CSE 512: Homework 6 Due Nov. 21

1. **Multiclass classification** Consider the multiclass logistic regression optimization problem

$$\underset{\Theta\in\mathbb{R}^{n\times K}}{\text{maximize}} \quad f(\Theta) = \frac{1}{m}\sum_{i=1}^m \left(\sum_{k=1}^K y_{ik} x_i^T\theta_k - \log\sum_{k=1}^K \exp(x_i^T\theta_k)\right).$$

where $y_{ik} = 1$ if data sample i is in class k, and 0 otherwise. As usual, $x_i \in \mathbb{R}^n$ is the ith data feature. Here, we write the entire matrix variable as

$$\Theta = \begin{bmatrix}\theta_1 & \theta_2 & \cdots & \theta_K\end{bmatrix}.$$

(a) In terms of each θ_k, write the gradient of f with respect to θ_k.

(b) Argue that this function has a smoothness parameter $L = 2\cdot L_X$ where $L_X = \frac{1}{m}\sum_{i=1}^m \|x_i\|_2^2$

(c) The function

$$f(\theta) = \log\left(\sum_{i=1}^m \exp(\theta_i)\right)$$

is sometimes called the log-sum-exp function. As we saw in lecture, it has the nice property of acting like a soft-max function, by "pulling" away the largest values of θ_i, to somewhat exaggerate their "lead".

A downside of using the log-sum-exp function is that it can have numerical issues. If θ_i is somewhat big, then $\exp(\theta_i)$ becomes very big, and can cause overflow. Conversely if θ_i is very negative, then all the values may be too close to 0 and cause underflow.

The "log-sum-exp-trick" is a numerical trick which deals with this issue, by adding and subtracting a constant whenever necessary. In effect, we simply do

$$f(\theta) = \underbrace{\log\left(\sum_{i=1}^m \exp(\theta_i - D)\right)}_{f_1(\theta)} + D.$$

Then, for the right choice of D, we can prevent overflow and underflow.

Propose a value of D such that $f_1(\theta) \leq 1$ (preventing overflow), and another value such that $f_1(\theta) \geq 1$ (preventing underflow).

(d) **Coding.** Run multiclass logistic regression on MNIST dataset, against each of the 10 classes.

While usually we pick a stepsize of $2/L$, I have tried this and found a larger stepsize of 10^{-5} will work well. Use this stepsize and run for 1000 iterations, or however many you need to see reasonable "working" behavior. Show the train/test loss plot and the train/test misclassificaton plot.

2. **Directed graphical models and probability inference** I have 4 tops: a red sweater, a blue T-shirt, a green hoodie, and a white tank top. I need your help to decide what to wear.

(a) I decide what to wear based on four factors:

- if it's raining
- if I want to take a walk outside
- if I feel sick
- the day of the week it is

Using the Naive Bayes assumption, draw a graphical model that indicates how I will make a decision of what to wear each day.

(b) To be more specific,

- I only wear the green hoodie when I walk outside, regardless of all other factors.

- If I feel sick, I will walk outside 10% of the time. If I feel well, I will walk outside 60% of the time.
- When it rains, I feel sick 70% of the time; otherwise, I feel sick 15% of the time.

Draw a corresponding graphical model for determining whether I wear a green hoodie. Given that it is raining, infer the probability that I am wearing a green hoodie.

(c) The probability that I wear a tank top, independently of all the other clothes, is 75% if it's raining and 25% if it's not raining.

- Today is Monday and it is raining.
- The probability that it will rain, given that the previous day rained, is 70%. The probability that it will rain, given that the previous day did not rain, is 10%.

Draw a graphical model predicting whether I will wear a tank top on Wednesday, and calculate this probability.

3. Run the ipython notebook `gmm_release.ipynb`. It will load a vector $x \in \mathbb{R}^{10000}$, which contains 10000 samples in $\mathbb{R}$ (scalars). It will also plot a histogram of the data.

(a) Implement a K-means method that identifies the cluster centers of this dataset. Plot the points vs class label using

```
plt.plot(x,class_label)
```

to see the separation by x value. Plot also the cluster centers in a different color.

(b) Implement a Gaussian mixture model that identifies not just the cluster centers but also their standard deviation. Plot the p.d.f. against the histogram, to see how close it is.

In both cases, try several different initiations to get something that looks reasonable. If you do need many initializations, describe your strategy, e.g. picking random centers, eyeing and guessing, etc.

4. **Hidden Markov Model spellchecker** In this exercise we will make a spell-checker using a HMM. To do this, download `alice_nlp_release.ipynb` and follow the instructions.

- Read through the first two blocks to get an idea of what the task is. The idea is to go through the corrupted corpus, identify words which have probably been corrupted, and correct them probabilistically.
- In the 4th box, fill in the functions to construct the word probabilities (weighted frequencies in uncorrupted corpus) and transition matrix (which gives Pr(word | prev word)). If done correctly, the lines printed out should read

```
prob. of "alice" 0.014548615047424706
prob. of "queen" 0.002569625514869818
prob. of "chapter" 0.0009069266523069947
prob. of "the alice" 0.00025406504065040653
prob. of "the queen" 0.016514227642276422
prob. of "the chapter" 0.012957317073170731
```

- In the 5th box, fill in the function for computing the emission probability. The first 10 words closest to Alice should be

```
['abide', 'alice', 'above', 'voice', 'alive', 'twice', 'thick', 'dance', 'stick', 'prize']
```

- Construct and run your Hidden Markov Model spell checker using the functions computed for the prior probabilities, emission probabilities, and transition probabilities. List some words whose spelling was corrected correctly, and some examples where the spell-correcter did not work as expected. Report the recovery rate of the "fixed" corpus.

