VIỆN HÀN LÂM KHOA HỌC VÀ CÔNG NGHỆ VIỆT NAM

HỌC VIỆN KHOA HỌC VÀ CÔNG NGHỆ VIỆT NAM

BÁO CÁO CUỐI KỲ
KHAI PHÁ DỮ LIỆU

Tên đề tài: Tự động phân loại tin tức văn bản dựa trên các phương pháp học máy

Giảng viên : PGS.TS. Nguyễn Đức Dũng

Học viên : Phan Thanh Hoài

Lớp : ITT18A-01

Hà Nội, 2018

VIỆN HÀN LÂM KHOA HỌC VÀ CÔNG NGHỆ VIỆT NAM

HỌC VIỆN KHOA HỌC VÀ CÔNG NGHỆ VIỆT NAM

BÁO CÁO CUỐI KỲ
KHAI PHÁ DỮ LIỆU

Tên đề tài: Tự động phân loại tin tức văn bản dựa trên các phương pháp học máy

Giảng viên : PGS.TS. Nguyễn Đức Dũng

Học viên : Phan Thanh Hoài

Lớp : ITT18A-01

Hà Nội, 2018

MỤC LỤC

DANH MỤC HÌNH ẢNH

LỜI MỞ ĐẦU

Trong khuôn khổ môn học Khai phá dữ liệu, nằm nắm vững hơn các thuật toán, đồng thời thể hiện được việc nghiên cứu và học tập của mình, em xin được thực hiện đề tài "Tự động phân loại tin tức văn bản dựa trên các phương pháp học máy".

Đề tài của em xuất phát từ nhu cầu môn học và tìm hiểu tin tức sự kiện được đăng trên các báo điện tử.

Từ nhu cầu thực tế và thông qua các kiến thức đã được học, em đề xuất giải pháp sử dụng thu thập tự động bằng cách sử dụng jsoup phiên bản 1.8.3 để tìm kiếm và trích xuất thông tin. Dữ liệu được lấy từ báo điện tử Dân Trí (http://dantri.com.vn) với 5 chủ đề nổi bật được lựa chọn là Sức khoẻ, Pháp luật, Kinh doanh, Thể thao, và Xe.

Trong quá trình tìm hiểu em có sử dụng một số tài liệu tham khảo bao gồm tài liệu môn học và các tài liệu chuyên ngành.

Trong quá trình tìm hiểu, do thời gian và kiến thức còn hạn chế nên trong báo cáo còn có sai sót, em rất mong thầy giáo và các bạn đóng góp ý kiến để đề tài của em được hoàn thiện hơn.

Em xin chân thành cảm ơn!

1. GIỚI THIỆU

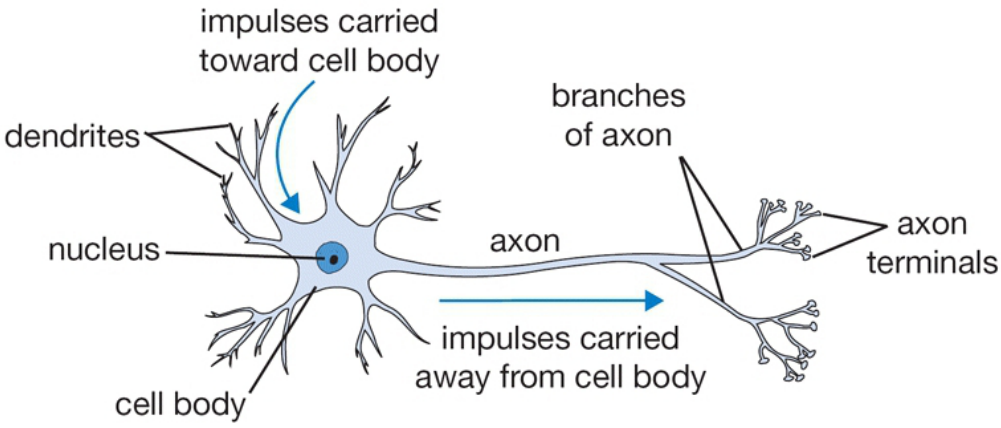
Trong báo cáo giữa kì, em đã giới thiệu sơ lược về bài toán phân lớp tin tức văn bản dựa trên các mô hình học máy, cũng như giới thiệu một phương pháp thu thập, tiền xử lý, và vector hoá dữ liệu văn bản. Dữ liệu này sau đó sẽ được sử dụng như đầu vào cho các bộ phân lớp. Trong báo cáo này, em trình bày 2 phương pháp để phân loại văn bản đã thu thập, đó là mô hình mạng nơ-ron nhân tạo, và xgboost.

2. MẠNG NƠ-RON NHÂN TẠO

Được xây dựng dựa trên ý tưởng về cấu trúc của mạng nơ-ron sinh học trong bộ não người, mạng nơ-ron nhân tạo gồm nhiều nơ-ron, mỗi nơ-ron được gọi là *perceptron* [1], kết nối với nhau qua các hàm kích hoạt. Lấy cảm hứng từ mạng nơ-ron sinh học, mạng nơ-ron nhân tạo được hình thành từ các tầng nơ-ron nhân tạo. Có 3 kiểu tầng chính là tầng vào (input layer) biểu diễn cho đầu vào, tầng ra (output layer) biểu diễn cho kết quả đầu ra và tầng ẩn (hidden layer) thể hiện cho các bước suy luận trung gian. Mỗi nơ-ron sẽ nhận tất cả đầu vào từ các nơ-ron ở tầng trước đó và sử dụng một hàm kích hoạt dạng (activation function) phi tuyến như sigmoid, ReLU, tanh để tính toán đầu ra.

