BITS F464: MACHINE LEARNING

SECOND SEMESTER 2022-23

ASSIGNMENT-2

DATE: 30/04/23

SUBMITTED BY

NAME - ID

Mufaddal Jiruwala	2020A7PS1720H
Sai Hemanth Ananthoju	2020A7PS0116H
Darshan Chandak	2020A7PS2085H

Contents:

PART - A

Naive Bayes Classifier to Predict Income:

We have implemented a Naive Bayes Classifier to predict if the income of a household is greater than $50K using numpy and pandas libraries.

Naive Bayes Classification is based on Bayes theorem, which describes the relationship between the probability of an event occurring and the probability of the event given some evidence.

Naive Bayes assumes the probability of an instance belonging to a certain class is determined by the probabilities of its features given the class. The algorithm then calculates the conditional probabilities of each feature given each class, and uses these probabilities to predict the most likely classes for a new instance.

Data Pre-Processing:

Our Dataset consists of 32,000 instances, with 14 features and one target variable, income. We first drop all the duplicate values in our dataframe.

We also observed that there were several missing values in our DataFrame [approximately 4300]. The missing values have been replaced with NaN's and then subsequently replaced.

Methods of Imputation:

1) **Mode Imputation**: The missing values in the column 'native_country' have been replaced with the mode of the column.
2) **Proportional Imputation**: Any other missing values have been replaced by proportional imputation, i.e., the missing values have been replaced with values that are proportional to the non-missing values in the same variable.

Test-Train Splits:

We have implemented 10 test-train splits, each with different training and testing data.

We have split the data into a 80:20 ratio, 80% for training and 20% for testing.

Naive Bayes Classifier Implementation:

Calculating Prior Probability:

$$Prior\ Probability\ =\ \frac{(Number\ of\ desired\ outcomes)}{Total\ number\ of\ outcomes}$$

We have calculated the prior probabilities for each of the 10 splits as follows:

```
[{' <=50K': 0.7582696223443083, ' >50K': 0.24173037765569172},
 {' <=50K': 0.7583848784048561, ' >50K': 0.24161512159514387},
 {' <=50K': 0.7576549233547197, ' >50K': 0.24234507664528027},
 {' <=50K': 0.7588074839601983, ' >50K': 0.24119251603980177},
 {' <=50K': 0.7581543662837604, ' >50K': 0.24184563371623957},
 {' <=50K': 0.7604210688078682, ' >50K': 0.23957893119213186},
 {' <=50K': 0.758231203657459, ' >50K': 0.241768796342541},
 {' <=50K': 0.7581543662837604, ' >50K': 0.24184563371623957},
 {' <=50K': 0.7595758576971839, ' >50K': 0.2404241430281609},
 {' <=50K': 0.7589227400207461, ' >50K': 0.2410772597925392}]
```

Calculating Likelihoods:

$$Likelihood\ =\ \frac{\left(\#occurences\ of\ F_n\ |\ Income\ >\ 50K|\right)}{\#\ occurences\ where\ Income\ >50K}$$

For each value in the training dataset, we have calculated:

1) The likelihood of each instance of the dataset with respect to the target variable
2) The likelihood of each class:

```
{'age': {' <=50K': {'mean': 36.826620053706236,
   'variance': 197.64363061531125},
  ' >50K': {'mean': 44.21169739351558, 'variance': 109.47894119724006}}
```

Predicting the class of a given Instance:

For features having continuous data, we have fit the data into a gaussian curve such that values that are present in the testing dataset, but are absent in the training dataset can be assigned probabilities.
However, for categorical data, if there is such a value that is present in the testing data that is absent in the training data, its probability is assumed to be 1.
This is changed later on while implementing smoothing techniques.

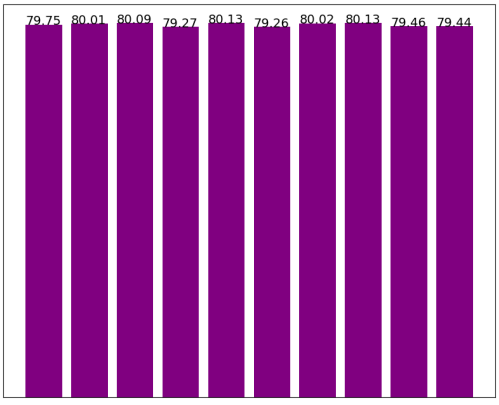
$$P\left(\frac{(Income\ >50K)}{F_1, F_2, F_3, \ldots F_{14}}\right) = \frac{\left(\prod_{n=1}^{14} P\left(\frac{F_n}{Income\ >50K}\right).P(Income\ >50K)\right)}{\left(\prod_{n=1}^{14} P\left(\frac{F_n}{Income\ >50K}\right).P(Income\ >50K)\right)+\left(\prod_{n=1}^{14} P\left(\frac{F_n}{Income\ \leq 50K}\right).P(Income\ \leq 50K)\right)}$$

This is the formula used for predicting the class (predicting if income >50K). Here, F1 - F14 are the values of the 14 features present in the dataset.

