CSCC11 Introduction to Machine Learning, Winter 2021

Assignment 2, Due Thursday, February 25, 10am

This assignment makes use of material from week 3 to week 5 (specifically Chapter 9.6). To begin the programming component, download `a2.tgz` from the course website and untar it. A directory A2 will be created; please don't change its structure, nor the headers of the python methods therein. Please read the assignment handout carefully. Submission instructions are at the end.

We prefer you ask questions on Piazza. Otherwise, for your first point of contact, please reach out to your TA, Mustafa Ammous, through email: mustafa.ammous@mail.utoronto.ca.

Written Component

1) Understanding Binary Class Logistic Regression

In this question, we investigate the assumptions and limitations of the binary class logistic regression model. Suppose we have two classes with labels c_1 and c_2, we can express the posterior probability of class c_1 given input $\mathbf{x}$ as

$$P(c_1|\mathbf{x}) = \frac{1}{1+e^{-\mathbf{w}^T\mathbf{x}}} = g(\mathbf{w}^T\mathbf{x}), \tag{1}$$

where $\mathbf{w}$ is the weight vector with bias included and g is the Sigmoid function.

Written Problems.

1. First, let's investigate the decision boundary of a binary class logistic regression model. Recall that the decision boundary is the set of points where $P(c_1|\mathbf{x}) = 0.5$ (i.e. when the decision function $\alpha(\mathbf{x}) = 0$). Show that when $P(c_1|\mathbf{x}) = 0.5$, the decision boundary $\alpha(\mathbf{x}) = 0$ is a linear function.

2. Knowing that logistic regression has a linear decision boundary, what datasets would this model perform poorly on? In this case, performing poorly means the model is unable to classify all the training data correct. Let's look at a toy example – the XOR (Exclusive OR) dataset. XOR is a binary input Boolean function that outputs **TRUE** when both inputs are the same and outputs **FALSE** otherwise. The dataset is represented as the table below. Our goal is to predict the output $y \in \{0, 1\}$ given the input vector $\mathbf{x} = [x_1, x_2]^T$, where $x_1, x_2 \in \{0, 1\}$.

$\mathbf{x}$		
x_1	x_2	y
0	0	0
0	1	1
1	0	1
1	1	0

   (a) Assume we are using binary class logistic regression on the XOR dataset. What is the maximum classification accuracy we can obtain? Explain.

   (b) As in basis function regression, we can apply basis functions to create a more sophisticated model. Consider the following feature map (basis functions) on the inputs:

$$\psi(\mathbf{x}) = \begin{pmatrix} \psi_1(\mathbf{x}) \\ \psi_2(\mathbf{x}) \\ \psi_3(\mathbf{x}) \end{pmatrix} = \begin{pmatrix} x_1 \\ x_2 \\ x_1 x_2 \end{pmatrix}. \tag{2}$$

Then, we can express the posterior probability of class c_1 given input $\mathbf{x}$ as $P(c_1|\mathbf{x}) = g(\mathbf{w}^T\psi(\mathbf{x}))$, where $\mathbf{w} = [w_1, w_2, w_3]^T$ is the weight vector of the model. Note that we exclude the bias term in this question. Specify the conditions and provide an example for the weight vector $\mathbf{w}$ such that this model perfectly classifies the XOR dataset.

2) Multiclass Logistic Regression

In this question we will consider multi-class logistic regression. We will begin by extending the formulation of 2-class logistic regression to the multi-class case. For background, see Chapter 9.6 of the online lecture notes.

