VIETNAM NATIONAL UNIVERSITY, HO CHI MINH CITY
UNIVERSITY OF TECHNOLOGY
FACULTY OF COMPUTER SCIENCE AND ENGINEERING

Machine Learning - CO3117

Assignment

MNIST Classification
using several Machine Learning methods

Instructor: Nguyen Duc Dung
Student: Vo Tran Minh Hieu - 2152560

HO CHI MINH CITY, JUNE 2024

Contents

1 Introduction

The MNIST database of handwritten digits is a subset of a larger set available from NIST. It has a training set of 60,000 examples, and a test set of 10,000 examples of handwritten digits from 0 to 9. The digits have been normalized and centered in a fixed-size image. Each image is a grayscale image of size 28x28 pixels.

The MNIST dataset is widely recognized as being one of the foundational datasets for learning and practicing how to develop, evaluate, and use machine learning models to classify image from scratch.

The following picture represents many random images from the training dataset.

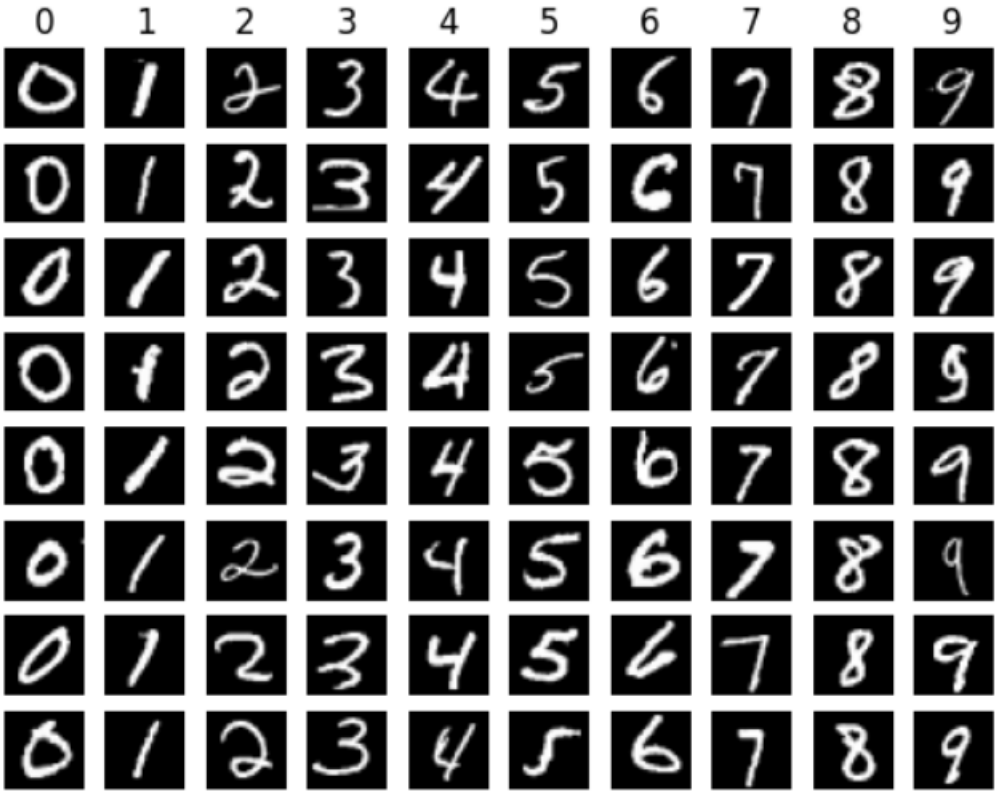

Figure 1: Some random examples.

From this study in MNIST, I learnt to prepare the dataset, design and implement machine learning models and bring theories to real-life problem. Furthermore, I know how to deal with image data.

2 Literature Review

In this assignment, I applied three machine learning models: K-nearest Neighbor, Naive Bayes classifier and Convolutional Neural Network.

2.1 K-Nearest Neighbors

The K-nearest neighbors is one of the simplest machine learning algorithms in the supervised learning techniques. K-NN is a **nonparametric model** which has also been described as **instanced-based learning**. A nonparametric model is one that cannot be characterized by a bounded set of parameters.

To do classification, we calculate distances from examples to the query point and keep the k examples that give the lowest distances. Then, we take the most common output value. To avoid ties on binary classification, k is usually chosen to be an odd number.

