COMP4650 Assignment 2 Answers

Jieli Zheng

u6579712

Question 1 A simple linear classifier

Generally I make three major changes on the classic Logistic Regression model.
The first major change I make in preprocessor is using SnowballStemmer with language=english. SnowballStemmer is a nltk package built-in stemmer that can transform any english words back to their stems which is easier for tokenizer to find the right tokens from the raw text.
The seconde major change I make is using tokenizer to WordPunctTokenizer instead of WhitespaceTokenizer. The WordPunctTokenizer can split sentence into a bunch of words and single punctunations. It is stronger than the original setting with WhitespaceTokenizer because WhitespaceTokenizer can't split punctunations from word. It definitely has different tokens like "word," "word." which should be the same meaning in human's perspective.
Last major change I make in CountVectorizer is setting the parameter max features to 5000. The advantage of extracting more features is that model can have larger observation on low frequency words with explicit tendency.
training accuracy: 0.9165333333333333
validation accuracy: 0.8768
testing accuracy: 0.8778

Question 2 Embedding based classifier

Code in Pycharm

Question 3 Tuning a pytorch model

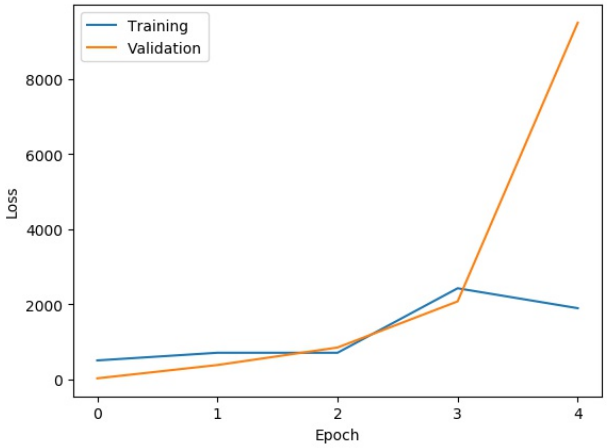
First setting:
Preprocessor: SnowballStemmer set language to english, WordPunctTokenizer
CountVectorizer: set stopwords to english, max features to 5000
FastText: hidden neurons set to 64
Optimizer: SGD set lr to 0.1, set momentum to 0.9
Epoch set to 5

Training accuracy: 0.5046419501304626
Validation accuracy: 0.4978678226470947
Testing accuracy: 0.4981499910354614

Generally, the default SGD Optimizer is not an expressive model with a single linear layer. accuracy around 0.5 is almost the same as a random choice generator. I think the main problem here is the Optimizer. **Second setting**:
Preprocessor: SnowballStemmer set language to english, WordPunctTokenizer
CountVectorizer: set stopwords to english, max features to 5000
FastText: hidden neurons set to 64
Optimizer: Adam set lr to 0.5
Epoch set to 5

training accuracy: 0.8862606883049011
validation accuracy: 0.755041778087616
testing accuracy: 0.7515999674797058

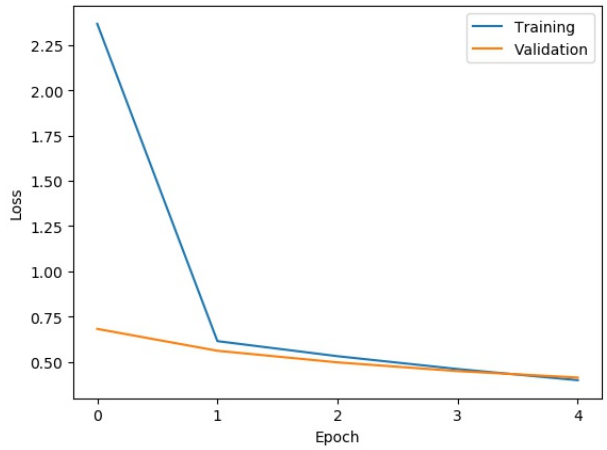
The second setting uses Adam Optimizer which is a more powerful optimizer than SGD. As a result, the final accuracy is much better than using SGD. However, I find that the validation curve is not very stable among several trials, because large learning rate causes overfitting problem and can only simulate training set instead of generally classifying all possible data.



Loss graph for Second setting by each epoch.

Third setting:
Preprocessor: SnowballStemmer set language to english, WordPunctTokenizer
CountVectorizer: set stopwords to english, max features to 5000
FastText: hidden neurons set to 64
Optimizer: Adam set lr to 0.001
Epoch set to 5

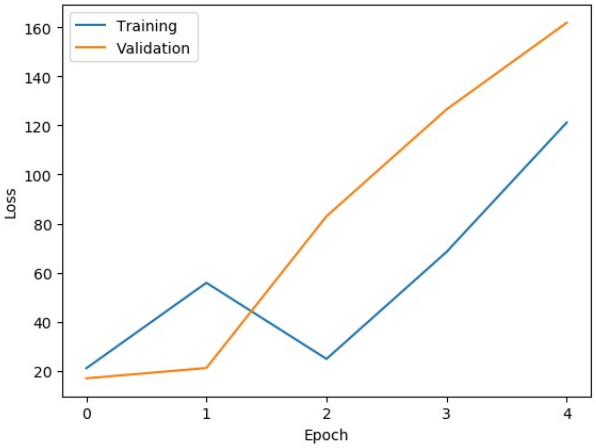
Training accuracy: 0.868869960308075
Validation accuracy: 0.8369536399841309
Testing accuracy: 0.8298999667167664



Loss graph for Third setting by each epoch.

