Report

Navie Bayes Classifier

The goal is to build a model which can predict the income class of an individual (more than $50k or less). The model is trained on a dataset : https://archive.ics.uci.edu/ml/datasets/Adult.

The dataset has over 32,000 instances of individuals with about 14 attributes defining demographics and employment information. All these attributes are independent.

What is Naive Bayes Classification?

Naive Bayes classification is a generative model which aims to calculate probability of a quantity belonging to a class based on the likelihood of it's happening and the prior probability of it's occurence. Below is the Bayes theorem on two independent events A and B :

$$P(A|B) = P(B|A) * P(A)/P(B)$$

Data Preprocessing

We load our data in pandas dataframe and perform a 'whitespace' correction in our data by following code snippet :

```
data = pd.read_csv("adult.csv", names=column_names)
replace_values = {" <=50K":"1", " >=50K":"-1", " >50K":"-1", " <50K":"1"}
data['salary'] = data['salary'].replace(replace_values)
```

We had our data in the form of continuous numerical values which need to be changed to discrete classes to improve accuracy. Below is the code for same :

```
data['fnlwgt'] = pd.cut(data['fnlwgt'], 5,labels=["1", "2", "3", "4", "5"])
data['capital-gain'] = pd.cut(data['capital-gain'], 5,labels=["1", "2", "3",
"4", "5"])
data['capital-loss'] = pd.cut(data['capital-loss'], 5,labels=["1", "2", "3",
"4", "5"])
data['hours-per-week'] = pd.cut(data['hours-per-week'], 5,labels=["1", "2",
"3", "4", "5"])
```

Finally, we need to replace **?** values with **NaN** by the following code :

```
data = data.replace('?', np.nan)
data.isna().sum().sum()
data.dropna(inplace=True)
```

Model Construction

To construct the model, we will make use of class 'NaiveBayes()' which has following important methods :

1. **calc_prior_prob** : This method is responsible for calculating prior probabilities of our training data.

```
def calc_prior_prob(self, y):
      self.priors = y.groupby(y).count().add(50).div(len(y))
      self.class1 = y.groupby(y).count()[0]
      self.class2 = y.groupby(y).count()[1]
      return
```

2. **calc_conditional_prob** : This method is responsible for calculating the conditional probability (likelihood of happening given the prior).

```
def calc_conditional_prob(self, X, y):
      self.cond_prob_of_feature = {}
      train = pd.concat([X,y], axis=1)
      i=0
      for col in train.columns:
        if i%1000==0 and i!=0:
           print(i)
        self.cond_prob_of_feature[col] = train.groupby(['salary',
col]).size()
        self.cond_prob_of_feature[col]/=len(train)
        self.cond_prob_of_feature[col]/=self.priors
      return
```

3. **classify** : This method is responsible for classifying our testing example based on the probabilities we calculated earlier by finding posterior probabilities.

```
def classify(self, x):
    prob_class1 = 1
    prob_class2 = 1
    count=0
    for i in x.index:
      if(x[count] not in self.cond_prob_of_feature[i]['-1']):
          prob_class1 *= 0.001
      else:
          prob_class1 *= self.cond_prob_of_feature[i]['-1'][x[count]]
      if(x[count] not in self.cond_prob_of_feature[i]['1']):
          prob_class2 *= 0.001
      else:
          prob_class2 *= self.cond_prob_of_feature[i]['1'][x[count]]
      count+=1
    if prob_class1*self.priors[0] >= prob_class2*self.priors[1]:
      return -1
    elif prob_class1*self.priors[0] > prob_class2*self.priors[1]:
      return 1
    else:
       return 1
```

Metrics Class

This is a helper class that helps us calculate useful metrics like 1) Accuracy 2) Precision 3) Recall for our models.
It takes in two numpy.array object for initialization (y_pred and y_test) and then we can calculate the metrics.

It has 4 functions:

```
__init__(self, y_pred, y_test)
    self.y_pred=y_pred
    self.y_test=y_test

accuracy(self):

precision(self):

recall(self):
```

that work pretty much how you might expect them to.

