training dataset to the logistic function and uses the function to classify the test data instead of using the independent probabilistic approach. We can conclude that the features are nearly independent of each other but not completely independent which the LR function is able to capture however NB is unable to capture the same. Nevertheless, the probabilistic conditional independence of the features captured by NB is a good approximation for the classification task. LR is also able to capture the categorical and continuous features of the training dataset which helps to easily classify the test dataset.

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

We used Tensorflow to implement 27 different Artificial Neural Networks by varying the number of neurons in each layer and its activation function. We tested all the ANNs on the same test-train split of the MNIST dataset. Here are the results:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| No. of neurons(HL1) | Activation(HL1) | No. of neurons(HL2) | Activation(HL2) | Accuracy |
| 100 | relu | 50 | relu | 0.9777 |
| 100 | relu | 50 | tanh | 0.9777 |
| 50 | relu | 100 | tanh | 0.9772 |
| 100 | relu | 50 | sigmoid | 0.9770 |
| 100 | tanh | 50 | tanh | 0.9758 |
| 100 | tanh | 50 | relu | 0.9755 |
| 100 | sigmoid | 50 | tanh | 0.9753 |
| 100 | tanh | 50 | sigmoid | 0.9749 |
| 100 | sigmoid | 50 | relu | 0.9738 |
| 50 | relu | 100 | relu | 0.9738 |
| 50 | relu | 50 | tanh | 0.9731 |
| 50 | tanh | 100 | relu | 0.9729 |
| 50 | tanh | 50 | relu | 0.9729 |
| 100 | sigmoid | 50 | sigmoid | 0.9727 |
| 50 | tanh | 100 | sigmoid | 0.9722 |
| 50 | relu | 50 | relu | 0.9721 |
| 50 | tanh | 50 | sigmoid | 0.9719 |
| 50 | relu | 100 | sigmoid | 0.9716 |
| 50 | tanh | 50 | tanh | 0.9713 |
| 50 | sigmoid | 100 | relu | 0.9709 |
| 50 | tanh | 100 | tanh | 0.9704 |
| 50 | sigmoid | 50 | tanh | 0.9704 |
| 50 | relu | 50 | sigmoid | 0.9698 |
| 50 | sigmoid | 100 | tanh | 0.9695 |
| 50 | sigmoid | 50 | relu | 0.9688 |
| 50 | sigmoid | 100 | sigmoid | 0.9669 |
| 50 | sigmoid | 50 | sigmoid | 0.9648 |

The model with highest accuracy had 100 neurons in the first hidden layer and 50 neurons in the second hidden layer with ReLU as activation function for both layers. Although all the 27 ANNs have similar accuracy, only the top 8 fall within a 95% Normal confidence interval of the best model, meaning that the remaining models are worse by a statistically significant margin.

Here are the confusion matrices for all the classifiers:

