CS5740, Spring 2020: Assignment 1

Due: Feb 26, 11:59pm

Report page limit: 3 pages

In this assignment, your group will build maximum entropy, perceptron, and multilayer perceptron text classifiers using bag of words features (and other task-specific features of your own design).

Forming Groups Please form groups of two early and work together. We will not handle pairing or monitor it. We will penalize heavily the grades of groups that include more of fewer than two students. If we identify cases of students that do not work together, we will be forced to penalize the assignment heavily. Do not create Github Classroom teams until your group is finalized on CMS.

Submission **Please see the submission guidelines at the end of this document.** In your writeup, at the least, make sure to describe (1) which features you tried, (2) which features were most successful in your classifiers, (3) how you represented your features in your three models, and (4) any important decisions you made when implementing or training your models.

Starter Repository:
https://classroom.github.com/g/cgTZPHBM (GitHub Classroom invitation link)

20 Newsgroups Leaderboard:
https://github.com/cornell-cs5740-20sp/leaderboards/blob/master/assignment1/leaderboard_newsgroup.csv

Proper Names Leaderboard:
https://github.com/cornell-cs5740-20sp/leaderboards/blob/master/assignment1/leaderboard_propername.csv

Report Template:
https://www.overleaf.com/read/qxfpdvcbnrzb

GitHub Classroom Setup Follow the *starter repository invitation* link. If your partner has already created a team, join their team. Otherwise, enter a name for your team and continue. GitHub Classroom will provide a link to your team's private repository for you to clone to your computer. All team code should exist in the repository. *Only the code that is present in the master branch of your repository by the due date will be evaluated as part of your submission!*

The starter repository contains training and development data for the *Proper Names* and *20 Newsgroups* datasets, in the `data/` directory. You will also find the test data for these two datasets without the labels.

The repository contains minimal starter code, including for the classification models and processing of the data. An evaluation script (`evaluate.py`) is available to check your progress on the development sets and verify that your output, in the form of predicted classes for names and newsgroups, is in the correct format for creating the leaderboard.

```
python evaluate.py -d newsgroups/dev/dev_labels.csv -p [YOUR_PREDICTED_LABELS]
```

You will need to read in the training and development/test data separately in order to train your classifier, and you are free to use code from the evaluation script to help with classifier training.

Proper Name Classification Proper name classification is the task of taking proper names like *Eastwood Park* and deciding whether they are places, people, etc. In the more general task of named entity recognition (NER), which we do not consider here, one also has to detect the boundaries of the phrases. In general, we might use a variety of cues to make these kinds of decisions, including looking at the syntactic environment that the phrases occur in, whether or not the words inside the phrase occur in lists of known names of various types (gazetteers), and so on. In the first part of this assignment, however, you will write classifiers which attempt to classify proper names purely on the basis of their surface strings alone. This approach is more powerful than you might think: for example, there aren't too many people named *Xylex*. Since even the distribution of characters is very distinctive for each of these categories, we will start out trying to make classification decisions on the basis of character n-grams alone (so, for example, the suffix *-x* may indicate drugs while *-wood* may indicate places).

The labels are one of five strings: `person`, `place`, `movie`, `drug`, or `company`.

Newsgroup Classification The 20 Newsgroups dataset contains 18,846 newsgroup documents written on a variety of topics. In the second part of this assignment, you will use the text of each document to predict its newsgroup. Unlike proper names, that can only belong to one of six categories, each document could belong to one of twenty newsgroups. Additionally, many of these newsgroups share similar themes, such as computing, science, or politics (see Table 1 for the full list). However, the distributions of words across each of these newsgroups are also fairly distinctive. For example, a document that uses the names of weapons will likely be in `talk.politics.guns`, a document mentioning "computer" will probably be in a `comp.*` group, and if a document uses the word "ice," it was likely written for `rec.sport.hockey` rather than `talk.politics.mideast`.

comp.graphics comp.os.ms-windows.misc comp.sys.ibm.pc.hardware comp.sys.mac.hardware comp.windows.x	rec.autos rec.motorcycles rec.sport.baseball rec.sport.hockey	sci.crypt sci.electronics sci.medsci.space
misc.forsale	talk.politics.misc talk.politics.guns talk.politics.mideast	talk.religion.misc alt.atheism soc.religion.christian

Table 1: Twenty newsgroup labels partitioned into six themes.

Features

1. *Character n-grams.* For the proper name classification dataset, start by using character n-grams. How does experimenting with different values of n improve or harm your accuracy? You may want to include features from several values of n in your final classifier.
2. *Bag of words, n-grams.* For the newsgroup dataset, start with a bag-of-words classifier. Increasing n for character n-grams improves the representation of context and ordering in your features, at the expense of data sparsity. How do word n-grams compare? You may want to experiment with using binary (presence/absence) features instead of count features and excluding stopwords (e.g., the, is, a) to improve your accuracy.
3. *Your own features.* Develop and use task-specific features you choose in order to improve your accuracy in the leaderboard (for both domains). In your final submission, describe how you represent these features in your models.