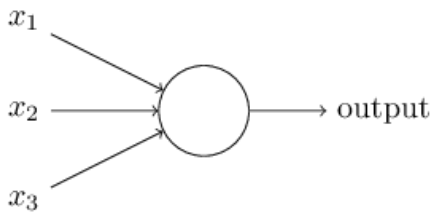
2.1. Perceptrons

2.1.1. Perceptron cơ bản



Hình 1. Kiến trúc nơ-ron sinh học

Hình 1. mô tả kiến trúc một nơ-ron sinh học. Giống với nơ-ron sinh học, một nơ-ron nhân tạo có thể nhận nhiều đầu vào và cho ra một kết quả duy nhất. mô hình perceptron có thể biểu diễn như sau:



Hình 2. Mô hình Perceptron

với x_1, x_2, x_3 là các đầu vào, và 1 đầu ra output tương ứng. Một perceptron sẽ nhận một hoặc nhiều đầu *x* vào dạng nhị phân và cho ra một kết quả *o* dạng nhị phân duy nhất. Các đầu vào được điều phối sự ảnh hưởng bởi các tham số trọng lượng tương ứng *w* của nó, còn kết quả đầu ra được quyết định dựa vào một ngưỡng *b*:

$$o = \begin{cases} 0 \ if \ \sum_i w_i x_i \leq threshold \\ 1 \ if \ \sum_i w_i x_i > threshold \end{cases}$$

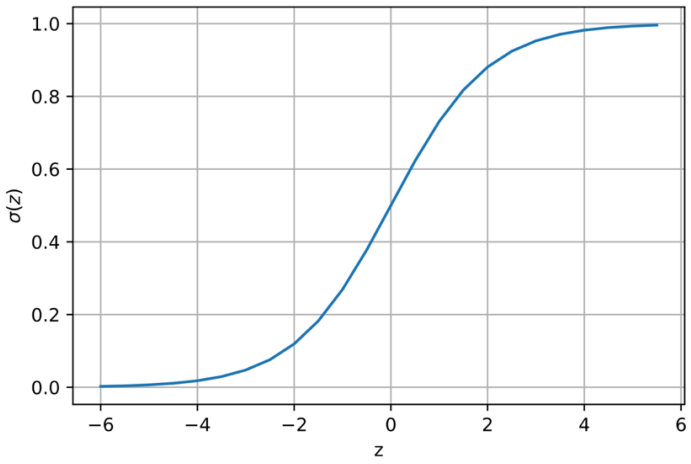
Đặt $b = -threshold$, ta có thể viết lại thành:

$$o = \begin{cases} 0 \ if \ \sum_i w_i x_i + b \leq 0 \\ 1 \ if \ \sum_i w_i x_i + b > 0 \end{cases}$$

2.1.2. Sigmoid Neuron

Với đầu vào và đầu ra dạng nhị phân, ta rất khó có thể điều chỉnh một lượng nhỏ đầu vào để đầu ra thay đổi chút ít, nên để linh động, ta có thể mở rộng chúng ra cả khoảng [0,1]. Lúc này đầu ra được quyết định bởi một hàm sigmoid $\sigma(w^T x)$. Hàm sigmoid có công thức:

$$\sigma(z) = \frac{1}{1 + e^{-z}}$$



Hình 3. Đồ thị hàm Sigmoid

Đặt z = w^Tx, công thức của perceptron lúc này sẽ có dạng:

$$o = \sigma(\mathrm{z}) = \frac{1}{1 + \exp(-w^T\mathrm{x})}$$

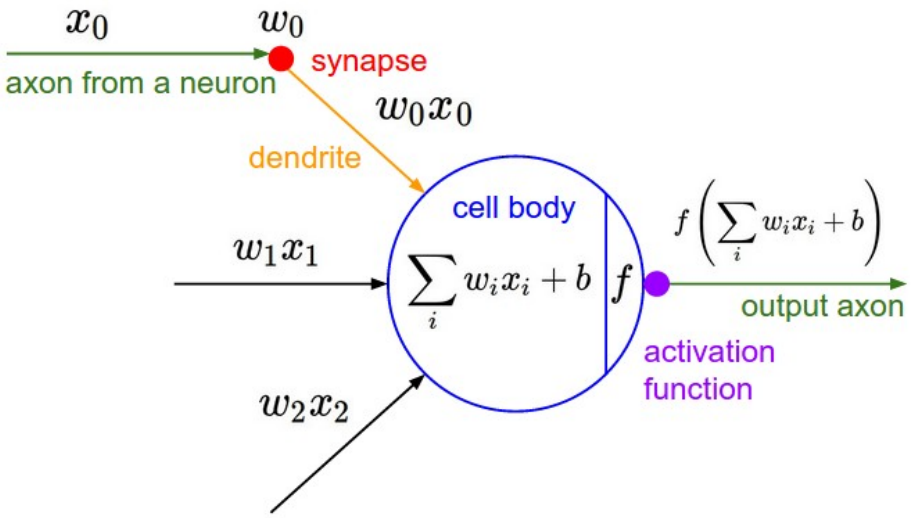
Tới đây thì ta có thể thấy rằng mỗi sigmoid neuron cũng tương tự như một bộ phân loại tuyến tính (logistic regression) bởi xác suất:

$$P(y_i = 1|x_i; w) = \sigma(w^T\mathrm{x}).$$

Ngoài hàm sigmoid, ta còn có thể một số hàm khác như hàm tanh, ReLu để thay thế hàm sigmoid bởi dạng đồ thị của nó cũng tương tự như sigmoid. Một cách tổng quát, hàm perceptron được biểu diễn qua một hàm kích hoạt (activation function):

$$o = f(\mathrm{z}) = \mathrm{f}(w^T\mathrm{x})$$

Bằng cách biểu diễn như vậy, ta có thể coi một neuron nhân tạo được thể hiện như sau:

