Iran University of Science and Technology
Deep Learning (Spring 2019)

Ad classification using the text description on Divar dataset

Mehrdad Nasser
Department of Computer Engineering
Iran University of Science
and Technology
`mehrdad_nasser@comp.iust.ac.ir`

Abstract

In this project, we use different classification techniques for the task of classifying advertisements in the Divar dataset. we first use a Support Vector Machine along with bag-of-words as features and set a baseline. We then use a simple Artificial Neural Network with word embeddings as features and study the effects of different settings for these word embeddings on the performance of the model. Next, we use Convolutional Neural Networks and experiment with various hyper-parameters and find the best setting for this model. We also use Recurrent Neural Networks and study the effects of applying different techniques to the last layer's hidden states. In the last part, we use a combination of CNN and RNN layers to see how the performance changes.

1 Introduction

Text classification is the task of assigning a label to a sentence or a group of sentences based on their features. There four main steps in text classification: Pre-processing, Feature extraction, Choosing a classifier, Evaluation. Pre-processing step mainly consists of tokenization and generally removing noise from the data. Noise could be numbers, certain punctuation marks, which all depends on the dataset. this step also depends on the language, for example, in English, word are usually transformed to lowercase while some languages don't have lowercase or uppercase. Feature extraction is the task of converting the text to a feature space suitable for using with classification models. These features could be n-grams in the text or word embeddings. Choosing the appropriate classifier is probably the most important step of the way. In the proposed method section we elaborate on the various methods that we used in our project. We first evaluate traditional methods like Support Vector Machines(SVM) and then use deep learning methods which have achieved surpassing results in almost every field of machine learning. The last step is choosing an appropriate evaluation metric which depends on the kind of task at hand.

2 Related work/Background

Before the reemerging of deep learning models, SVMs were the state-of-the-art models in text classification. In 2014, [1] proposed a CNN-based model for text classification that outperformed SVMs. This model used convolution and pooling layers to encode text to a vector and perform classification using that vector. [4] uses a heirarchical attention mechanism to perform document classification. This model uses attention for words and sentences seperately. This model is suitable for datasets whose instances are made up of several sentences. Since the dataset used in this project is comprised of mainly small sentences, we didn't explore that architecture.

3 Proposed method

There are four models that we use in this project: SVM, ANN, CNN, RNN and a combination of CNN and RNN. For SVM we use different combination of n-grams as features which we discuss in the next section. for the ANN architecture, we simply use the average of the word embeddings of the words in a sentence as the feature of that sentence. For the CNN model we use the architecture used in [1]. In this architecture, each sentence is represented by an $S \times d$ matrix where S is the length of the sentence and d is the dimension of the word embeddings. One-dimensional convolutional layers of length k and size N are used to extract features from the sentences. The one-dimensional convolution layer acts like a sliding window of size k on the sentence which means that every k consecutive words are convolved into a vector of size N. Features are then extracted using these one-dimensional convolutional layers and these features are then passed through a pooling layer and then are used as inputs for a fully-connected network to make predicitons. For the RNN method, we use a bi-directions Long Short-term memory (bi-LSTM) network. RNNs process the input sequentially which means they process each sentence one word at a time. This characteristic enables them to take in to account the order in which words appear in a sentence, something that CNNs lack. In last architecture, we make use of the feature extraction power of CNNs and the sequential nature of RNNs. We explore two approaches for this model: Using a CNN to extract features and then feed these features to an RNN, or using an RNN on the sentences and apply convolution to the hidden states of the last layer of the RNN. We compare the results of these two approaches in the next section.

4 Results

We use the Divar dataset which consists of 947653 ads grouped into 6 main categories: Vehicles, Electronic devices, Businesses, For the home, Personal, Leisure Hobbies. We used 236913 ads for test and 710740 for training while preserving the relative size of the classes. The best setting we found for pre-processing was to only do tokenization since removing punctuations or stemming or removing numbers didn't didn't improve the results. We use the accuracy on the test data as our metric. The results of our experiments for each are shown throughout this section.

4.1 SVM

For SVM, we use three sorts of features which are: Bag of uni-grams, Bag of uni-grams and bi-grams, Bag of uni-grams and bi-grams and tri-grams. We also used three sets of values for these features which are: Binary, Count, Tf-idf. Binary means that the value of a feature is 1 if the corresponding n-gram exists in the text and 0 otherwise. Count means that we assign the number of occurences of that feature in the text. Tf-idf means we use the Tf-idf of that feature in the text as the value of the feature. Having three sets of features and three sets of values, we ran nine experiments in total. The accuracies on test data for each setting can be seen in the table below.

	Binary	Count	Tf-idf
Uni-gram	86.9%	86.92%	87.46%
Uni-gram, Bi-gram	87.14%	87.19%	88.59%
Uni-gram, Bi-gram, Tri-gram	86.82%	86.81%	88.46%

We see that using uni-grams and bi-grams as feature along with Tf-idf yields the best accuracy on test data.