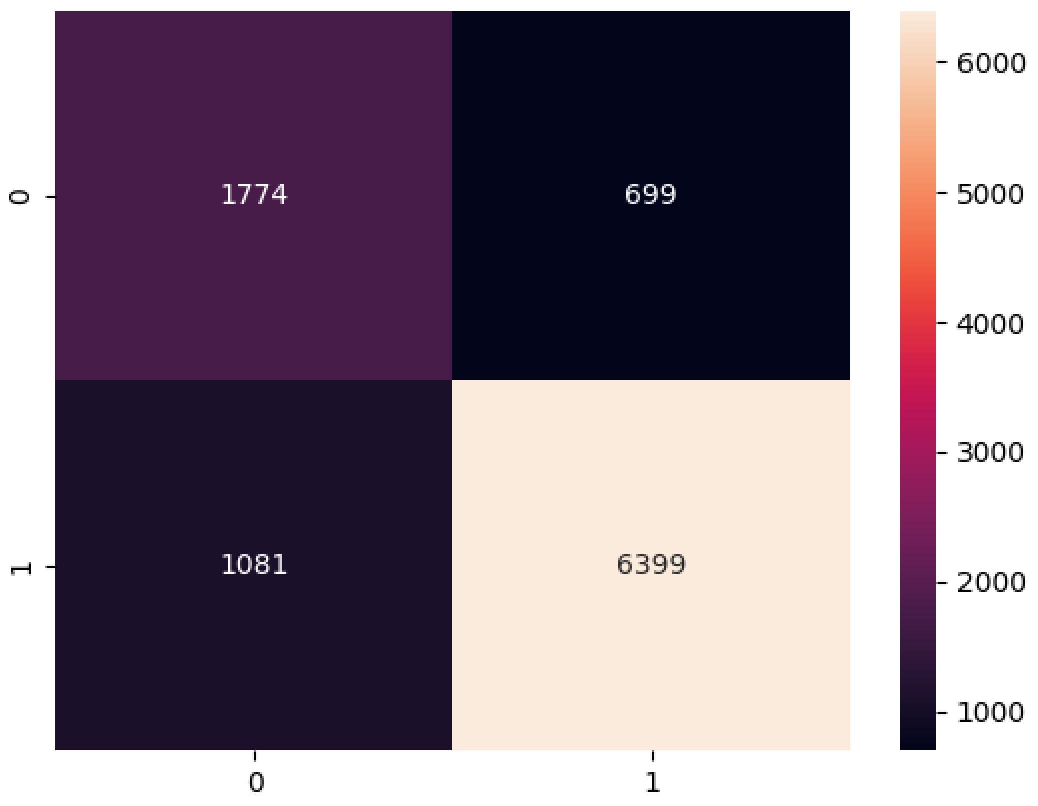
Preprocessing Class

It helps us with our test and train splits. It has one function of our interest:

```
def train_test_split(self,df, train_size=0.67, test_size=0.33,
random_state=0)
    # <!--       Some Code -->
    return train, test
```

It splits our test and train data from the data read from adult.csv in the ratio 0.67:0.33.

Confusion Matrix obtained from naive-bayes model



Model Analysis and Comparision

The above model has the following values of analysis parameters (Accuracy, Precision, Recall, F1-score) :
Accuracy: 82.11%
Precision: 85.54%
Recall: 90.15%
F1: 90.15%

Comparing to KNN and Logistic Regression, We observe:

1. KNN gave lower Accuracy, Recall and F1 score but higher Precision :
   Accuracy: 81.37%
   Precision: 85.54%
   Recall: 90.15%
   F1: 86.74%
2. Logistic gave higher Accuracy, Precision but lower F1 score and Precision :
   Accuracy: 83.93%
   Precision: 91.72%
   Recall: 87.50%
   F1: 87.50%

So, it can be said that our naive bayes model performed poorly compared to Logistic Regression. This may be because we have not done hyperparameter tuning and proper preprocessing of the numerical data. It could also be because we have not used enough laplace smoothing.

Basic Neural Networks

The goal is to build a basic neural network that can classify images of handwritten digits from the MNIST dataset.

Dataset

To implement our neural network, we are utilising the MNIST dataset. This dataset consists of 70,000 images handwriiten digits (0 - 9). We have used Keras framework for building and consequently training this dataset. We included this dataset in our code as follows :

```
from keras.datasets import mnist
(x_train, y_train), (x_test, y_test) = mnist.load_data()
```

Architecture

A basic neural network consists of an input layer, one or more hidden layers and a final ouput layer. For this project, we are required to build models by varying the following parameters:

1. Hidden layers : 2 or 3
2. Total Neurons in hidden layers : 100 or 150
3. Activation function : tanh, sigmoid or ReLu

There will be total 72 models possible, out of which we need to build any 15 distinct models.