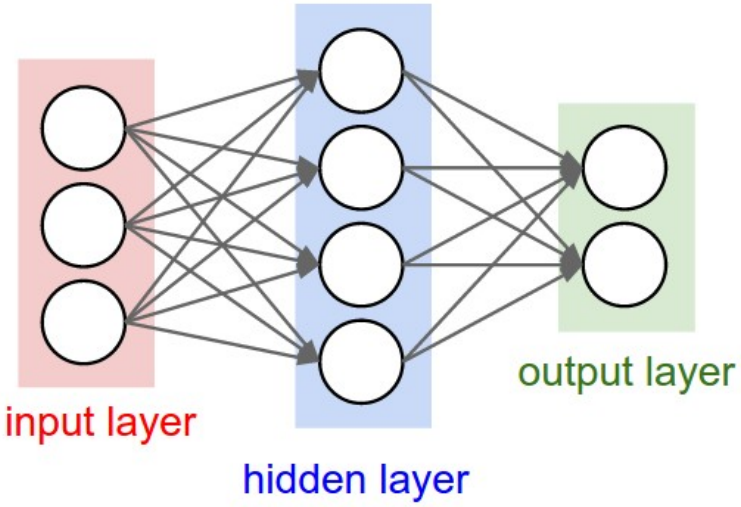


Hình 4. Mô hình tương ứng với mạng nơ-ron sinh học

Một điểm cần lưu ý là các hàm kích hoạt buộc phải là hàm phi tuyến. Vì nếu nó là tuyến tính thì khi kết hợp với phép toán tuyến tính w^Tx thì kết quả thu được cũng sẽ là một thao tác tuyến tính, từ đó, nó trở nên vô nghĩa.

2.2. Kiến trúc mạng Nơ-ron nhân tạo

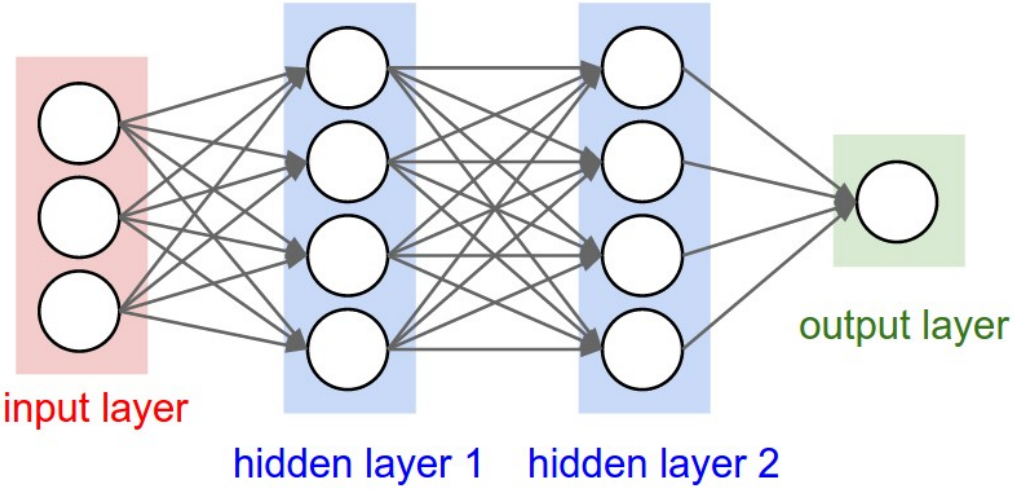
Mạng nơ-ron nhân tạo là sự kết hợp của của các tầng perceptron hay còn được gọi là perceptron đa tầng (multilayer perceptron) [2]:



Hình 5. Mạng nơ-ron với 1 tầng ẩn

Một mạng nơ-ron nhân tạo có 3 kiểu tầng:

- *Tầng vào (input layer):* Là tầng bên trái cùng của mạng thể hiện cho các đầu vào của mạng.
- *Tầng ra (output layer):* Là tầng bên phải cùng của mạng thể hiện cho các đầu ra của mạng.
- *Tầng ẩn (hidden layer):* Là tầng nằm giữa tầng vào và tầng ra thể hiện cho việc suy luận logic của mạng. Một mạng nơ-ron chỉ có 1 tầng vào và 1 tầng ra nhưng có thể có nhiều tầng ẩn.



Hình 6. Mạng nơ-ron với nhiều tầng ẩn

Trong mạng nơ-ron nhân tạo, mỗi nút mạng là một sigmoid nơ-ron nhưng hàm kích hoạt của chúng có thể khác nhau. Tuy nhiên trong thực tế người ta thường để

chúng cùng dạng với nhau để tính toán cho thuận lợi. Ở mỗi tầng, số lượng các nút mạng (nơ-ron) có thể khác nhau tuỳ thuộc vào bài toán và cách giải quyết. Nhưng thường khi làm việc người ta để các tầng ẩn có số lượng nơ-ron bằng nhau. Ngoài ra, các nơ-ron ở các tầng thường được liên kết đôi một với nhau tạo thành mạng kết nối đầy đủ (full-connected network).