We will denote input feature vectors by $\mathbf{x}$, and let's assume the first element of $\mathbf{x}$ is 1 to allow for a bias term; i.e., $\mathbf{x} = (1, x_1, ..., x_d)^T$. For now, suppose there are 2 classes, with class labels 1 and 2. And let c denote the class variable. In these terms, the conditional probability of class 1, given the data, is simply

$$p(c = 1 \,|\, \mathbf{x}) \;=\; \frac{p(c = 1)\; p(\mathbf{x} \,|\, c = 1)}{p(c = 1)\; p(\mathbf{x} \,|\, c = 1) \;+\; p(c = 2)\; p(\mathbf{x}|c = 2)} \tag{3}$$

For the purposes of logistic regression, we define a mapping from an input feature vector to what we might call *unnormalized probabilities* as follows:

$$e^{\mathbf{w}_k^T \mathbf{x}} \;=\; \alpha\; p(c = k)\; p(\mathbf{x} \,|\, c = k)\;.$$

for some unknown constant α, where k is either 1 or 2 in our 2-class case. Such unnormalized probabilities are just an exponentiated linear functions of the inputs (with the addition of the bias term which is the first element of the weight vector). We call them unnormalized because we don't know the value of α, and hence the exponentiated inner products $e^{\mathbf{w}_k^T \mathbf{x}}$ don't sum to one like a probability distribution should.

For two classes the model is specified by two weight vectors, $\mathbf{w}_1$ and $\mathbf{w}_2$. And according to the model, Eqn. (3) can be rewritten as follows

$$p(c = 1 \,|\, \mathbf{x}, \mathbf{w}_{1:2}) \;=\; \frac{e^{\mathbf{w}_1^T \mathbf{x}}}{e^{\mathbf{w}_1^T \mathbf{x}} \;+\; e^{\mathbf{w}_2^T \mathbf{x}}} \;=\; \frac{1}{1 + e^{(\mathbf{w}_2 - \mathbf{w}_1)^T \mathbf{x}}}\;.$$

This is immediately recognizable as the sigmoidal function $g(\cdot)$ from Eqn. (39) in Chapter 9.6 of the online lecture notes, but here it is applied to $(\mathbf{w}_1 - \mathbf{w}_2)^T\mathbf{x}$ instead of $\mathbf{w}^T\mathbf{x}$. In other words, in the online notes on 2-class logistic regression, the weights we learned were equal to the difference of the weights used here, i.e., $\mathbf{w}_1 - \mathbf{w}_2$, which defines the normal vector to the decision boundary $(\mathbf{w}_1 - \mathbf{w}_2)^T\mathbf{x} = 0$.

Now we're ready to look at the more general case with K classes. Assume again that inputs are $\mathbf{x} = (1, x_1, ..., x_d)^T$. And for each of K classes, let there be a weight vector, $\mathbf{w}_k = (b_k, w_{k,1}, ..., w_{k,d})^T$. Assuming c denotes the class variable, we can write the probability for class $c = k$, where $1 \le k \le K$, as

$$p(c = k \,|\, \mathbf{x}) \;=\; \frac{p(c = k)\; p(\mathbf{x} \,|\, c = k)}{\sum_{\ell=1}^{K} p(c = \ell)\; p(\mathbf{x} \,|\, c = \ell)} \tag{4}$$

As above, the logistic regression model defines a parametric mapping from inputs to unnormalized probabilities. Given the model parameters, $\mathbf{w}_{1:K}$, the model specifies that

$$p(c = k \,|\, \mathbf{x}, \mathbf{w}_{1:K}) \;=\; \frac{e^{\mathbf{w}_k^T \mathbf{x}}}{\sum_{\ell=1}^{K} e^{\mathbf{w}_\ell^T \mathbf{x}}} \tag{5}$$

$$\;=\; \frac{1}{1 + \sum_{\ell \in \{1,...K\} \setminus k} e^{(\mathbf{w}_\ell - \mathbf{w}_k)^T \mathbf{x}}}\;. \tag{6}$$

The sum in the denominator in Eqn. (6), ℓ takes on all values between 1 and K, except k. Note that while Eqn. (6) more closely resembles the sigmoidal function in 2-class logistic regression, it will be more convenient to use

Eqn. (5) in what follows. The function in Eqn. (5), which maps K *score* values (i.e., the inner products $\mathbf{w}_k^T\mathbf{x}$) to probabilities, is used often in deep learning; in neural network jargon it is called a *softmax function*.

