MATH4828B Machine Learning for Natural Language Processing

Project 2 – Sentiment Analysis

Report

Name: TSE, Ho Nam
Student ID: 20423612
Kaggle Team Name: 20423612
Kaggle Username: Peter Tse (`mcreng`)
Ranking (Score): 7th (0.64506)

Abstract

This report discusses the methodology used to train the neural network used in participating the Kaggle competition. In short, `Jupyter Notebook` was used to run the codes and they were mainly written with help of the libraries `numpy`, `sklearn` and `nltk`. Before the data were fed into the neural network, preprocessing such as tokenization, words removal, n-grams and tfidf were use. Word lemmatization was also attempted but it did not increase the accuracy sufficiently and it cost much longer processing time so it was no longer used. The neural network used was Multilayer Layer Perceptron with Xaiver initialization, cross-entropy loss and Adam update rule. Its regularization coefficient, hidden layer size and activation function were considered as hyperparameters and k-fold cross validation was used to consider which combination of the hyperparameters are the best. At last, the neural network with hidden size $(50, 50,)$, regularization coefficient of ~ 0.03 and logistic activation function was used to predict the test case. This network scored 0.64506 in the competition, ranking seventh.

1 Preprocessing

1.1 Tokenization

Before any preprocessing, the documents are read and tokenized through `nltk` tokenizer.

1.2 Stop Word Removal

As the documents are read, some stop words are removed from the documents. These stop words contain commonly used words that carry insignificant meanings in the document. They are a composition of `nltk` stop words and a self-made list of

```
["'d", "'ll", "'re", "'s", "'ve", "`'", "could", \
"might", "must", "n't", "need", "sha", "wo", "would"].
```

These additional stop words are used because the `nltk` stop words could not remove these words and these words appear to be insignificant in meaning as well.

I also added a check of `len(word) > 1` which remove all words with length of 1. These words are not meaningful and sometimes might be created through a typo.

> we didnt eat expensive.the staff ok. just not special 140 **e** excluding tax no breakfast. confirmation said internet facilities none.

1.3 Punctuation Removal

The punctuation marks in the documents are removed since they do not give much insights towards the sentiment analysis. They are removed by considering

`list(string.punctuation).`

1.4 Word Lowercase

Cases seems to be not useful in the document since by briefing looking at the document, the words are mainly in lowercase.

> cozy stay rainy city. husband spent 7 nights **monaco** early **january** 2008. business trip chance come ride. we booked **monte carlo** suite proved comfortable longish stay.

The bolded words in the quote above should be in uppercase when written in standard English, but since they are not, it seems we cannot gather useful information by looking at cases. Since it is not sure if there are some documents with words in uppercase, to prevent the same words in different cases being counted separately, all the words are set to lowercase.

1.5 Number Removal

It is suspected that this task is in fact about hotel reviews, that the documents given are the reviews, and the categorical outputs are the number of stars given in this review. It might be useful to look for numbers in the text since reviewers might put 'Great hotel! 5 stars!' in the reviews and we hence can directly get the category corresponding to the review. However, there are too many other unrelated occurrences of numbers which makes it difficult to check whether the number is about the number of stars the reviewers give. Also, sometimes we have pieces of text like

> lawn mowers 5 a.m. given room golf course. woke 5 morning roar lawn mowers. 5 numerous fairways. wear tear rooms facilities obvious. beautiful old hotel **not 3 stars**. bring earplugs.

which might fool the network that this review corresponds to 3 stars (which in fact corresponds to 1 star) if we consider the numbers. Therefore, numbers are removed to reduce the ambiguity caused by the presence of numbers. The removal of numbers is done by `word.isnumeric()`.

1.6 Lemmatization

Lemmatization is the process of change the word into its corresponding common base form, which is achieved by using morphological analysis of words. For example[1], we have

Before		**After**
the	$\rightarrow$	the
boy's	$\rightarrow$	boy
cars	$\rightarrow$	car
are	$\rightarrow$	be
different	$\rightarrow$	differ
colors	$\rightarrow$	color

This might help increase the accuracy since with lemmatization, words with same base form can be grouped together. This is achieved by using `WordNetLemmatizer` in the `nltk` library. Since this lemmatizer requires the part of speech tag of each word as well, we also use `nltk.pos_tag(tokens)`.

