Intelligence Artificielle 2 Report

Building an Anti-Spam

Faculty of Science and Technology
University of Limoges

presented by

Doan Thi Van Thao
Nguyen Thi Mai Phuong

Master 1 CRYPTIS

Instructor: Mr. Karim Tamine

April 26, 2021

Contents

1 Anti-Spam

Anti-spam refers to services and solutions that focus on blocking and mitigating the effects of illegal emails – or spam – on email users. To achieve this objective, different types of anti-spam systems have been integrated with the email systems.

In this scope of the project, we implement an Anti-Spam based on 2 main classification algorithms: **Naive Bayes Network** and **Artificial Neural Network**. By using a supplied DataSet (see subsection 3.1), the Anti-Spam can detect spam and divert it to a spam folder (junk mailbox) based on the normal and abnormal activities of given electronic messaging system.

2 Classification Algorithms

2.1 Naive Bayes Network

Naive Bayes classifier is one supervised learning method based on Bayes's theorem. Naive Bayes network is applied if the values of the attributes of an example are independent given the class of the example.

We assume that a data set contains n instances (or rows) $\mathbf{x}_i, i = 1..n$; each consists of p attributes, that means $\mathbf{x}_i = (x_{i1}, x_{i2}, .., x_{ip})$. Each instance is assumed to belong to one (and only one) class $y \in \{y_1, y_2, ..., y_c\}$. Based on Bayes' theorem:

$$P(A|B) = \frac{P(B|A)P(A)}{P(B)} \tag{1}$$

Naive Bayes algorithm calculates a posterior class probability to an instance: $P(Y = y_j|X = \mathbf{x}_i)$ and then refers to the construction of a Bayesian probabilistic model. From Equation 1, we achieve:

$$P(y_j|\mathbf{x}_i) = \frac{P(\mathbf{x}_i|y_j)P(y_j)}{P(\mathbf{x}_i)} \tag{2}$$

Because of the independence between these attributes x_{i1} to each other, Equation 2 can be presented as:

$$P(y_j|\mathbf{x}_i) = \frac{\prod_{k=1}^{p} P(\mathbf{x}_{ik}|y_j)P(y_j)}{P(\mathbf{x}_i)} \tag{3}$$

In the binary (two-class) classification problems like Anti-spam, each instance is assigned to exactly one class, thus Equation 3 showed that it is necessary to only calculate the value of the numerator corresponding to each class and select that class for which this value is maximal. The denominator, $P(\mathbf{x}_i)$, does not depend on the class, so $P(\mathbf{x}_i)$ acts as a scaling factor and ensures that the posterior probability $P(y_j|\mathbf{x}_i)$ is properly scaled (i.e., a number between 0 and 1)[1]. This rule is called the *maximum posterior rule* (Equation 4).

[1]Daniel Berrar, Bayes' Theorem and Naive Bayes Classifier, Tokyo Institute of Technology (January 2018).

Also, the resulting "winning" class is also known as the *maximum a posteriori* (MAP) class, and it is calculated as $\hat{y}$ for each instance $\mathbf{x}_i$ as follows:

$$\hat{y} = \underset{y_j}{\operatorname{argmax}} \prod_{k=1}^{p} P(\mathbf{x}_{ik}|y_j)P(y_j) \tag{4}$$

In case the expression values of an attribute in p follows a normal distribution, we can model the probability density for class y_i as:

$$f(x|y_i) = \frac{1}{\sqrt{2\pi\sigma_i}} e^{\frac{-(x-\mu_j)^2}{2{\sigma_i}^2}} \tag{5}$$

where μ_i and σ_i denote the mean and standard deviation of the attribute 's expression value for class y_i, respectively.

2.2 Artificial Neural Network (ANN)

Artificial neural networks (ANN) are a set of algorithms, modeled loosely after the human brain, that are designed to recognize patterns. Each parameter in the network, sometimes also referred to as neurons, is a function which produces an output, after receiving one or multiple inputs from previous layers. Those outputs are then passed to the next layer of neurons until the terminal neurons, which then output the final result for the model.

2.2.1 ANN without hidden layers

An artificial neuron is a function f_j of the input $x = (x_1, ..., x_d)$ weighted by a vector $w_j = (w_{j,1}, ..., w_{j,d})$, completed by a neuron bias b_j, and associated to an activation function ϕ, namely:

$$y_j = f_j(x) = \phi(\langle w_j, x \rangle + b_j) \tag{6}$$

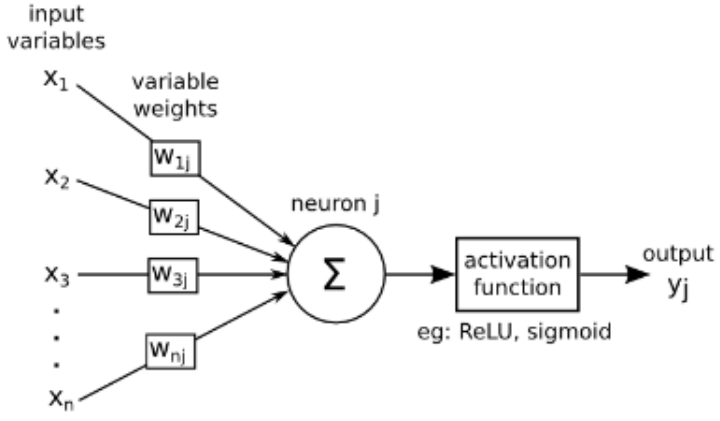


Figure 1: ANN without hidden layers.