Now let's consider the objective for learning. As with 2-class logistic regression we are going to use a log loss. Given training data $\{(\mathbf{x}_i, y_i)\}_{i=1}^N$, assumed to be IID, we form the loss under the model as

$$E(\mathbf{w}_{1:K}) \;=\; -\log \prod_{i=1}^{N} p(y_i \,|\, \mathbf{x}_i, \mathbf{w}_{1:K}) \,.$$

In the lectures and notes on the 2-class case we manipulated the form of the log probability of the model in terms of the sigmoid function with the class labels being either 0 or 1. We can do the same thing here but it's just a little trickier. Rather than assume that y_i takes on a value from 1 to K, instead we let y_i be a so-called *one-hot* binary vector. That is, we define y_i to be a vector of length K whose elements are 0 or 1. When the correct class for the ith data point is k, then all elements of y_i are 0 except for the kth; i.e.,

$$y_{i,k} \;=\; \begin{cases} 1 & \text{when } k \text{ is the correct class} \\ 0 & \text{otherwise} \end{cases}$$

With this *one-hot label encoding* we can express the negative log likelihood of the training data as follows:

$$\begin{aligned} E(\mathbf{w}_{1:K}) \;&=\; -\log \prod_{i=1}^{N} \prod_{k=1}^{K} p(y_{i,k} \,|\, \mathbf{x}_i, \mathbf{w}_{1:K})^{y_{i,k}} \\ &=\; -\sum_{i=1}^{N} \sum_{k=1}^{K} y_{i,k} \, \log p(y_{i,k} \,|\, \mathbf{x}_i, \mathbf{w}_{1:K}) \end{aligned}$$

Next we need to derive the form of the gradients of E so we can figure out the necessary conditions for the optimal values of the parameters $\mathbf{w}_{1:K}$. To this end, just as we defined the sigmoid function for 2-class logistic regression, here we use a similar notation to define the softmax function: i.e.,

$$\sigma(z_{1:K},\, k) \;=\; \frac{e^{z_k}}{\sum_{\ell=1}^{K} e^{z_\ell}} \,. \tag{7}$$

With this notation, and defining $z_{i,k} = \mathbf{w}_k^T\mathbf{x}_i$, for $k = 1 \ldots K$, we can express the loss as

$$E(\mathbf{w}_{1:K}) \;=\; -\sum_{i=1}^{N} \sum_{k=1}^{K} y_{i,k} \log \sigma(z_{i,1:K},\, k) \,. \tag{8}$$

We will solve the learning problem by minimizing E using gradient descent to find the weight vectors. But we still need to find the gradient first.

Written Problems.

1. Find the gradient of $\sigma(z_{1:K}, k)$ in Eqn. (7) with respect to z_j:

$$\frac{\partial \sigma(z_{1:K},\, k)}{\partial z_j}. \tag{9}$$

   Hint: Consider the cases when $k = j$ and $k \neq j$. You may find it helpful to look at the structure of the gradient in the 2-class case in Eqn. (46) in Chapter 9.6 of the online lecture notes.

2. Find the gradient of the log likelihood for a single point $(\mathbf{x}, y)$ with respect to $\mathbf{w}_j$:

$$\frac{\partial}{\partial \mathbf{w}_j} \sum_{k=1}^{K} y_k \log \sigma(z_{1:K},\, k) \tag{10}$$

3. Find the gradient of the loss with respect to $\mathbf{w}_j$:

$$\frac{\partial E(\mathbf{w}_{1:K})}{\partial \mathbf{w}_j}. \tag{11}$$

Hint: Use the results above. And the gradient should have a form similar to Eqn. (48) in Chapter 9.6 of the online lecture notes.