Lemmatization is **not** used in the final models though, since it requires a long time to compute, and the accuracy does not increase significantly after lemmatization.

1.7 n-gram

In our data preprocessing model, each word is captured separately which means phrases like `Hong Kong` cannot be captured. Therefore, apart from considering word by word, we also consider n-gram, which means we also consider contiguous sequence of n words in the documents as well. For example, a sentence of

"I love Hong Kong."

can have n-grams of following.

n-gram	Examples
1-gram	I, love, Hong, Kong
2-gram	I love, love Hong, Hong Kong
3-gram	I love Hong, love Hong Kong
4-gram	I love Hong Kong

In particular, our model uses n-grams with $n = 1$ to 5.

1.8 Head/Tail Word Removal

Consider the example in n-gram above, some phrases like "love Hong" does not make sense. Some phrases that appear rarely may also not contribute to the meaning of the

[1]Example adapted from Stanford NLP: `https://nlp.stanford.edu/IR-book/html/htmledition/stemming-and-lemmatization-1.html`

documents. These can be regarded as the tail phrases and can be removed. In our model, we remove phrases that do not appear more than 3 times in total.

Some words may appear very often, for example in our dataset, `hotel` and `room` appear very frequently. They may also not help classifying the documents since they are ubiquitous. These words are regarded as head words and can also be removed. In our model, phrases that appear in more than 99.9% of all documents are removed.

1.9 TFIDF

TFIDF is a weighting applied to a document-vocabulary matrix. Usually when as a word appears more often in a single document, this word might be important in that document so we might want to weigh it more. However, if a word appears frequently across all documents, this might mean this word is not useful in distinguishing the documents. Therefore, a weighting of the product of term frequency (tf) and inverse document frequency (idf) can be used to apply to the document-vocabulary matrix.

In our model, after processing all documents, the document-vocabulary matrix uses the following tf and idf functions.

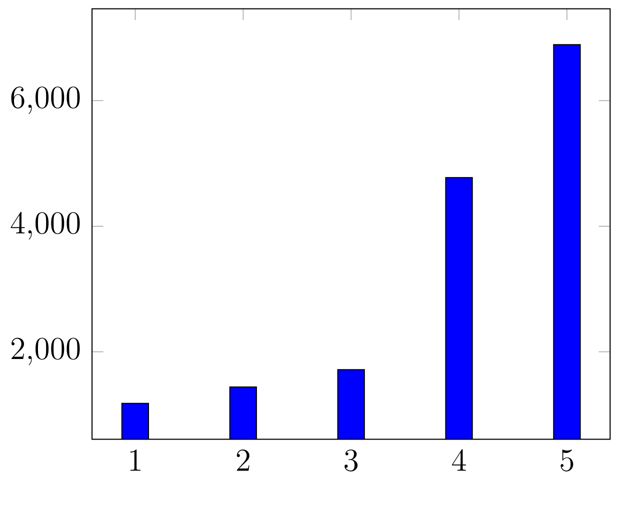
$$\mathrm{tf}(t,d) = f_{t,d}, \qquad \mathrm{idf}(t) = 1,$$

where $f_{t,d}$ is the frequency of term t in document d. This means we just use the number of occurrences of each word in each document as the entry of the data matrix. Other choices of idf were tried such as

$$\mathrm{idf}(t) = \log \frac{N}{n_t},$$

where N is the number of documents and n_t is the number of document term t appears in, but they do not seem to give better results. One hypothesis is that the training data is skewed towards 5-star reviews which after applying idf, classifications with positive reviews might perform worse since the positive words are treated with less significance.

Distribution of training data according to label



1.10 Saving Processed Data

To avoid preprocessing the same set of data every time a model is trained, the preprocessed data are stored as `.npy` files, a compressed binary file for `numpy` arrays. It is achieved by `np.save(filename, obj)` where `filename` is the location to store the files and `obj` is the array to be stored.