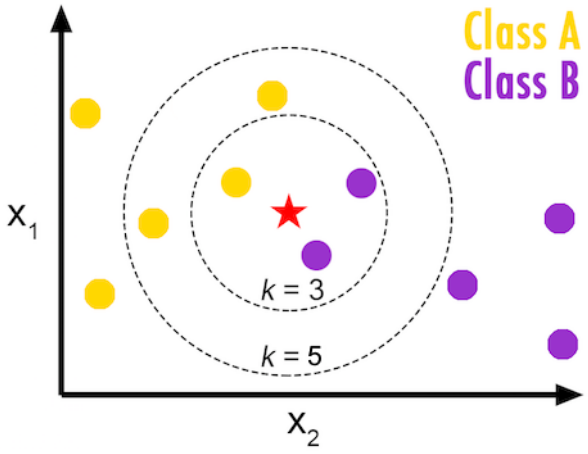

Figure 2: Demonstration of KNN.

There are many distance measurement. Typically, distances are measured with a **Minkowski distance** or L^p norm, defined as

$$L^p(x_j, x_q) = (\sum_i |x_{j,i} - x_{q,i}|^p)^{1/p}.$$

With $p = 2$ this is Euclidean distance and with $p = 1$ it is Manhattan distance. Other distance measurements such as Hamming distance, etc.

Advantages:

- Easy to implement as the complexity of the algorithm is not that high.
- In low-dimensional spaces with plenty of data, nearest neighbors works very well: we are likely to have enough nearby data points to get a good answer.

Disadvantages:

- The complexity is O(N), instance-based methods are designed for large datas sets, so we would like something faster.
- Does not scale – As we have heard about this that the KNN algorithm is also considered a Lazy Algorithm. The main significance of this term is that this takes lots of computing power as well as data storage. This makes this algorithm both time-consuming and resource exhausting.

- The model faces **curse of dimensionality** problem as the number of dimensions is large. In that case, the nearest neighbors in high-dimensional spaces are usually not very near.

2.2 Naive Bayes Classifier

Naive Bayes classifiers is a family of algorithms based on Bayes' Theorem. In that, every pair of features is conditionally independent of each other given the class. The full joint distribution can be written as

$$P(Cause, Effect_1, ..., Effect_n) = P(Cause)\prod_i P(Effect_i|Cause).$$

$$\implies P(Cause|Effect_1, ..., Effect_n) \propto P(Cause)\prod_i P(Effect_i|Cause).$$

$$\implies \hat{y} = argmax_y P(y)\prod_{i=1}^{n} P(x_i|y)$$

Such a probability distribution is called a Naive Bayes model because it is often used in cases where the "effect" variables are *not* strictly independent given the cause variable. In practice, naive Bayes systems often work very well, even when the condition independence assumption is not strictly true.

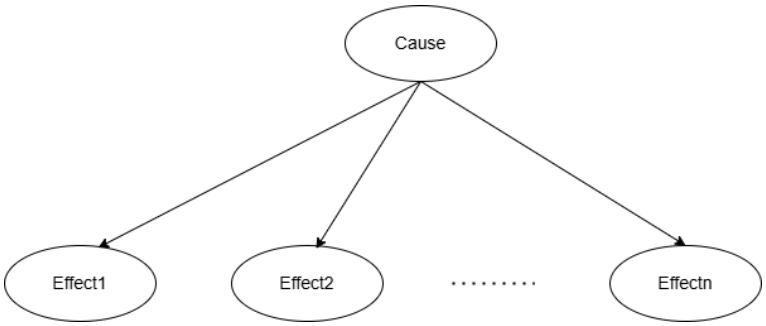

Figure 3: Naive Bayes graph.

Naive Bayes models are generative models, they are widely used for language determination, document retrieval, spam filtering, and other classification tasks. For tasks such as medical diagnosis, where the actual values of the posterior probabilities really matter - for example, in deciding whether to perform an appendectomy - one would usually prefer to use the more sophisticated models.

2.3 Convolutional Neural Networks

In deep learning, a **convolutional neural network** (CNN/ConvNet) is a class of deep neural networks, most commonly applied to analyze visual imagery. The cnn architecture uses a special technique called Convolution instead of relying solely on matrix multiplications like traditional neural networks.

A convolutional neural network contains spatially local connections, at least in the early layers, and has patterns of weights that are replicated across the units in each layer. A pattern

of weights that is replicated across multiple local regions is called a kernel and the process of applying the kernel to the pixels of the image (or to spatially Kernel organized units in a subsequent layer) is called **convolution**.

