CS5740, Spring 2017: Assignment 1

Due: February 22, 2017

In this assignment, your group will build maximum entropy, perception, and multilayer perceptron text classifiers using bag of words features (and other task-specific features of your own design). After training and evaluating these classifiers on two different datasets, one of newsgroups and another of proper names, you will submit the results of your most successful models to our in-class Kaggle competitions.

Starter Repository: `https://goo.gl/sMLu7Y` (GitHub Classroom invitation link)
20 Newsgroups Kaggle: `https://inclass.kaggle.com/c/cs5740-20-newsgroups-classification`
Proper Names Kaggle: `https://inclass.kaggle.com/c/cs5740-proper-names-classification`

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. You will also find the test data for these two datasets without the labels.

The repository does not contain any starter code: you are responsible for implementing your own classification system and designing features. However, 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 Kaggle format.

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

You will need to read in the training and 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.* You may use any task-specific features you choose in order to improve your accuracy on the Kaggle competitions. 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. You are allowed to use NumPy only for this part.

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. Given a paired feature vector x and label y in a training set, a MaxEnt model uses a weight vector w to compute $P(y \mid x, w)$ as follows:

$$P(y \mid x, w) = \frac{e^{w \cdot f(x,y)}}{\sum_{y'} e^{w \cdot f(x,y')}}$$

   Your model adjusts w in order to maximize the log likelihood of the training labels given the set of feature vectors. This is calculated as:

$$L(w) = \sum_i \log P(y_i \mid x_i, w)$$

   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:

$$\frac{\partial L}{\partial w_y} = \sum_i I(y_i = y) f(x_i, y) - \sum_i P(y \mid x_i, w) f(x_i, y)$$

   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.

3. *Multilayer Perceptron.* Finally, implement a multilayer perceptron (MLP) using TensorFlow. One big advantage of 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 what TensorFlow is doing, read https://www.cs.cornell.edu/courses/cs5740/2016sp/resources/backprop.pdf.)

   TensorFlow matrix operations are very similar to NumPy. After creating a matrix variable with `tf.placeholder`, you can multiple and add these matrices with `tf.matmul` and `tf.add` (following the convention to `import tensorflow as tf`). Using these functions, you can create your hidden layers. Your MLP must have multiple hidden layers, although the exact number is your own choice (Don't forget to randomly initialize the weights and biases of each layer, for example with `tf.random_normal`!).

   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. Activation functions in TensorFlow that you should try are: `tf.nn.relu`, `tf.sigmoid`, and `tf.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 TensorFlow, you can use `tf.nn.softmax_cross_entropy_with_logits` along with `tf.reduce_mean` 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 TensorFlow comes with others built in, such as: `tf.train.GradientDescentOptimizer`, `tf.train.AdagradOptimizer`, `tf.train.MomentumOptimizer`, `tf.train.AdamOptimizer`.

Kaggle Instructions. We will use private Kaggle competitions to manage the evaluation and provide a leaderboard. *Despite the rules posted on Kaggle, no sharing between teams is allowed.* Until the assignment deadline, you may submit once per day to each competition (two daily submissions in total).

The competitions use 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 "classifier" 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!

Kaggle in-class registration: https://inclass.kaggle.com/account/register

In order to receive access to the in-class competition, you must sign up with a cornell.edu email address or add your Cornell email address to an existing account. Every person in your group must have their own Kaggle account. When you join each competition, you will be able to join as a group.

After using your classifier to predict labels for the test data (`propernames/test/test_data.csv` and `newsgroups/test/test_data.csv`), you must export the results to a CSV file in the same format as the development and test labels. Kaggle accepts your prediction CSV in any order, as long as each label is paired with the correct ID from the data file (you can output a CSV and test against the development labels to verify format before submitting to Kaggle).

Submission. Your submission on CMS should include a writeup in PDF format. In your write-up, be 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. 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. The writeup page limit is four pages. You do not need to submit your GitHub Classroom repository.

Grading. You will be graded to approximate evaluation in real life. The following factors will be considered: your technical solution, your development and learning methodology, the quality of your code, and the performance of your model on our held-out super secret test data.