HW 4 Due: Tuesday, April 18, 2023 at 2:00 pm.

Please hand in this Homework as follows:
• Upload to Gradescope HW4: A pdf of problems 1,2,3.
• Upload to Gradescope HW4 programming `ipynb` file for problem 4.

1. *Logistic regression* (15 points)

   (a) Recall the in class we wrote the logistic regression loss as

   $$L(\theta) = \sum_{i=1}^{n} \log(1 + e^{-Y_i X_i^t \theta}),\ Y_i \in \{-1, 1\}$$

   (I've removed the factor 2 in the exponent as it can be absorbed in θ.)
   Give a detailed derivation of the Newton algorithm for logistic regression and show that each iteration corresponds to solving a weighted least square problem, i.e. starting with θ^{old}, θ^{new} solves:

   $$\arg\min_{\theta}(Z - X\theta)^t W(Z - X\theta), \text{or } X^t W X\theta = X^t W Z,$$

   where $X \in R^{n\times d}$ is the training data matrix, $W \in R^{n\times n}$ is a diagonal matrix and $Z \in R^n$. Write W and Z in terms of $X, Y, \theta^{(old)}$. Hint: Note that in this model $p_i = P(Y_i = 1|X_i) = \frac{1}{1+e^{-\eta_i}} = \frac{e^{\eta_i}}{1+e^{\eta_i}}$, where $\eta_i = X_i^t\theta$, and $\frac{e^{Y_i\eta_i}}{[1+e^{Y_i\eta_i}]^2} = p_i(1 - p_i)$.

   (b) Assume the data are perfectly linearly separable, i.e. there exist θ such that $x_i^t\theta < 0$ if $y_i = 0$ and $x_i^t\theta > 0$ if $y_i = 1$. Show that the maximum likelihood estimator for the logistic regression model does not exist. Comment on the behavior of the iteratively reweighted least squares algorithm. Hint: If θ is a perfect separator then $\alpha\theta$ is also for any $\alpha > 0$. It may be easier to work with the likelihood instead of the log-likelihood.

   (c) What happens with the hinge and the quadratic losses in the perfectly separable setting. In both cases discuss whether there is a minimizer, and explain your conclusions.
   Hinge: $L(\theta) = \sum_{i=1}^{n}[1 - Y_i X_i^t\theta]_+$.
   Quadratic: $L(\theta) = \sum_{i=1}^{n}[1 - Y_i X_i^t\theta]^2$.

2. *Lasso minimization* (15 points) We are given data $X_1, Y_1, \ldots, X_n, Y_n$, with $X_i \in R^d, Y_i \in R$. We assume each coordinate of the X_i has mean zero and variance 1. In class we discussed coordinatewise minimization of the Lasso loss function:

   $$L(\theta) = \frac{1}{2n}\sum_{i=1}^{n}(Y_i - X_i^t\theta)^2 + \lambda\sum_{j=1}^{d}|\theta_j|,$$

   with $\lambda > 0$.

(a) Fixing all coordinates except for the k'th coordinate we minimize:
$f(\theta_k) = \frac{1}{2n}\sum_{i=1}^{n}(Y_i - c_i - X_{ik}\theta_k)^2 + C + \lambda|\theta_k|$.
Write out the expressions for c_i and C.

(b) Show that minimizing $f(\theta_k)$ is equivalent to minimizing
$g(\theta_k) = \frac{1}{2}\theta_k^2 - \frac{1}{n}\sum_{i=1}(Y_i - c_i)X_{ik}\theta_k + \lambda|\theta_k|$.

(c) Define the function $h(x) = \frac{1}{2}x^2 - tx + \lambda|x|,\ \lambda > 0$, show that it is strictly convex, and thus has a unique minimum.

(d) Show that the minimum is given by $x^* = sign(t)[|t| - \lambda]_+$. (Hint: If a strictly convex function h is smoothly differentiable at a point x and $h'(x) = 0$ then it is minimized at x.)

3. *Multinomial gradient* (10 points) You have C classes and labeled data $X_1, Y_1, \ldots, X_n, Y_n$, with $X_i \in R^d$ and $Y_i \in \{1, \ldots, C\}$. Let Z_i be the 'one-hot' vector corresponding to Y_i, i.e. $Z_{ij} = 1_{j=Y_i}, j = 1, \ldots, C$. Let $\mathbf{X}$ be the $n \times d$ data matrix. Let $\mathbf{Z}$ be the $n \times C$ label matrix.

We model

$$P(Y = c|X = x) = \frac{\exp \theta_c^t x}{\sum_{k=1}^{C} \exp \theta_k^t x},$$

for $\theta_c, c = 1, \ldots, C$ unknown parameters in R^d.

