Hoemwork 4 Logistic Regression, the Perceptron and the Support Vector Machine

Spring 2019
(Due: March 22, 2019 Friday)

Objective

The objective of this project is twofold:

(a) Consolidate and further your understanding of the logistic regression, the perceptron and the SVM;

(b) Implement the three linear classification algorithms in Python on a synthetic 2D dataset, and compare their performances.

Important Concepts

The table below summarizes the three linear classification algorithms we will study in this project. Please still review your lecture notes and textbook before proceeding to the project problems, as the table does not elaborate on the details of the classifiers.

Learning Algorithm	**Optimization Problem**	**Solution/Iterate**
Logistic Regression $y_j \in \{0,1\}$	$$\underset{\boldsymbol{\theta}}{\text{argmin}}\, J(\boldsymbol{\theta})$$ $$J(\boldsymbol{\theta}) = \sum_{j=1}^{N} -\Big\{ y_j \log h_{\boldsymbol{\theta}}(\boldsymbol{x}_j) + (1-y_j)\log(1-h_{\boldsymbol{\theta}}(\boldsymbol{x}_j))\Big\} \quad (1)$$	$$\boldsymbol{\theta}^{(k+1)} = \boldsymbol{\theta}^{(k)} - \alpha_k \Big(\sum_{j=1}^{N} (h_{\boldsymbol{\theta}^{(k)}}(\boldsymbol{x}_j) - y_j)\boldsymbol{x}_j \Big) \quad (2)$$
Perceptron $y_j \in \{-1,1\}$	$$\underset{\boldsymbol{\theta}}{\text{argmin}}\, J(\boldsymbol{\theta})$$ $$J(\boldsymbol{\theta}) = \sum_{i=1}^{N} \max\{-y_j g_{\boldsymbol{\theta}}(\boldsymbol{x}_j), 0\} = \sum_{i\in\mathcal{M}(\boldsymbol{\theta})} -y_j g_{\boldsymbol{\theta}}(\boldsymbol{x}_j) \quad (3)$$	$$\textbf{Batch}: \boldsymbol{\theta}^{(k)} = \boldsymbol{\theta}^{(k)} + \alpha_k \sum_{j\in\mathcal{M}_k} \begin{bmatrix} y_j\boldsymbol{x}_j \\ y_j \end{bmatrix}$$ $$\textbf{Online}: \boldsymbol{\theta}^{(k)} = \boldsymbol{\theta}^{(k)} + \alpha_k \begin{bmatrix} y_{j'}\boldsymbol{x}_{j'} \\ y_{j'} \end{bmatrix}$$ $$j' \in \mathcal{M}_k \quad (4)$$
Hard-Margin SVM $y_j \in \{-1,1\}$	$$\underset{\boldsymbol{\theta}}{\text{argmin}}\, \frac{1}{2}\|\boldsymbol{w}\|_2^2$$ subject to $y_j g_{\boldsymbol{\theta}}(\boldsymbol{x}_j) \geq 1, j = 1,...,N$ (5)	Quadratic programming solver (6)

Soft-Margin SVM (general) $y_j \in \{-1,1\}$	$$\underset{\boldsymbol{\theta},\boldsymbol{\xi}}{\text{argmin}} \frac{1}{2}\|\boldsymbol{w}\|_2^2 + C\rho(\boldsymbol{\xi})$$ subject to $y_j g_{\boldsymbol{\theta}}(\boldsymbol{x}_j) \geq 1-\xi_j$, $\xi_j \geq 0, j = 1, ..., N$ (7)	Quadratic programming solver (8)
Soft-Margin SVM ($\rho(\boldsymbol{\xi}) = \|\boldsymbol{\xi}\|_2^2$) $y_j \in \{-1,1\}$	$$\underset{\boldsymbol{\theta},\boldsymbol{\xi}}{\text{argmin}} \frac{1}{2}\|\boldsymbol{w}\|_2^2 + C\|\boldsymbol{\xi}\|_2^2$$ subject to $y_j g_{\boldsymbol{\theta}}(\boldsymbol{x}_j) \geq 1-\xi_j$, $j = 1, ..., N$ (9)	Quadratic programming solver (10)
Soft-Margin SVM ($\rho(\boldsymbol{\xi}) = \|\boldsymbol{\xi}\|_1$) $y_j \in \{-1,1\}$	$$\underset{\boldsymbol{\theta}}{\text{argmin}} \sum_{i=1}^{N} [1 - y_j g_{\boldsymbol{\theta}}(\boldsymbol{x}_j)]_+ + \frac{\lambda}{2}\|\boldsymbol{w}\|_2^2$$ (11)	Quadratic programming solver (12)

