Generalized Linear Classifiers Practical

1. Implement the multi class perceptron algorithm

Training data: $\mathcal{T} = \{(\boldsymbol{x}_t, \boldsymbol{y}_t)\}_{t=1}^{|\mathcal{T}|}$

1. $\mathbf{w} = 0$
2. for $n : 1..N$
3. for $t : 1..T$
4. Let $\boldsymbol{y}' = \arg\max_{\boldsymbol{y}'} \mathbf{w} \cdot \mathbf{f}(\boldsymbol{x}_t, \boldsymbol{y}')$
5. if $\boldsymbol{y}' \neq \boldsymbol{y}_t$
6. $\mathbf{w} = \mathbf{w} + \mathbf{f}(\boldsymbol{x}_t, \boldsymbol{y}_t) - \mathbf{f}(\boldsymbol{x}_t, \boldsymbol{y}')$
7. return $\mathbf{w}$

- Located at http://ryanmcd.googlepages.com/perceptron.tar.gz is some starter code in Java
- This code should make things easier, but you can use whatever you like
  - The code contains simple files for processing the data
  - There are comments where you need to fill things in Perceptron.java and PerceptronModel.java
  - Perceptron.java – Main class, where you will code the main algorithm
  - PerceptronModel.java – contains all model parameters (i.e., weights w). Also has methods for predicting a label given an instance
  - Instance.java – Represents a pair (x,y) of inputs and labels. Methods allow you to get f(x,y') for any y' or the score w*f(x,y')
  - DataReader.java – File to read in the data
- Test it on train.txt and test.txt in the data directory
- Data format is:

  label f1:v1 f2:v2 ... fm:vm
  label f1:v1 f2:v2 ... fm:vm
  ...
  - Each line is a new instance
  - label is an integer that is either 0, 1, 2 (three class prediction problem)
  - f1 is an integer feature and v1 is the value of the feature (in this case all binary)
  - The representation is sparse – not all features are included. You can assume this means that that features value is 0.
  - Of course, if you use the DataReader.java file, you should not have to worry about this
  - Everything is represented as integers instead of strings, just to make it easier to use primitives in whatever language you are using
  - The data is a set of reviews and the labels indicate whether it is a review about italian, chinese or mexican food

- To compile starter code type "javac Perceptron.java"
- To run code type "java -Xmx500m java Perceptron data/train.txt data/test.txt"
- Once you have filled in the sections the training accuracy should be in the high 90s and the testing accuracy in the low to mid 80s. What accuracy did you get?

2. Averaged perceptron

   - Maintain a weight vector $\mathbf{w}_{\text{avg}}$ that is the average of all weight vectors after every iteration. After training return this weight vector instead of the final weight vector.

   Training data: $\mathcal{T} = \{(\boldsymbol{x}_t, \boldsymbol{y}_t)\}_{t=1}^{|\mathcal{T}|}$

   1. $\mathbf{w} = 0;;\ \mathbf{w}_{\text{avg}} = 0$
   2. for $n : 1..N$
   3. for $t : 1..T$
   4. Let $\boldsymbol{y}' = \arg\max_{\boldsymbol{y}'} \mathbf{w} \cdot \mathbf{f}(\boldsymbol{x}_t, \boldsymbol{y}')$
   5. if $\boldsymbol{y}' \neq \boldsymbol{y}_t$
   6. $\mathbf{w} = \mathbf{w} + \mathbf{f}(\boldsymbol{x}_t, \boldsymbol{y}_t) - \mathbf{f}(\boldsymbol{x}_t, \boldsymbol{y}')$
   7. $\mathbf{w}_{\text{avg}} += \mathbf{w}$
   8. return $\mathbf{w}_{\text{avg}} / (N \times T)$

   - How does this change accuracy? Try training with the data/train_sort.txt. What happens?
   - General discussion.

What you need to submit

- All the .java files from the assignment (including the ones you did not modify)
- A short report that contains
  - The accuracies on the training and on the testing data for the normal perceptron and the averaged perceptron
  - These accuracies when using train_sort.txt
  - A brief explanation for why you think the average perceptron improves performance over the original. There is no single correct answer here since there are actually a couple of valid reasons.
- Send all these by email to Joakim and Ryan