2.3. Lan truyền tiến trong mạng nơ-ron

Các node trong mạng (nơ-ron) được kết hợp đôi một với nhau theo một chiều duy nhất từ tầng vào tới tầng ra. Nghĩa là mỗi nốt ở một tầng nào đó sẽ nhận đầu vào là tất cả các nốt ở tầng trước đó mà không suy luận ngược lại. Hay nói cách khác, việc suy luận trong mạng nơ-ron là suy luận tiến (feedforward):

$$z_i^{(l+1)} = \sum_{j=1}^{n^l} w_{ij}^{(l+1)} a_j^{(l)} + b_i^{(l+1)}$$

$$a_i^{(l+1)} = f(z_i^{(l+1)})$$

Trong đó, n^l là số lượng nút ở tầng l tương ứng, và a_j^l là nút mạng thứ j của tầng l. $w_{ij}^{(l+1)}$ là tham số trọng lượng của đầu vào a_j^l đối với nút mạng thứ i của tầng $l+1$ và $b_i^{(l+1)}$ là độ lệch (bias) của nút mạng thứ i của tầng $l+1$. Đầu ra của nút mạng này được biểu diễn bằng $a_i^{(l+1)}$ ứng với hàm kích hoạt $f(z_i)$ tương ứng.

Để tiện tính toán, ta coi a_0^l là một đầu vào và $w_{i0}^{(l+1)} = b_i^{(l+1)}$ là tham số trọng lượng của đầu vào này. Khi đó, công thức trên có thể được viết lại thành:

$$z_i^{(l+1)} = w_i^{(l+1)} . a^{(l)}$$

$$a_i^{(l+1)} = f(z_i^{(l+1)})$$

Nếu nhóm các tham số của mỗi tầng thành một ma trận có các cột tương ứng với tham số mỗi nút mạng thì ta có thể tính toán cho toàn bộ các nút trong một tầng bằng vector:

$$\boldsymbol{z}^{(l+1)} = \boldsymbol{W}^{(l+1)} . \boldsymbol{a}^{(l)}$$

$$\boldsymbol{a}^{(l+1)} = f(\boldsymbol{z}^{(l+1)})$$

2.4. Học với mạng Nơ-ron nhân tạo

Cũng tương tự như các bài toán học máy khác thì quá trình học vẫn là tìm lấy một hàm lỗi để đánh giá và tìm cách tối ưu hàm lỗi đó để được kết quả hợp lý nhất có thể. Như đã đề cập mỗi nút mạng của nơ-ron có thể coi là một bộ phân loại (logistic regression) có hàm lỗi là:

$$\mathrm{J}(\boldsymbol{W}) = -\frac{1}{m}\sum_{i=1}^{m}(y^{(i)}\log\left(\sigma^{(i)}\right) + \left(1 - y^{(i)}\right)\log\left(1 - \sigma^{(i)}\right))$$

Trong đó, *m* là số lượng dữ liệu huấn luyện, y^i là đầu ra thực tế của dữ liệu thứ *i* trong tập huấn luyện. Còn $\sigma^{(i)}$ là kết quả ước lượng được ứng với dữ liệu thứ *i*.

Hàm lỗi của mạng nơ-ron cũng tương tự như vậy, chỉ khác là đầu ra của mạng nơ-ron có thể có nhiều nút nên khi tính đầu ra ta cũng cần phải tính cho từng nút ra đó. Giả sử số nút ra là *K* và y_k là đầu ra thực tế của nút thứ *k*, còn σ_k là đầu ra ước lượng được cho nút thứ *k* tương ứng. Khi đó, công thức tính hàm lỗi sẽ thành:

$$J(\mathbb{w}) = -\frac{1}{m}\sum_{i=1}^{m}\sum_{k=1}^{K}(y_k^{(i)}\log\left(\sigma_k^{(i)}\right) + \left(1 - y_k^{(i)}\right)\log\left(1 - \sigma_k^{(i)}\right))$$

Lưu ý rằng, các tham số lúc này không còn đơn thuần là một ma trận nữa mà là một tập của tất cả các ma trận tham số của tất cả các tầng mạng nên ta có thể biểu diễn nó dưới dạng tập hợp **W**. Để tối ưu hàm lỗi ta vẫn sử dụng các phương pháp đạo hàm như đã đề cập ở các bài viết trước. Nhưng việc tính đạo hàm lúc này không đơn thuần như logistic regression, bởi để ước lượng được đầu ra ta phải trải qua quá trình lan truyền tiến. Tức là để tính được σ_k ta cần một loạt các phép tính liên hợp nhau.

2.5. Lan truyền ngược và đạo hàm

Để tính đạo hàm của hàm lỗi $J(\mathbb{w})$ trong mạng nơ-ron, ta sử dụng một giải thuật đặc biệt là giải thuật lan truyền ngược (backpropagation) [3]. Nhờ có giải thuật được sáng tạo vào năm 1986 này mà mạng nơ-ron thực thi hiệu quả được và ứng dụng ngày một nhiều cho tới tận ngày này. Về cơ bản phương pháp này được dựa theo quy tắc chuỗi đạo hàm của hàm hợp và phép tính ngược đạo hàm để thu được

đạo hàm theo tất cả các tham số cùng lúc chỉ với 2 lần duyệt mạng. Giải thuật lan truyền ngược được thực hiện như sau:

1. *Lan truyền tiến*

   Lần lượt tính các $\boldsymbol{a}^{(l)}$ từ $l = 2 \rightarrow L$ theo công thức:

$$\boldsymbol{z}^{(l)} = \boldsymbol{W}^{(l)}.\boldsymbol{a}^{(l-1)}$$

$$\boldsymbol{a}^{(l)} = f(\boldsymbol{z}^{(l)})$$

   Trong đó, tầng vào $\boldsymbol{a}^{(l)}$ chính bằng giá trị vào của mạng **x**

2. *Tính đạo hàm theo z ở tầng ra*

$$\frac{\partial J}{\partial \boldsymbol{z}^{(L)}} = \frac{\partial J}{\partial \boldsymbol{a}^{(L)}} \frac{\partial \boldsymbol{a}^{(L)}}{\partial \boldsymbol{z}^{(L)}}$$

   với $\boldsymbol{a}^{(L)}$ và $\boldsymbol{z}^{(L)}$ tính được ở bước 1.

3. *Lan truyền ngược*

   Tính đạo hàm theo *z* ngược lại từ $l = (L-1) \rightarrow 2$ theo công thức:

$$\frac{\partial J}{\partial \boldsymbol{z}^{(l)}} = \frac{\partial J}{\partial \boldsymbol{z}^{(l+1)}} \frac{\partial \boldsymbol{z}^{(l+1)}}{\partial \boldsymbol{a}^{(l)}} \frac{\partial \boldsymbol{a}^{(l)}}{\partial \boldsymbol{z}^{(l)}}$$

$$= ((\boldsymbol{W}^{(l+1)})^T \frac{\partial J}{\partial \boldsymbol{z}^{(l+1)}}) \frac{\partial \boldsymbol{a}^{(l)}}{\partial \boldsymbol{z}^{(l)}}$$

   với $\boldsymbol{z}^{(l)}$ tính được ở bước 1 và $\frac{\partial J}{\partial \boldsymbol{z}^{(l+1)}}$ tính được ở vòng lặp ngay trước.

4. *Tính đạo hàm*

   Tính đạo hàm theo tham số *w* bằng công thức:

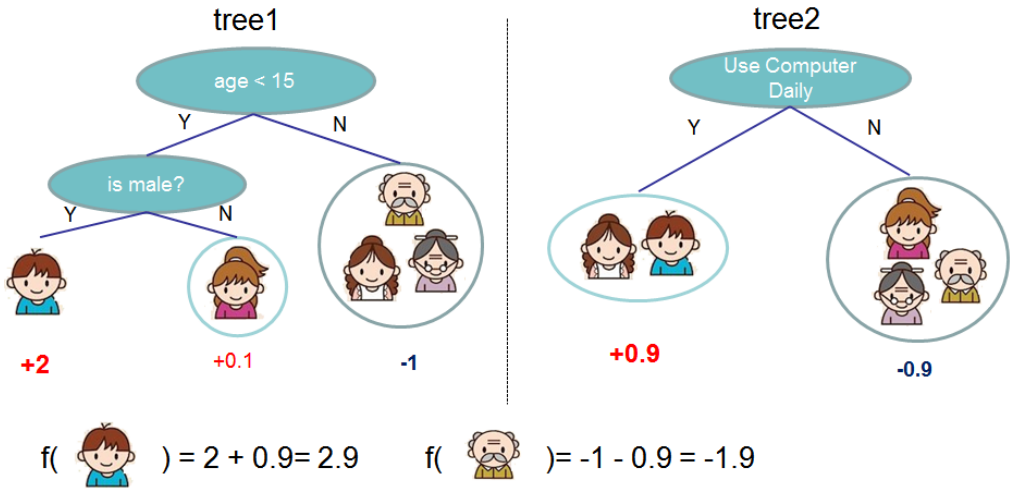
$$\frac{\partial J}{\partial \boldsymbol{W}^{(l)}} = \frac{\partial J}{\partial \boldsymbol{z}^{(l)}} \frac{\partial \boldsymbol{z}^{(l)}}{\partial \boldsymbol{W}^{(l)}}$$

$$= \frac{\partial J}{\partial \boldsymbol{z}^{(l)}} (\boldsymbol{a}^{(l-1)})^T$$

   với $\boldsymbol{a}^{(l-1)}$ tính được ở bước 1 và $\frac{\partial J}{\partial \boldsymbol{z}^{(l)}}$ tính được ở bước 3.

3. MÔ HÌNH XGBOOST

XGBoost là viết tắt của Extreme Gradient Boosting [4]. Đây là thuật toán state-of-the-art nhằm giải quyết bài toán supervised learning cho độ chính xác khá cao bên cạnh các mô hình Deep learning khác. Ý tưởng cơ bản của XGBoost là giải thuật Decision Tree (cây quyết định), áp dụng các kỹ thuật để kết hợp các bộ phân lớp cây quyết định yếu, làm mịn Training Loss (sai số khi huấn luyện) và Regularization (chuẩn hóa sai số, hệ số và số biến).



Hình 7. Ý tưởng cơ bản của XGBoost

Nếu Deep Learning chỉ nhận đầu vào là dữ liệu thô dạng numerical (thường phải chuyển đổi sang n-vector trong không gian số thực) thì XGBoost nhận đầu vào là dữ liệu dạng bảng với mọi kích thước và dạng dữ liệu bao, gồm cả dữ liệu nhị phân và dữ liệu đa lớp. Dữ liệu đa lớp thường được tìm thấy nhiều hơn trong thực tế.