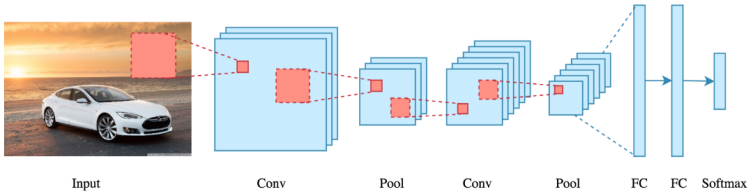

Figure 4: Naive Bayes graph.

Kernels and convolutions are easiest to illustrate in one dimension rather than two or more, so we will assume an input vector $\mathbf{x}$ of size n, corresponding to n pixels in a one-dimensional image, and a vector kernel $\mathbf{k}$ of size l. (For simplicity we will assume that l is an odd number.) All the ideas carry over straightforwardly to higher-dimensional cases.

We write the convolution operation using the symbol, for example: $\mathbf{z} = \mathbf{x} * \mathbf{k}$. The operation is defined as follows:

$$z_i = \sum_{j=1}^{l} k_j x_{j+i-(l+1)/2}.$$

In other words, for each output position i, we take the dot product between the kernel $\mathbf{k}$ and a snippet of $\mathbf{x}$ centered on x_i with width l. **Stride** is the distance between center pixels of the kernels

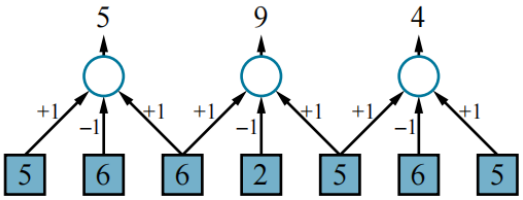

Figure 5: An example of a one-dimensional convolution operation with a kernel of size l =3 and a stride s=2. The peak response is centered on the darker (lower intensity) input pixel. The results would usually be fed through a nonlinear activation function (not shown) before going to the next hidden layer.

$$\begin{pmatrix} +1 & -1 & +1 & 0 & 0 & 0 & 0 \\ 0 & 0 & +1 & -1 & +1 & 0 & 0 \\ 0 & 0 & 0 & 0 & +1 & -1 & +1 \end{pmatrix} \begin{pmatrix} 5 \\ 6 \\ 6 \\ 2 \\ 5 \\ 6 \\ 5 \end{pmatrix} = \begin{pmatrix} 5 \\ 9 \\ 4 \end{pmatrix}.$$

3 Methodology

3.1 Data Description and Collection

The dataset was obtained from datasets of keras. The original source is http://yann.lecun.com/exdb/mnist/. Title of the dataset is "THE MNIST DATABASE of handwritten digits". Contributors are Yann Lecun-from Courant Institute, NYU, Corinna Cortes-Google Labs, New York, and Christopher J.C. Burges, Microsoft Research, Redmond. The dataset consists of 60,000 training examples, and a test set of 10,000 examples.

3.2 Data Preprocessing

Because the digits have been self-normalized and centered in a fixed-size image. Minimal efforts are spending on preprocessing and formatting, users just need to load the image and change the shape to serve for inputting.

For Naive Bayes and KNN, I change the shape of images from 28x28 to 1-dimensional array with 784 elements. The python code for loading images is as follows:

```
from keras.datasets import mnist
(X_train, y_train), (X_test, y_test) = mnist.load_data()
X_train = X_train.reshape(60000, 784)
X_test = X_test.reshape(10000, 784)
```

For CNN, I use one hot encoding to the target class, transform labels from one-dimension array to three-dimension array. Meanwhile, pixels' values in training are normalized images to improve the performance of the model.

```
def load_dataset():
  # load dataset
  (trainX, trainY), (testX, testY) = mnist.load_data()
  # reshape dataset to have a single channel
  trainX = trainX.reshape((trainX.shape[0], 28, 28, 1))
  testX = testX.reshape((testX.shape[0], 28, 28, 1))
  # one hot encode target values
  trainY = to_categorical(trainY)
  testY = to_categorical(testY)
  return trainX, trainY, testX, testY

def prep_pixels(train, test):
  # Convert from integer to floats
  train_norm = train.astype('float32')
  test_norm = test.astype('float32')
  # Normalize to range 0-1
  train_norm = train_norm / 255.0
  test_norm = test_norm / 255.0
  # Return normalized images
  return train_norm, test_norm
```