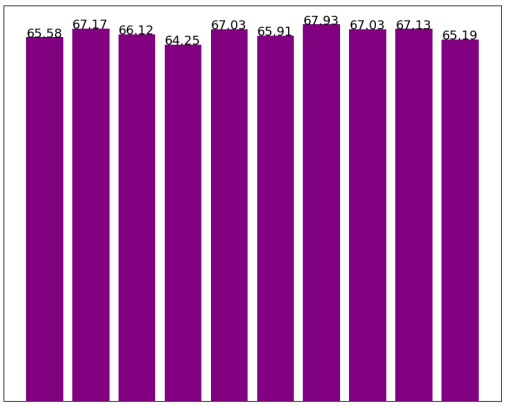
Calculating Performance Metrics:

We have calculated the accuracy for each of the 10 test-train splits as follows:

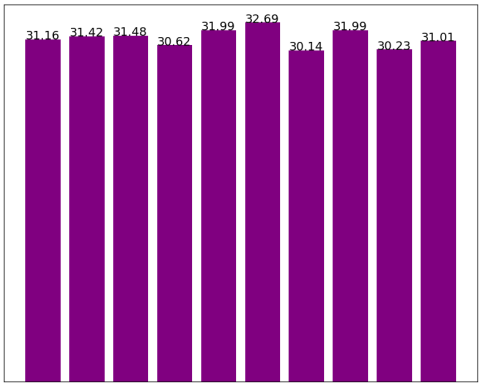
```
Accuracy of Split 1: 79.75
Accuracy of Split 2: 80.01
Accuracy of Split 3: 80.09
Accuracy of Split 4: 79.27
Accuracy of Split 5: 80.13
Accuracy of Split 6: 79.26
Accuracy of Split 7: 80.02
Accuracy of Split 8: 80.13
Accuracy of Split 9: 79.46
Accuracy of Split 10: 79.44
```



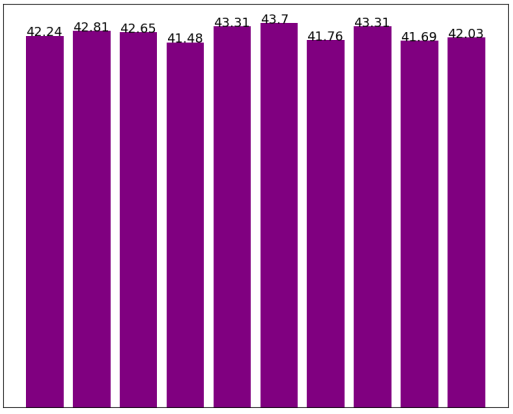
Accuracies of each of the 10 splits



Precisions of each of the 10 splits



Recall values of each of the 10 splits



F1 scores of each of the 10 splits

We observe that the maximum accuracy we get is 80.13% for splits 5 and 8, but maximum precision, recall and F1-score for split 6.

Smoothing Technique: Laplace Smoothing:

We have used Laplace smoothing with 4 different values of the parameter, alpha.

$$p_{i,\ \alpha\text{-smoothed}} = \frac{x_i + \alpha}{N + \alpha d};$$

Formula for Laplace Smoothing

Here, alpha = smoothing parameter and d = number of classes.

The values of alpha we have used is [1, 10, 100, 1000].

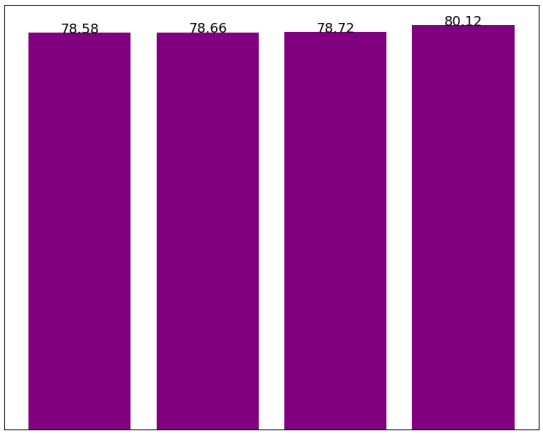
We have used Laplace smoothing for all the categorical data, as the continuous data has already been set on a Gaussian distribution.
We then calculated the accuracy for each split across the parameters as follows:

```
Accuracy for Split 1 and smoothing parameter 1 is: 78.33
Accuracy for Split 1 and smoothing parameter 10 is: 78.33
Accuracy for Split 1 and smoothing parameter 100 is: 78.56
Accuracy for Split 1 and smoothing parameter 1000 is: 78.76

Accuracy for Split 2 and smoothing parameter 1 is: 78.21
Accuracy for Split 2 and smoothing parameter 10 is: 78.24
Accuracy for Split 2 and smoothing parameter 100 is: 78.43
Accuracy for Split 2 and smoothing parameter 1000 is: 79.59

Accuracy for Split 3 and smoothing parameter 1 is: 78.43
Accuracy for Split 3 and smoothing parameter 10 is: 78.43
Accuracy for Split 3 and smoothing parameter 100 is: 78.63
Accuracy for Split 3 and smoothing parameter 1000 is: 79.5

Accuracy for Split 4 and smoothing parameter 1 is: 78.01
Accuracy for Split 4 and smoothing parameter 10 is: 78.09
Accuracy for Split 4 and smoothing parameter 100 is: 78.27
Accuracy for Split 4 and smoothing parameter 1000 is: 79.35

Accuracy for Split 5 and smoothing parameter 1 is: 78.58
Accuracy for Split 5 and smoothing parameter 10 is: 78.66
Accuracy for Split 5 and smoothing parameter 100 is: 78.72
Accuracy for Split 5 and smoothing parameter 1000 is: 80.12

Accuracy for Split 6 and smoothing parameter 1 is: 77.95
Accuracy for Split 6 and smoothing parameter 10 is: 77.95
Accuracy for Split 6 and smoothing parameter 100 is: 78.04
Accuracy for Split 6 and smoothing parameter 1000 is: 78.37
```

```
Accuracy for Split 7 and smoothing parameter 1 is: 78.21
Accuracy for Split 7 and smoothing parameter 10 is: 78.33
Accuracy for Split 7 and smoothing parameter 100 is: 78.5
Accuracy for Split 7 and smoothing parameter 1000 is: 79.75

Accuracy for Split 8 and smoothing parameter 1 is: 78.58
Accuracy for Split 8 and smoothing parameter 10 is: 78.66
Accuracy for Split 8 and smoothing parameter 100 is: 78.72
Accuracy for Split 8 and smoothing parameter 1000 is: 80.12

Accuracy for Split 9 and smoothing parameter 1 is: 77.64
Accuracy for Split 9 and smoothing parameter 10 is: 77.64
Accuracy for Split 9 and smoothing parameter 100 is: 77.93
Accuracy for Split 9 and smoothing parameter 1000 is: 78.58

Accuracy for Split 10 and smoothing parameter 1 is: 78.13
Accuracy for Split 10 and smoothing parameter 10 is: 78.13
Accuracy for Split 10 and smoothing parameter 100 is: 78.23
Accuracy for Split 10 and smoothing parameter 1000 is: 78.43
```