2 Neural Network

After preprocessing the data, they are fed into a Multilayer Perceptron neural network initialized with uniform distribution with cross-entropy loss and Adam as update rule. Logistic activation function is used in the hidden layers and Softmax is used in the output layer in the final model.

2.1 Multilayer Perceptron

Definition 1. *A neuron can be considered as a function that takes n inputs, m outputs and an activation function $\sigma : \mathbb{R}^m \to \mathbb{R}^m$ that has the structure*

$$\mathbf{y} = \sigma(\mathbf{W}\mathbf{x} + \mathbf{b}),$$

where $\mathbf{y} \in \mathbb{R}^m$ *is the output,*
$\mathbf{W} \in \mathbb{R}^{m\times n}$ *is the weight matrix,*
$\mathbf{b} \in \mathbb{R}^m$ *is the bias, and*
$\mathbf{x} \in \mathbb{R}^n$ *is the input.*

Remark. This definition of neuron is actually a generalization of a perceptron, that a perceptron only takes in 1 input and gives out 1 output, and its activation function is defined.

With the definition of a neuron, we can then define multilayer perceptron.

Definition 2. *Multilayer Perceptron (MLP) is a feed-forward network that consists of multiple layers: Input layer, Hidden layers and Output layer. Except the input layer, every layer has multiple neurons with nonlinear activation functions. Neurons in one layer take inputs from all neurons in their previous layer, and output to all neurons to their next layer.*

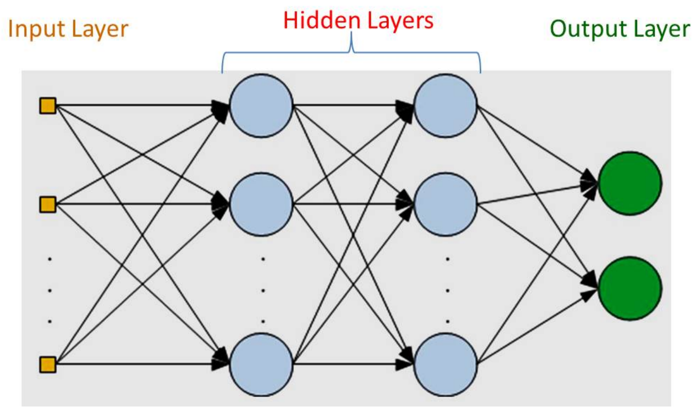
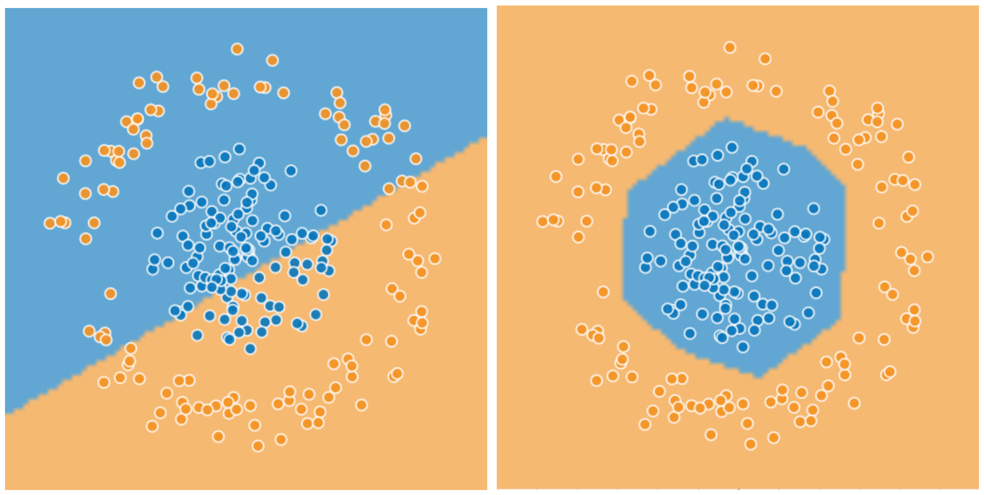


Figure 1: Multilayer Perceptron Structure

This network usually outperforms other classifiers such as Naïve Bayes, Perceptron and Logistic Regression because MLP can handle data that is not linearly separable.

