Homework 02: Text Classification

2021 Spring, CS 6501 Natural Language Processing

Due on **March 18, 2021 11:59 PM**

1. **Linear Classifiers** (2 points) On page 18 of lecture slides 02, we give the definition of logistic regression for multi-class classification

$$P(y \mid \boldsymbol{x}) = \frac{\exp(\boldsymbol{w}_y^\mathsf{T} \boldsymbol{x} + b_y)}{\sum_{y' \in \mathcal{Y}} \exp(\boldsymbol{w}_{y'}^\mathsf{T} \boldsymbol{x} + b_{y'})} \tag{1}$$

   where $\boldsymbol{x}$ is an input document, $y \in \mathcal{Y}$ could be any pre-defined label. Equation 1 defines the conditional probability of any label y for the given input $\boldsymbol{x}$. $\boldsymbol{w}_y$ is the classification weight associated with the label y. As shown in equation 1, logistic regression uses the softmax function to make sure $P(y \mid \boldsymbol{x})$ is a valid probability. Although the softmax function is a nonlinear function, please prove that the classifier defined equation 1 can still be considered as a linear classifier.

2. **Binary logistic regression classifiers** (2 points) On page 20 of lecture slides 02, we showed that for binary classification, logistic regression can be written in a more concise form as

$$P(Y = 1 \mid \boldsymbol{x}) = \frac{1}{1 + \exp(-\boldsymbol{w}^\mathsf{T} \boldsymbol{x})} \tag{2}$$

   where $Y \in \{0, 1\}$ and $\boldsymbol{w}$ is the only parameter of this model. Please verify the model defined in equation 2 is a special form of equation 1 when the label set $\mathcal{Y}$ only contains two elements.

3. **NLL and Cross entropy** (2 points) For a given probabilistic classifier $P(Y = y \mid \boldsymbol{x}; \boldsymbol{\theta})$ with $\boldsymbol{\theta}$ as the classification parameter and $y \in \mathcal{Y}$, there are two ways of defining the loss function. In the lecture of logistic regression, we discussed the definition of *negative log-likelihood* (NLL) as

$$L(\boldsymbol{\theta}) = -\sum_{i=1}^{m} \log P(y^{(i)} \mid \boldsymbol{x}^{(i)}; \boldsymbol{\theta}) \tag{3}$$

   where $\{(\boldsymbol{x}^{(i)}, y^{(i)})\}_{i=1}^m$ is the collection of training examples. In the lecture of neural network classifiers, we introduced another loss function called *cross entropy*. With the empirical distribution

$$Q(Y = y' \mid \boldsymbol{x}^{(i)}) = \begin{cases} 1 & y' = y^{(i)} \\ 0 & y' \neq y^{(i)} \end{cases} \tag{4}$$

   and the definition of cross entropy as

$$H(Q(Y \mid \boldsymbol{x}), P(Y \mid \boldsymbol{x})) = -\sum_{y' \in \mathcal{Y}} Q(Y = y' \mid \boldsymbol{x}) \log P(Y = y' \mid \boldsymbol{x}), \tag{5}$$

   please show that the following equation holds

$$L(\boldsymbol{\theta}) = \sum_{i=1}^{m} H(Q(Y \mid \boldsymbol{x}^{(i)}), P(Y \mid \boldsymbol{x}^{(i)})) \tag{6}$$

4. **Building a Multi-class Classifier** (6 points) With the demo code, we explained how to build a logistic regression classifier for binary classification problem. In this project, we will follow a similar procedure to build a text classifier for multi-class classification problems. We are going to build a text classifier that can predict the topic of a new article based its headline. You can download the data from the following link,

   https://yangfengji.net/uva-nlp-course/data/news.zip

which contains three files

- `news-trn.tsv`: the training examples with labels
- `news-dev.tsv`: the validation example with labels
- `news-tst.tsv`: the test examples

Please follow the requirements to build a text classifier from the beginning

(a) **Load data** (1 point): please implement a data reader function that can load the examples and store both texts and their labels into suitable data structures. With this data reader function, now you can load the training and validation sets respectively, and get a sense about what it looks like with the following two distributions

i. the number of examples in each set

ii. the label distribution in each set, which can tell you which label is the major class

(b) **Construct bag-of-words representations** (1 point) Construct the bag-of-words representations of both sets. The vocab of the bag-of-words representations should be built from the training set. For the `CountVectorizer` function, please use the default parameter values.

i. Report the size of the vocab

ii. Plot the word frequency with a decreasing order and verify whether it looks like similar to the Zipf's law.