A schematic representation of an artificial neuron without hidden layers[2] is as on Figure 1. Neural network without hidden layers is so-called a single perceptron, where the final output is directly calculated from the beginning input.

2.2.2 ANN with one hidden layer

Being different to a single perceptron is a multilayer perceptron whose structure is composed by several hidden layers of neurons. A neural network with one hidden layer is illustrated on Figure 2.

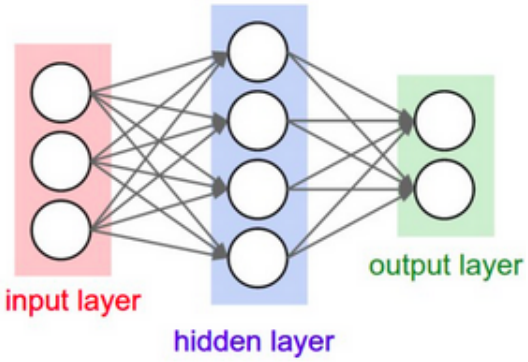


Figure 2: Neural network with one hidden layer.

Each unit (or neuron) of a layer is linked to all the units of the next layer but has no link with the neurons of the same layer. In addition, the parameters of the architecture are the number of hidden layers and of neurons in each layer. The mathematical formulation of the neural network is presented as follows:

We set $h^{(0)}(x) = x$, for the hidden layer,

$$\begin{aligned} a^{(1)}(x) &= b^{(1)} + W^{(1)} h^{(0)}(x) \\ h^{(1)}(x) &= \phi(a^{(1)}(x)) \end{aligned} \tag{7}$$

For the output layer,

$$\begin{aligned} a^{(2)}(x) &= b^{(2)} + W^{(2)} h^{(1)}(x) \\ h^{(2)}(x) &= \psi(a^{(2)}(x)) := f(x, \theta). \end{aligned} \tag{8}$$

where ϕ is the activation function and ψ is the output layer activation function. $W^{(i)}$ is a matrix with number of rows being the number of neurons in the layer k and number of columns being the number of neurons in the layer $k-1$.

[2]Neural Networks and Introduction to Deep Learning, WikiState

2.2.3 The Backpropagation

The backpropagation is the most fundamental in ANN, which aims to minimize the cost function by adjusting network's weights and biases using the method of gradient descent. In simple terms, after each forward pass through a network, backpropagation performs a backward pass while adjusting the model's parameters. Suppose n is the number of training instances, ϵ_r is learning rate, and L is the number of layers, then the empirical quadratic loss[2] is proportional to $\sum_{i=1}^{n} R_i(\theta)$, with:

$$R_i(\theta) = \sum_{k=1}^{K} (Y_{i,k} - f_k(X_i, \theta))^2 \tag{9}$$

At step $r+1$, we have:

$$\begin{aligned} W_{k,m}^{(L+1,r+1)} &= W_{k,m}^{(L+1,r)} - \varepsilon_r \sum_{i \in B} \frac{\partial R_i}{\partial W_{k,m}^{(L+1,r)}} \\ W_{m,l}^{(L,r+1)} &= W_{m,l}^{(L,r)} - \varepsilon_r \sum_{i \in B} \frac{\partial R_i}{\partial W_{m,l}^{(L,r)}} \end{aligned}$$

Figure 3: The backpropagation in neural network.

3 Implementation

3.1 Spam Dataset

"Spam" is a provided data set, which includes 2 subsets of data as follows:

- "Spam detection - For model creation.csv": so-called training data with 2972 rows, each row has 57 attributes, except the target column;
- "Spam detection - For prediction.csv": so-called test data with 1274 rows, each row has 57 attributes.

	GOAL-Spam	word_freq_make	word_freq_address	word_freq_all	word_freq_3d	word_freq_our	word_freq_over	word_freq_remove	word_freq_internet	word_freq_order	word_freq_mail	word_freq_receive	word_freq_will	word_freq_people	word_freq_report	word_freq_addresses	w
2	No	0	0	0	0	0.19	0	0	0	0.09	0	0.09	0.59	0	0	0	
3	No	0	0	0	0	0	0	0	0	0	0	0	1.75	0	0	0	
4	Yes	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	
5	No	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	
6	No	0	0	1.33	0	0	0	0	0	0	1.33	0	0	0	0	0	
7	Yes	0	0	1.25	0	1.25	0.62	0	0	0	0	0	0	0	0	0	
8	No	0	0	0.79	0	0	0	0	0	0	0	0	1.58	0	0	0	
9	Yes	0	0	0	0	0	0	1.96	0	0	0	0	0	0	0	0	
10	No	0	1.23	0	0	0	0	0	0	0	1.23	0	0.61	2.46	0	0	
11	Yes	0.08	0.08	0.76	0	0.85	1.02	0.25	0.17	0.59	0.08	0.17	0.59	0.17	0	2.21	

Figure 4: A part of training data set.