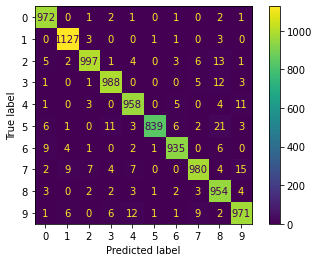


Figure : 50\_relu\_50\_relu

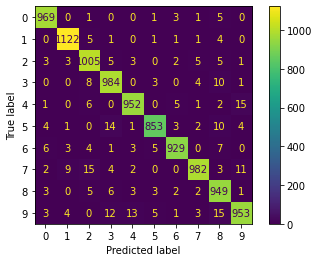


Figure : 50\_relu\_50\_sigmoid

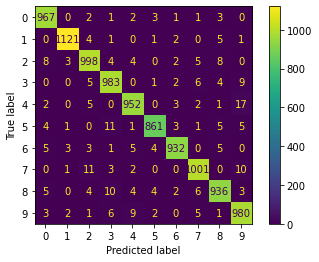


Figure : 50\_relu\_50\_tanh

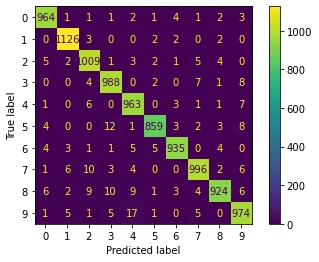


Figure : 50\_relu\_100\_relu

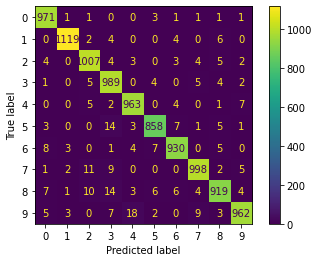


Figure : 50\_relu\_100\_sigmoid

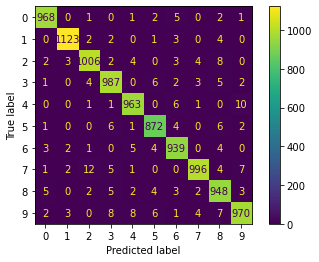


Figure : 50\_relu\_100\_tanh

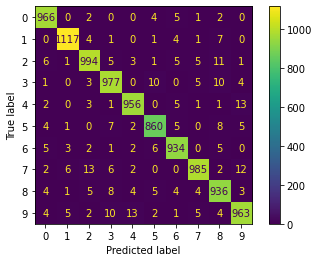


Figure : 50\_sigmoid\_50\_relu

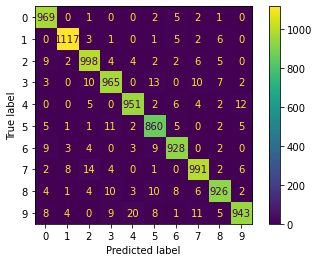


Figure : 50\_sigmoid\_50\_sigmoid

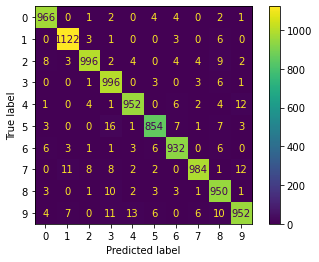


Figure : 50\_sigmoid\_50\_tanh

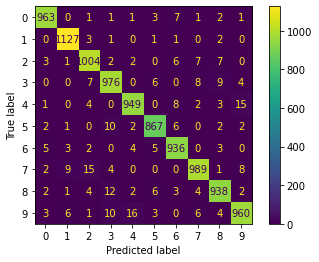


Figure : 50\_sigmoid\_100\_relu

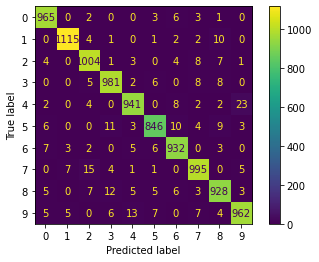


Figure : 50\_sigmoid\_100\_sigmoid

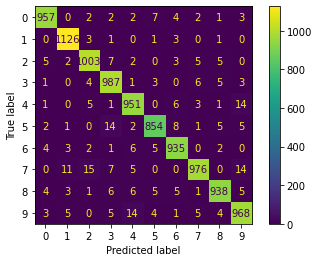


Figure : 50\_sigmoid\_100\_tanh

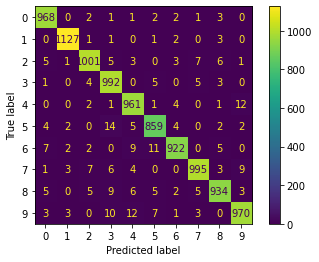


Figure : 50\_tanh\_50\_relu

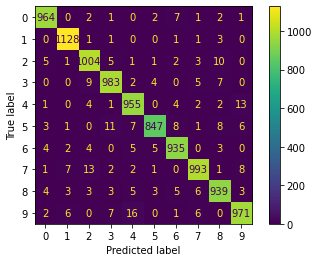