This is because even the term $\mathbf{Wx} + \mathbf{b}$ is a linear function, the activation function is not linear and thus, the network can model nonlinear functions. Figure 2 shows experimental results of different classifiers on nonlinear dataset. Since it is generally not true that the training data is linearly separable, MLP is used in this case.



(a) Perceptron on Nonlinear Dataset (b) MLP on Nonlinear Dataset

Figure 2: Classifers on Nonlinear Dataset[2]

For each forward pass of the network, we have some input $\mathbf{x}$ and it corresponds to a class of 1 to 5. As $\mathbf{x}$ is passed through the network, we would expect the output vector has the highest value in the i-th entry where i is the corresponding class of $\mathbf{x}$.

2.2 Initialization

Before training, the weights and biases of the neurons need to be initialized. In our model, the initialization suggested by Xaiver and Yoshua[3] is used for the weights and the biases are initialized as 0. Xaiver et al. suggests to initialize the weight W with

$$W \sim U\left[-\sqrt{\frac{6}{n+m}}, \sqrt{\frac{6}{n+m}}\right],$$

where n and m are the number of nodes in the previous and next layer respectively. The randomness is necessary since otherwise all neurons would have exactly the same behaviour. The weight is normalized by the number of nodes in neighbouring layers so that during back-propagation, the gradient would not explode or vanish over a deep network (since in each back-propagation step, the gradient w.r.t. the input is multiplied by the weight once).

2.3 Activation Functions

In the design of the network, two activation functions for hidden layers are considered and they are ReLU and Logistic function.

[2]Images generated from Tensorflow Playground: `https://playground.tensorflow.org/`
[3]`http://proceedings.mlr.press/v9/glorot10a/glorot10a.pdf`

Definition 3. *The rectified linear unit (ReLU) is an activation function $f : \mathbb{R} \to \mathbb{R}$ applied elementwise to an output, with*

$$f(x) = \max\{0, x\}.$$

Definition 4. *The logistic function is an activation function $f : \mathbb{R} \to \mathbb{R}$ applied elementwise to an output, with*

$$f(x) = \frac{1}{1 + \exp(-x)}.$$

These two functions are not linear, which helps the network to model nonlinear functions. Logistic function is not recommended in training deep neural network (which will be explained in subseqent section), however, since the network is around 2 to 3 layers deep, logistic function works alright. There is also an activation function for the output, which is the softmax function.

Definition 5. *The softmax function is an activation function $\sigma : \mathbb{R}^m \to \mathbb{R}^m$ for the output layer, with*

$$\sigma(\mathbf{y})_j = \frac{\exp(y_j)}{\sum_{i=1}^m \exp(y_i)}.$$

The softmax function can be considered as finding the probabilities of the input for each class. This is because the reduce sum of a softmax output is

$$\sum_{j=1}^m \frac{\exp(y_j)}{\sum_{i=1}^m \exp(y_i)} = 1.$$

The softmax function also allows us to compare the output of the network and the ground truth easily. Suppose for the input $\mathbf{x}$, its corresponding class if k. We then can construct an one-hot vector of class k as $\mathbf{e}_k = \begin{bmatrix} 0 & 0 & \cdots & 1 & 0 & \cdots & 0 \end{bmatrix}^\intercal$ where the 1 is placed at the k-th entry. Thus, we can compare the softmax output and the one-hot vector using some loss functions.

2.4 Loss Function

A naïve way to compare the softmax output and the one-hot ground truth vector is to use the 2-norm squared distance, but in reality cross-entropy loss works well in classification problems. In both cases, we wish to minimize the loss so as to get greater accuracy.

Definition 6. *The 2-norm squared distance of two vectors $\mathbf{u}, \mathbf{v} \in R^m$ is defined as*

$$\|\mathbf{u} - \mathbf{v}\|_2^2 = \sum_{i=1}^m (u_i - v_i)^2.$$

Definition 7. *The cross entropy between two discrete probability distributions p, q is defined as*

$$H(p, q) = -\sum_x p(x) \log q(x).$$

The 2-norm squared distance usually does not work as well because it is harder to optimize[4]. Therefore, in classification problems, cross-entropy is chosen as the loss function.