Each row in these 2 datasets will be considered as an instance $\mathbf{x}_i$ with $\mathbf{x}_i = (x_{i1}, x_{i2}, .., x_{ip})$ where $p = 57$ as mentioned on subsection 2.1. Also, the first column **"GOAL-Spam"** is corresponding to the class $y \in \{y_1, y_2, ..., y_c\}$. Here, there is only 2 classes: *Yes* and *No*, so $c = 2$.

3.2 Naive Bayes Model for Anti-spam

3.2.1 Library

As required, it is not allowed to use any specific libraries for this model, thus we import only 3 basic libraries, namely *csv, math, time* for file handling and mathematical calculation.

3.2.2 Implementation

We have noticed that each of 57 attributes of $\mathbf{x}_i$ also follows Gaussian distribution. Therefore, the main idea is to calculate normal distribution parameters: μ_i and $\sigma_i, i = 1, 2...57$ based on training data set corresponding to each class y_i; and then measure probability density of each $\mathbf{x}_i$ in test data by following Equation 5.

```
def clasify_xi(x_i):
    info_x_i = calculate_class_prob(x_i)
    prob_by_class = probByClasses
    idx = 0
    prob_x_i = {}
    for label in info_x_i:
        prob = info_x_i[label] * prob_by_class[idx]
        idx += 1
        prob_x_i[label] = prob

    max_prob = -1
    lable_xi = None
    for label, prob in prob_x_i.items():
        if lable_xi is None or prob > max_prob:
            max_prob = prob
            lable_xi = label
    return lable_xi
```

Listing 1: Classify each $\mathbf{x}_i$ of test set.

The function `classisy_xi` calculates *maximum a posteriori* as shown on Equation 4 by the variable `prob` in each `label` of `info_x_i`. After all input vectors $\mathbf{x}_i$ of test set are classified, the accuracy of Naive Bayes classifier had been done by making a comparison between their labels in test set and their label assigned by the classification.

3.3 Artificial Neural Network for Anti-spam

3.3.1 Library

In contrast to Bayes Model presented above, ANN classifier implements Keras library for classification.

3.3.2 Implementation

As same as Bayes Network, here the data set also includes 2 parts, training set and validation set as in subsection 3.1. Each row of both data sets then separated into 2 parts: the first is for parameter's prediction (57 columns), and the second is target column (column **"GOAL-Spam"**). Next step is for data processing, namely to normalize data and to one hot encode the classes. Normalization aims to change the values of an array to a common scale, without distorting differences in the ranges of values.

```
model = Sequential()
model.add(Dense(2, activation='softmax', input_dim=57))
model.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'])
graph_data = model.fit(X_train, y_train, epochs=180, batch_size=20,
             validation_data=(X_eval, y_eval), verbose=0)
```

Listing 2: ANN without hidden layers.

```
model = Sequential()
model.add(Dense(500, activation='sigmoid', input_dim=57))
model.add(Dropout(0.2))
model.add(Dense(2, activation='softmax'))
model.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'])
graph_data = model.fit(X_train, y_train, epochs=300, batch_size=20,
             validation_data=(X_eval, y_eval), verbose=0)
```

Listing 3: ANN with one hidden layer.

The main part of ANN classification is building Neural Network. In our data set, the input is of 57 values and output is of 2 values. So the input and output layer is of 57 and 2 dimensions respectively. For ANN without hidden layers, the output layer implement softmax activation function as on Listing 2. However, this function will be changed differently to evaluate the performance of the algorithm (see more in section 5).

For ANN with hidden layers, we are using one hidden layer of 500 dimensions and one output layer of 2 dimensions as on Listing 3. Here loss is cross entropy loss.

`Categorical_crossentropy` specifies that we have multiple classes. In this case, the data set is big and we cannot fit complete data at once so we use batch size. This divides data into batches each of size equal to `batch_size`. After all, we found that batch_size also affects classifier's performance, which will be clarified in section 5.

4 Graphical User Interface (GUI)

In this project, we use the library `tkinter` to build the GUI. For the basic setup, we can choose the model which we want to run.

Moreover, with ANN, we can set up the parameters `epochs` and `batch_size`. If two parameters keep the value 0, the most satisfy value that we find out in the subsection 5.2 will be apply.