Training & Testing

For the purpose of training we will use Adam optimizer. Below is the code snippet for our model construction :

```
model = keras.Sequential()
            for i in range(hidden_layer-1):
                if i==0:
                    model.add(Dense(neuron/hidden_layer, input_shape=
(784,)))
                    model.add(Activation(activation_function))
                else:
                    model.add(Dense(neuron/hidden_layer))
                    model.add(Activation(activation_function))
                model.add(Dense(neuron%hidden_layer + neuron/hidden_layer))
                model.add(Activation(activation_function))
            model.add(Dense(10))
            model.add(Activation('softmax'))
            model.compile(loss="categorical_crossentropy", optimizer="adam",
metrics=["accuracy",keras.metrics.Recall(),keras.metrics.Precision()])
            history = model.fit(x_train, y_train, batch_size=batch_size,
epochs=epochs)
            y_pred = model.predict(x_test)
```

After the creation of our model (with a permutation of parameters), we will test it by using our testing data and also analyse it via confusion matrix. Here is a code snippet for the same :

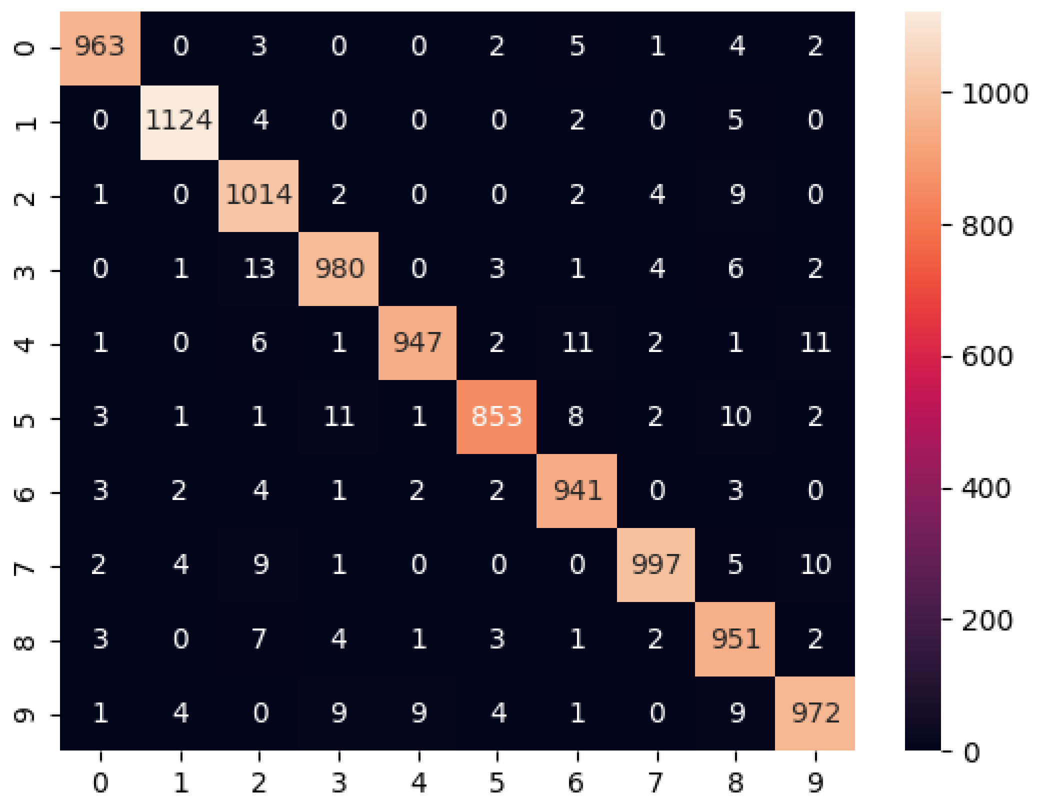
```
 y_pred = np.argmax(y_pred, axis=1)
            confusion = confusion_matrix(y_true=y_test.argmax(axis=1),
y_pred=y_pred)
            metrics = model.evaluate(x_test, y_test)
            metrics = pd.Series(metrics,index=
['Loss','Accuracy','Precision','Recall'])
            confusion = pd.DataFrame(confusion)
            print(confusion)
            confusion.to_csv("./data/confusion/"+str(activation_function)+"-
"+str(hidden_layer)+"-"+str(neuron)+".csv")
            metrics.to_csv("./data/metrics/"+str(activation_function)+"-
"+str(hidden_layer)+"-"+str(neuron)+".csv",header=False)
```

We used **15 epochs** and **128 batch size**.

Example

Here is an example of a confusion matrix for a model which has following parameter values :

1. Hidden layers : 3
2. Total neurons in hidden layers : 150
3. Three activation functions (in order) : ReLu, sigmoid and tanh.



