CSE 512: Homework 5 — Due Wednesday November 13

1. **LogitBoost (4 pts)** In our lecture we saw how Adaboost can be seen as a greedy coordinate-wise method that minimizes the empirical exponential loss at each step. Now, we attempt to make similar conclusions for LogitBoost (Alg. 1)

Algorithm 1: Logitboost

```
Data: Samples: $x_1, ..., x_m$, training labels $y_i \in \{-1, 1\}$, weak learners $\mathcal{H}$.
Result: Classifier $H(x) = \sum_{t=1}^T h^{(t)}(x)$
Initial weights $w_i^{(0)} = \frac{1}{m}$ for $i = 1, ..., m$;
$H^{(1)}(x) = 0$. for $t = 1, ..., T$ do
    Update
        $z_i^{(t)} = \frac{\frac{y_i}{2} + \frac{1}{2} - p_i^{(t-1)}}{p_i^{(t-1)}(1 - p_i^{(t-1)})}, \qquad w_i^{(t)} = p_i^{(t-1)}(1 - p_i^{(t-1)}), \qquad p_i^{(t)} = \sigma(H^{(t)}(x_i))$
    Train weak learner over weighted regression
        $h^{(t+1)} = \operatorname*{argmin}_{h \in \mathcal{H}} \sum_{i=1}^m w_i^{(t)} (z_i^{(t)} - h(x_i))^2 \qquad (*)$
    Merge learner
        $H^{(t+1)} = H^{(t)} + h^{(t+1)}$
end
```

(a) **Greedy behavior.** Begin by considering the logistic minimization

$$\underset{H \in \mathcal{H}}{\text{minimize}} \sum_{i=1}^m \underbrace{\log(1 + \exp(-y_i h(x_i)))}_{\mathcal{L}(h)} \qquad \text{(Logistic loss)}$$

In particular, let's say I'm updating the tth weak learner, by approximating

$$\underset{h_t \in \mathcal{H}}{\text{minimize}} \sum_{i=1}^m \mathcal{L}(H_{t-1}(x_i) + h_t(x_i))$$

i. **(1pt)** Write the second order Taylor approximation of $\mathcal{L}(H_{t-1} + h_t)$, over variable h_t, in terms of $p_i^{(t-1)}$

ii. **(2pts)** Show that minimizing the second order Taylor series approximation over $\mathcal{H}$, summed over all samples, yields the same minimization problem as (*).

(b) **Coding. (1 pt)** Open the mnist_logitboost_release directory and in the iPython notebook, download the data. We are again going to do 4/9 handwriting disambiguation. Only minimal preprocessing was used in this dataset. You may now use sklearn's regression tree implementation.

- Write a function that computes the train and misclassification rates , as well as the train exponential loss value, using just this decision stump (tree of depth = 1).
- **Deep trees.** Using the tree library from sklearn, train trees with depth 1 to 20, and plot the train and test misclassification rate, as well as the train exponential loss, as a function of depth. Report also the smallest train and test misclassification rate, and smallest train exponential loss, over the sweep.
- **Boosted decision stumps.** Now build a decision stump, e.g. a tree with depth = 1. Initialize weights as $w_i = 1/m$ for all $i = 1, ..., m$, and fit the decision tree over the *weighted* misclassification error, using the code snippet

```
clf = clf.fit(Xtrain, ytrain, sample_weight = w).
```

Implement the Logitboost method, as shown in algorithm 1. Plot the training logistic loss, and train and test misclassification rate. Comment on how the boosted decision stumps performed compared to the deep decision tree.

2. **GMM problem. (3pts)**

We will now walk through the steps of coding up a Gaussian mixture model.

(a) First, we will implement the procedure on a curated 2-D problem, which will help us work out bugs, and visualize our result, before figuring out how to extend this to a higher dimensional problem.

- Open `mnist_GMM.ipynb`. We will load the MNIST dataset, and build a Gaussian mixture model over a 2-D PCA representation of it. The reduced dimensionality will help us with some numerical stability, and can make the landscape easier to visualize.
- Before doing anything, fill in the `gaussfn(x,mu,C)` function with the formula for the pdf of a multivariate Gaussian with mean vector μ and covariance matrix C. The input x should be of dimension $2 \times m$, where m may be the number of training samples in 2 dimensions, and the output should be a vector number of length m, representing the PDF at the m points.
- I have provided for you two functions: `plot_label(y)`, which plots the datapoints with different colors for each label, and `plot_gauss(alpha,mu,Clist)` which plots contours of Gaussians, weighted by α, with centers μ and covariances in the list `Clist`. Note that this function depends on `gaussfn` being coded correctly!
- Fill in the four functions `get_pi`, `get_alpha`, `get_mu`, `get_Clist`.
- If you've done the previous part correctly, you should just be able to run the last two boxes and get the GMM!
- Turn in two plots, one with the Gaussian contour plot overlaying the points colored by their true labels, and one overlaying the points colored by their learned labels. Comment on any discrepancies.

(b) Now, we will extend this to the full mnist dataset, without any curation. This problem is harder because, in its full 784 dimensional form, we get a lot of singular matrices.

