

EE 511 Machine Learning, Winter 2018: Homework 4

Due: Friday, Mar 2, beginning of class

1 Programming: Language Identification [100 points]

For this homework assignment you will be building a system to identify the language of written text. The text is taken from Twitter and there are nine Western European languages to choose from: English, Spanish, Portuguese, Galician, Basque, Catalan, French, Italian and German. You will be making a generative model, i.e., modeling $p(\text{text}, \text{language})$ instead of $p(\text{language}|\text{text})$. **You can use either a Markov model (or higher-order Markov model) or a recurrent neural network (RNN) but are not required to do both.** (You can also do an HMM, but that is likely to be more work than it is worth.) If you want to use a deep-learning library like Tensorflow then the RNN is the best option. Otherwise, the Markov model is probably the best option. Both models are capable of getting good results at this task.

Separate files are provided for the training, validation and evaluation data. The data is tab separated. The first column uses a two letter code to indicate the language and the second column has the text of the Tweet.

1. Warm-up: Perplexity of a Unigram Model

- (a) Generative models for text are called language models. The first thing to do when building a language model is to define the vocabulary. We will be processing the text one character at a time. A common way of selecting the vocabulary is to choose all symbols that occur at least k times in the training data. Define your vocabulary to be the set of characters that occur at least 10 times in the training data. Everything else should be mapped to a special token for out-of-vocabulary tokens that should also be included in the vocabulary. There are two additional special tokens that should be included in the vocabulary: the start token $\langle S \rangle$ and the end token $\langle /S \rangle$. These are added to the beginning and the end of each Tweet.

Any method for splitting the text into characters is fine. If you are using Python just do `list(text)` to get a list of characters for the string `text`. Sometimes certain Unicode characters will get split into two or more items and sometimes they won't depending on how you do it or which version of Python you use. Don't worry about that issue for this assignment.

Report the size of the vocabulary you found and the percentage of out of vocabulary tokens.

- (b) The most popular way to measure the performance of a language model is using perplexity. Perplexity is defined as $2^{H(X)}$ where $H(X)$ is the entropy defined by

$$H(X) = - \sum_x p(x) \log_2 p(x) \approx - \frac{1}{n} \log_2 p(x)$$

where we approximate $H(X)$ with the cross-entropy associated with the empirical distribution (n is the number of test or validation set samples) when we don't know the true distribution and $p(x)$ is our estimate.

The unigram model corresponds to an assumption that the process is independent and identically distributed (i.i.d.). Compute the unigram probability distribution over the vocabulary using a relative frequency estimate. The end of sentence token should be included for the perplexity calculation but the start of sentence token should not. (For this problem, treat all the text as if

it were from a single language.) Report the entropy and the perplexity for this distribution using the validation data.

Sanity check: What are the maximum and minimum possible values of the perplexity for any distribution over the same set of characters?

2. Option A: Language-dependent Markov Models

Since you know the language of each tweet in the training data, you can partition it based on language and learn separate Markov models for each language. In other words, estimate $\Theta_l = \{\pi_j^{(l)}, a_{ij}^{(l)}\}$ for each language l , where $\pi_j = P(x_1 = j)$ and $a_{ij} = p(x_t = j | x_{t-1} = i)$. You can use the formulas derived in class, except that you need some sort of smoothing. Alternatively, you can use a language modeling toolkit such as SRILM,¹ which would provide smoothing options and allow you to use higher-order Markov process models. You can choose model order and smoothing method using the validation set. Be sure to use the vocabulary that you found in part (1) for all languages, since you will need to cover all possible characters in testing. If you are not using a toolkit that does it for you, make sure to add the start and end of sentence tokens to the beginning and end of each Tweet.

Create a table with the perplexity of each of your language models on each of the language-dependent validation sets. Sanity check: you should get lower perplexity for a Markov model on the language that it was trained on than what you got with the unigram model in part 1. You should also get higher perplexity when the training and validation languages do not match.

3. Option B: Recurrent Neural Network

At each timestep the RNN will read in a character and output a prediction for the next character in the sequence. The training objective is to maximize the log-likelihood of the data. One possibility is to train a separate model for each of the seven languages. Some downsides of this approach is that it can take much longer to train all of those models and it does not allow the model to share information between languages. A better approach is to train a single RNN that jointly models all of the languages. At each time step feed in both a character and a language embedding. The language embedding tells the RNN which language it is modeling and allows it to adjust its predictions accordingly. During evaluation, we can iterate over the set of language embeddings to find which one maximizes the probability of the text.

