Tutorial: Generalized Iterative Scaling for Maximum Entropy Modeling

In class, we discussed the idea that maximum entropy models can be trained using the conditional maximum likelihood criterion. In this tutorial we explore the following problem: you have a number of candies in a bag, each of which are either Cherry or Strawberry flavored. The candies have features that can be observed: they are either square (S) or circular (C), and they either have wrappers that are red (R) or yellow (Y). Your job is to construct a maximum entropy model that associates the features with the candy flavor, train it using the given training data, and is able to predict the probability of flavor given shape and color.

To generate the training data, I have created a python script called `gen_data.py`, which creates a biased sample of different flavors given a sampling of different feature combinations. If you examine the code closely, you can see the priors on the feature combinations (which are correlated) and the probability of flavor given each feature set.

To generate data, run

```
python gen_data.py --num 1000 > data
```

Remember that a Maximum Entropy model defines its probability distribution in terms of a normalized exponential sum of weighted features:

$$P(\text{flavor}|\text{features}) = \frac{e^{\sum_i \lambda_i \times f_i(\text{flavor},\text{features})}}{z}$$

where $z = \sum_x e^{\sum_i f_i(x,\text{features})}$. In this case, we will model the distribution using eight feature functions:

$$\begin{aligned}
f_0(\text{Cherry}, \text{R}) = 1 &\iff \text{label} = \text{Cherry} \,\&\, \text{color} = \text{Red} \\
f_1(\text{Cherry}, \text{Y}) = 1 &\iff \text{label} = \text{Cherry} \,\&\, \text{color} = \text{Yellow} \\
f_2(\text{Cherry}, \text{S}) = 1 &\iff \text{label} = \text{Cherry} \,\&\, \text{shape} = \text{Square} \\
f_3(\text{Cherry}, \text{C}) = 1 &\iff \text{label} = \text{Cherry} \,\&\, \text{shape} = \text{Circle} \\
f_4(\text{Strawberry}, \text{R}) = 1 &\iff \text{label} = \text{Strawberry} \,\&\, \text{color} = \text{Red} \\
f_5(\text{Strawberry}, \text{Y}) = 1 &\iff \text{label} = \text{Strawberry} \,\&\, \text{color} = \text{Yellow} \\
f_6(\text{Strawberry}, \text{S}) = 1 &\iff \text{label} = \text{Strawberry} \,\&\, \text{shape} = \text{Square} \\
f_7(\text{Strawberry}, \text{C}) = 1 &\iff \text{label} = \text{Strawberry} \,\&\, \text{shape} = \text{Circle}
\end{aligned}$$

The key to understanding the training is that the conditional maximum likelihood is maximized when the expected counts of label-feature pairs given the model matches the observed feature counts when the label is known. This falls directly out of taking the derivative of the log posterior with respect to each weight and setting it to zero (we showed this in class).

One algorithm for updating weights in a maximum entropy model is the Generalized Iterative Scaling algorithm (Algorithm 1)– this is a relatively simple algorithm that updates the lambda weights for each feature by directly comparing the expected counts of each label-feature pair with the observed counts of the label-feature pair in the training data. Take a close look at the algorithm - it iterates over the data several times, computing a probability of each label (i.e. ignoring the true label) given the features, and then uses that probability to derive the feature expectation.

input : a set of data, each datum with a label and associated features
output: a set of lambda weights that are parameters of the MaxEnt model

F = number of distinct features;
cmax = maximum count of features;
observed[0..F] = count of each observed feature;
weights $\lambda_{[0..F]} = 0$;
while *not converged* **do**
  expected[0..F] = 0;
  foreach *training instance j* **do**
    z=0;
    /* Step 1: Compute the sum of the active feature functions for each label, and compute normalization term z */
    foreach *output label y* **do**
      s[y] = 0;
      foreach *feature i such that* $f_i(x_j, y) \neq 0$ **do**
        $s[y] += \lambda_i \times f_i(x_j, y)$;
      end
      $z += e^{s[y]}$;
    end
    /* Use feature function sums to compute probability of class label y given features, and use to compute expectation of features */
    foreach *output label y* **do**
      foreach *feature i such that* $f_i(x_j, y) \neq 0$ **do**
        expected[i] += $f_i(x_j, y) \times e^{s[y]}/z$;
      end
    end
  end
  /* Update lambda parameters by making a step in the direction to equalize expected and observed counts */
  foreach *feature function i* **do**
    $\delta_i = \frac{1}{cmax} \log \frac{\text{observed}[i]}{\text{expected}[i]}$;
    $\lambda_i += \delta_i$
  end
end

Algorithm 1: Pseudocode for Generalized Iterative Scaling, adapted from Goodman (2002)

Given that the data features are fixed, the model is adjusting the parameters of the probabilistic model to change the expected feature count. Convince yourself that the algorithm is doing the right thing generally for the following three cases:

1. When the expected count of a feature is lower than observed (i.e., the model underpredicts the label).
2. When the expected count of a feature is greater than observed (i.e., the model overpredicts the label).

3. When the expected count of a feature is equal to the observed (i.e., the model correctly predicts the probability of the label).

Code up the algorithm; if you would like a start (particularly for reading in the data) take a look at `maxent_gis_template.py`. As you are coding, can you identify where the probability of the label given the features gets calculated? I've also put the answer I came up with in `maxent_gis_template.py` (no fair peeking until you've tried).

If you finish this quickly here are some extensions to think about:

1. Add a feature that talks about both shape and color (e.g., $f(\text{Cherry}, \text{S}, \text{R})$) for square red Cherry candies. Does this change the lambda distribution at all? Does it predict more accurately? (You should evaluate on a test set generated using the same distributions as the data set.)
2. Add a third feature column (you need to change the data generator). Add a third flavor (lime).
3. Look up Goodman's paper on a faster GIS algorithm and implement it. Compare update times for different data sizes.

References:

J. Goodman, "Sequential Conditional Generalized Iterative Scaling," Proc. Annual Meeting of the Association for Computational Linguistics, Philadelphia, PA, 2002.