Bên cạnh đó, XGboost có tốc độ huấn luyện khá nhanh, có khả năng mở rộng để tính toán song song trên nhiều server, có thể tăng tốc bằng cách sử dụng GPU, nhờ vậy mà Big Data không phải là vấn đề của mô hình này. Vì thế, XGBoost hiện nay đang rất được ưa chuộng, nhất là trong các cuộc thi về Khoa học dữ liệu như Kaggle.

4. THỰC NHIỆM

4.1. Dữ liệu huấn luyện

Dữ liệu sử dụng trong báo cáo này được thu thập từ trang báo điện tử Dân Trí, gồm 5303 bài báo với 5 chủ đề: Sức khoẻ (1038), Kinh doanh (1065), Pháp luật

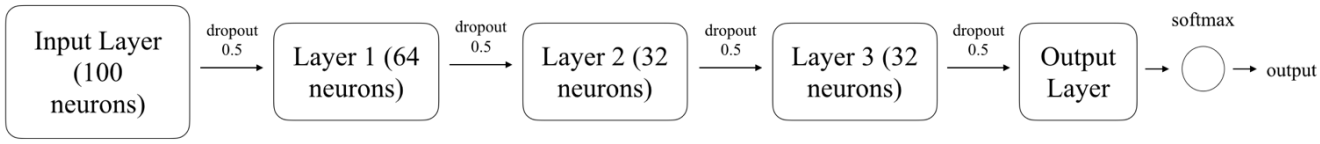
(1042), Thể thao (1102), và Xe (1056). Sau khi thu thập, dữ liệu được tiền xử lý, làm sạch và vector hoá (trong báo cáo giữa kì). Mỗi điểm dữ liệu của văn bản *v* gồm có:

- Vector 100 chiều, là đặc trưng biểu diễn phân bố chủ đề ẩn cho *v* (sử dụng mô hình LDA để học, được đề cập trong báo cáo giữa kì)
- Nhãn tương ứng: Sức khoẻ (0), Kinh doanh (1), Pháp luật (2), Thể thao (3), Xe (4). Trong khi thu thập dữ liệu, nhãn của tin tức được lưu lại cùng văn bản text.

Sau đó, dữ liệu được tách thành các tập con riêng biệt: tập huấn luyện và tập test (tỉ lệ 80%, 20%). Tập huấn luyện sẽ được sử dụng để huấn luyện mô hình (mạng nơ-ron hoặc XGBoost), sau đó, độ chính xác được đánh giá trên tập test, được đo bằng số lượng dự đoán đúng trên tổng số bài báo trong tập test.

4.2. Mô hình mạng nơ-ron nhân tạo

Để thực hiện phân loại tin tức, em thiết kế mạng nơ-ron gồm 5 tầng, gồm 1 tầng input, 1 tầng output và 3 tầng ẩn, hàm kích hoạt sử dụng ở tầng input và tầng ẩn là ReLu, hàm kích hoạt sử dụng ở tầng cuối là Softmax.



Hình 8. Kiến trúc mạng nơ-ron được thiết kế

Các tham số sử dụng:

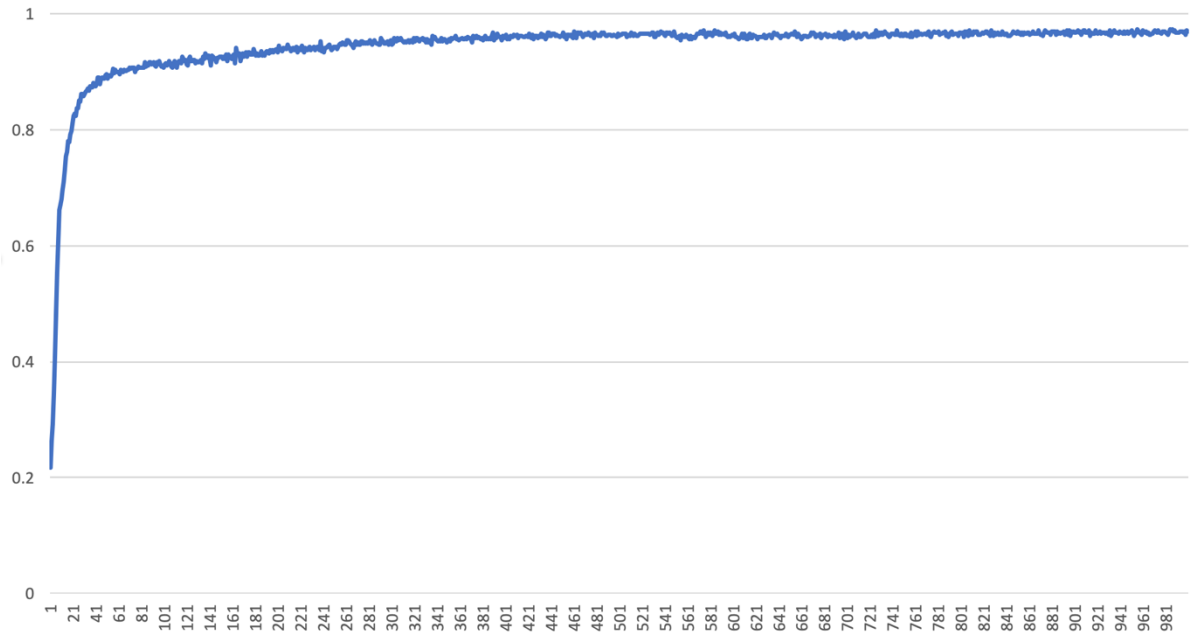
- Hàm mất mát: categorical crossentropy
- Accuracy metric: accuracy
- Optimizer: Adam
- Learning rate: 0.001
- Epochs: 1000
- Batchsize: 256