(c) **Build a Classifier** (1 point) Build a classifier with the following parameter setting

- penalty = L2
- fit_intercept = True
- C = 1 (in our lecture note, we use $\lambda = \frac{1}{C}$)

Report the classification accuracy on the *training and validation* sets.

(d) **Hyper-parameter Tuning** (2 points) In the whole process of building text classifiers, there are many different parameters that can affect the model performance. For this question, we will mainly consider the following four parameters with the given sets of values

- $C = \{10.0, 1.0, 0.1, 0.01\}$
- min_df $= \{1, 5\}$
- max_df $= \{0.8, 1.0\}$
- fit_intercept $= \{\text{Ture}, \text{False}\}$

to find the best value combination of these four parameters. In machine learning, the process of finding the best values of these parameters is called **Hyper-parameter tuning**. We call them as *hyper-parameters*, because, unlike classification weights, these parameters need to be specificed before the actual learning process starts.

You can *manually* try each combination at a time and report the the best value combination and the corresponding training and validation accuracies.

(e) **Find the Best Performance** (1 point) In the dataset files, you will also find a test file named `news-tst.tsv`. Based on the validation accuracy, please use the best model from the previous step to predict the labels for the headlines in the test file, and submit the label prediction file with the name `news-tst.pred` and the following format for each line (the same format for the training and validation sets)

label\t headline_text

The prediction accuracy on those test examples will be calculated and used for grading this question.

5. **CNN for Text Classification** (4 points) Based on the demo code, if we want to build another neural network classifier, the only thing that we need to do is to specify the model architecture in the forward() function (and the corresponding parameter definition in the __init__() function). In section 2.2 of the demo code, we already have a scaffolding for a convolutional neural network classifier. The goal of this problem is to practice the implementation of a simple CNN and test its performance.

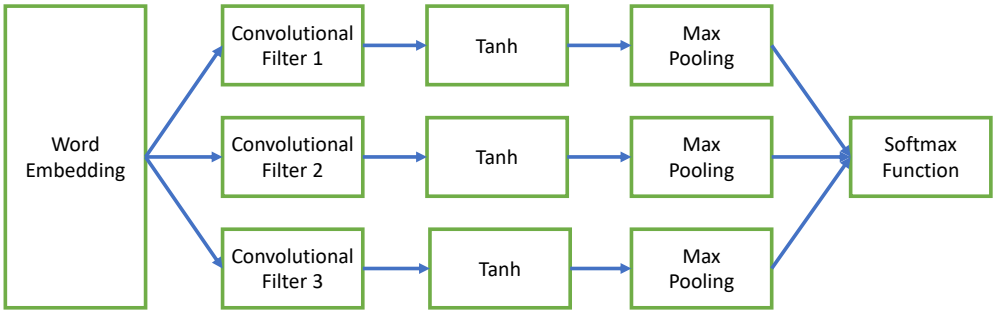

Figure 1: The basic architecture of the CNN classifier.

Please implement the CNN classifier with the following specification in the `forward()` function of the `SimpleCNN` class.

- The neural network architecture is illustrated in Figure 1. Although some of the lines in the `NeuralClassifier` can still be used here, please use them with caution.
- About convolution layer: this layer includes three convolutional filters with window size (or kernel size) as $2, 3, 4$ respectively. For each convolutional filter, the input and output dimensions are the same as the word embedding dimension. To define each convolutional filter, you will need the torch.nn.Conv1d() function.
- Following the convolutional layer is the nonlinear activation function `tanh`, which is implemented in PyTorch as torch.tanh()
- The next operation is the 1-dimensional max-pooling, which should only operate along the dimension. For example, in a 3-D tensor $\boldsymbol{T} \in \mathbb{R}^{B \times E \times L}$, where the first dimension is the mini-batch size B, the second dimension is the word embedding dimension E, and the third dimension is the number of words (with paddings) L. This max-pooling operator should only be applied on the third dimension, and the output from this operator should be a $B \times E$ matrix. The max-pooling operations can be implemented with torch.max() with a specified dim parameter.
- The outcomes from the max-pooling layer should be three $B \times E$ matrices. The last operations before the (log-)softmax function is the concatenation — concatenating three matrices together to form a $B \times 3E$ matrix with the PyTorch function torch.cat().

After the implementation, please change the `model_name` parameter in section 2.2.2 as

```
model_name = 'cnn'
```

Then, you should be able to run the code with the rest of the functions. Please

- report the validation accuracy of this model in your homework submission, and
- submit the IPython notebook file with `[ComputingID]-cnn.ipynb`