Comparison between different Smoothing Parameters:

From the above results, we take maximum accuracy for each of the parameters across all splits and we find it to be as follows:



We find that the accuracy is highest for the value of the smoothing parameter, alpha = 1000.

Comparing Naive Bayes with KNN and Logistic Regression:

We have performed Logistic Regression and KNN on the same dataset, using scikit-learn library.

We have found the metrics to be as follows:

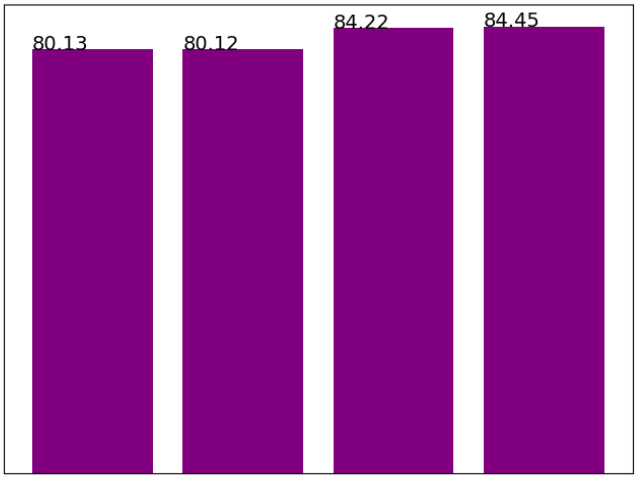
Metrics for Logistic Regression:

Accuracy: 84.22
Precision: 73.21
Recall: 57.43
F1 Score: 64.76

Metrics for KNN:

Accuracy: 84.45
Precision: 69.52
Recall: 67.62
F1 Score: 68.32

Comparing all the models, we find the following:



X-axis: Accuracies of [Naive Bayes, Naive Bayes with Laplace Smoothing, Logistic Regression, KNN] respectively.

We observe that Logistic Regression and KNN have greater accuracies than Naive Bayes in this case. This may be because we have assumed the features to be independent, but that may not be true in some cases.

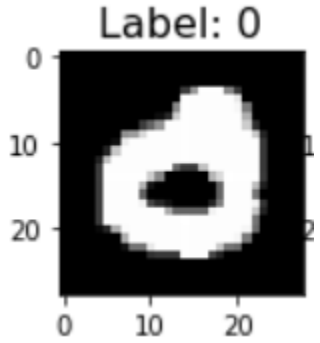
PART - B

Building a Neural Network for Image Classification:

We have built a basic Neural Network that can classify images from the MNIST dataset. The dataset contains 70,000 28 x 28 images of handwritten digits. We have implemented our Neural Network using Keras.

Pre-Processing:

SInce the problem statement required us to split the training and testing datasets into 67% and 33% respectively, we have merged the default datasets mnist_train and mnist_test into one dataset and then used train_test_split() of scikit-learn to split the dataset into 67% training and 33% testing data. [Since there are 70,000 images in total, training data consists of 46,900 images and testing data consists of 23,100 images]



A sample image from the dataset.

Dividing into Classes:

Since there are 10 possible digits (from 0 - 9), we have made 10 classes for classification into each of the digits.

We have created a vector for each of the digits and filled it with zeros except for the index of the digit.

For example, 7 is represented as: `[0. 0. 0. 0. 0. 0. 0. 1. 0. 0.]`

Data Normalisation:

Since there are 256 pixel values, ranging from 0 - 255, we have normalized our data by dividing each of the dataset values with 255. This ensures that all the pixel values are between 0 and 1.

Creating a Neural Network using Keras:

We have created 15 models using Keras, each differing in each of the parameters:

- There are 4 models each with activation functions set to [‘ReLU’, ‘sigmoid’, ‘tanh’] functions respectively. Each of these 4 models differ with respect to either number of hidden layers (2 or 3) or the number of neurons in each hidden layer (100 or 150).
- The remaining 3 models were done by fixing two hidden layers and randomly picking the number of nodes for each, such that the total number of neurons for both the hidden layers is 150. The activation function for these models is also randomly selected.

IMPORTANT

1) We have used Adam [Adaptive Moment] optimizer for all of our models. Adaptive Moment is a stochastic Gradient method that is based on the adaptive estimation of first-order and second-order moments.

2) Across all models for our output layer, we have used the ‘softmax’ function as our activation function because it is ideal for multi-class classification. Softmax ensures that the output of the layers sums up to 1 (normalizes them), hence making it ideal.