The cross-entropy is a concept in probability where it is used to compare the similarity of two probability distributions. Since we can regard the softmax output and the one-hot vector as probabilities, we can compare the similarity of these two vectors (or probability distributions) using cross entropy. The lower the value, the more similar they are. In our case, let $\mathbf{e}_k$ be the one-hot ground truth vector and $\mathbf{y}$ be the softmax output, we compute the loss as $H(\mathbf{e}_k, \mathbf{y})$.

In addition to the cross entropy loss, we also place a regularization constraint on the weights of the neurons.

Definition 8. *The L2 regularization on a weight* $\mathbf{W} \in \mathbb{R}^{n\times m}$ *is*

$$\frac{1}{2}\lambda \left\|\mathbf{W}\right\|_2^2 = \frac{1}{2}\lambda \sum_{i=1}^{n}\sum_{j=1}^{m} w_{ij}^2.$$

Since in neurons, $\mathbf{W}$ is multiplied with the input $\mathbf{x}$, without the regularization, $\mathbf{W}$ may be chosen that it heavily relies on certain inputs and disregard other inputs. This might lead to overfitting since the weights are chosen to make the training set work the best. Regularization encourages the weights to be diffuse and can help with overfitting. λ is a hyperparameter in the model that sets how heavily should the weights be regularized. The sum between the cross-entropy loss and the regularization is the final loss that we consider in the model.

2.5 Optimization

After the forward pass, we compute the loss and afterwards we wish to modify the weights and biases in the neurons so the loss is reduced. In general, let l be the loss, and p be some parameter in the network, we can minimize l by modifying p as

$$p \leftarrow p - \alpha \nabla_p l.$$

When the loss l is taken over all samples, this is known as the gradient descent algorithm. However, this is very costly since the number of samples can be very large, so one variation is the stochastic gradient descent (SGD), which the loss l is taken over some minibatch of samples, instead of all. In this model, we use another very common stochastic update rule, known as Adam, since Adam sometimes gives a better result. The update rule of Adam is as follows, with g being the upstream gradient, p being the parameter to be updated and α, β_1, β_2 being some hyperparameters. $\varepsilon = 10^{-8}$ is placed to prevent division by zero.

Step 1. $m \leftarrow \beta_1 m + (1-\beta_1)g$

Step 2. $v \leftarrow \beta_2 v + (1-\beta_2)g^2$

Step 3. $m \leftarrow \dfrac{m}{1-\beta_1^t}$

[4]From Stanford CS231n Notes `http://cs231n.github.io/neural-networks-2/#losses`

STEP 4. $v \leftarrow \dfrac{v}{1-\beta_2^t}$

STEP 5. $p \leftarrow p - \alpha \cdot \dfrac{m}{\sqrt{v}+\varepsilon}$

Intuitively, m and v are the running averages of the first and second moment of the gradient. We can think of m as an estimate of the update direction with previous updates considered, and v as an estimate of the size of the current update. We then wish to update p in the directon of m but we wish to normalize the update step by $\sqrt{v}$ so each step would be around equal size. In our model, $\beta_1 = 0.9$ and $\beta_2 = 0.99$, and α, the learning rate, equals 0.001.

We then train the network and runs through multiple epochs until the loss converges.

3 Hyperparameters Tuning

In our model, there are multiple hyperparameters.

Symbol	Meaning
α	Learning rate
β_1	First moment ratio
β_2	Second moment ratio
λ	Regularization coefficient

In particular, $\beta_1 = 0.9$ and $\beta_2 = 0.99$ are widely used and there is no need to tune these two hyperparameters. The learning rate is usually predetermined and in our model, it is chosen to be 0.001. The only hyperparameter left for us to tune is λ.

