1 Logistic Regression

In this exercise we focus on the classification problem. This is just like the regression problem, except that the values y we now want to predict take on only a small number of discrete values. For now, we will focus on the **binary** classification problem in which y can take on only two values, 0 and 1. For instance, if we are trying to build a spam classifier for email, then $\mathbf{x}$ may be some features of a piece of email, and y may be 1 if it is a piece of spam mail, and 0 otherwise. The value 0 is also called the negative class, whereas the value 1 the positive class. Additionally, they are sometimes denoted by the symbols '–' and '+'. Given a training example $\mathbf{x}^{(i)}$, the corresponding $y^{(i)}$ is also called the **label** of it.

1.1 Theory

We could approach the classification problem ignoring the fact that y is discrete-valued, and use our old linear regression algorithm to try to predict y given $\underline{x}$. However, it is easy to construct examples where this method performs very poorly. Intuitively, it also does not make sense for $h(\underline{x})$ to take values larger than 1 or smaller than 0 when we know that $y \in [0, 1]$. To fix this, let change the form for our hypotheses $h(\underline{x})$. We will choose

$$h(\underline{x}) = g\left(\underline{\theta}^T \underline{x}\right) = \frac{1}{1 + e^{-\underline{\theta}^T \underline{x}}} \tag{1}$$

where

$$g(z) = \frac{1}{1 + e^{-z}} \tag{2}$$

is called the **logistic** (or the **sigmoid**) function. Notice that $g(z)$ tends towards 1 as $z \to \infty$, and $g(z)$ tends towards 0 as $z \to -\infty$. Moreover, $g(z)$, and hence also $h(\underline{x})$, is always bounded between 0 and 1. As before, we are keeping the convention of letting $x_0 = 1$, so that $\underline{\theta}^T \underline{x} = \theta_0 + \sum_{j=1}^{m} \theta_j x_j$.

So, given the logistic regression model, how do we fit $\underline{\theta}$ for it? Following how we saw least squares regression could be derived as the maximum likelihood estimator under a set of assumptions, let endow our classification model with a set of probabilistic assumptions, and then fit the parameters via maximum likelihood. Let us assume that $P(y = 1|\underline{x}; \underline{\theta}) = h(\underline{x})$ and

$P(y = 0|\underline{x}; \underline{\theta}) = 1 - h(\underline{x})$. Note that this can be written more compactly as

$$P(y|\underline{x}; \underline{\theta}) = [h(\underline{x})]^y \times [1 - h(\underline{x})]^{1-y}.$$

Assuming that we have n training examples, we can then write down the likelihood of the parameters as

$$\begin{aligned}\mathcal{L}(\underline{\theta}) &= P(\underline{y}|\mathbf{X}; \underline{\theta}) \\ &= \prod_{i=1}^{n} P\left(y_i|\underline{x}_i; \underline{\theta}\right) \\ &= \prod_{i=1}^{n} \left[h\left(\underline{x}_i\right)\right]^{y_i} \times \left[1 - h\left(\underline{x}_i\right)\right]^{1-y_i}\end{aligned}$$

The goal is to maximize the log likelihood

$$\log \mathcal{L}(\underline{\theta}) = \sum_{i=1}^{n} \left(y_i \log h\left(\underline{x}_i\right) + (1 - y_i) \log\left(1 - h(\underline{x}_i)\right)\right), \tag{3}$$

or, alternatively, to minimize the cost function $\mathcal{J}(\underline{\theta})$ is defined as

$$\mathcal{J}(\underline{\theta}) = -\frac{1}{n} \log \mathcal{L}(\underline{\theta}) \tag{4}$$

How do we minimize the cost function? Similar to our derivation in the case of linear regression, we can use Gradient Descent.

The gradient of the cost function $\mathcal{J}(\underline{\theta})$ is given by

$$\frac{\partial \mathcal{J}(\underline{\theta})}{\partial \theta_j} = \frac{1}{n} \sum_{i=1}^{n} \left[h\left(\underline{x}_i\right) - y_i\right] x_{ij}, \;\; j = 1, 2, \ldots, m. \tag{5}$$

and the corresponding matrix form

$$\frac{\partial \mathcal{J}(\underline{\theta})}{\partial \underline{\theta}} = \frac{1}{n} \mathbf{X}^T [h(\mathbf{X}) - \underline{y}] \tag{6}$$

then, the gradient descent algorithm updates the parameters using

$$\underline{\theta} = \underline{\theta} - \frac{\partial \mathcal{J}(\underline{\theta})}{\partial \underline{\theta}} \tag{7}$$

1.2 Python Exercise

In this part of the exercise, you will implement the logistic regression to predict if a person has cancer or not. We employ the existing dataset from `sklearn` named *breast cancer wisconsin*[1]. This dataset contains in total 569 examples, among them 212 examples are labelled as malignant (M or 0) and 357 examples are marked as benign (B or 1). Features are computed from a digitalized image of a fine needle aspirate (FNA) of a breast mass. A feature vector has 30 dimensions.