3) The number of neurons in the input layer has been set to 784 (28 x 28), that is the result of flattening the 2 - Dimensional arrays (as given in our dataset) into a single long continuous vector.

```
Activation:  relu
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense (Dense)               (None, 784)               615440

 dense_1 (Dense)             (None, 100)               78500

 dense_2 (Dense)             (None, 100)               10100

 dense_3 (Dense)             (None, 10)                1010

=================================================================
Total params: 705,050
Trainable params: 705,050
Non-trainable params: 0
_________________________________________________________________
```

```
Activation:  sigmoid
Model: "sequential_3"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense_13 (Dense)            (None, 784)               615440

 dense_14 (Dense)            (None, 150)               117750

 dense_15 (Dense)            (None, 150)               22650

 dense_16 (Dense)            (None, 150)               22650

 dense_17 (Dense)            (None, 10)                1510

=================================================================
Total params: 780,000
Trainable params: 780,000
Non-trainable params: 0
```

The above images are examples of 2 models that have been generated. The differences in the number of hidden layers, number of neurons per hidden layer and the activation functions determine the number of parameters for both.

Training the Models:

We have trained our models with our training data and have trained them through 10 epochs. On average, you can see the accuracy increasing per epoch in each model.

```
Fit Model: 1
Epoch 1/10
1466/1466 [==============================] - 8s 3ms/step - loss: 0.2187 - accuracy: 0.9334
Epoch 2/10
1466/1466 [==============================] - 8s 5ms/step - loss: 0.0926 - accuracy: 0.9711
Epoch 3/10
1466/1466 [==============================] - 6s 4ms/step - loss: 0.0637 - accuracy: 0.9799
Epoch 4/10
1466/1466 [==============================] - 5s 4ms/step - loss: 0.0492 - accuracy: 0.9842
Epoch 5/10
1466/1466 [==============================] - 6s 4ms/step - loss: 0.0380 - accuracy: 0.9880
Epoch 6/10
1466/1466 [==============================] - 4s 3ms/step - loss: 0.0317 - accuracy: 0.9901
Epoch 7/10
1466/1466 [==============================] - 5s 3ms/step - loss: 0.0278 - accuracy: 0.9911
Epoch 8/10
1466/1466 [==============================] - 5s 3ms/step - loss: 0.0238 - accuracy: 0.9924
Epoch 9/10
1466/1466 [==============================] - 5s 4ms/step - loss: 0.0218 - accuracy: 0.9937
Epoch 10/10
1466/1466 [==============================] - 6s 4ms/step - loss: 0.0182 - accuracy: 0.9940

Fit Model: 2
```

Training result for model 1: From the above image, you can see the training results and training accuracy for model 1. Please note that in our code, the batch-size is not specified. Keras uses a default batch size of 32, and hence you see the number 1466 (46900/32 = 1466).

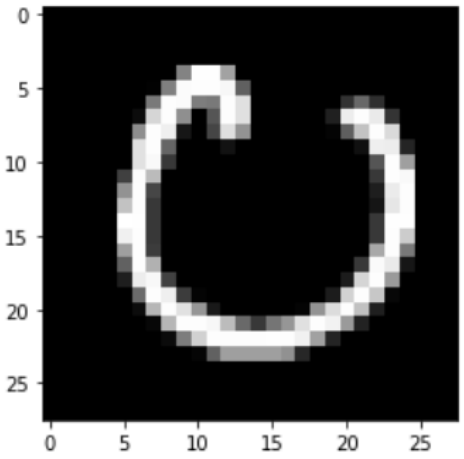
Testing the Models:

All the 15 models have been tested with the testing data, and the results are as follows:

```
722/722 [==============================] - 2s 3ms/step - loss: 0.1135 - accuracy: 0.9760
Test Loss for Model 1: 0.11349620670080185, Test Accuracy for Model 1: 0.9759740233421326

722/722 [==============================] - 2s 2ms/step - loss: 0.1212 - accuracy: 0.9768
Test Loss for Model 2: 0.12122821062803268, Test Accuracy for Model 2: 0.9767532348632812

722/722 [==============================] - 2s 2ms/step - loss: 0.1200 - accuracy: 0.9716
Test Loss for Model 3: 0.12003445625305176, Test Accuracy for Model 3: 0.9716449975967407

722/722 [==============================] - 2s 3ms/step - loss: 0.0911 - accuracy: 0.9776
Test Loss for Model 4: 0.09113399684429169, Test Accuracy for Model 4: 0.9775757789611816

722/722 [==============================] - 2s 2ms/step - loss: 0.1066 - accuracy: 0.9723
Test Loss for Model 5: 0.10661759227514267, Test Accuracy for Model 5: 0.9722510576248169

722/722 [==============================] - 2s 2ms/step - loss: 0.1111 - accuracy: 0.9682
Test Loss for Model 6: 0.11110080778598785, Test Accuracy for Model 6: 0.9682251214981079

722/722 [==============================] - 2s 3ms/step - loss: 0.1167 - accuracy: 0.9771
Test Loss for Model 7: 0.11669335514307022, Test Accuracy for Model 7: 0.977142870426178

722/722 [==============================] - 2s 2ms/step - loss: 0.0951 - accuracy: 0.9792
Test Loss for Model 8: 0.09514343738555908, Test Accuracy for Model 8: 0.9791774749755859

722/722 [==============================] - 2s 2ms/step - loss: 0.0870 - accuracy: 0.9784
Test Loss for Model 9: 0.08701613545417786, Test Accuracy for Model 9: 0.9783982634544373

722/722 [==============================] - 2s 2ms/step - loss: 0.1120 - accuracy: 0.9734
Test Loss for Model 10: 0.11201383173465729, Test Accuracy for Model 10: 0.9733766317367554

722/722 [==============================] - 2s 2ms/step - loss: 0.1123 - accuracy: 0.9684
Test Loss for Model 11: 0.11232063919305801, Test Accuracy for Model 11: 0.9683982729911804

722/722 [==============================] - 2s 3ms/step - loss: 0.1151 - accuracy: 0.9679
Test Loss for Model 12: 0.11505765467882156, Test Accuracy for Model 12: 0.9678787589073181

722/722 [==============================] - 2s 3ms/step - loss: 0.1033 - accuracy: 0.9721
Test Loss for Model 13: 0.10330615937709808, Test Accuracy for Model 13: 0.9720779061317444

722/722 [==============================] - 2s 3ms/step - loss: 0.1152 - accuracy: 0.9666
Test Loss for Model 14: 0.11519297957420349, Test Accuracy for Model 14: 0.9665800929069519

722/722 [==============================] - 2s 2ms/step - loss: 0.0945 - accuracy: 0.9798
Test Loss for Model 15: 0.09453130513429642, Test Accuracy for Model 15: 0.9798268675804138
```

