University of Toronto Scarborough
Department of Computer and Mathematical Sciences

Introduction to Machine Learning and Data Mining
CSCC11H3, Fall 2021

Assignment 2

Due December 8, 2021 at 11:49 pm

1 Logistic Regression

1.1 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

$$P\left(c_1 \mid \mathbf{x}\right)=\frac{1}{1+e^{-\mathbf{w}^T \mathbf{x}}}=g\left(\mathbf{w}^T \mathbf{x}\right) \tag{1.1}$$

where w is the weight vector with bias included and g is the Sigmoid function.

1.2 Written Problems.

1. First, let's investigate the decision boundary of a binary class logistic regression model. The decision boundary is the set of points where $P\left(c_1 \mid \mathbf{x}\right)=0.5$ (i.e. when the decision function $\alpha(\mathbf{x})=0$). Show that when $P\left(c_1 \mid \mathbf{x}\right)=0.5$, 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 $\mathbf{x}=\left[x_1, x_2\right]^T$, 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:

$$\psi(\mathbf{x}) = \begin{pmatrix} \psi_1(\mathbf{x}) \\ \psi_2(\mathbf{x}) \\ \psi_3(\mathbf{x}) \end{pmatrix} = \begin{pmatrix} x_1 \\ x_2 \\ x_1 x_2 \end{pmatrix} \tag{1.2}$$

Then, we can express the posterior probability of class c_1 given input $\mathbf{x}$ as $P(c_1 \mid \mathbf{x}) = g\left(\mathbf{w}^T \psi(\mathbf{x})\right)$, 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 w such that this model perfectly classifies the XOR dataset.

1.3 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.
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., $\mathbf{x} = (1, x_1, \ldots, x_d)^T$. 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

$$p(c=1 \mid \mathbf{x}) = \frac{p(c=1) p(\mathbf{x} \mid c=1)}{p(c=1) p(\mathbf{x} \mid c=1) + p(c=2) p(\mathbf{x} \mid c=2)} \tag{1.3}$$

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

$$e^{\mathbf{w}_k^T \mathbf{x}} = \alpha p(c=k) p(\mathbf{x} \mid c=k) \tag{1.4}$$

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, w_1 and w_2, where we can write:

$$p\left(c=1 \mid \mathbf{x}, \mathbf{w}_{1:2}\right) = \frac{e^{\mathbf{w}_1^T \mathbf{x}}}{e^{\mathbf{w}_1^T \mathbf{x}} + e^{\mathbf{w}_2^T \mathbf{x}}} = \frac{1}{1 + e^{(\mathbf{w}_2 - \mathbf{w}_1)^T \mathbf{x}}} \tag{1.5}$$

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 $\mathbf{x}=(1,x_1,\ldots,x_d)^T$. And for each of K classes, let there be a weight vector, $\mathbf{w}_k=(b_k,w_{k,1},\ldots,w_{k,d})^T$. Assuming c denotes the class variable, we can write the probability for class $c=k$, where $1\leq k\leq K$, as

$$p(c=k\mid\mathbf{x})=\frac{p(c=k)p(\mathbf{x}\mid c=k)}{\sum_{\ell=1}^{K}p(c=\ell)p(\mathbf{x}\mid c=\ell)} \tag{1.6}$$

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

$$\begin{aligned}p\left(c=k\mid\mathbf{x},\mathbf{w}_{1:K}\right)&=\frac{e^{\mathbf{w}_k^T\mathbf{x}}}{\sum_{\ell=1}^{K}e^{\mathbf{w}_\ell^T\mathbf{x}}}\\&=\frac{1}{1+\sum_{\ell\in\{1,\ldots K\}\setminus k}e^{(\mathbf{w}_\ell-\mathbf{w}_k)^T\mathbf{x}}}\end{aligned} \tag{1.7}$$

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

$$E\left(\mathbf{w}_{1:K}\right)=-\log\prod_{i=1}^{N}p\left(y_i\mid\mathbf{x}_i,\mathbf{w}_{1:K}\right) \tag{1.8}$$

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 i th data point is k, then all elements of y_i are 0 except for the k th; i.e.,

$$y_{i,k}=\begin{cases}1 & \text{when } k \text{ is the correct class}\\ 0 & \text{otherwise}\end{cases} \tag{1.9}$$