This exercise re-uses the piece of code for dataset loading and pre-processing from the previous exercises. Therefore, this part has been done for you.

Step 1: Implement the sigmoid, cost and gradient functions. Similar to the previous exercises, you have to implement the functions `compute_cost` and `compute_gradient`. Also, you need to write code to calculate the output of our hypothesis, namely implement the `sigmoid` function (logistic function). At the end of this step, you will use the function `approximate_gradient` to verify if your implementation is correct.

Step 2: Update the model's parameters using mini-batch gradient descent. In this step, we re-use the implementation of the mini-batch gradient descent algorithm from the previous exercise. You need to write your code to update the model's parameters using the function `compute_gradient` that you have implemented. Again, you have to compute the cost across the training process and visualize it.

Step 3: Predict the probabilities of having caner and drawing the confusion matrix. In this step, you use the trained model to predict the probabilities of having caner using the measurements from the test set. You will need to call function `sigmoid` you have implemented before. Moreover, you will evaluate the performance of your model using accuracy measure, which is the percentage of correctly classified examples over the total number of examples in the test set. Then, you draw a confusion matrix illustrating your classification result. The accuracy of your model should be greater than 90%

[1] https://archive.ics.uci.edu/ml/datasets/Breast+Cancer+Wisconsin+(Diagnostic)

2 Recommendation System

Recommendation System (RS) concerns the task of giving recommendations to users. Nowadays, this task is everywhere over the Internet, for example, Netflix uses its RS engine to recommend suitable movies to viewers, Amazon uses RS to recommend potential products to customers, Facebook uses RS to rank and suggest posts of interest to users, etc. As a result, research in RS has been receiving a lot of attention in recent years.

One of the most common approaches for RS is *collaborative filtering*. In this exercise, we will consider the most basic but effective approach to perform collaborative filtering, namely *matrix factorization*.

2.1 Short Theory

Let's consider the Netflix problem: recommending movies to users. Suppose we are given a rating matrix $R \in \mathbb{R}^{n\times m}$, in which each row corresponds to a user and each column corresponds to a movie. An element R_{ij} in R corresponds to the rating user i gives to movie j. For movie rating, the rating values are normally integers in the range $1, \ldots, 5$. Since the number of movies is very large and each user can only give ratings for small number of movies, the rating matrix R is highly sparse, meaning that there is only a small portion of the entries is known. The task is to predict the unknown rating entries from the known rating entries. If we can accurately do this, we can predict the rating of a user to movies he or she has not seen, and then give suitable recommendations.

In matrix factorization, we assume that the rating matrix R can be factorized into two matrices $U \in \mathbb{R}^{n\times d}, V \in \mathbb{R}^{m\times d}$: $R = UV^T$. U can be considered as containing the users' features, i.e. each row U_i encodes the preference of user i; while V can be considered as containing the items' features, i.e. each row V_j encodes the characteristics of item j.

Denote Ω the set of known entries. The objective function which we need to minimize is the mean square error over the knwon entries, as follows:

$$L = \frac{1}{2}\frac{\sum_{i,j\in\Omega}\left(U_i^T V_j - R_{ij}\right)^2}{|\Omega|}, \tag{8}$$

with $|\Omega|$ the number of known entries.

Clearly if we fix V and only solve eq. (8) for U_i, we will arrive at the linear regression (least square) problem. As a result, an approach to minimize L in eq. (8) is to alternate between solving for U and for V. This approach is called *Alternating least square* (ALS). However, as we know that gradient descent algorithm is more scalable than least square method, we will minimize the loss L using the former.

From Ω, we can create a mask $M \in \{0,1\}^{n \times m}$ for the rating matrix, such that $M_{ij} = 1$ if $ij \in \Omega$ (or R_{ij} is known. The loss in eq. (8) can be written in Numpy form as:

$$L = \frac{1}{2} \frac{\texttt{sum}\left(M \odot \left(UV^T - R\right)^{**} 2\right)}{\texttt{sum}(M)}, \tag{9}$$

where `sum` represents the summation of all elements in a matrix, $\odot$ represent the element-wise matrix multiplication.