Matrices for other 14 models can be found in github repository at the location : **ml-models-2 / data / plots**

Below is the precision, accurcy and recall for the same model :

Loss	0.11280009895563126
Accuracy	0.9721999764442444
Precision	0.9717000126838684
Recall	0.9739400744438171

Metrics for other 14 models can be found in github repository at the location : **ml-models-2 / data / metrics**

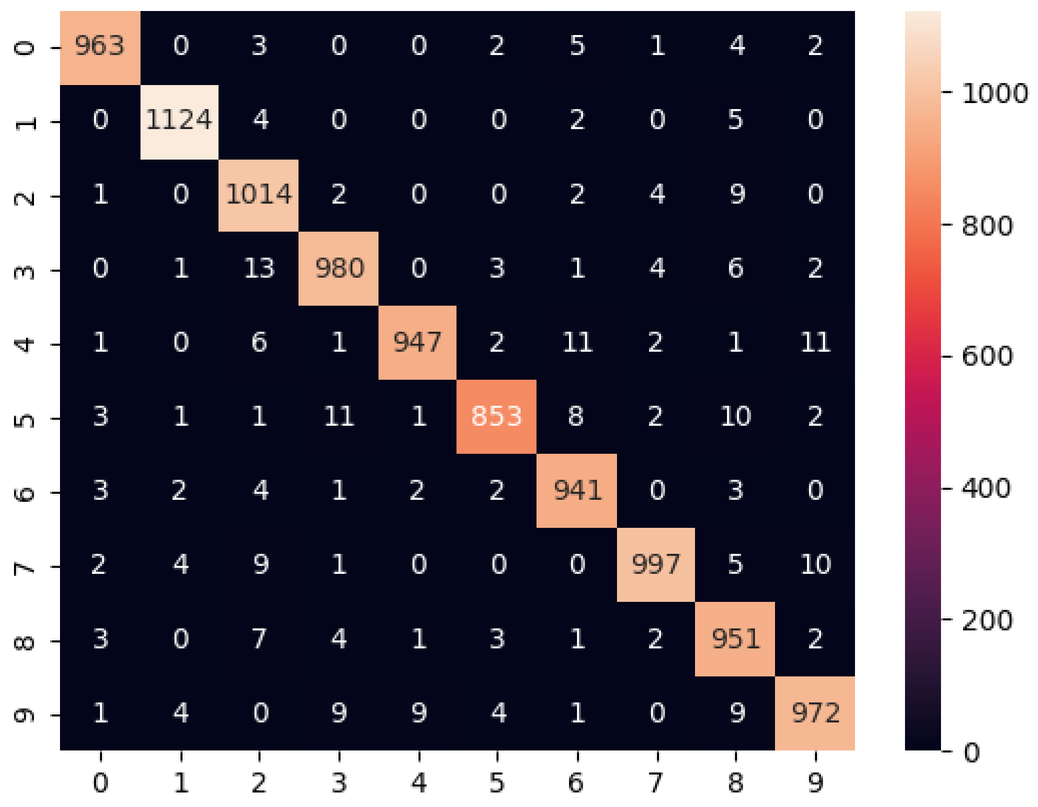
Best Classifier

Based on the metrics (precision, accuracy and recall) of all the models, our best model has following parameter values :

1. Hidden layers : 2
2. Total neurons in hidden layers : 150
3. Activation function : ReLu

Below are the confusion matrix and metrics table for this classifier :

Loss	**0.08407726883888245**
Accuracy	0.9783999919891357
Precision	0.9775999784469604
Recall	0.9791666865348816



	0	1	2	3	4	5	6	7	8	9
0	963	0	3	0	0	2	5	1	4	2
1	0	1124	4	0	0	0	2	0	5	0
2	1	0	1014	2	0	0	2	4	9	0
3	0	1	13	980	0	3	1	4	6	2
4	1	0	6	1	947	2	11	2	1	11
5	3	1	1	11	1	853	8	2	10	2
6	3	2	4	1	2	2	941	0	3	0
7	2	4	9	1	0	0	0	997	5	10
8	3	0	7	4	1	3	1	2	951	2
9	1	4	0	9	9	4	1	0	9	972

Is our classifier statistically significant from the best classifier:

We found that a classifier https://towardsdatascience.com/going-beyond-99-mnist-handwritten-digits-recognition-cfff96337392 achieved 99.4% accuracy on the same MNIST dataset.

This is much superior to our basic neural network which achieved only a mere 97.8% accurcy.

While our model is also very good, it is statistically significant from the best classifier. This is due to a few reasons:

1. Basic Neural Network vs Complex Neural Network
2. Simple Variance vs Batch Normalization, Data Augmentation, Regularization
3. Fixed LR vs Variable LR
4. etcetera