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

$$\begin{aligned}E\left(\mathbf{w}_{1:K}\right)&=-\log\prod_{i=1}^{N}\prod_{k=1}^{K}p\left(y_{i,k}\mid\mathbf{x}_i,\mathbf{w}_{1:K}\right)^{y_{i,k}}\\&=-\sum_{i=1}^{N}\sum_{k=1}^{K}y_{i,k}\log p\left(y_{i,k}\mid\mathbf{x}_i,\mathbf{w}_{1:K}\right)\end{aligned} \tag{1.10}$$

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.,

$$\sigma\left(z_{1:K}, k\right)=\frac{e^{z_k}}{\sum_{\ell=1}^{K} e^{z_\ell}} \tag{1.11}$$

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

$$E\left(\mathbf{w}_{1:K}\right)=-\sum_{i=1}^{N} \sum_{k=1}^{K} y_{i,k} \log \sigma\left(z_{i,1:K}, k\right) \tag{1.12}$$

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.

1.4 Written Problems.

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

$$\frac{\partial \sigma\left(z_{1:K}, k\right)}{\partial z_j}. \tag{1.13}$$

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 $\mathbf{w}_j$:

$$\frac{\partial}{\partial \mathbf{w}_j} \sum_{k=1}^{K} y_k \log \sigma\left(z_{1:K}, k\right) \tag{1.14}$$

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

$$\frac{\partial E\left(\mathbf{w}_{1:K}\right)}{\partial \mathbf{w}_j} \tag{1.15}$$

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, $\mathbf{w}_k=\left(b_k, w_{k,1}, \ldots, w_{k,d}\right)^T$. If we include regularization, with a (diagonal) Gaussian prior, the negative log-posterior becomes, up to an additive constant,

$$\hat{E}\left(\mathbf{w}_{1:K}\right)=-\log \left[p\left(\mathbf{w}_{1:K}\right) \prod_{i=1}^{N} p\left(y=y_i \mid \mathbf{x}_i, \mathbf{w}_{1:K}\right)\right] \tag{1.16}$$

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

$$p\left(\mathbf{w}_{1:K}\right)=\prod_{k=1}^{K}\left(\frac{1}{\left((2\pi)^{(D+1)}\beta\alpha^{D}\right)^{1/2}}\exp\left(-\frac{\mathbf{w}_k^T C^{-1}\mathbf{w}_k}{2}\right)\right) \tag{1.17}$$

where the covariance matrix is given by $C=\operatorname{diag}(\beta,\alpha,\alpha,\ldots,\alpha)\in\mathbb{R}^{(D+1)\times(D+1)}$. 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 $\hat{E}\left(\mathbf{w}_{1:K}\right)=E\left(\mathbf{w}_{1:K}\right)+E_2\left(\mathbf{w}_{1:K}\right)$, where $E_2\left(\mathbf{w}_{1:K}\right)$ is the regularization term.

1.5 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 $\langle i\rangle=\{1,2,3\}$, 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 $\mathrm{train\_}x\in\mathbb{R}^{100\times 2}$ and train_y $\in\mathbb{R}^{100\times 1}$ form 100 input/output pairs of the training set. The arrays test_x $\in\mathbb{R}^{50\times 2}$ and test_y $\in\mathbb{R}^{50\times 1}$ 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, OD $280/\mathrm{OD}315$ 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 $\in\mathbb{R}^{148\times 13}$ and train_y $\in\mathbb{R}^{148\times 1}$ form 148 input/output pairs of the training set. The arrays test_x $\in\mathbb{R}^{30\times 13}$ and test_y $\in\mathbb{R}^{30\times 1}$ 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 answer the following questions:

1. Do you expect logistic regression to perform well on generic_1? Why? What if we apply the feature map defined in Eqn. 1.2 ?

2. Do you expect logistic regression to perform well on generic_2? Why? What if we apply the feature map defined in Eqn. 1.2 ?

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. 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 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. 1.2. 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:

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?

2. 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?

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

2 Clustering

You are asked to implement K-Means and Gaussian Mixture Model (GMM). Starter code is in directory Clustering_Problem. You will then apply these algorithms on different datasets for various applications. This problem also asks a number of questions. For each, you are expected to write short answers (no more than 2 or 3 sentences).