4.2 ANN

In this part, we used a simple ANN with 3 layers using the average of the word embeddings as the feature for each instance in the training set. We also used Dropout and set the dropout rate to 0.3 and also used Batch normalization. The most important factor here was the choice of initialization for word embeddings. We tested three different scenarios: Randomly initializing embeddings, Using pre-trained embeddings, Training embeddings on the training set. We trained word embeddings on the training data using Cbow method.We used fastText [2] for pre-trained embeddings. The results are reported in the table below.

embedding initialization	**accuracy**
Randomly initialized embeddings	86.38%
Pre-trained embeddings	86.4%
Embeddings trained on training set	88.06%

We gained a 1.6 percent increase in accuracy when we trained our own embeddings on the training data.

4.3 CNN

In our experiments with the CNN model, there were 2 main hyper-parameters that we needed to tune: kernel lenght and the number of kernels. Kernel length is the size of the convolution sliding window which means the number of consecutive words that are convolved into a feature vector and number of kernels is the dimension of this vector. We first ran some tests using a only one kind of kernel and set the number of kernels to 128 to find the best kernel length. We use global average pooling for all the experiments because it resulted in better accuracy. The results are shown in the table below.

Kernel length	**accuracy**
2	87.73%
3	88.00%
4	87.98%
5	88.00%
10	87.90%
30	87.73%

As we can see, smaller kernel lengths generally give better results. now we set kernel length to 3 and test the effect of different number of kernels which is basically the dimension of the output feature space. Results are shown below.

number of kernels	**accuracy**
128	88.00%
256	88.16%
512	88.21%
1024	88.36%

We can see that increasing the kernel size increases the accuracy but it also increases the training time. Training a network with a kernel size of 1024 takes 5 times longer that a network with kernel size of 128. All the above experiments were done using only a single kind of kernel. We can use multiple kernel lengths and then concatenate the output feature maps. we ran two experiments with different combinations of kernel lengths and set the kernel length to 512 for a shorter training time. Results are shown in the table below.

kernel lengths	**accuracy**
2,3,4	88.46%
3,4,5	88.39%

We can see that using kernel length with a smaller average resulted in better accuracy probably because we extract features from a smaller number of consecutive words on average.

4.4 RNN

For the RNN method, we used a bi-directional LSTM with 1 layer. We ran three main experiments all related to how we handled the hidden states of the last layer. We first used only the hidden state of the last time step. In the next experiment we used global max pooling on the hidden states and in the last experiment we used an attention layer to calculate the weighted sum of all the hidden states. We implemented the attention layer as it is mentioned in **?**. The formula for the attention layer is shown below:

$$A = tanh(H)$$

$$\alpha = softmax(w^T A)$$

$$O = H\alpha^T$$

where H denotes the hidden states of the last layer and α is the attention weight vector. Result of the three experiments are shown below:

Model	**accuracy**
Last time step hidden state	88.48%
Attention	88.87%
global max pooling	88.78%

As we can see using an attention layer yields the best accuracy. The reason is that by using attention weight, we are giving different values to different time steps, and this is intuitively understandable because some words play a more important role in prediction of the class, for example a word like "is" is much less valuable than the word "sofa" when trying to predict the label of an ad.

4.5 Combination of CNN and RNN

In the previous two models, we saw that the CNN model outperformed the ANN model because of its feature extraction capabilities and the LSTM model outperformed the CNN model because of its sequential nature. In this section we experiment with two models which are both results of combining LSTMs and CNNs. For the first model, we use a CNN to extract features from the input data and then perform pooling to make the sequence smaller, and then feed the resulting vectors to an LSTM. In our experiment, we used average pooling of size 5 which made input sequences 5 times shorter. The main advantage of this model is that it makes the sequences smaller thus reduces the training time of the LSTM but the LSTM doesn't use all the information of the sequence because of the pooling. The second model is comprised of an LSTM followed by a CNN. In this model the hidden states of the last layer of the LSTM are fed to a CNN. The second model suffers from long training time but performs better than the first model. The second model outperformed the attention based LSTM model in the previous section. The results are shown in the table below:

textbfModel	**accuracy**
CNN + LSTM + attention	88.46%
LSTM + CNN + average pooling	88.9%

5 Discussion

The model that we proposed in this project was a combination of and an LSTM and a CNN. CNNs lack the ability to take advantage of the sequential properties of the input data. LSTMs on the other hand specialize in sequential data but the real challenge is the way we use the hidden states of all the time steps of the last layer. We used the feature extraction power of CNNs to extract features from hidden states of the last layer of an LSTM and we improved the results compared to the attention-based LSTM.

References

[1] Kim, Yoon (2014) Convolutional Neural Networks for Sentence Classification, *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pp. 1746–1751.

[2] Joulin, Armand , Grave, Edouard , Bojanowski, Piotr & Mikolov, Tomas (2016) Bag of Tricks for Efficient Text Classification. *arXiv preprint arXiv:1607.01759*

[3] Zhou, Peng , Shi, Wei , Tian, Jun , Qi, Zhenyu , Li, Bingchen , Hao, Hongwei & Xu, Bo (2016) Attention-Based Bidirectional Long Short-Term Memory Networks for Relation Classification. *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*, pp. 207–212.

[4] Yang, Zichao , Yang, Diyi , Dyer, Chris , He, Xiaodong , Smola, Alex & Hovy, Eduard (2016) Hierarchical Attention Networks for Document Classification. *Proceedings of the 2016 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pp. 1480–1489