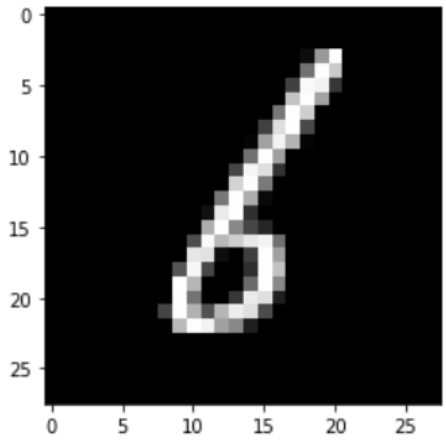
As you can see from the results, Model 15 gives us the most accuracy among the models. We have also plotted examples from the testing dataset to give a general idea of how the prediction works:

Predicted: 0, True: 0



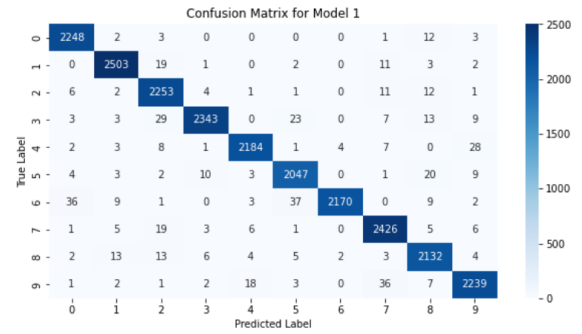
In this case, the model has classified it correctly.

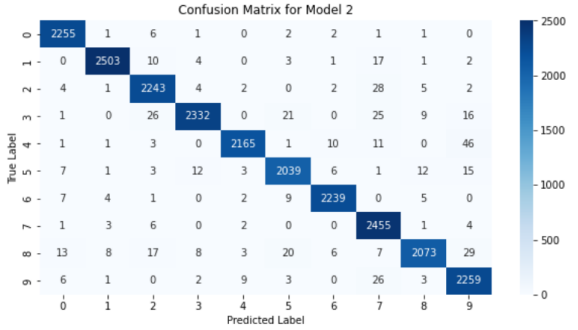
Predicted: 1, True: 6

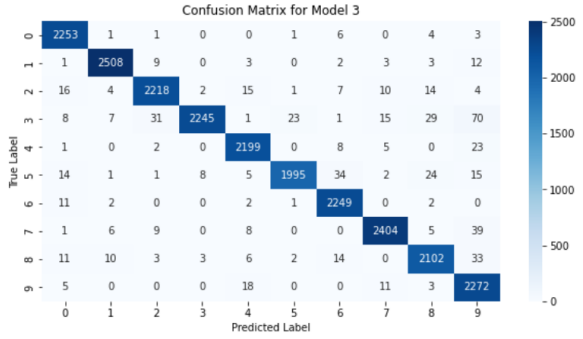


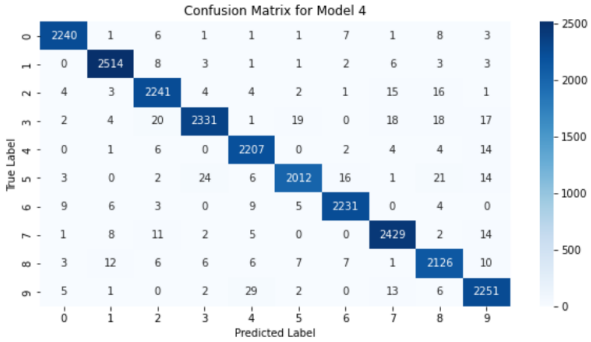
This is a rare case, where the model has failed to classify correctly.