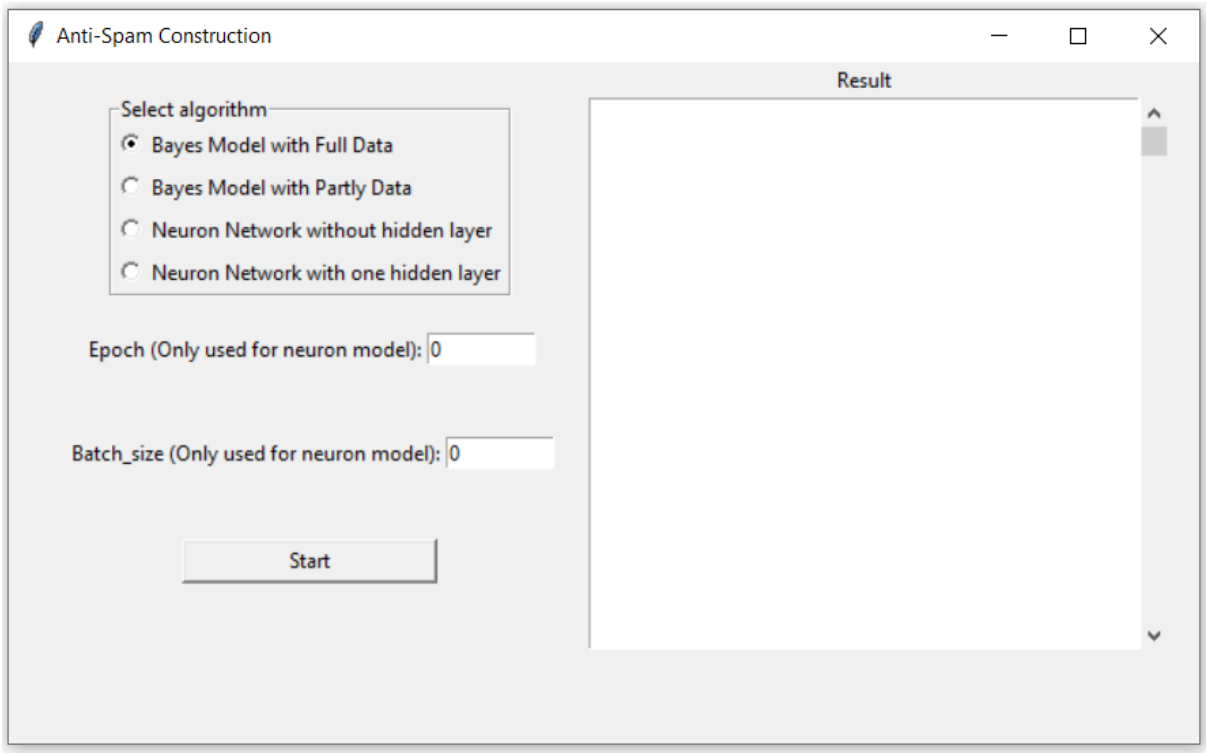


Figure 5: Graphical User Interface - Setup State

The result will be displayed on the right side of the window. Within it, we list all the necessary information like Accuracy/Error on training data, Accuracy/Error on evaluating data, and the total processing time as illustrated on Figure 6 below. We decide to round its float number to 8 decimal places for easier to compare results between models.

Besides, if we choose to run ANN models, a new window with an accuracy comparison graph will appear.