Implementing K-Means: Implement the K-Means clustering algorithm explained in Chapter 16 of the online lecture notes. You only need to implement the train method in the Python file kmeans.py, which performs K-Means clustering on a given the dataset. It iteratively updates K centers, and assigns each data point to the closest center. Upon convergence, the method returns the assignments of the dataset. The cluster centers are stored within the KMeans object under the variable self. centers. Once you are confident that your code is correct, you can run test_clustering.py to test your implementation. Note that you can visualize the clusters by setting the visualize=True.

With K-Means implemented, let's look at an application. Read document_clustering ·py carefully. It loads a dataset from BBC that contains word frequency vectors for thousands of documents on five different topics. Our goal is to cluster these documents to discover what those topics might be. This general problem is often called topic modeling. Here we explore some of the issues involved in determining the topics through clustering and careful data pre-processing. For this task, we only use K-Means clustering. Note: You could assume all initial centers are unique.

Step 0: Before you start, have a look at the data. Load BBC_data.pkl from the data directory. This file contains three main arrays: dat a contains the raw data, terms contains the words corresponding to the features of the data vectors, and labels contains the assignment of documents to classes (we'll only use this at the end). Plot a few vectors from data, inspect some of the words in the terms array, print the word contents (and frequencies) for several document vectors, and get a good idea of how many different words you might expect in a document

1. How sparse are the document term vectors (i.e. on average, how many of the entries in each vector are zero)?
2. What are the 10 most common terms? What are the 10 least common terms?
3. What is the average value for word-frequencies? (only counting vector entries that are non-zero).

Step 1: Run the document clustering script with $K = 5$, $\text{norm\_flag} = 0$, $\text{diffuse} = 0$, and random center initialization (see center_initializations.py). This will apply your K-Means clustering code to the original input vectors. The result will be stored in the results directory with the correspond experiment configuration you used. Inspect the resulting clusters, labels and record:

1. Can you figure out which topics the clusters represent?
2. What are the factors that make clustering difficult?
3. Should we expect better results if we get a lucky guess at the cluster centers?

Step 2: Run the document clustering script with $K = 5$, $\mathrm{norm\_flag} = 1$, $\mathrm{diffuse} = 0$, and random center initialization. This will run clustering after pre-processing the input document vectors so they are normalized to have unit length. In effect, we are now treating each input vector as a probability distributions over terms.

1. What problem in Step 1 does this solve?
2. Based on the data points in each cluster, what do you think topics for these clusters?
3. Would you consider this result better or worse than Step 1? Why?

Step 3: Run the document clustering script with $K = 5$, $\mathrm{norm\_flag} = 1$, $\mathrm{diffuse} = 2$, and random center initialization. This will pre-process each document by doing 2-steps of random-walk diffusion based on word co-occurrence probabilities (estimated from the entire data-set). That is, the pre-processed vectors will not only have the original terms, but they will also contain words that are strongly associated with those originally in the document (e.g., they might be synonyms). Run the clustering a few times, checking the output results, and pick a good one. Answer these questions:

1. What would you say are the topics for clusters?
2. Why is the clustering suddenly better?
3. What would you say is the general lesson to be learned from trying to cluster high-dimensional sparse data?

Implementing GMM: K-Means "struggles" to cluster specific type of data. We now look at GMM clustering and compare it with K-Means. Implement the GMM clustering algorithm explained in Chapter 16 of the online lecture notes. You only need to modify the Python file gmm. py. The train method is already provided and you will only need to implement the following methods:

1. e_step performs the expectation step of the EM algorithm. It computes the responsibilities (probabilistic assignment of data to mixture components), given the current parameters.
2. m_step performs the maximization step of the EM algorithm. This updates the mean and variance of each Gaussian component in the mixture, along with the mixture proportions.

Similar to KMeans, The parameters of the GMM are stored within the GMM object under the variables self.centers, self.covariances, and self.mixture_proportions. Test the code again with test_clustering.py by setting test_method='gmm'. Set test_method='all' to compare the methods. For some datasets, you will notice that the clusters found by K-Means and GMM are very different.

How can you characterize these differences and why do they occur? If you are curious (not marked), you can also run document clustering with GMM as well, but you will quickly notice why it's not a great idea. What can we do if we want to run GMM for this task?