Apart from these hyperparameters, the structure of the model can also vary. In particular, the number of layers, number of neurons in each layer and the activation function used. Since training the model once does not take that long, these properties of the model are also considered as hyperparameters. In our model, the hidden layer size of the model can be $(5, 5), (10, 10), (5, 5, 5), (10, 10, 10), (25, 25), (50, 50)$. The activation functions can be ReLU or logistic.

To find the best set of hyperparameters, random search with cross validation ($k = 5$) is used. There are two ways to find the best hyperparameters, one is grid search and one is random search. The first is to choose hyperparameters with a grid layout to evaluate the model at these hyperparameters. The second is to choose hyperparameters randomly instead.

In the example of Figure 3, both uses 9 points to search for the hyperparameters. Given some hyperparameters are more important than some others, the global best set of hyperparameters can be approximated by choosing the best settings of important hyperparameters. In grid search, each hyperparameter is only explored in 3 distinct values, but in random search, each is explored in 9 distinct values. This shows the random search may find the best hyperparameters more efficiently since it covers more distinct points, in particular, for those important hyperparameters. Then, we can find approximately the best important hyperparameters settings and achieve a good accuracy overall. Therefore, in our model, random search is used.

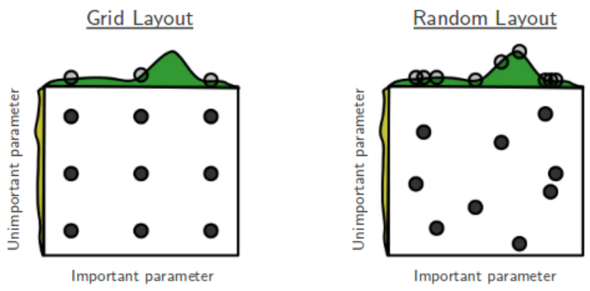


Figure 3: Grid Search vs Random Search

On the other hand, cross validation is a technique used in training that we separate the data we have into k equal-sized sets. Then, we pick $k-1$ of them as the training set and the remaining as the validation set. We use the training set to train and use the validation set to check the accuracy of the model on data that it have not seen. We then repeat the process k times with each time have a different validation set. This helps us to get a brief idea of how the model performs in average by considering the average validation accuracy.

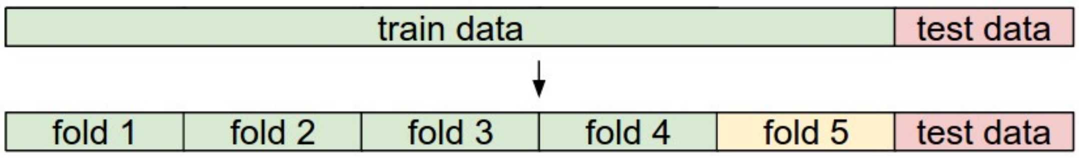


Figure 4: k-fold cross validation

Combining both, we ran random search for 100 times, each performing 5-fold cross validation. Then, we pick the model that performs the best as our final model. The performance of each of the model is included in the `Jupyter Notebook` file. To speed up the training process, early stopping is also used, which means when the model does not improve in several epochs, we stop the training of that model and move on to the next one.

After the tuning, the best model we have is a MLP with hidden size $(50, 50,)$ with logistic activation function. The regularization coefficient is about 0.03. In the competition, this model scores 0.64506, ranking seventh.

4 Implementation

In this model, the following libraries are used.

Name	Version	Description
`Python`	`3.6.6`	Language used
`nltk`	`3.3.0`	For text preprocessing
`numpy`	`1.15.3`	For handling data matrix
`pandas`	`0.23.4`	For handling `csv` files
`scikit-learn`	`0.20.0`	For machine learning
`ipycache`	`0.1.5dev`	For caching `Jupyter` output

In particular, since there are 100 models, each running 5 times, I trained my model on an Amazon Web Service instance. Normally `Jupyter Notebook` only works when the browser connecting to the notebook stays opened, but since the training took quite some time (930 minutes), I used the library `ipycache` which helps cache the output of a notebook cell into a pickle (`pkl`) file. Then, I can close the browser connection and after the cell finished running, I can get back the cached output by running the cell again. I also ran the training concurrently on the 4 cores of CPU to speed up the training time.

End of Report