4. Now, suppose we have D dimensional inputs and K classes. For each of K classes, let there be a weight vector, $\mathbf{w}_k = (b_k, w_{k,1}, ..., w_{k,d})^T$. If we include regularization, with a (diagonal) Gaussian prior, the negative log-posterior becomes, up to an additive constant,

$$\hat{E}(\mathbf{w}_{1:K}) = -\log \left[p(\mathbf{w}_{1:K}) \prod_{i=1}^{N} p(y = y_i \,|\, \mathbf{x}_i, \mathbf{w}_{1:K}) \right], \tag{12}$$

where $p(\mathbf{w}_{1:K})$ is the joint distribution over K indepdendent Gaussian densities with the same diagonal covariance. That is,

$$p(\mathbf{w}_{1:K}) = \prod_{k=1}^{K} \left(\frac{1}{((2\pi)^{(D+1)} \beta \alpha^D)^{1/2}} \exp\left(-\frac{\mathbf{w}_k^T C^{-1} \mathbf{w}_k}{2} \right) \right). \tag{13}$$

where the covariance matrix is given by $C = \text{diag}(\beta, \alpha, \alpha, \ldots, \alpha) \in \mathbb{R}^{(D+1)\times(D+1)}$. Here, α denotes the variance of the prior on each of the weights, and β is the prior variance on the bias term. Usually we don't want a strong prior on the bias term so $\beta \gg \alpha$. Derive $\frac{\partial \hat{E}}{\partial \mathbf{w}_k}$ for this regularized objective function.
Hint: Your negative log-posterior should have form $\hat{E}(\mathbf{w}_{1:K}) = E(\mathbf{w}_{1:K}) + E_2(\mathbf{w}_{1:K})$, where $E_2(\mathbf{w}_{1:K})$ is the regularization term.

Programming Component

You are asked to implement a logistic regression classification algorithm (we provide gradient descent code). The starter code is in directory `A2`. You will then apply this algorithm on four datasets, each comprises a training set and a test set. You should train your algorithm on the training data, then evaluate performance on the test set. The test performance should be measured as the average 0-1 test loss; i.e., once you've fit a model to the training data, evaluate it by counting the number of correctly labelled test points and dividing the count by the size of the test set.

Datasets: In the starter file there is a `datasets` directory. In that directory there are several pickle files for the different datasets, as described below.

- Generic datasets: There are three generic datasets, `generic_<i>.pkl` where `<i> = {1, 2, 3}`, all of which have the same format. Given 2D, real-valued measurements (features), we want to classify whether a test point is class 0 or class 1 for the first two datasets, and class 0, class 1, or class 2 for the third dataset. Each pickle file contains four arrays, `train_X`, `train_y`, `test_X`, and `test_y`. The arrays `train_X` $\in \mathbb{R}^{100\times2}$ and `train_y` $\in \mathbb{R}^{100\times1}$ form 100 input/output pairs of the training set. The arrays `test_X` $\in \mathbb{R}^{50\times2}$ and `test_y` $\in \mathbb{R}^{50\times1}$ form 50 input/output pairs of the test set.
- Wine dataset: The wine dataset, `wine.pkl`, has 13 attributes: Alcohol, Malic acid, Ash, Alcalinity of ash, Magnesium, Total phenols, Flavanoids, Nonflavanoid phenols, Proanthocyanins, Color intensity, Hue, OD280/OD315 of diluted wines and Proline (i.e. 13D measurements/features). Given the 13D measurements, we want to classify whether the wine class is wine 0, wine 1, or wine 2 (i.e. 3 classes). The file contains four arrays, `train_X`, `train_y`, `test_X`, and `test_y`. The arrays `train_X` $\in \mathbb{R}^{148\times13}$ and `train_y` $\in \mathbb{R}^{148\times1}$ form 148 input/output pairs of the training set. The arrays `test_X` $\in \mathbb{R}^{30\times13}$ and `test_y` $\in \mathbb{R}^{30\times1}$ form 30 input/output pairs of the test set.

Visualizing the Generic Datasets: You need to modify `visualize_generic.py` to visualize the three generic datasets. Once you have visualized the datasets (**NOTE**: You do not need to submit the generated plots), answer the following questions under **Visualization** in `Questions.txt`.