Để thực thi chương trình, chạy lệnh: `python ann_classifier.py`

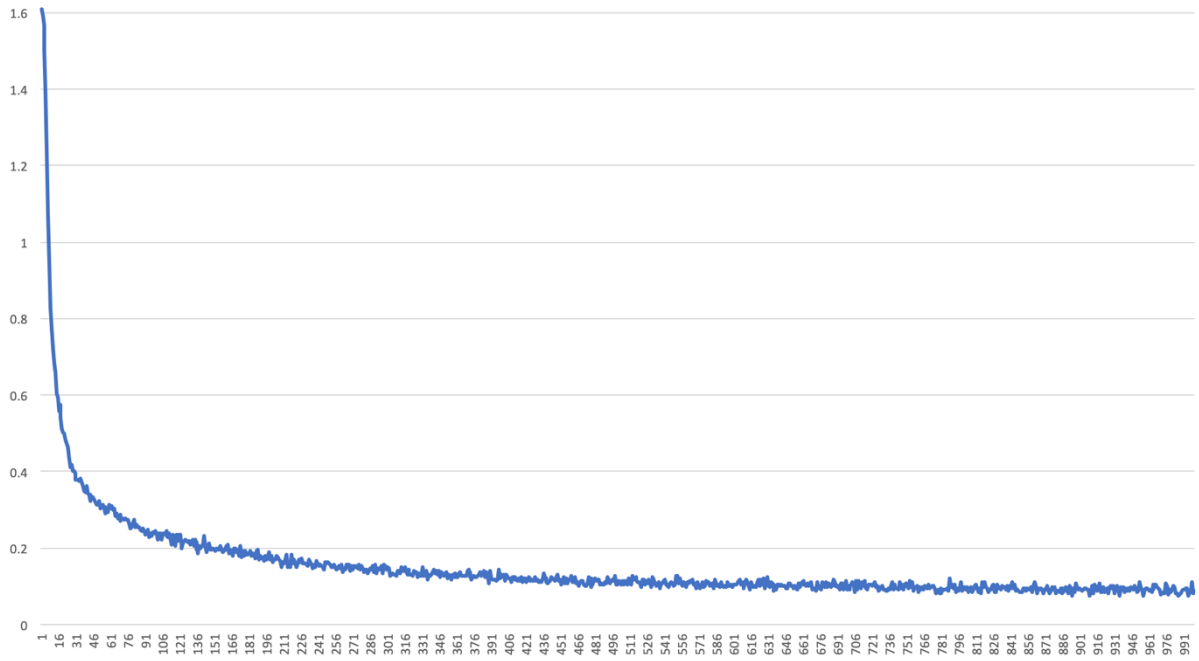
```
import time
import numpy as np
import pandas as pd
from keras import layers
from keras import models
from keras import optimizers
from keras.utils import np_utils
from keras.wrappers.scikit_learn import KerasClassifier
from sklearn.model_selection import KFold
from sklearn.model_selection import cross_val_score
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import LabelEncoder

seed, dim, data_dir = 13, 100, 'data/'
np.random.seed(seed)

def encode(y):
    encoder = LabelEncoder()
    encoder.fit(y)
    encoded_Y = encoder.transform(y)
    return np_utils.to_categorical(encoded_Y)

def load(path, dim):
    print('Loading data', path, '...')
    data_set = pd.read_csv(path, header=None).values
    x, y = data_set[:, 0:dim].astype(float), data_set[:, dim]
    return x, encode(y)

def evaluate(model, seed, x, y, ep, bz):
    print('+ Evaluating by K-fold validation ... ', end='',
flush=True)
    estimator = KerasClassifier(build_fn=model, epochs=ep,
batch_size=bz, verbose=0)
    estimator.fit(x, y)
    results = cross_val_score(estimator, x, y,
cv=KFold(n_splits=5, shuffle=True, random_state=seed))
    print('%.2f%%' % (results.mean() * 100))

def create_model():
    model = models.Sequential([
        layers.Dense(dim, input_dim=dim, activation='relu'),
        layers.Dropout(0.5),
        layers.Dense(64, activation='relu'),
        layers.Dropout(0.5),
        layers.Dense(32, activation='relu'),
        layers.Dropout(0.5),
        layers.Dense(16, activation='relu'),
        layers.Dropout(0.5),
        layers.Dense(5, activation='softmax')
```

```
    ])
    adam = optimizers.Adam(lr=0.001)
    model.compile(loss='categorical_crossentropy',
optimizer=adam, metrics=['accuracy'])
    return model

x, y = load(data_dir + "data.csv", dim)
x_train, x_test, y_train, y_test = train_test_split(x, y,
test_size=0.2, random_state=seed)

print('+ Training ANN model ...')
start_time = time.time()
ep, bz = 1000, 256
model = create_model()
model.fit(x_train, y_train, epochs=ep, batch_size=bz, verbose=2)
print('+ Training time: ', time.time() - start_time, 's')
evaluate(create_model, seed, x, y, ep, bz)

prediction = model.predict(x_test)
count_acc = 0
for p_index in range(len(prediction)):
    if prediction[p_index].argmax() == np.argmax(y_test[p_index]):
        count_acc = count_acc + 1
print('-> Accuracy in test data: ', count_acc / len(prediction))
```

Hình 9. Một đoạn mã nguồn thực hiện việc huấn luyện và phân loại sử dụng mô hình mạng nơ-ron nhân tạo



Hình 10. Độ chính xác trên tập huấn luyện sau mỗi vòng lặp



Hình 11. Giá trị hàm mất mát sau mỗi vòng lặp

Hình 10. và 11. mô tả độ chính xác trên tập huấn luyện và giá trị hàm mất mát sau mỗi vòng lặp. Về cơ bản, giá trị hàm mất mát sẽ giảm sau mỗi vòng lặp, và độ chính xác sẽ tăng dần. Tuy nhiên nếu huấn luyện quá mức, mô hình sẽ bị overfitting. Sau khi chạy thuật toán với 1000 vòng lặp, giá trị của hàm mất mát là 0.0869 và giá trị của độ chính xác đạt được là 96.96% trên tập huấn luyện.