The third setting uses Adam Optimizer and a fairly small learning rate to 0.001. In this setting, the validation process is stable during training, which means that smaller learning rate can prevent this model from overfitting problem and therefore it has better accuracy. However the model can't learn enough from training data due to small learning rate.
Forth setting:
Preprocessor: SnowballStemmer set language to english, WordPunctTokenizer
CountVectorizer: set stopwords to english, max features to 5000
FastText: hidden neurons set to 64
Optimizer: Adam set lr to 0.01
Epoch set to 5

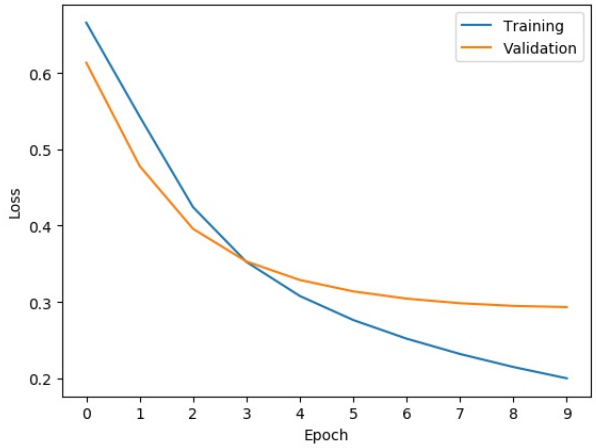
Training accuracy: 0.8751111030578613
Validation accuracy: 0.8643390536308289
Testing accuracy: 0.8671000003814697
The Forth setting I use is Adam Optimizer with learning rate to 0.1. Here the model still has a bit overfitting but we got a trade-off between overfitting and higher accuracy. Although small learning rate can prevent overfitting, the model may not



Loss graph for Forth setting by each epoch.

have good accuracy because the model has not learned enough features with small learning rate and few epochs.
Fifth setting:
Preprocessor: SnowballStemmer set language to english, WordPunctTokenizer
CountVectorizer: set stopwords to english, max features to 5000
FastText: hidden neurons set to 128
Optimizer: Adam set lr to 0.001
Epoch set to 10

training accuracy: 0.9328136444091797
validation accuracy: 0.8872379660606384
testing accuracy: 0.8869499564170837



Loss graph for Fifth setting by each epoch.

The final setting I use is to handle both overfitting and accuracy. Firstly, to learn more features from training data, I set more epochs and smaller learning rate to fit the data with more steps and make full use of sparse data in training set. Also, more hidden neurons can make the model more expressive. Finally, the validation accuracy as well as testing accuracy reach a good balance point. The main advantage of the fifth setting over the forth is that model has more generalization ability from the training data and also prevents overfitting problem.

Question 4 Comparison of models

LogisticRegression model performs a bit better if I take testing accuracy and training time into account. LogisticRegression is used for classify whether candidate belongs to a certain class, movie review problem is a binary classify problem and LogisticRegression is suitable for this problem. FastText is a text classification algorithm based on neural network that can be used both unsupervised and supervised learning. FastText here uses one linear layer as well as Vectorizer to extract features. I think there are several reasons for FastText not taking advantage over simple logistic regression. First of all, the

tendency of a movie review is relevant to the order of the words, which means the average aggregation may also confuse the model. For example,"The movie is not very good , but i still like it." should be positive, "The movie is very good , but i still do not like it ." should be negative, but average aggregation will take them similar and has worse classification. Secondly, FastText can easily faces overfitting after less than 10 epochs, which make it hard to train when the data set is small. LogisticRegression is faster and a bit better than FastText on testing accuracy, therefore I choose LogisticRegression.

Question 5 Implement Word2Vec

Code in Pycharm

Question 6 Compare pre-trained embeddings

Classifier without pre-train performs better. There are several reasons for pretrained classifier performing bad. First of all, the pretrained embedding definitely has negative effect on the initial model of movie review problem because model quickly converges into a local minimum for loss function. Secondly, pretrained model still faces overfitting problem. Thirdly, the pretraining process uses unlabelled text, which is an unsupervised learning process, and the embedding after pretraining may not be relevant to the attitude of a movie review. Therefore, the pretrained embeddings have negative effect on the Q6 classifier.
First setting:
epochs: 10
hidden neurons: 128
optimizer: Adam with lr set to 0.001
Results:
training accuracy: 0.7016000151634216
validation accuracy: 0.7015999555587769
testing accuracy: 0.7032999992370605

Other Three setting:
Second setting:
epochs: 5
hidden neurons: 32
optimizer: Adam with lr set to 0.001
Results:
training accuracy: 0.5785999894142151
validation accuracy: 0.585599958896637
testing accuracy: 0.5831999778747559
Third setting:
epochs: 5
hidden neurons: 32
optimizer: Adam with lr set to 0.001
Results:
training accuracy: 0.6610000133514404
validation accuracy: 0.6523000001907349
testing accuracy: 0.6656999588012695

Forth setting:
epochs: 5
hidden neurons: 32
optimizer: Adam with lr set to 0.1
Results:
training accuracy: 0.6171333193778992
validation accuracy: 0.5895999670028687
testing accuracy: 0.5947999954223633

Question 7 Further Modification

The main change I make is replacing the linear classifier in FastText with a multi-layer neural network. The multi-layer contains 3 linear layer as well as dropout layers. To give it non-linear features, I use Leaky-ReLU to maintain negative neurons. Results:
training accuracy: 0.9234333634376526

validation accuracy: 0.8883000016212463
testing accuracy: 0.8956999778747559