Assignment 2

CMSC 473/673 — Introduction to Natural Language Processing

Item	Summary
Assigned	Monday October 9th
Due	Wednesday October 25th, 11:59 PM Baltimore time
Topic	Classification and Evaluation
Points	110

In this assignment you will gain experience with implementing, training, and evaluating classifiers.

You are to *complete* this assignment on your own: that is, the code and writeup you submit must be entirely your own. However, you may discuss the assignment at a high level with other students or on the discussion board. Note at the top of your assignment who you discussed this with or what resources you used (beyond course staff, any course materials, or public Discord discussions).

The following table gives the overall point breakdown for this assignment.

Question	1	2	3	4	5
Points	30	20	25	10	25

How you structure your code is up to you. Your code does not have to be pristine. However, we should be be able to follow what you write. Ensure your code is appropriately commented and described in your writeup. Try to make the coding easy on yourself.

What To Turn In Turn in a writeup in PDF format that answers the questions; turn in all requested code necessary to replicate your results. Be sure to include specific instructions on how to build (compile) your code. Answers to the following questions should be long-form. Provide any necessary analyses and discussion of your results.

How To Submit Submit the assignment on the submission site:

https://www.csee.umbc.edu/courses/undergraduate/473/f23/submit

Be sure to select "`Assignment 2`."

Full Questions

1. (**30 points**) Consider the problem of determining whether or not a sentence is acceptable (grammatical) due solely to noun-verb agreement. (This is one of the most common instance of English's inflectional morphology.) For example, "she loves NLP" and "they love NLP" would be considered acceptable, while "he love NLP" would not. In this problem, you will work through developing a *very* small scale classifier for this problem. This question requires small-scale computation, but there is no implementation required for this problem. That said, you may find it helpful to use Pytorch to aid your computations. However, if you do use Pytorch and you do use `nn.Linear`, you'll probably want `bias=False` when instantiating it.

   Specifically, assume that your input will be an English sentence. For a sentence, determine whether it is grammatically acceptable or not. You can assume that grammatical acceptability will be determined solely based on noun-verb agreement. You'll model this with a maxent classifier.

   (a) As formulated, is this a binary or multi-class classification problem?

   (b) Provide 2-3 example features that you might define. For each of these features, provide an English explanation, a light mathematical / pseudo-code formulation, and the values they would result in for x_1 ="she loves NLP" and x_2 ="he love NLP."

   (c) Let θ be the collection of weights in your maxent model. How many weights are there?

   (d) Define your own non-trivial, non-uniform values for θ, and let θ_l be the vector of weights associated with label l. For each instance i and label l, we define the **logit** as

   $$u_{i,l} = \theta_l^T f(x_i).$$

   Using the values you've set for θ, compute the logits for each of x_1 and x_2.

   (Non-trivial and non-uniform θ: you can pick "nice" values to use, e.g., -1, 0, 1, 2, etc. But, θ should not uniformly be any of those values, e.g., θ should not be uniformly 0.)

   (e) Using these logits, compute the normalizers $Z(x_1)$ and $Z(x_2)$ that will let us turn the logits u_1 and u_2 into proper posterior distributions $p(y|x_1)$ and $p(y|x_2)$.

   (f) Using these normalizers, compute $p(y|x_1)$ and $p(y|x_2)$.

   (g) Use the `argmax` decoding rule (e.g., see deck 4, slide #5) to identify the labels that would be predicted for each of x_1 and x_2 using the current values of θ.

   (h) Now, what if θ were uniformly 0. What are the values of $p(y|x_1)$ and $p(y|x_2)$?

2. (**20 points**) Each of the following scenarios describes the result of some (made up) classifier: there is a list of the correct labels, and the corresponding predictions. For each of the following situations, compute accuracy, recall, precision, and F_1 score. You may verify your answers with code, but you must show work, or some intermediate steps, to receive full credit on this problem.

(a) A binary classification result, where the correct labels are `[T, T, F, T, F, T, F, T]` and the predicted labels are `[T, F, T, T, F, F, F, T]`. Assume `T` means "true" (the desired class) and `F` ("false") is the "default" class. Compute accuracy, and compute recall, precision, and F1 for `T`.

