Assignment 4: Naive Bayes Classifier and Deep Learning for Natural Language Processing (DIFFICULT!)

UVA CS 4774
Machine Learning Foundation, Deep Learning and Good Uses

January 20, 2022

a *The assignment should be submitted in the PDF format through Collob. If you prefer hand-writing QA parts of answers, please convert them (e.g., by scanning or using an app like Genuis Scan) into PDF form.*

b *For questions and clarifications, please post on Slack.*

c *Policy on collaboration:*

Homework should be done individually: each student must hand in their own answers. It is acceptable, however, for students to collaborate in figuring out answers and helping each other solve the problems. We will be assuming that, with the honor code, you will be taking the responsibility to make sure you personally understand the solution to any work arising from such collaboration.

d *Policy on late homework:*

Homework is worth full credit at the midnight on the due date. Each student has 10 *extension days to be used at his or her own discretion throughout the entire course. You could use the days in whatever combination you like. For example, all late days on 1 assignment or 1 each day over 10 assignments (for a maximum grade of 90% on each). After you've used all 10 days, you cannot get credit for anything turned in late.*

e *: we will use the EXTRA points in YOUR final grade calculation...*

e *Policy on grading:*
***1**: 50 points in total. 30 points for code submission (and able to run). 20 points for displaying results in the report.*
***2**: 50 points in total. 25 points for code submission (and able to run). 25 points for the results in your report.*

1 Dataset

In this programming assignment, we will classify text-based movie reviews into positive or negative. A ZIP file "data_sets_naive_bayes.zip" including two sets of data samples (i.e. training set and test set) for movie reviews is provided to you through collab attachments. We will compare the following methods:

- Naive Bayes
- Deep Learning using pretrained state-of-the-art NLP models (BERT)

Please follow the following naming rules wrt files and functions.

2 Naive Bayes Classifier

We expect you to submit a source-code file named as "naiveBayes.py" containing the necessary and required functions for training, testing and evaluations.

For a naive Bayes classifier, when given an unlabeled document d, the predicted class

$$c_d^* = \underset{c}{\operatorname{argmax}}\, \mathbb{P}(c|d)$$

, where c is the value of the target class variable. For the target movie review classification task, $c =$ **pos** or **neg**. For example, if $\mathbb{P}(c = pos|d) = \frac{3}{4}$ and $\mathbb{P}(c = neg|d) = \frac{1}{4}$, we use the MAP rule to classify the document d into the "positive" class. The conditional probability $\mathbb{P}(c|d)$ is calculated through **Bayes' rule**,

$$\mathbb{P}(c|d) = \frac{\mathbb{P}(c)\mathbb{P}(d|c)}{\mathbb{P}(d)} \propto \mathbb{P}(c)\mathbb{P}(d|c)$$

This assignment requires you to implement three types of naive bayes classifiers, among which the first two follow the Multinomial assumption and the third uses the multivariate Bernoulli assumption.

2.1 Preprocessing

- (Q1) You are required to implement the preprocessing for the dataset. The first step is stemming and stopword removal. (you can use a python package such as NLTK).

  You will build a vocabulary of unique words from the training corpus, being ranked with their frequency. Then you will just use the top $K = 100$ words that appearing more than a certain times (e.g. 3 times) in the whole training corpus. You can optionally add a 'UNK' token to the vocabulary to represent the words not a part of the vocabulary.

- You are required to implement a generic function to load the datasets from the provided zip and build the vocabulary.
  $XtrainText, XtestText, ytrain, ytest, vocabulary = loadData(textDataSetsDirectoryFullPath)$

- "textDataSetsDirectoryFullPath" is the real full path of the file directory that you get from unzipping the datafile. For instance, it is "/HW/data_sets/" on the instructor's laptop.

2.2 Build "bag of words" (BOW) Document Representation

- (Q2) The next step is converting the text data into a Bag of Words representation. You are required to provide the following function to convert a text document into a feature vector:

  $BOWDj = transfer(review, vocabulary)$

  where fileDj is the location of file j

- (Q3) Read in the training and test documents into BOW vector representations using the above function. Then store features into matrix Xtrain and Xtest, and use ytrain and ytest to store the labels.

  $Xtrain, Xtest = getBOWRepresentation(XtrainText, XtestText, vocabulary)$

- getBOWRepresentation should call $transfer(review, vocabulary)$

  Note: Xtrain and Xtest are matrices with each row representing a document (in BOW vector format). ytrain and ytest are vectors with a label at each position. These should all be represented using a python list or numpy matrix.

2.3 Multivariate Bernoulli Naive Bayes Classifier (BNBC)

- We need to learn the $\mathbb{P}(c_j)$, $\mathbb{P}(w_i = false|c_j)$ and $\mathbb{P}(w_i = true|c_j)$ through the training. MLE gives the relative-frequency as the estimation of parameters. We will add with Laplace smoothing for estimating these parameters.
- Essentially, we simply just do counting to estimate $\mathbb{P}(w_i = true|c)$.

$$\mathbb{P}(w_i = true|c) = \frac{\#\text{files which include } w_i \text{ and are in class c} + 1}{\#\text{files are in class c} + 2}$$

$$\mathbb{P}(w_i = false|c) = 1 - \mathbb{P}(w_i = true|c)$$

- Since we have the same number of positive samples and negative samples, $\mathbb{P}(c = -1) = \mathbb{P}(c = 1) = \frac{1}{2}$.
- (Q10) You are required to provide the following function (and module) for grading:

  $thetaPosTrue, thetaNegTrue = naiveBayesBernFeature\_train(Xtrain, ytrain)$

- (Q11) Provide the resulting parameter estimations into the writing.
- (Q12) You are required to provide the following function (and module) for grading:

  $yPredict, Accuracy = naiveBayesBernFeature\_test(Xtest, ytest, thetaPosTrue, thetaNegTrue)$

  Add the resulting Accuracy into the writing.