4.3. Mô hình XGBoost

Để sử dụng mô hình XGBoost, em dùng thư viện xgboost kết hợp với thư viện Scikit-learn [5] trên Python. Để thực thi chương trình, chạy lệnh: `python xgboost_classifier.py`

```
import time
import numpy as np
from sklearn.metrics import accuracy_score
from sklearn.model_selection import train_test_split
from xgboost import XGBClassifier

def evaluate(data, test_size):
    dim = len(data[0]) - 1
    x, y = data[:, 0:dim], data[:, dim]
    sum_score = 0.0
    x_train, x_test, y_train, y_test = train_test_split(x, y,
test_size=test_size, random_state=fold)
    model = XGBClassifier()
    model.fit(x_train, y_train)
```

```
    predictions = [round(p) for p in model.predict(x_test)]
    sum_score += accuracy_score(y_test, predictions)
    acc = sum_score / fold
    print('-> Accuracy on test data:', acc)

print('+ Evaluation using XGBoost ...')
start_time = time.time()
evaluate(np.loadtxt('data/data.csv', delimiter=','), 0.2)
print('+ Training time: ', time.time() - start_time, 's')
```

Hình 12. Một đoạn mã nguồn thực hiện việc huấn luyện và phân loại, sử dụng XGBoost

4.4. Kết quả

Trong phạm vi báo cáo, em tiến hành thực nghiệm 2 phương pháp này trên 5303 bài báo, chạy trên máy tính macOS, CPU Core i5 (2.9Ghz), 8GB RAM. Kết quả cho thấy, XGBoost chạy rất nhanh, thời gian huấn luyện dữ liệu khoảng 16s, bằng 1/5 so với mạng nơ-ron nhân tạo, tuy nhiên độ chính xác lại thấp hơn 0.57%. Trên thực tế, với số lượng dữ liệu khổng lồ (hàng triệu bản ghi), thời gian huấn luyện sẽ lâu hơn rất nhiều.

Mô hình	Mạng nơ-ron nhân tạo	XGBoost
Thời gian huấn luyện	81.73 s	**15.86 s**
Độ chính xác (acc)	**95.57%**	95.00%

Hình 13. Kết quả thực nghiệm và so sánh giữa 2 phương pháp

Do thời gian hạn chế, dữ liệu thu thập chưa quá nhiều (chỉ hơn 5300 bài báo), nên độ chính xác chưa thực sự ấn tượng. Tuy nhiên, kết quả 95.57% là khá tốt. Khi dữ liệu lớn, mô hình được huấn luyện trên hàng triệu bài báo trở lên, độ chính xác sẽ được cải thiện hơn rất nhiều.

5. TỔNG KẾT

5.1. Kết quả đạt được

Báo cáo đã áp dụng thành công 2 mô hình học máy phổ biến, là mạng nơ-ron nhân tạo, và xgboost cho bài toán phân loại tin tức tiếng Việt, sử dụng dữ liệu

đã thu thập và tiền xử lý được từ báo cáo giữa kì, gồm 5303 bài báo từ báo điện tử Dân Trí, kết quả thu được khá tốt với độ chính xác tốt nhất 95.57%.

Kết quả cho thấy, cách biểu diễn văn bản bằng phân bố chủ đề ẩn là khả quan, cho ra kết quả khá tốt với số lượng dữ liệu huấn luyện không nhiều.

5.2. Định hướng tiếp theo

Trong tương lai, em dự định sẽ có thêm 1 vài cải tiến cũng như bổ sung thêm dữ liệu cần thiết:

- Tiếp tục thu thập dữ liệu từ báo điện tử Dân Trí, và một số đầu báo uy tín khác để làm phong phú thêm tập dữ liệu
- Thử một số mô hình học máy mới, cũng như thay đổi tham số các mô hình cũ và thử nghiệm để phù hợp với dữ liệu hơn.

TÀI LIỆU THAM KHẢO

1. Rosenblatt, Frank. "The perceptron: a probabilistic model for information storage and organization in the brain." Psychological review 65.6 (1958): 386.
2. Gardner, Matt W., and S. R. Dorling. "Artificial neural networks (the multilayer perceptron)—a review of applications in the atmospheric sciences." Atmospheric environment 32.14-15 (1998): 2627-2636.
3. Plaut, David C. "Experiments on Learning by Back Propagation." (1986).
4. Chen, Tianqi, and Carlos Guestrin. "Xgboost: A scalable tree boosting system." Proceedings of the 22nd acm sigkdd international conference on knowledge discovery and data mining. ACM, 2016.
5. Pedregosa, Fabian, et al. "Scikit-learn: Machine learning in Python." Journal of machine learning research 12.Oct (2011): 2825-2830.