Similar to the logistic regression case, to avoid overfitting, we need to add regularization terms to our parameters, which are U and V in this case. This results in the final loss function (in Numpy form):

$$L = \frac{1}{2} \frac{\texttt{sum}\left(M \odot \left(UV^T - R\right)^{**} 2\right)}{\texttt{sum}(M)} + \frac{1}{2}\lambda * \texttt{sum}\left(U^{**}2\right) + \frac{1}{2}\lambda * \texttt{sum}\left(V^{**}2\right) \tag{10}$$

As we are using gradient descent to solve for U and V, we need to calculate the gradients of the loss L in eq. 10 w.r.t. U and V. These gradients can be expressed as follows:

$$\frac{\partial L}{\partial U} = \left(\left(UV^T - R\right) \odot M\right) V + \lambda U \tag{11}$$

$$\frac{\partial L}{\partial V} = \left(\left(UV^T - R\right) \odot M\right)^T U + \lambda V \tag{12}$$

$$\tag{13}$$

Having the gradients calculated using the equations above, we can apply gradient descent for a number of iterations to learn for U and V. Obtaining U and V, we can then make prediction for the whole rating matrix as UV^T.

2.2 Python Exercise

In this exercise, you will implement the collaborative filtering using matrix factorization algorithm. You will use a standard movie rating datasetm namely,"MovieLens100K". The dataset contains 943 users and 1682 movies. There are $100,000$ integer ratings, with values in range $[1,5]$. As you can see, this dataset is highly sparse with only 6.3% of ratings are known. For your convenience, the function to load the dataset has been given and the dataset has been split into a training and a testing set.

Step 1: Load dataset and create mask matrix. Finish the `create_mask` function. In the original rating matrix, known entries have positive values while unknown entries are all set to 0. As a result, you can create the mask by comparing the original rating matrix with 0.

Step 2: Implement cost and gradient functions. You will implement functions *compute_cost* and *compute_gradient*. Follow the equations presented above for your implementation.

Step 3: Learn the users' features and movies' features. Use gradient descent with a predifined number of learning iteration and regularization weights to learn for U and V.

Step 4: Write evaluation function. In this step, you need to write the two functions which are commonly used to evaluate matrix factorization methods, namely the *root mean square error (RMSE)* and *mean absolute error (MAE)*. Note that both RMSE and MAE are calculated **only** over the entries in a given mask.

Step 5: Make prediction and evaluate. Use the learned features U and V to complete the rating matrix. After that, you can evaluate your results.

3 K-means Clustering

3.1 Theory

K-means is a clustering algorithm. Clustering algorithms are unsupervised techniques for sub-dividing a larger data set into smaller groups. The term "unsupervised" indicates that the data do not originate from clearly defined groups that can be used to label them a priori. For example, say that you're exploring a group of 300 hens in a barn and you've measured a bunch of parameters from each bird: height, weight, egg size, laying regularity, egg color, etc. Based on all these parameters, you want to figure out if the hens fall into a small number of distinct groups or if they constitute just a single homogeneous population. Note that this question has two features: 1) At the outset you don't know how many groups there are and therefore 2) you have no way of assigning a given hen to a given group. All hens are treated equally. For problems such as this, you can use k-means clustering to objectively assign each hen to a group. Using further techniques (described below) you can objectively test whether the assignments you've made are reasonable.

The k-mean clustering includes some steps:

- Step 1: Initlialize randomly centroids of clusters using points from the dataset.
- Step 2: Assign datapoints to the clusters using a distrance metric.
- Step 3: Compute new centroids using the mean of the clusters.
- Repeat step 2 and step 3 until there is no change in the clusters.

3.2 Python Exercise

In this exercise, you will implement k-means clustering algorithm to subdivide the `sklearn`'s *digits* dataset. The dataset contains images of handwritte digits from 0 to 9. Table 3.2 provides information regarding this dataset.

You are going to do the following steps.

- Step 1: In this step, you will load the dataset *Digits* with 5 classes. You can choose how much data you want to use, but the whole dataset

Table 1: *Digits* dataset.

Classes	10
Samples per class	~ 180
Samples total	1797
Dimensionality	64
Features	integers $0 - 16$

will take you more time to process. Then, you will visualize some images in this dataset using `matplotlib`. Also, you will be provided the code to visualize the dataset using `t-SNE`, which is one of the strongest method for visualizing high dimensional data. An optional step is to try different data reduction methods for visualization such as PCA[2].

- Step 2: You have to implement function `kmeans` in this step. The function takes two inputs: the data from `Digits` and the number of clusters. The expected outputs are the centroids of clusters and the labels of the data.

- Step 3: This step will evaluate your clustering results. You will use the function `kmeans` to make clusters. Also, you will visualize some images from the clusters to make sure that members of a cluster are indeed similar.

[2]http://scikit-learn.org/stable/auto_examples/manifold/plot_lle_digits.html