Models

1. *Perceptron.* To begin, implement a simple perceptron-based model from scratch. Remember, instead of having to compute the derivative over the entire training set, the perceptron simply picks up each example in sequence, and tries to classify it given the current weight vector. If it gets it right, it simply moves on to the next example, otherwise it updates the weight vector with the difference of the feature counts in the correct example and in the prediction. It is recommended not to use a deep learning framework for this model, but instead use NumPy or just simple Python code with dictionaries.

2. *Maximum Entropy.* Second, implement a maximum entropy (MaxEnt) classifier. Training a MaxEnt classifier requires more work, but we'll offload the more difficult optimization component to SciPy. For an example X, label y, a paired feature vector $\phi(X, y)$, a MaxEnt model uses a weight vector w to compute:

   $$p(y \mid X; w) = \frac{e^{w \cdot \phi(X,y)}}{\sum_{y'} e^{w \cdot \phi(X,y')}}$$

   Learning adjusts w in order to maximize the log-likelihood of the training data $\{(X^{(i)}, y^{(i)})\}_{i=1}^{N}$ given the set of feature vectors. This is calculated as:

   $$L(w) = \sum_{i=1}^{N} \log p(y^{(i)} \mid X^{(i)}; w)$$

   You should use `scipy.optimize.fmin_l_bfgs_b` (part of the SciPy package) for optimization. In addition to providing the objective function L, you must also compute its derivatives:

   $$\frac{\partial L}{\partial w_j} = \frac{1}{N} \sum_{i=1}^{N} \left(\phi_j(X^{(i)}, y^{(i)}) - \sum_{y} p(y \mid X^{(i)}; w) \phi_j(X^{(i)}, y) \right)$$

   Here, we multiply by $\frac{1}{N}$ to make the objective invariant to the data size and avoid very large updates, which are often numerically unstable. Recall that the left sum is the total feature count vector over examples with true class y in the training, while the right sum is the expectation of the same quantity, over all examples, using the label distributions the model predicts. Two things to note about `scipy.optimize.fmin_l_bfgs_b`: (1) it *minimizes* the objective function using its gradient, meaning you will need to negate both L and its gradient and (2) even though its documentation suggests otherwise, it only accepts single-dimensional `ndarrays` (vectors) as arguments to the objective function. It is recommended not to use a deep learning framework for this model, but instead use NumPy or just simple Python code with dictionaries.

3. *Multilayer Perceptron.* Finally, implement a multilayer perceptron (MLP). One big advantage of DyNet (and other similar frameworks like PyTorch and TensorFlow) is that it abstracts away the details of backpropagating error, allowing you to focus on designing your network's architecture. (If you want to learn about backpropagation, read `https://www.cs.cornell.edu/courses/cs5740/2016sp/resources/backprop.pdf`.)

   Matrix operations in deep learning frameworks are very similar to NumPy. Here, we illustrate this using DyNet. In DyNet, you can create a matrix (called expressions) from a Python or Numpy array using `dy.inputTensor`. DyNet funtions like `dy.cmult` can be used to perform matrix operations on one or more matrices. Also, Python unary operators like `+` and `*` can be used with DyNet expressions. For reference, see the DyNet Python reference page: `http://dynet.readthedocs.io/en/latest/python_ref.html`. Using these functions, you can create your hidden layers. Your MLP must have multiple hidden layers, although the exact number is your own. There are two types of parameters in Dynet: `dy.Parameters` and `dy.LookupParameters`. `dy.Parameters` contain vectors or matrices that are updated using backpropagation. `dy.LookupParameters` allow you to look up a single vector given an index in the set of parameters, and is often used for tasks like looking up word vectors. *Hint: your feature vectors will probably be sparse, so it is more efficient to use LookupParameters than Parameters in the first layer of the network.* Although DyNet takes care of initializing your parameters, this is critical to getting your experiment to work. You can choose different ways of initializing your parameters by using a different `dy.PyInitializer`. A short tutorial on DyNet basics, including creating a computation graph and using parameters, is available at `https://github.com/alsuhr-c/dynet_tutorials/blob/master/dynet_basics.ipynb`.

   When designing your MLP, experiment with different activation functions. Activation functions add nonlinearity to your MLP, allowing it to capture more complex aspects of your training data. You should use at least ReLu, sigmoid, and *tanh* activation functions. In DyNet, they are available as `dy.recfity`, `dy.logistic`, and `dy.tanh`.

In your final layer, you will want to transform the output of your network to a probability distribution (using the *softmax* function) and compare this distribution to your training labels. In DyNet, you can use `dy.log_softmax` along with `dy.mean_elems` to define the cost function used by the optimizer.