3.3 Model Designing and Training

The training of different models in this project is outlined as follows:

1. **K-nearest neighbors**: For K-NN, the number of neighbors is set to 3. The model is fit on the training dataset (X_train, y_train). Having finished training, the model will be used to predict test images (X_test). The model's performance is evaluated by accuracy score.

```
from sklearn.neighbors import KNeighborsClassifier

kNN = KNeighborsClassifier(n_neighbors=3)
kNN.fit(X_train, y_train)
```

2. **Naive Bayes Model**: In this assignment, the naive bayes classifier is constructed from scratch, the conditional probabilities of attributes are determined by the frequency of existence in a specific class. The manhattan distance is used for simplicity. Like KNN, Naive Bayes classifier is evaluated by accuracy score.

```
        def NaiveBayesLearner(dataset):
        target_values = dataset.values[dataset.target]
        target_dist = CountingProbDist(target_values)
        attr_dist = {(attr, class_val):
        ↪  CountingProbDist(dataset.values[attr]) for attr in dataset.inputs
        ↪  for class_val in target_values}
        for example in dataset.examples:
                target_val = example[dataset.target]
                target_dist.add(target_val)
                for attr in dataset.inputs:
                        attr_dist[attr, target_val].add(example[attr])

        def predict(sample):
                def class_probability(target_val):
                        return
                        ↪  target_dist[target_val]*product(attr_dist[attr,
                        ↪  target_val][sample[attr]] for attr in
                        ↪  dataset.inputs)
                return max(target_values, key=class_probability)

        return predict
```