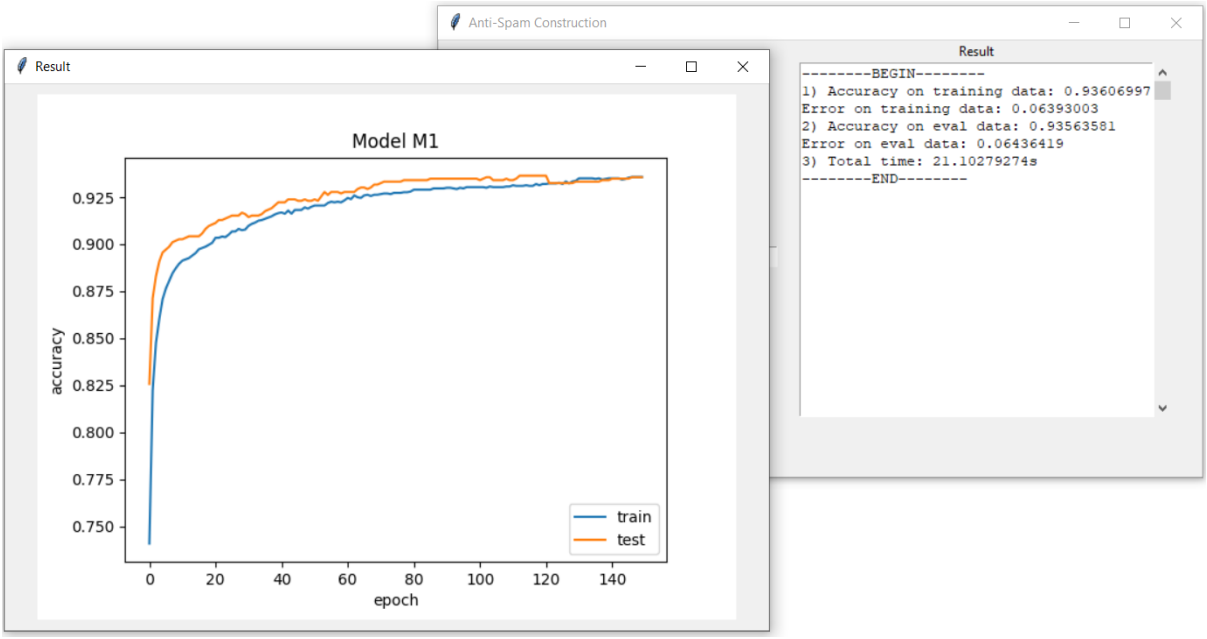


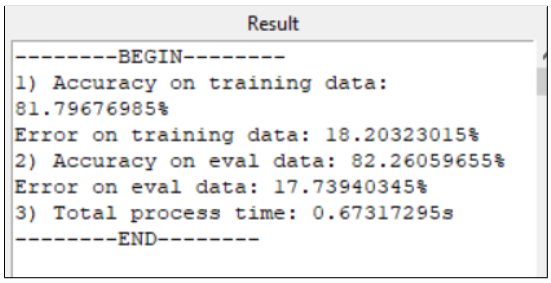
Figure 6: Graphical User Interface - Result State

5 Performance Evaluation

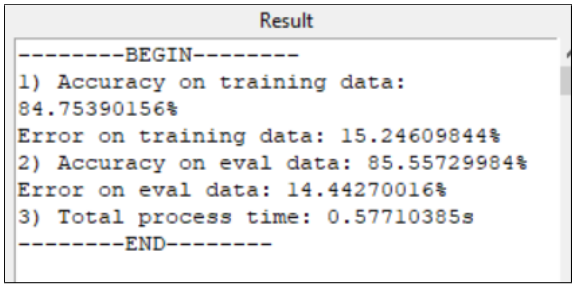
In this section, we will illustrate the results returned when apply Naive Bayes Network and Artificial Neural Network for Anti-spam.

5.1 Naive Bayes Network for Anti-spam

For the sake of performance comparison for Naive Bayes Model, we have done data processing by removing some noisy data in **Spam detection - For model creation.csv** before training it for model creation. As a result, the training data set after processed includes 2500 rows, compared to 2972 rows in original data set. The different results which get from two cases: using the original data (fully data) and using processed data (partly data) are depicted on Figure 7.



(a) Bayes Model with original data



(b) Bayes Model with processed data

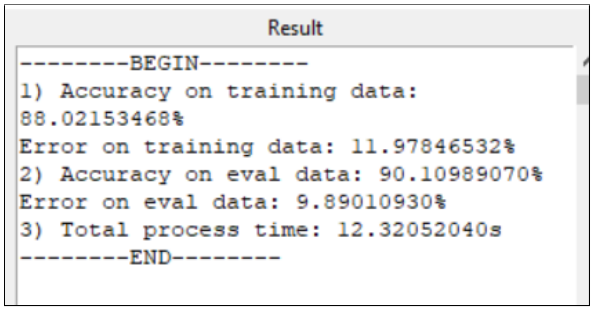
Figure 7: Bayes Model for Anti-Spam

As it might be seen, the accuracy rate on processed data gave a better performance than that we get from original data. Thus, it can be concluded that noise in the data set will lead to decreased classification accuracy and poor prediction results. For future work, we will try to apply more noise handling techniques to this implementation.

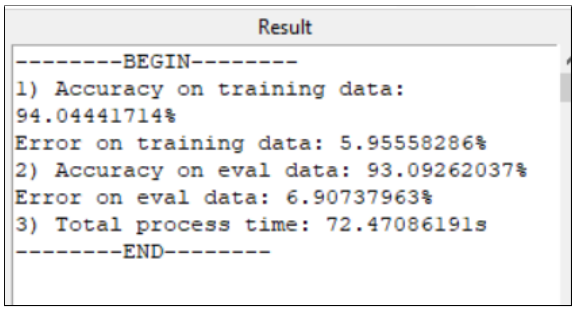
5.2 ANN for Anti-spam

5.2.1 Batch Size on performance

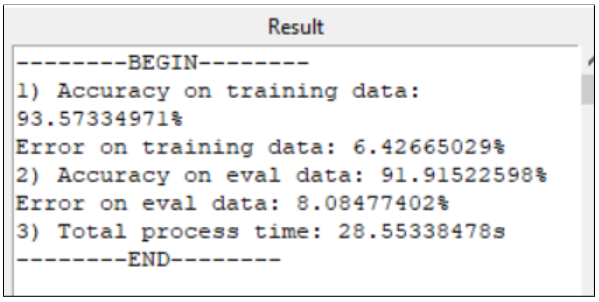
The `batch_size` is a hyper-parameter that defines the number of samples to work through before updating the internal model parameters. To find a satisfying `batch_size`, we try to implement different `batch_size` values: 5, 20 and 500.