In class we wrote the likelihood $\theta = (\theta_1, \ldots, \theta_C)$ as

$$L(X, Y, \theta) = \prod_{i=1}^{n}\prod_{c=1}^{C}\left[\frac{\exp \theta_c^t X_i}{\sum_{k=1}^{C} \exp \theta_k^t X_i}\right]^{Z_{ic}}.$$

Denote by $\pi_{ic} = P(Y = c|X_i, \theta)$, let $\pi_c = (\pi_{1c}, \ldots, \pi_{nc})$ and let π be the $n \times C$ matrix with columns $\pi_c, c = 1 \ldots, C$.

(a) Write the log-likelihood $\log L(X, Y, \theta)$.

(b) Write the gradient of $\nabla_{\theta_c} \log L(\theta)$ w.r.t to θ_c in terms of $\mathbf{Z}, \pi_c$ and $\mathbf{X}$.

(c) Write the $C \times d$ matrix of the C gradients $\nabla_{\theta_c} \log L, c = 1, \ldots, C$ as a matrix product in terms of $\mathbf{Z}, \pi$ and X, yielding a $d \times C$ matrix.

4. *MNIST classification* (60 points)

Load the MNIST data and normalize it as in HW3. Split the data into 60% training, 20% development/validation, 20% test.

- **Bayes' rule with Bernoulli mixtures.**

  In HW3 you wrote an EM algorithm to cluster the entire MNIST dataset. Now we will try to use Bernoulli mixtures as a generative model for *each* class. In other words now we are providing an explicit model for $P(X|Y = c)$ for each c and then applying Bayes rule with the estimated models.

  Read in the MNIST data set and binarize as in HW3. Estimate a separate mixture model *for each class* with $M = 1, 5, 10, 20$. So in total you will have $M \cdot 10$ models. For $M = 1$ you don't need the EM algorithm. Use the development set to choose the optimal M and then retrain the mixtures with the training and development sets. Classify the test set data using Bayes' rule with your estimated class mixture models. You can assume that $\pi_c = 1/10, c = 1, \ldots, 10$.

- **Logistic regression.**

  - *Raw data.*
    Next you will explore how well logistic regression on the raw data compares with the Bayes method you implemented.
    Multinomial logistic regression is a linear model for multi-class classification. The conditional probabilities of the outcome class $c \in \{0, \ldots, C-1\}$ are modeled using the softmax function:

    $$P(y = c|x, \beta_c, b_c) = \frac{\exp(\beta_c^t x + b_c)}{\sum_{d=1}^{C} exp(\beta_d^t x + b_d)}.$$

    Train the model on the training set, evaluate the model on development set, and calculate the error rate. You will need the sklearn.linear model implementation of multiclass logistic regression:

    ```
    from sklearn.linear_model import LogisticRegression
    lg=LogisticRegression(fit_intercept=True, C=100000, penalty=l2,
              multi_class=multinomial,solver=lbfgs)
    ```

    This implementation of logistic regression always assumes a regularization term and $C = 1/\lambda$. So a small regularization term corresponds to very large C. Use the development set to pick the optimal C, just try a few C's with a wide range of orders of magnitude. Once you pick the optimal C retrain with the training and development set together.

  - *PCA projections.* The next step is to determine how well PCA works in terms of classification. Use the PCs you computed in HW3. Train a classifier to predict the class label given those principal components, using logistic regression. To predict the label for a new data point, we use its projection onto the principal eigenvectors. After setting up the basic work flow use the development set to select the optimal number of principal components k and the optimal C. Retrain the model using both training and development data with these values. Compute the error rate on testing data. Compare the result to multinomial logistic regression using raw features, and describe your findings.

- **Stochastic gradient descent.** Your next job is to train an ℓ_2-regularized logistic regression classifier using stochastic gradient descent. You need to write your own SGD routine for this problem. Recall the SGD framework that was covered in class using minibatches and use the formulas from problem 3. There are two functions in `scipy.special` called `softmax` and `logsumexp` that you will find very useful.

  i. Initialize the model with $\theta = 0$.

  ii. Randomly split the training data into mini-batches. Make one pass of the data, processing one mini-batch in every iteration. This is called one training epoch.

  iii. Repeat the last step a few times. Make sure you compute the negative log-likelihood of the training and validation data after each epoch and stop the process when the validation data loss stops decreasing.

  Remember the role that the learning rate plays in SGD. Tiny learning rates lead to slow convergence, while large rates can impede convergence—it might make the objective value oscillate.

  To select the regularization parameter λ, you should try different values on the validation set. I would try with $\lambda = 0, 10, 100$. Select those that yield the smallest validation error. Plot the error rate and negative log-likelihood for both training and validation set as a function of iterations.

  Once you've chosen the values of the learning rate, λ, and confirmed that your method is working train the model using the training data together with validation to retrain. Finally report the error on the test set. How does it compare in terms of time and error rate to the Logistic regression function in the previous item.