(b) A binary classification result, where the correct labels are `[T, F, F, F, F, F, F, T]` and the predicted labels are `[F, T, F, F, F, F, F, F]`. Assume `T` means "true" (the desired class) and `F` ("false") is the "default" class. Compute accuracy, and compute recall, precision, and F1 for `T`.

(c) A multiclass classification result, where the correct labels are `[T, F, U, F, F, F, U, T]` and the predicted labels are `[F, T, U, F, F, F, F, T]`. Assume `T` means "true," `U` means "maybe," and `F` ("false") is the "default" class. Compute accuracy. Then, compute, at both the micro and macro levels, average recall, precision, and F1. Explain why you should *not* include the `F` class in the precision, recall, and F1 computations.

(d) A multiclass classification result, where the correct labels are `[C, C, A, C, C, C, C, C, B, A, C, C, C]` and the predicted labels are `[C, C, C, C, C, C, C, C, B, A, A, C, C]`. In this example, there is no "good" default option, so we can consider `A`, `B`, and `C` to be all possible classes/labels of interest. Compute accuracy. Then, compute, at both the micro and macro levels, average recall, precision, and F1. Explain why you *should* include all classes in the precision, recall, and F1 computations.

3. (**25 points**) In the previous assignment, you explored the UD EWT dataset using the tokenization that was provided. However, in class, we showed how sometimes you may need to use alternative tokenizations, especially if you want to use some modern models. For this question,

   - First, load up the `bert-base-cased`. Try tokenizing the following three "words": November, Novembrer, and Novembr. What do you notice about the output? Speculate on why you're seeing that output.

     You can load the tokenizer by using the key "bert-base-cased" as the argument to the `from_pretrained` function (rather than "bert-base-uncased" as used in class).
   - Using the `bert-base-cased` tokenizer, tokenize the training portion of the UD EWT corpus.
   - In Assignment 1, you defined the vocabulary based on the words that occurred as the tokenized text in the training corpus. Compare how large that vocabulary is compared to the vocabulary that is defined by the Bert tokenizer.
   - Identify some of the tokens that are defined by the Bert tokenizer that were not part of the Assignment 1 vocab. (Here, "some" means more than 2. But, you can keep it to under 7 for ease.) What do you notice about these tokens? To help answer this, you may want to look at the contexts in which they would appear.

- What is one potential benefit from defining a vocab based off of a particular corpus (vs. using a vocabulary predefined by an existing tokenizer)? Likewise, what is one potential benefit from using a vocabulary predefined by an existing tokenizer (vs. defining a vocab based off of a particular corpus)?

4. (**10 points**) One very important aspect of NLP is the ability to identify good and reasonable baseline systems. These baseline systems help you contextualize any progress (or harm) your proposed approach makes. Coming up with these baselines is not always easy. However, a very common one is called the "most frequent class" baseline (also sometimes called the R0 classifier). This most frequent class baseline simply identifies the most common label y' from the training set, and then when presented with *any* evaluation instance, simply returns y'.

   For this question, I strongly recommend using the existing implementations of accuracy, recall and precision in the Python library `sklearn`, e.g., `sklearn.metrics.precision_score` and `sklearn.metrics.recall_score`.

   (A) Consider a small training set with 5 instances, where the labels of each of those 5 instances are $\alpha, \alpha, \beta, \gamma, \alpha$. What label would the most frequent baseline return?

   (B) Working from the GLUE RTE corpus, implement a most frequent class baseline for predicting entailment. On only the dev, evaluate this baseline with 5 measures: accuracy, macro precision, macro recall, micro precision, and micro recall.

   Turn in your code, these 5 scores, and a brief paragraph addressing your observations and analysis of how reasonable the predictions made by this model are.

5. (**25 points**) Now implement a maxent classifier to predict entailment. You will only evaluate on the dev set. Use the same metrics as above. You may use the scaffolding code provided to start your solution. Specifically:

   (a) Train a model using the lexically-defined word overlap features. If you use the scaffolding code, you'll need to implement the classifier as a `nn.Module` subclass, choose the appropriate loss function, successfully train the model, load and tokenize the dev set, and then evaluate the predictions made on the dev set.

   (b) Train and evaluate a model using dense embeddings, e.g., those returned by a `BertModel`.

   (c) Compare all the results you've gotten (including those of the most-common baseline). As part of this comparison, briefly discuss the strengths and limitations of each of the lexically-defined and embedding-based features.