(a) Result with `batch_size` 500



(b) Result with `batch_size` 5



(c) Result with `batch_size` 20

Figure 8: Result of ANN with different `batch_size`

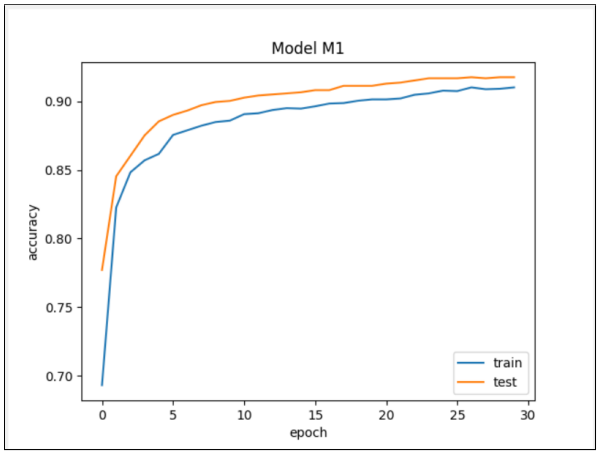
As it has been pointed out, the result of `batch_size` 500 is not good as we expected despite its process time is the most quickly. Performances of the two remaining cases are better and almost equal. However, the processing time is too long with `batch_size`-5 implementation.

According to this experiment, it is clearly seen that the `batch_size` impacts how quickly ANN model can learn, as well as the stability of the learning process. In addition, it shows that `batch_size` 20 is a good choice.

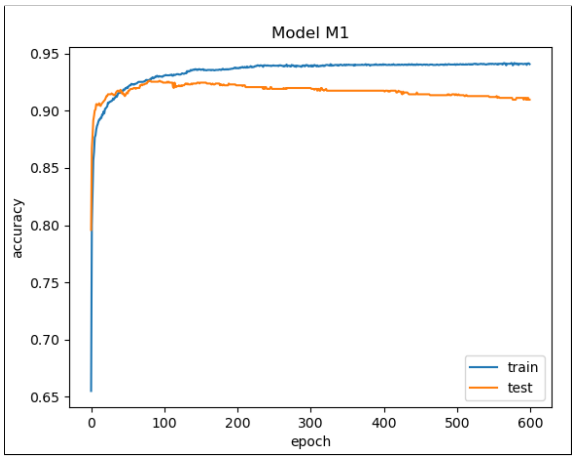
5.2.2 Epoch on performance

The `epochs` is a hyper-parameter that defines the number times that the learning algorithm will work through the entire training data set. A correct `epochs` should make the learning curves good-fit, not under-fit or over-fit.

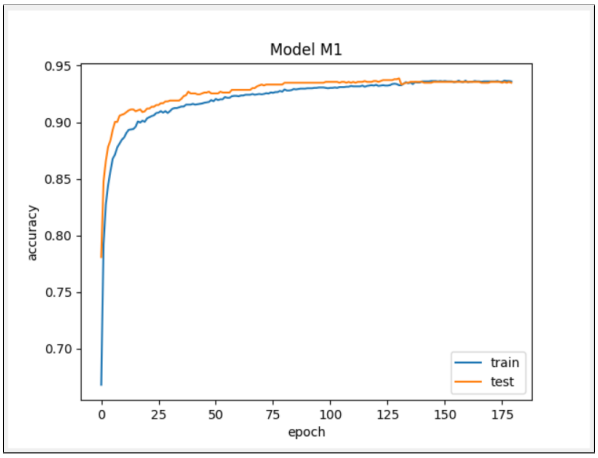
Similar with `batch_size`, we try to run many `epochs` values to find a good `epochs` for each Neural Network model. Figure 9 below illustrates the effect of `epochs` on the performance of ANN without hidden layers (called M1) implemented on our training data set. As being shown, `epochs` 180 for M1 model is a good choice. Doing the same type of the experiment, we found out that `epochs` 300 is the better for ANN with one hidden layer (called M2).



(a) Under-fit with `epochs` 30



(b) Over-fit with `epochs` 600



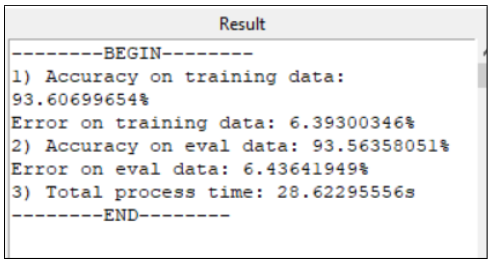
(c) Good-fit with `epochs` 180

Figure 9: Learning curves of model M1 with different `epochs`

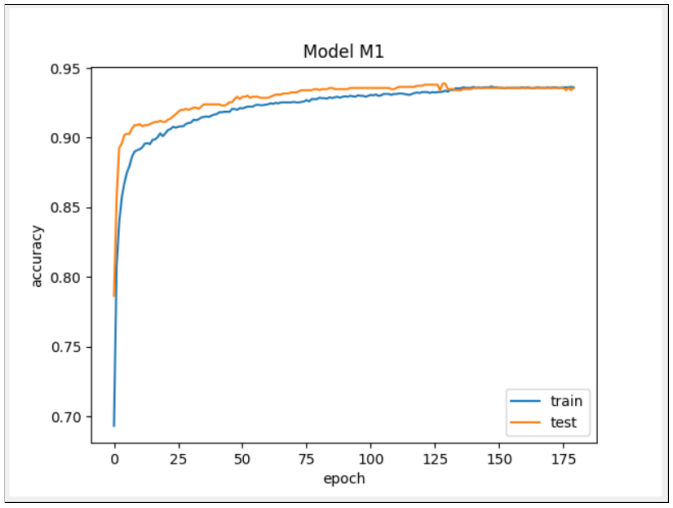
5.2.3 Activation function on performance

When we implement ANN without hidden layers, we conduct many experiments with different activation functions, such as `sigmoid` and `softmax` to see which brings the best

performance. The `sigmoid` activation function transforms an input into a value between 0.0 and 1.0, whereas `softmax` function normalizes input into a probability distribution.