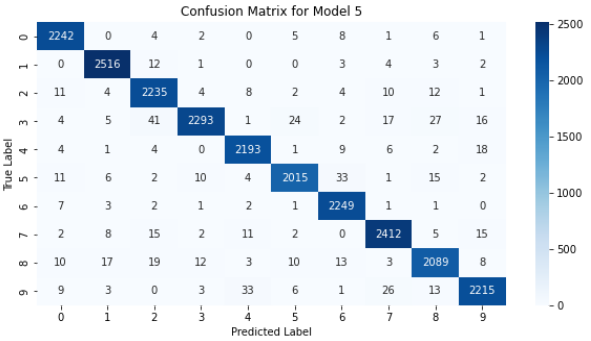
Confusion Matrices for all Classifiers:

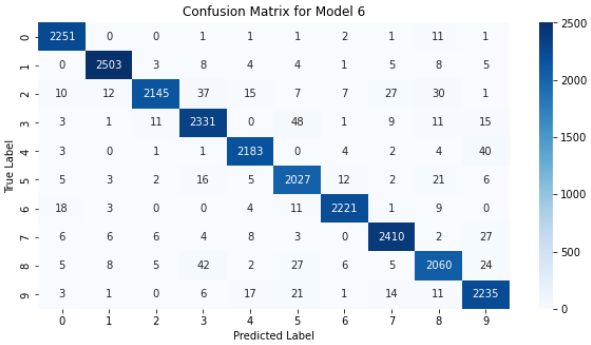


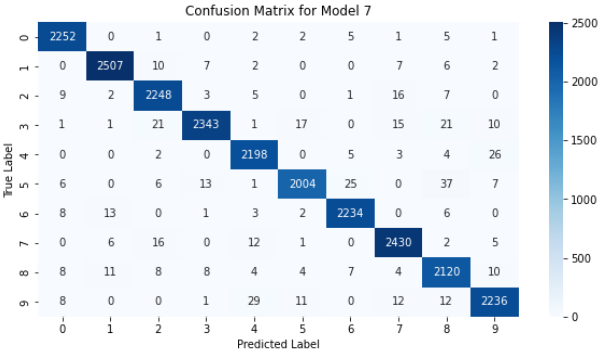


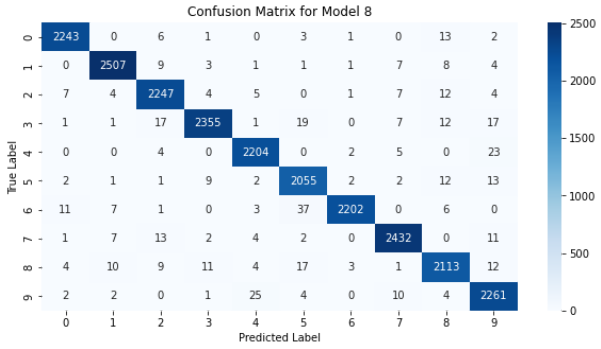


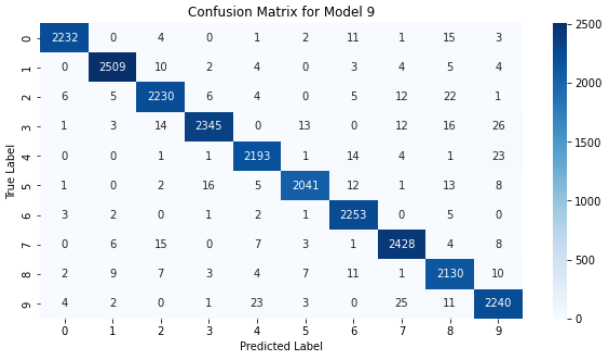


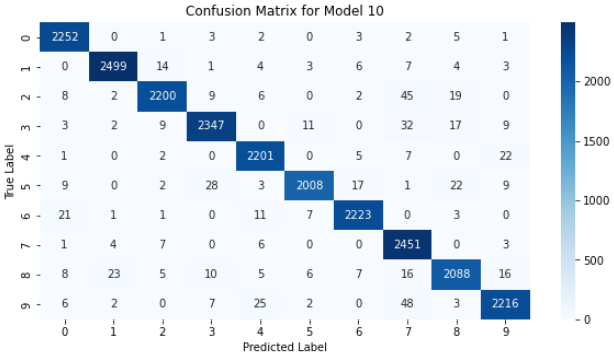


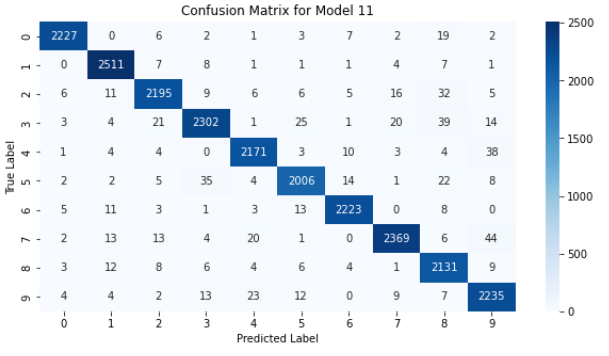


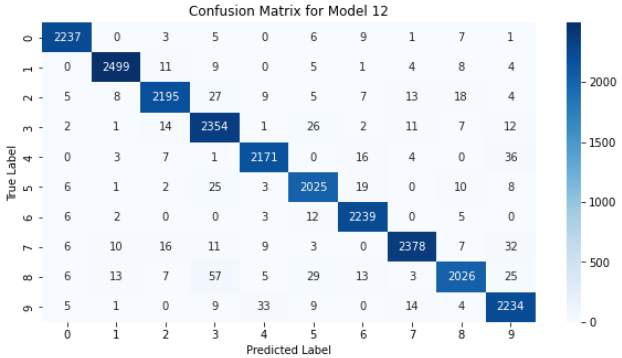


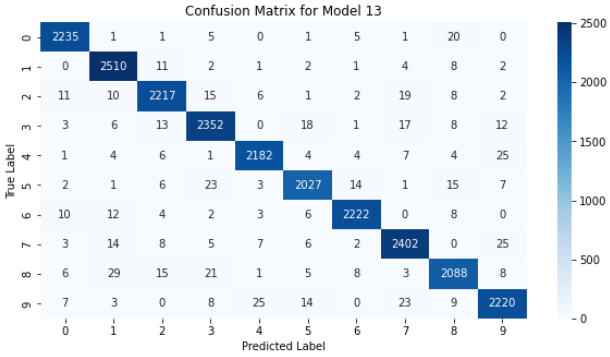


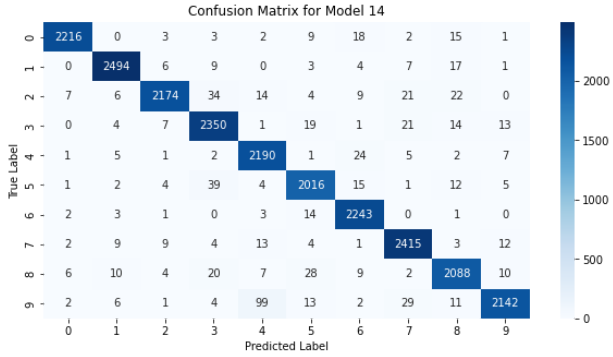

Confusion Matrix for Model 9
True Label
Predicted Label
0 1 2 3 4 5 6 7 8 9
2232 0 4 0 1 2 11 1 15 3
0 2509 10 2 4 0 3 4 5 4
6 5 2230 6 4 0 5 12 22 1
1 3 14 2345 0 13 0 12 16 26
0 0 1 1 2193 1 14 4 1 23
1 0 2 16 5 2041 12 1 13 8
3 2 0 1 2 1 2253 0 5 0
0 6 15 0 7 3 1 2428 4 8
2 9 7 3 4 7 11 1 2130 10
4 2 0 1 23 3 0 25 11 2240