1. Do you expect logistic regression to perform well on `generic_1`? Why? What if we apply the feature map defined in Eqn. 2?
2. Do you expect logistic regression to perform well on `generic_2`? Why? What if we apply the feature map defined in Eqn. 2?
3. Do you expect logistic regression to perform well on `generic_3`? Why?
4. Why can't we directly visualize the wine dataset? What are some ways to visualize it?

Implementing Logistic Regression: You should implement the multi-class logistic regression with regularization explained in the written component and Chapter 9.6 of the online lecture notes. The gradient for the model is derived in the written component above. You only need to modify the file `logistic_regression.py`. You might find `utils.py` to be helpful for your implementation. The relevant methods are

- `_compute_loss_and_gradient` computes the negative log likelihood and its gradient given the inputs and outputs. It is essential for learning the logistic regression model parameters. When the hyperparameters α^{-1} and β^{-1} are non-zero, we compute the negative log posterior instead. **IMPORTANT NOTE**: For numerical stability, divide the negative log likelihood by the number of data points, drop all log constants, and drop all constant factors. Your loss should be in the form $\hat{E}_{avg}(\mathbf{w}) = \frac{E(\mathbf{w})}{N} + E_2(\mathbf{w})$, where N is the number of data points. Then, compute the gradient based on $\hat{E}_{avg}$.
- `learn` is a template for a gradient descent method, with line search, for minimizing the negative log likelihood of the data. It uses `_compute_loss_and_gradient` extensively, and it provides a facility to check your gradients numerically. Specifically, by setting the `check_grad` flag, it computes the numerical gradient using finite difference. It uses your implementation of negative log likelihood and negative log posterior for the computation.
- `predict` evaluates the logistic regression model given an array of exemplars and the weight parameters. It computes the posterior class probability given the data and model parameters. This function is used mainly for classification.

Training Logistic Regression and Evaluating Performance: To train and evaluate the classifier, you will learn models with different parameter settings, and compute the test errors for them on different datasets. The Python file `train_logistic_regression.py` allows you to train and evaluate a model given a dataset, random seed, hyperparameters, and initial parameters for training. You will need to implement the `train` function and the `feature_map` in `train_logistic_regression.py`. Specifically, you will need to implement the code to initialize logistic regression model, initialize its parameters, train the model, and evaluate the training and test performance with and without the feature map defined in Eqn. 2. Note that only the generic datasets can be used with the feature map since it is meant for 2D inputs. To apply feature map on the inputs, set `apply_data_preprocessing = True`.

Finally, run logistic regression using `train_logistic_regression.py` and answer the following questions under **Analysis** in `Questions.txt`:

1. First, we have an experiment on `generic_1` dataset. Run logistic regression without regularization and without feature map. Did you run into any numerical errors? If so, why do you think this is the case? Now, run logistic regression with regularization. What happens? What are the train and test accuracies?

2. Now, let's look at `generic_2` dataset. Run logistic regression without regularization and without feature map. What are the train and test accuracies? Run it with feature map now, did the performance get better? Why do you think that is the case?

3. `generic_3` is a multi-class classification problem. Run logistic regression without regularization and without feature map. What are the train and test accuracies? What if we run it with feature map?

4. Wine dataset: Does logistic regression perform well on this dataset?

Submission

You will need to submit two files, one for each component. The specifications of the file format and naming convention are specified below.

1. Written Component

   You may hand write or type up (e.g. Word, LateX, etc.) the solutions. We strongly recommend typing it up for legibility. Compile the document as a PDF named **A2sol_UTORID.pdf** (e.g. **A2sol_student1.pdf**). Then you should submit the pdf file onto Quercus.

2. Programming Component

   Compress the `A2` directory into a tar file named **A2sol_UTORID.tgz** (e.g., **A2sol_student1.tgz**).

   Please don't modify the method/function headers or the structure of the A2 directory tree since the automarker will fail to run. Failing to follow the instructions will result in a 25% penalty.

   To tar and zip the contents, use the command: `tar -cvzf name_of_tar.tgz A2`. Make sure to decompress the tar file to ensure the entire A2 directory is there. Then you should submit the tar file onto Quercus.