Here is an outline of how to make the RNN model. Some of the details are tailored towards a Tensorflow implementation. You can ignore those if you use a different library like Keras or Torch.

(a) Preprocessing

In this problem, sequence length refers to the number of characters in a Tweet and batch size refers to the number of Tweets processed in parallel. Batch sizes greater than one are used to get better estimates of the gradients.

Prepend all of the sequences with the start of sentence token $\langle S \rangle$ and append the end of sentence token $\langle /S \rangle$. It is convenient to define a maximum sequence length and then pad all the sequences to have that length. You can use the $\langle /S \rangle$ token for padding. Keep track of the original (before padding) length of each sequence. You also need to create a table that can map characters to ids and a separate table that maps languages to ids.

(b) Embedding Layer

The first layer of the RNN is an embedding layer. You will have two types of embeddings, one for characters and one for languages. The size of these embeddings can be small (less than 15 for the characters and less than 6 for languages). At each timestep, you will input a character id and a language id and your model will lookup the embeddings for those ids in the two embedding tables. After the lookup, the embedding vectors should be concatenated together to form a single vector representing the character-language pair.

¹<http://www.speech.sri.com/projects/srilm/>

(c) **Recurrent Layer**

There are several different types of RNN's besides the simplest one (known as `tf.nn.rnn_cell.BasicRNNCell` in Tensorflow). Feel free to use one of the other variants such as Gated Recurrent Units (`tf.nn.rnn_cell.GRUCell`) or long short-term memory cells (`tf.nn.rnn_cell.LSTMCell`). LSTMs usually perform the best, but also take longer to train.

You can use the `tf.nn.dynamic_rnn` command to process an entire sequence with the RNN. This command outputs two objects. You just need the first one, which is a concatenation of the output vectors from each timestep.

(d) **Output Layer**

The output from the RNN hidden layer is a three dimensional tensor with dimensions corresponding to the batch size, sequence length, and the size of the RNN. The next step is to multiply this tensor with the output character embeddings. In Tensorflow, you will have to reshape the 3D tensor to 2D in order to do the matrix multiplication. You can either use the transpose of the input character embeddings or create new parameters for the output character embeddings. If you do the former, you will need to do a linear projection to get the output from the RNN down to the right size. Add a bias vector to the product of the character embeddings and the RNN output.

(e) **Compute Loss & Train Model**

Train the model to maximize the log probability, equivalent to minimizing the cross-entropy. The labels are the ids of the tokens in the next position in each sequence. When computing the loss it is important to mask the padding tokens at the end of each sequence.

After training the model, report the perplexity on the validation data. As a reminder, perplexity is computed by exponentiating the average log probability of each token. As a sanity check, see if your perplexity is less than the number you computed in the warm-up problem. You can choose your parameters based on what gives the best perplexity on the validation data. The best possible model on this data will probably take more than 24 hours to train without a GPU. When preparing the assignment, your TA was too lazy to wait more than five to ten minutes for each model to train. So, don't feel obligated to train longer than that. You aren't expected to get a state-of-the-art result.

4. Language Identification

After training the model (either an RNN or a Markov model), you can now use it to identify the language of a Tweet. The classification rule is to assign the label according to $p(l|text)$. Don't use the class priors $p(l)$ to make classification decisions in order to give more weight to the rarer languages.

Report the overall accuracy on the valid data. The second key metric is the average F1 score. The F1 score is a way of summarizing precision and recall and is calculated as $2 \times \frac{precision \times recall}{precision + recall}$. The average F1 score is the (unweighted) simple average of the F1 scores for the individual languages. Compared to accuracy, the average F1 score places a much greater emphasis on doing well at identifying the most difficult languages.

Things to submit

- Submit all your code and include a README file explaining how to run your code.
- Run your model on the test data which does not have the language ID. Submit a file named **test.ids** containing the language ID your model predicts. Please make sure you do not change the order of the test samples. Each line should only contains the language ID for the corresponding test sample.