2500 2000 1500 1000 500 0

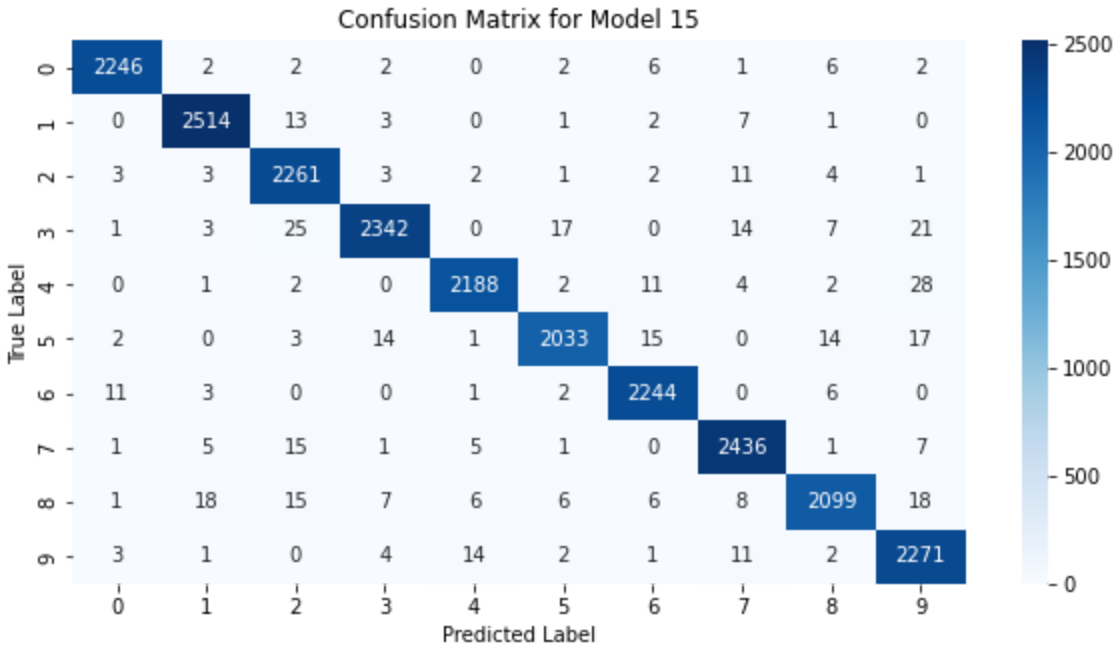

Confusion Matrix for Model 10
True Label
Predicted Label
0 1 2 3 4 5 6 7 8 9
2252 0 1 3 2 0 3 2 5 1
0 2499 14 1 4 3 6 7 4 3
8 2 2200 9 6 0 2 45 19 0
3 2 9 2347 0 11 0 32 17 9
1 0 2 0 2201 0 5 7 0 22
9 0 2 28 3 2008 17 1 22 9
21 1 1 0 11 7 2223 0 3 0
1 4 7 0 6 0 0 2451 0 3
8 23 5 10 5 6 7 16 2088 16
6 2 0 7 25 2 0 48 3 2216
2000 1500 1000 500 0


Confusion Matrix for Model 11
True Label
Predicted Label
0 1 2 3 4 5 6 7 8 9
2227 0 6 2 1 3 7 2 19 2
0 2511 7 8 1 1 1 4 7 1
6 11 2195 9 6 6 5 16 32 5
3 4 21 2302 1 25 1 20 39 14
1 4 4 0 2171 3 10 3 4 38
2 2 5 35 4 2006 14 1 22 8
5 11 3 1 3 13 2223 0 8 0
2 13 13 4 20 1 0 2369 6 44
3 12 8 6 4 6 4 1 2131 9
4 4 2 13 23 12 0 9 7 2235
2500 2000 1500 1000 500 0