Challenge!

How to deal with multiple advisers I have m advisers, and I don't know which one of them to trust. Every day $t = 1, ..., T$, I ask all m advisers a prediction question, which they answer yes $y_i^{(t)} = 1$ or no $y_i^{(t)} = -1$ $(i = 1, ..., m)$. The true answer on each day is denoted by $y_\star^{(t)} \in \{-1, 1\}$.

Each day, I have to make a guess as to what $y_\star^{(t)}$ has to be, which I do using the linear predictor

$$\hat{y}_{\text{pred}}(w, y^{(t)}) = \mathbf{sign}\left(\sum_{i=1}^m w_i y_i^{(t)}\right).$$

The variables w_i, $i = 1, ..., m$ are positive weights I place on each adviser, to indicate my trust in that adviser. I constrain $0 \le w_i \le 1$ and require $\sum_{i=1}^m w_i = 1$ (w represents a probability mass function.)

To learn how much I trust each adviser, I optimize the following convex optimization problem, over the weights w_i, over T days of observation:

$$\begin{array}{ll} \underset{w}{\text{minimize}} & f(w) = \displaystyle\sum_{t=1}^T \underbrace{\left(\sum_{i:y_i^{(t)} \neq y_*^{(t)}} w_i - \sum_{i:y_i^{(t)} = y_*^{(t)}} w_i\right)}_{=:f^{(t)}(w)} \\ \text{subject to} & 0 \le w \le 1 \\ & \displaystyle\sum_{i=1}^m w_i = 1 \end{array} \tag{1}$$

1. **Online learning method.** To learn the correct weighting, I run the following method. I start by setting $w_i^{(0)} = 1/m$ for all i. Then, every day, if an adviser is correct, I update that adviser's weight as $\hat{w}_i^{(t)} = w_i^{(t-1)} e^\mu$ for some $\mu \ge 0$. If the adviser is not correct, I leave the weight alone $\hat{w}_i^{(t)} = w_i^{(t-1)}$. Then I reweight, e.g.

$$w_i^{(t)} = \frac{\hat{w}_i^{(t)}}{\sum_{j=1}^m \hat{w}_j^{(t)}}.$$

   This method is known as the *exponential weighting method.*

   Suppose that each adviser has a probability p_i of being correct. Show that as $T \to +\infty$, this method converges to exclusively listening to the adviser which is most often correct, e.g.

$$w_i^{(t)} \overset{t \to +\infty}{\to} \begin{cases} 1 & \text{if } i = \underset{j}{\text{argmax}} p_j \\ 0 & \text{else.} \end{cases}$$

   Hint: Note that the method doesn't really change if you reweight at each iteration, vs reweighting at the very end.

2. **Projected incremental gradient descent.** We can also think of a gradient descent method to solve (1), where at each iteration, we take a gradient step

$$w^{(t+1)} = \mathbf{proj}_{\Delta_{m-1}}\left(w^{(t)} - \eta \nabla f^{(t)}(w^{(t)})\right)$$

   Show that taking a gradient step

$$\hat{w} = w^{(t)} - \eta \nabla f^{(t)}(w^{(t)})$$

   essentially results in subtracting or adding a constant value to $w^{(t)}$ each time an adviser gets an answer wrong or right, respectively.

3. **Mirror descent.** The set $\Delta_{m-1} = \{w : 0 \le w \le 1, \sum_{i=1}^m w_i = 1\}$ is called the *probability simplex.* Projecting on this set is actually not very trivial, and involves sorting all the elements (which, if we remember from algorithms, is

at least $O(m \log m)$. So, while we could run a projected gradient descent method to solve (1), we will try a different trick instead.

We do something called the *mirror descent method.* Basically, this method is a projected gradient method, but with a transformed variable. In particular, our *mirror map* is going to be the gradient of the strictly convex function

$$g(u) = \sum_{i=1}^{m} u_i \log(u_i) \qquad \text{(negative entropy function).}$$

Derive the mirror map, which is $\nabla g(u)$, and the inverse mirror map. That is, find Φ and Φ^{-1} where

$$\Phi(u) = \nabla g(u), \qquad \Phi^{-1}(\nabla g(u)) = u.$$

4. The incremental mirror descent method can then be summarized as

$$\begin{aligned} \Phi(\hat{w}) &= \Phi(w^{(t)}) - \nabla f(w^{(i)}) \\ \Phi(w^{(t+1)}) &= \Phi(\mathbf{proj}_{\Delta_{m-1}}(\hat{w})). \end{aligned}$$

Or, to write it in a way that is more implementable,

$$\begin{aligned} \hat{w} &= \Phi^{-1}\left(\Phi(w^{(t)}) - \nabla f(w^{(t)})\right) \\ w^{(t+1)} &= \Phi^{-1}\left(\Phi(\mathbf{proj}_{\Delta_{m-1}}(\hat{w}))\right). \end{aligned}$$

Show that this method is equivalent to the exponential weighted algorithm in part (a).