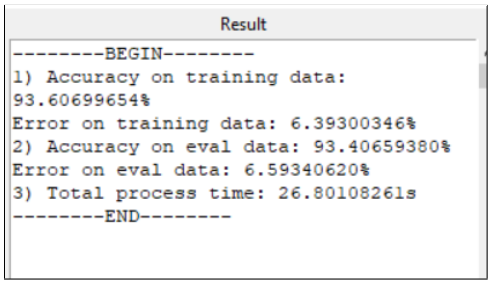


(a) Result

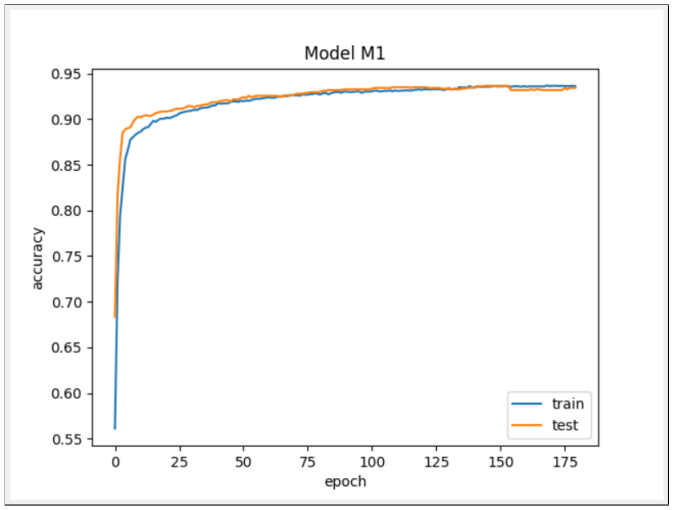


(b) Accuracy comparison graph

Figure 10: M1 using `sigmoid` function



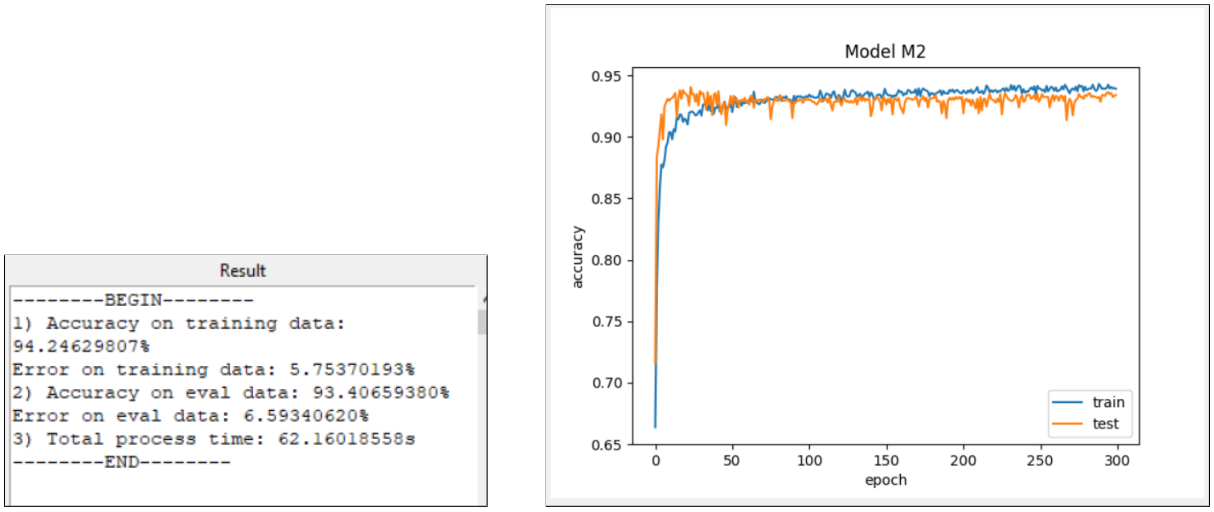
(a) Result



(b) Accuracy comparison graph

Figure 11: M1 using `softmax` function

As we can see, in this case of a binary classification, `sigmoid` and `softmax` give the equivalent results. Similarly, for M2 model, as displayed on Figure 12 we use `sigmoid` activation on the hidden layer and `softmax` for the output layer.