Confusion Matrix for Model 12
True Label
Predicted Label
0 1 2 3 4 5 6 7 8 9
2237 0 3 5 0 6 9 1 7 1
0 2499 11 9 0 5 1 4 8 4
5 8 2195 27 9 5 7 13 18 4
2 1 14 2354 1 26 2 11 7 12
0 3 7 1 2171 0 16 4 0 36
6 1 2 25 3 2025 19 0 10 8
6 2 0 0 3 12 2239 0 5 0
6 10 16 11 9 3 0 2378 7 32
6 13 7 57 5 29 13 3 2026 25
5 1 0 9 33 9 0 14 4 2234
2000 1500 1000 500 0

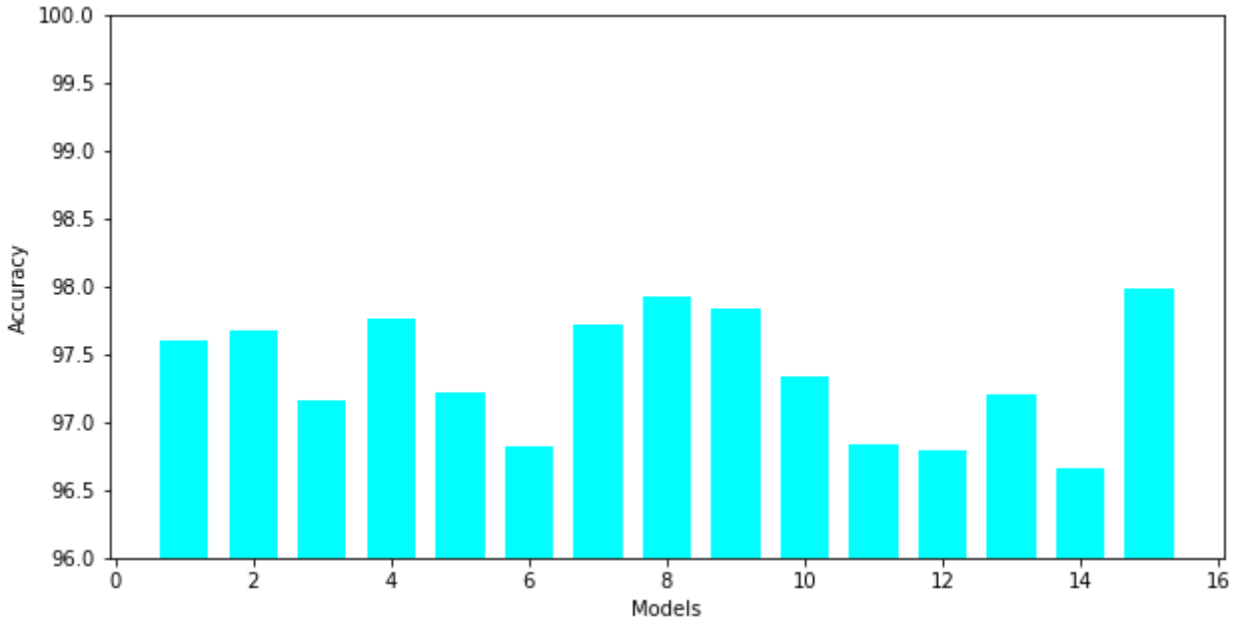

Confusion Matrix for Model 13
True Label
Predicted Label
0 1 2 3 4 5 6 7 8 9
2235 1 1 5 0 1 5 1 20 0
0 2510 11 2 1 2 1 4 8 2
11 10 2217 15 6 1 2 19 8 2
3 6 13 2352 0 18 1 17 8 12
1 4 6 1 2182 4 4 7 4 25
2 1 6 23 3 2027 14 1 15 7
10 12 4 2 3 6 2222 0 8 0
3 14 8 5 7 6 2 2402 0 25
6 29 15 21 1 5 8 3 2088 8
7 3 0 8 25 14 0 23 9 2220
2500 2000 1500 1000 500 0


Confusion Matrix for Model 14
True Label
Predicted Label
0 1 2 3 4 5 6 7 8 9
2216 0 3 3 2 9 18 2 15 1
0 2494 6 9 0 3 4 7 17 1
7 6 2174 34 14 4 9 21 22 0
0 4 7 2350 1 19 1 21 14 13
1 5 1 2 2190 1 24 5 2 7
1 2 4 39 4 2016 15 1 12 5
2 3 1 0 3 14 2243 0 1 0
2 9 9 4 13 4 1 2415 3 12
6 10 4 20 7 28 9 2 2088 10
2 6 1 4 99 13 2 29 11 2142
2000 1500 1000 500 0



Model 15 is the best classifier among all the models.

Comparative Analysis of Models:



From the graph, we can see the accuracy of model 15 is the highest. The characteristics of Model 15 are as follows:

```
Activation:  relu
Model: "sequential_14"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense_62 (Dense)            (None, 784)               615440

 dense_63 (Dense)            (None, 90)                70650

 dense_64 (Dense)            (None, 60)                5460

 dense_65 (Dense)            (None, 10)                610

=================================================================
Total params: 692,160
Trainable params: 692,160
Non-trainable params: 0
```

It has 2 hidden layers with 90 and 60 neurons respectively.

The maximum difference between the accuracies of the classifiers is 1.33%, and it is not statistically significant. Hence, in this case, we do not have a classifier that is not statistically significant from the best classifier.

However, if we were to make a model with, say, a hidden layer having only one neuron. In this case, the prediction will always be the same class, and thus, the accuracy of the model would be very low. Thus, this model would be statistically insignificant.