2.4 Extra Credit (10/100): Multinomial Naive Bayes Classifier (MNBC) Training + Testing Step from scratch

- We need to learn the $\mathbb{P}(c_j)$ and $\mathbb{P}(w_i|c_j)$ through the training set. Through MLE, we use the relative-frequency estimation with Laplace smoothing to estimate these parameters.
- Since we have the same number of positive samples and negative samples, $\mathbb{P}(c = -1) = \mathbb{P}(c = 1) = \frac{1}{2}$.
- (Q4) You are required to provide the following function (and module) for grading:

  $thetaPos, thetaNeg = naiveBayesMulFeature\_train(Xtrain, ytrain)$

- (Q5) Provide the resulting value of thetaPos and thetaNeg into the writeup.

  Note: Pay attention to the MLE estimator plus smoothing; Here we choose $\alpha = 1$.

  Note: thetaPos and thetaNeg should be python lists or numpy arrays (both 1-d vectors)

- (Q6) You are required to provide the following function (and module) for grading:

  $yPredict, Accuracy = naiveBayesMulFeature\_test(Xtest, ytest, thetaPos, thetaNeg)$

  Add the resulting Accuracy into the writeup.

  Important: Do not forget perform **log** in the classification process.

2.5 Multinomial Naive Bayes Classifier (MNBC) Evaluate Step

- (Q7) Use "sklearǹnaive_bayes.MultinomialNB" from the scikit learn package to perform training and testing.

  Add the resulting Accuracy into the writeup.

  [Extra Credit (10/100):] Compare the results with your own MNBC if you implement the above extra-credit scratch version.

2.6 How will your code be checked ?

In collab, you will find the sample codes named "naiveBayes.py". "textDataSetsDirectoryFullPath" is a string input. We will run the command line: "python naiveBayes.py textDataSetsDirectoryFullPath" to check your code if it can print the result of the following functions in the table.

thetaPos, thetaNeg = naiveBayesMulFeature_train(Xtrain, ytrain)
yPredict, Accuracy= naiveBayesMulFeature_test(Xtest, ytest, thetaPos, thetaNeg)
thetaPosTrue, thetaNegTrue= naiveBayesBernFeature_train(Xtrain, ytrain)
yPredict, Accuracy= naiveBayesBernFeature_test(Xtest, ytest,thetaPosTrue, thetaNegTrue)

3 State-of-the-art Deep Learning for NLP: BERT

Researchers have developed various techniques for training general purpose NLP models by pretraining using enormous piles of unannotated text. These general purpose pre-trained models can then be fine-tuned on smaller task-specific datasets, like the dataset we use for this HW.

One such model is BERT (blog post), that we will use in this homework.

3.1 Installation

- Please use Google Colab only for this question.
- Install transformers package, that provides thousands of state-of-the-art pretrained models for NLP tasks.

```
!pip install transformers
```

- Change Runtime type to GPU.

3.2 Preprocessing for BERT model

- Recall that for Naive Bayes you implemented your own tokenizer using the NLTK package. For this portion, we will use the "bert-base-uncased" tokenizer. BERT was trained using the WordPiece tokenization. It means that a word can be broken down into more than one sub-words. This process is a part of the training time and depends on the training corpus. We could use any other tokenization technique, but we'll get the best results if we tokenize with the same tokenizer the BERT model was trained on.
- In this section, we will not do any preprocessing to the data and let the BERT tokenization process the data.
- We have provided a wrapper for this part of the process. Note that BERT was pretrained using a maximum length sequence(MAX_LEN) of 512. You can specify a smaller MAX_LEN like 300 to finetune the model. You can also choose this heuristically based on the average length of reviews in the dataset.

3.3 Fine Tune a State-of-the-art NLP Model

- First, We'll be using $TFBertForSequenceClassification$. This interface enables us to load a pre-trained BERT model with an added untrained single linear layer on top for classification that we will use as a classifier. We will finetune the entire pre-trained BERT model and the additional untrained classification layer for our Movie Review classification task.
- (**Q1**) Your first task is to train this model. Remember to shuffle the data prior to training, also shuffle for each epoch. (a). Report the performance of BERT model on test dataset. (b) Also, include a figure showing the train and validation loss. (c) Report the batch size, learning rate, etc you used to train the model.
- You are free to tune your own hyperparameters like learning rate, batch size, number of epochs, etc. You will use 10% of the training data as validation set. The final test accuracy should be reported after training on the entire train dataset.

3.4 Training an MLP from scratch on Bag of Words Representation

In this section, we will compare the performance of MLP on the same dataset, represented in the Bag of Words form as used in the Naive Bayes Section.

- You can reuse the *loadData* and *getBOWRepresentation* functions from the NaiveBayes section.
- The input features are now the BOW representations learnt in the NaiveBayes Section.
- In this section, you are required to try different architectures of MLP, like varying hidden size and number of layers. You can reuse the *train_and_select_model* function from HW3. You will use 10% of the training data as validation set.
- (Q2) Report the accuracy of the best model (selected using validation accuracy, retrained on entire training data) on the test set.

Congratulations ! You have implemented a state-of-the-art machine-learning toolset for an important web text mining task !