Finally, you should experiment with different optimizers and learning rates to see if they allow faster training and/or better results. The standard approach is to use gradient descent, but DyNet, for example, comes with others built in, such as: `dy.SimpleSGDTrainer`, `dy.AdagradTrainer`, `dy.AdamTrainer`.

Leaderboard Instructions We will update the leaderboards once a day using your predictions on the testing set. The leaderboard uses classification accuracy as the evaluation metric, calculated the same way as in the evaluation script provided in the starter repository. For comparison: a simple baseline that always chooses the most frequent class would achieve 30% accuracy on the proper names dataset (the six categories do not occur with equal frequencies) and 5% accuracy on newsgroup dataset. If your classifier is performing worse than these baselines on either dataset, something has gone horribly wrong!

After using your classifier to predict labels for the test data (`data/propernames/test/test_data.csv` and `data/newsgroups/test/test_data.csv`), you must export the results to a CSV file in the same format as the development and test labels. The starter code contains example CSV files with (incorrect) test predictions for all domains. Your prediction files should be in the same format as these files for the update to the leaderboard to work correctly. If something did not work correctly, for example if the number of predictions is not the same as the number of labels, an error message will be displayed on the leaderboard, but will lose this evaluation chance. Your predictions file must use the correct filename for the leaderboard to be updated. All test predictions files must end in `newsgroup_test_predictions.csv` or `propername_test_predictions.csv`. You should also add a prefix indicating which method produced the results; one of `maxent_`, `mlp_`, and `perceptron_`.

Performance Grading Your test performance in the six experiments will be calculated as:

$$\sum_{m\in\{\textsc{MaxEnt},\textsc{Perceptron},\textsc{MLP}\},d\in\{\textsc{ProperName},\textsc{Newsgroups}\}} \frac{a_m^d}{80} \times 4 \ ,$$

where a_m^d is the test accuracy of model m on dataset d, $0 \leq a_m^d \leq 100$. There are three models and two datasets, a total of six test accuracy scores.

Current CS5740 State of the Art The highest performance on ProperName is 87.25 (Sp18), and on Newsgroups is 84.13 (Sp19).

Submission, Grading, and Writeup Guidelines Your submission on CMS is a writeup in PDF format. **The writeup must follow the template provided. Please replace the TODOs with your content. Do not modify, add, or remove section, subsection, and paragraph headers.** The writeup must include at the top of the first page: the names of the students, the NetIDs of both students, the team name, and the URL of the Github repository. The writeup page limit is **3 pages**. We will ignore any page beyond the page limit in your PDF (do not add a cover page). We have access to your repository, and will look at it. Your repository must contain the code in a form that allows to run it form the command line (i.e., Jupyter notebooks are not accepted).

The following factors will be considered: your technical solution, your development and learning methodology, and the quality of your code. If this assignment includes a leaderboard, we will also consider your performance on the leaderboard. Our main focus in grading is the quality of your empirical work and implementation. Not fractional difference on the leaderboard. We value solid empirical work, well written reports, and well documented implementations. Of course, we do consider your performance as well. The assignment details how a portion of your grade is calculated based on your empirical performance.

In your write-up, be sure to describe your approach and choices you made. Back all your analysis and claims with empirical development results, and use the test results only for the final evaluation numbers. It is sometimes useful to mention code choices or even snippets in write-ups — feel free to do so if appropriate, but this is not necessary.

Some general guidelines to consider when writing your report and submission:

- Your code must be in a runnable form. We must be able to run your code from vanilla Python command line interpreter. You may assume the allowed libraries are installed. Make sure to document your code properly.
- Your submitted code must include a `README.md` file with execution instructions. Make sure to document your code properly.
- Please use tables and plots to report results. If you embed plots, make sure they are high resolution so we can zoom in and see the details. However, they must be readable to the naked eye. Specify exactly what the numbers and axes mean (e.g., F1, precisions, etc).
- It should be made clear what data is used for each result computed.
- Please support all your claims with empirical results.
- All the analysis must be performed on the development data. It is OK to use tuning data. Only the final results of your best models should be reported on the test data.
- All features and key decisions must be ablated and analyzed.
- All analysis must be accompanied with examples and error analysis.
- Major parameters (e.g., embedding size, amount of data) analysis must include sensitivity analysis. Plots are a great way to present sensitivity analysis for numerical hyper parameters, but tables sometimes work better. Think of the best way to present your data.
- If you are asked to experiment with multiple models and multiple tasks, you must experiment and report on all combinations. It should be clear what results come from which model and task.
- Clearly specify what are the conclusions from your experiments. This includes what can be learned about the tasks, models, data, and algorithms.
- Make figures clear in isolation using informative captions.

This assignment was adapted from Dan Klein.