3. **Convolutional Neural Network**: The CNN consists of many convolutional layers, followed by maxpooling and fully connected layers. The learning rate is set to 0.01 and momentum = 0.9. I use categorical cross entropy as loss function. For validating, k-fold cross validation is used with k equal to 5 as default. The model is evaluated using accuracy score, confusion matrix and classification report.

```
        def define_model():
  model = Sequential()
  model.add(Conv2D(32, (3, 3), activation='relu',
  ↪  kernel_initializer='he_uniform', input_shape=(28, 28, 1)))
  model.add(MaxPooling2D((2, 2)))
  model.add(Conv2D(64, (3, 3), activation='relu',
  ↪  kernel_initializer='he_uniform'))
  model.add(Conv2D(64, (3, 3), activation='relu',
  ↪  kernel_initializer='he_uniform'))
  model.add(MaxPooling2D((2, 2)))
```

```
    model.add(Flatten())
    model.add(Dense(100, activation='relu', kernel_initializer='he_uniform'))
    model.add(Dense(10, activation='softmax'))
    # compile model
    opt = SGD(learning_rate=0.01, momentum=0.9)
    model.compile(optimizer=opt, loss='categorical_crossentropy',
    ↪  metrics=['accuracy'])
    return model
```

```
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d (Conv2D)             (None, 26, 26, 32)        320

 max_pooling2d (MaxPooling2  (None, 13, 13, 32)        0
 D)

 conv2d_1 (Conv2D)           (None, 11, 11, 64)        18496

 conv2d_2 (Conv2D)           (None, 9, 9, 64)          36928

 max_pooling2d_1 (MaxPoolin  (None, 4, 4, 64)          0
 g2D)

 flatten (Flatten)           (None, 1024)              0

 dense (Dense)               (None, 100)               102500

 dense_1 (Dense)             (None, 10)                1010

=================================================================
Total params: 159254 (622.09 KB)
Trainable params: 159254 (622.09 KB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

Figure 6: CNN summary.

4 Results

1. K-nearest neighbors: The accuracy score is 0.9705 97%, which is a quite good result.
2. Naive Bayes: The accuracy score got is 0.2687 27%, this is a low score which means that Naive Bayes is not suitable for classifying digits. The result can be improve by smooth for probability, choose significant attribute instead of input all pixels.
3. CNN: CNN model got a high accuracy score of 99.29%, this is the highest score in three models. Moreover, confusion matrix and classification report are plotted below:

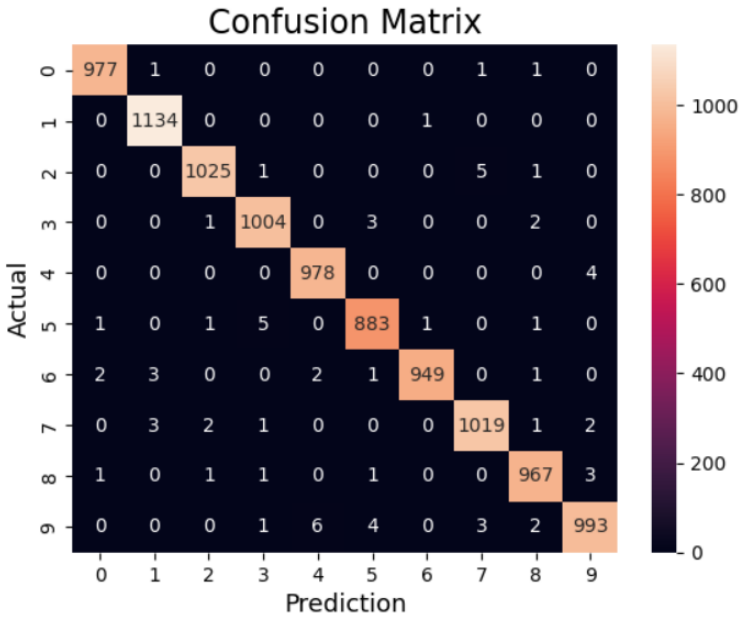

Figure 7: Confusion Matrix.

	precision	recall	f1-score	support
0	1.00	1.00	1.00	980
1	0.99	1.00	1.00	1135
2	1.00	0.99	0.99	1032
3	0.99	0.99	0.99	1010
4	0.99	1.00	0.99	982
5	0.99	0.99	0.99	892
6	1.00	0.99	0.99	958
7	0.99	0.99	0.99	1028
8	0.99	0.99	0.99	974
9	0.99	0.98	0.99	1009
accuracy			0.99	10000
macro avg	0.99	0.99	0.99	10000
weighted avg	0.99	0.99	0.99	10000

Figure 8: Classification Report.

5 Discussion

Deep learning approach performs better than machine learning approaches because CNNs have feature extraction layers, this character helps neural networks learn important features. In the opposite, traditional algorithms such as SVMs, k-NN require manual feature

extraction. The performance of these models heavily depends on the quality of the features extracted.

Moreover, CNNs can capture complex patterns and hierarchical structures in the data, as we increase capacity and depth of the network, the performance increase accordingly. Thirdly, models such as K-NN is prone to dataset with large number of dimensions, the MNIST dataset contains a huge number of dimensions which limit the performance of KNN, neural networks do not suffer from this effect.

For this MNIST classification task, Support Vector Machine approach with a radial basis function kernel would be the most appropriate classical machine learning technique. There are many reasons:

- SVMs are particularly effective in high-dimensional spaces and can perform well when the number of dimensions exceeds the number of samples.
- The RBF kernel allows the SVM to create complex decision boundaries, which is crucial for accurately classifying the digits. Additionally, SVMs have regularization parameters that can prevent overfitting, making them robust when dealing with the complexity of digit variations in the MNIST dataset.
- Historically, SVMs have been shown to perform competitively on the MNIST dataset, often achieving high accuracy, sometimes comparable to simpler neural networks before the widespread adoption of deep learning.
- SVMs construct a **maximum margin separator**-a decision boundary with largest possible distance to example points. This helps them generalize well.
- SVMs are instance-based models, which means they lead data to speak for themselve. Moreover, an SVM model keeps only the examples that are closest to the separating plane. Thus SVMs combine the advantages of nonparametric and parametric models: they have the flexibility to represent complex functions, but they are resistent to overfitting.

6 Conclusion

Github link for this project:

In this study, I successfully developed and evaluated machine learning models for the classification of digits in the MNIST database based on pixels.

Throughout the project, my objective was to do experiment and compare performance of different ML algorithms and between classical machine learning with deep learning. To achieve this, I use Python and libraries, packages involves numpy, scikit learn, and keras.

In the future, I want to try SVM model and compare its performance with CNN.

References

[1] https://www.geeksforgeeks.org/k-nearest-neighbours/

[2] Jason Brownlee PhD, (November 14, 2021), How to Develop a CNN for MNIST Handwritten Digit Classification, https://www.analyticsvidhya.com/blog/2021/05/convolutional-neural-networks-cnn/

[3] Stuart Russell, Peter Norvig (2021) *Artificial Intelligence: A Modern Approach*, Pearson, 4th ed.

[4] https://machinelearningmastery.com/how-to-develop-a-convolutional-neural-network-from-scratch