(a) Result (b) Accuracy comparison graph

Figure 12: Effect of activation function on M2 model

5.3 Analysis of Results

Recall, in this section, we use M1 stands for ANN without hidden layer, and M2 for ANN with one hidden layer. In order to compare ANN models to Naive Bayes one, we set their parameters as presented in section 5.2, then calculate average accuracy of each model on test data set (3.1) by running function `model.predict(X_test)` 10 times, and computing their average value. This average accuracy will be used to compare to Bayes model's accuracy as showed on Table 1 as follows:

Accuracy	**Results**		
	Naive Bayes Network	**M1**	**M2**
On training data	81.79676985%	93.58351725%	94.25718637%
On test data	82.26059655%	93.44637629%	93.65659106%
Total process time	**0.67317295s**	**15.80108261s**	**45.43263051s**

Table 1: Comparison of three models

As we can see, Naive Bayes model works very fast; however, its result is not as good as others. Meanwhile, the results of M1 and M2 are not much different. But M2 has pretty higher performance. On the other hand, the processing time of M2 is almost 3 times more than the process time of M1.

To clarify M1 and M2 performances, we add test data set into function `model.fit()` as indicated in the implementation part of subsection 3.3 and then compare their accuracy. Two models will be set up with the same parameters: `batch_size` 20 and `epochs` 300. Figure 13 below shows their difference.

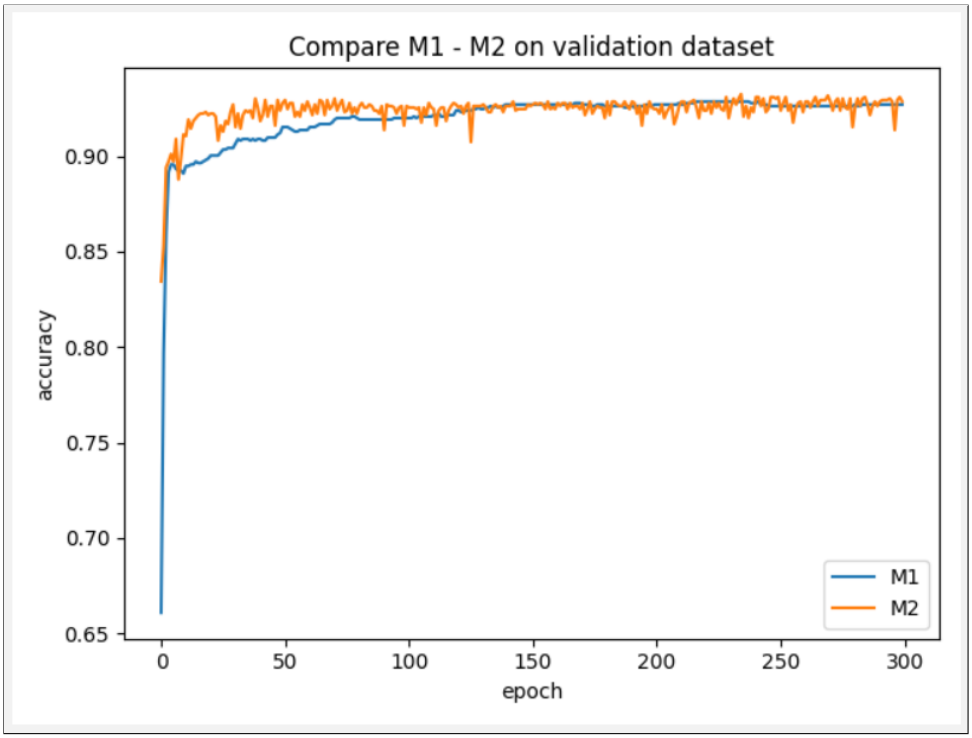


Figure 13: Compare M1 and M2

According to Figure 13, at very first epochs, M2 model already got better performance with about 93% accuracy, while M1 gradually increased from 90%. Even so, since epoch 180, their accuracy becomes not much different. Besides, it is apparent to see the fluctuation of Neural Network M2 compared to Neural Network M1.

6 Summary and Conclusion

Summary

In the scope of the project, we recalled the general principle of the two learning methods: Naive Bayes Network and Artificial Neural Network; as well as described the implementation method in these models produced for the given data set. Besides, we conducted many experiments in order to determine optimal parameters for each model, together with making analysis and comparison of the results obtained.

Conclusion

Based on experimental results, we conclude that Naive Bayes Model works quickly and can save tons of time compared to ANN. However, its worst limitation is the assumption of p independent attributes (or predictors) of vector $\mathbf{x}_i$. In real life, it is almost impossible to get a set of predictors which are completely independent. Additionally, the model supposes that each predictor follows Gaussian distribution while it might be not totally

exact in reality. These are the main reasons that make Naive Bayes Network in the Anti-spam problem perform not well as other models.

Comparing to Naive Bayes classifier, ANN Models are more complex to implement with many decisions about hidden layers, activation function, `batch_size`, or `epochs`. However, the results we obtain from ANNs are better than Naive Bayes Model. In particular, the model with one hidden layer in the Anti-spam problem is pretty better than the model without hidden layers. Although the highlights of M1 model are its processing time and the stability, we can say M2 model expects to bring higher performance especially when input data expands bigger in size.