- Actually, it is still impossible to get it to work with $n = 784$, so the problem is *still* reduced first to $n = 196$ (4x subsampling) to start.
- The issue that arises in the higher dimensional regime is that the covariance matrix C is often so close to singular, especially in the unimportant dimensions, that inverting it is a huge pain. So, we will use a "lazy linear algebra trick", which is to add a tiny diagonal, $+0.1I$, to the covariance matrices C each time we compute them. This will get your code to converge much better.
- Now, for 10 different initialization, run the GMM method on the $n = 196$ dataset for 100 trials. Plot the *average purity*, e.g.

$$\text{purity}(\mathcal{S}) = \frac{\#\text{ of most frequent element in }\mathcal{S}}{\text{size of }\mathcal{S}}$$

  and the average purity is the purity over sets

$$\mathcal{S}_k = \{y[i] : \hat{y}[i] = k\}$$

  e.g. the homogeneity of the true labels in each cluster.

  Put this plot in the PDF. Is GMM sensitive to initialization? Is it a successful unsupervised learning technique for clustering handwritten digits? Would you suggest any improvements?

3. **Movie recommendations. (3pts)** We are going to try to replicate the famous Netflix prize recommendation system, but using an incredibly baby version of the model. (You don't have two years to complete this assignment, after all.) Open `movie_recommendations.ipynb`. I have downloaded data from the MovieLens databank [1] partitioned into 33% train, 33% validation, and 34% test.

   We will build a movie recommendation system using solely a matrix factorization technique, but with as much respect to memory compression as possible. See also this benchmark [2] which gives an idea of what kind of values of RMSE we are looking for.

   - Fill in the predictor function

     ```
     def predictor(uID,mID):
         return global_avg*np.ones(len(uID)) # global average
     ```

     (All you really have to do is compute the global average of the train set.)
   - Next, fill in the functions for computing the RMSEs over the train, validation, and test set. Hereon out, you are not allowed to touch the validation and test sets, except through this function. You should get as values:
     `train_rmse = 1.0453, val_rmse = 1.0385, test_rmse = 1.0438`
   - Next, build two predictors, one which predicts

     $$\text{ratings(u,m)} = \text{global\_average} + \text{average\_user(u)}$$

     where average_user is a vector computed on the training data and gives each user's average rating, after the global average is subtracted away.

     The second predictor should do the same thing, but average over movies. Report the RMSE of the train, validation, and test set here.

     Note that some users or movies from the train set may not appear in the validation or test set. In those cases, you are stuck, and can only compute the global average score.
   - Finally, build a simple predictor that uses the global average, user average, and movie average vectors. Do this via the following

     $$\text{ratings(u,m)} = \text{global\_average} + \tfrac{1}{2}\ \text{average\_user(u)} + \tfrac{1}{2}\ \text{average\_movie(m)}.$$

     Report the RMSE of the train, validation, and test set here.
   - Now we run the matrix completion method, by minimizing

     $$\underset{U\in\mathbb{R}^{\#\ \text{users},r}, V\in\mathbb{R}^{\#\ \text{movies},r}}{\text{minimize}}\ f(U,V) := \frac{1}{2}\sum_{i,j\in\Omega}\left(u_i^T v_j + c + \frac{\bar{m}_j}{2} + \frac{\bar{w}_i}{2} - R_{i,j}\right)^2$$

     Here, c is the global average, $\bar{m}$ is the average movie rating after global average is subtracted away, and $\bar{w}$ is the average user rating after global average is subtracted away. We represent u_i as the ith row of U and v_j as the jth row of V.

     The matrix $R \in \mathbb{R}\#$ users, $\#$ movies is an extremely sparse rating matrix, that contains the observed ratings.

     We now do this in several steps. First, write down the gradient of this objective function w.r.t. u_i and v_j.
   - Now code up the gradients, by filling in the two helper functions

     ```
     def get_grad_ui(U,V,i):
         return np.zeros(r) # replace me

     def get_grad_vj(U,V,j):
         return np.zeros(r) # replace me
     ```

[1] https://grouplens.org/datasets/movielens/latest/
[2] http://www.recsyswiki.com/wiki/MovieLens_100k_benchmark_results

Note that given a row i or j, the gradient ∇U_i should only involve the rows of V_j where R_{ij} is nonzero. Carefully read through the scipy.sparse.coo_matrix documentation to see the correct way of pulling out those indices.

Hint: maybe something like...

```
j_nonzero = R.col[R.row==i]
```

Note also that you should never create a dense matrix R, and just work with its sparse form.

- Finally, code up the full gradient descent method. Note that you cannot use 0 initializations here for U and V, which would render all gradients 0 (super suboptimal saddle point).

  Pick a value of $r = 5$, step size 0.05, initialization of $U_{i,j} \sim \mathcal{N}(0, 1/5)$ and $V_{i,j} \sim \mathcal{N}(0, 1/5)$, i.i.d. Report the RMSE of the train, validation, and test set after 20 iterations. Include also a plot of the RMSE over the three datasets in this regime.

- Now, use the validation set to pick the *best* value of r and maxiter. Train the model under these hyperparameters, and report the final RMSE of the train, validation, and test set.