Figure : 50\_tanh\_50\_sigmoid

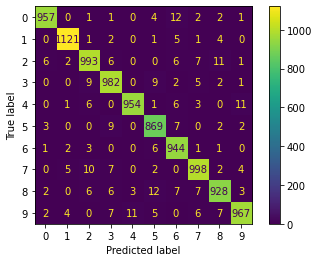


Figure : 50\_tanh\_50\_tanh

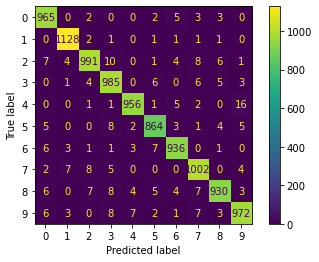


Figure : 50\_tanh\_100\_relu

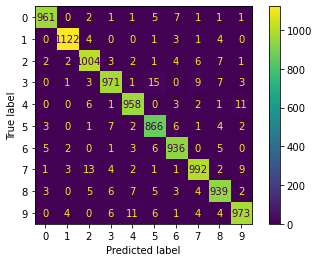


Figure : 50\_tanh\_100\_sigmoid

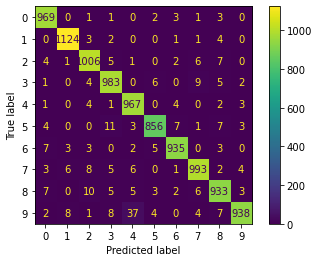


Figure : 50\_tanh\_100\_tanh

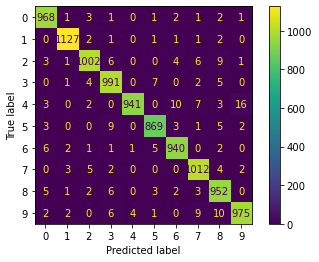


Figure : 100\_relu\_50\_relu

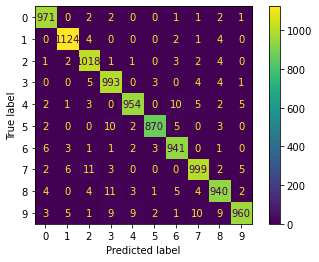


Figure : 100\_relu\_50\_sigmoid

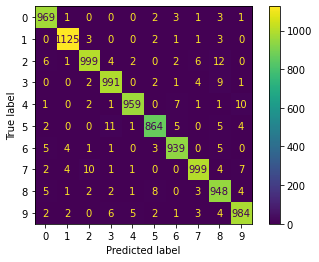


Figure : 100\_relu\_50\_tanh

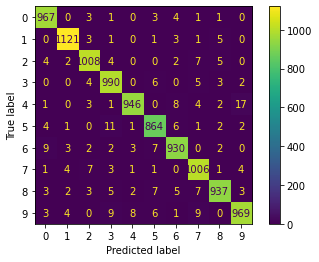


Figure : 100\_sigmoid\_50\_relu

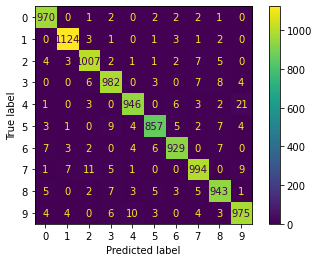


Figure : 100\_sigmoid\_50\_sigmoid

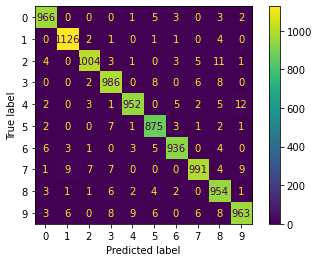


Figure : 100\_sigmoid\_50\_tanh

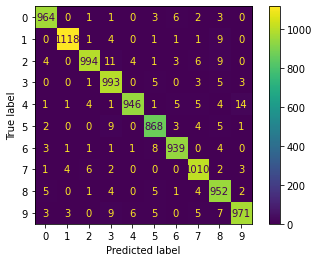


Figure : 100\_tanh\_50\_relu

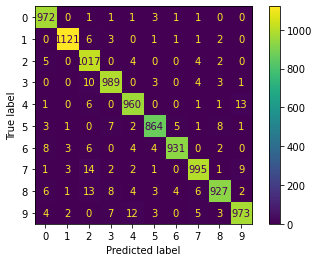


Figure : 100\_tanh\_50\_sigmoid

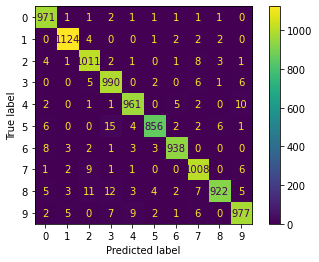


Figure : 100\_tanh\_50\_tanh