Table 1: 2-Class Classifiers

A few things about the table:

1. Notations and conventions: $\boldsymbol{\theta} = (\boldsymbol{w}^T, w_0) \in \mathbb{R}^{N+1}$, N is the total number of samples, and in any equation in the table, $g_{\boldsymbol{\theta}}(\boldsymbol{x}) = \boldsymbol{w}^T\boldsymbol{x} + w_0$;

2. In (1) and (2), $h_{\boldsymbol{\theta}}(\boldsymbol{x}) = 1/(1 + e^{-(\boldsymbol{w}^T\boldsymbol{x}+w_0)})$;

3. In (4), $\mathcal{M}_k = \mathcal{M}(\boldsymbol{\theta}^{(k)})$ is the set of indices of the misclassified samples when $\boldsymbol{\theta}^{(k)}$ is used to describe the decision boundary, i.e.

$$\mathcal{M}_k = \mathcal{M}(\boldsymbol{\theta}^{(k)}) = \{j \in \{1, ..., N\} \,|\, \text{sign}(g_{\boldsymbol{\theta}^{(k)}}(\boldsymbol{x}_j)) \neq y_j\} \tag{13}$$

Exercise 1: Theory

This exercise contains three problems whose purpose is to help you consolidate and further your understanding of the three linear classification algorithms: logistic regression, the perceptron and the SVM. More specifically, each of the problems concentrates on one of the aforementioned algorithms: the first problem focuses on the issue of nonconvergence of logistic regression when the data is linearly separable, the second problem focuses on proving convergence of a simplified perceptron algorithm, and the last problem focuses on finding the convex dual problem of the soft-margin SVM. Reading the lecture notes should be helpful for solving these problems.

Tasks:

(a) (**Logistic Regression**) We analyze the convergence behaviour of the logistic regression when the data is linearly separable.

(i) Prove that if two classes of data in $\mathbb{R}^n$ are linearly separable, then the magnitude of the slope and intercept parameters $\boldsymbol{w}$ and w_0 of the optimization would tend to ∞. Furthermore, prove that the gradient descent iterates in (2) would not converge in a finite number of steps, if we allowed the algorithm to run forever (i.e. only let it stop when $\|\boldsymbol{\theta}^{(k+1)} - \boldsymbol{\theta}^{(k)}\|_2 = 0$).

(ii) What happens if we restrict $\|\boldsymbol{w}\|_2 \leq c_1$ and $|w_0| < c_2$ for some $c_1, c_2 > 0$? What other ways can you come up with to counter the nonconvergence issue?

(iii) Does linear separability of data cause nonconvergence for the other linear classifiers that we have studied? Why?

(b) (**Perceptron**) We study a simplified online perceptral algorithm by absorbing w_0 into $\boldsymbol{w}^{(k)}$ and setting $\alpha_k = 1$:

$$\boldsymbol{w}^{(k+1)} = \boldsymbol{w}^{(k)} + y_j \boldsymbol{x}_j, \text{ some } j \in \mathcal{M}_k \tag{14}$$

(i) Show that as far as classifying $\boldsymbol{x}_j$ is concerned, the move from $\boldsymbol{w}^{(k)}$ to $\boldsymbol{w}^{(k+1)}$ is in the right direction.

Hint: Show $y_j(\boldsymbol{w}^{(k+1)})^T \boldsymbol{x}_j > y_j(\boldsymbol{w}^{(k)})^T \boldsymbol{x}_j$.

(ii) Let us show convergence of this algorithm. Let $\boldsymbol{w}^*$ be the ultimate solution, and let $\rho = \min_j y_j(\boldsymbol{w}^*)^T \boldsymbol{x}_j$.

1. Show that $(\boldsymbol{w}^{(k)})^T \boldsymbol{w}^* \geq k\rho$.
2. Show $\boldsymbol{w}^{(k)} \leq kR^2$ for $R = \max_j \|\boldsymbol{x}_j\|_2$, by proving that $\|\boldsymbol{w}^{(k)}\|_2^2 \leq \|\boldsymbol{w}^{(k-1)}\|_2^2 + \|\boldsymbol{x}_j\|_2^2$.
3. Using 1. and 2., prove that

$$\frac{(\boldsymbol{w}^{(k)})^T \boldsymbol{w}^*}{\|\boldsymbol{w}^{(k)}\|_2} \geq \sqrt{k}\frac{\rho}{R} \tag{15}$$

Show that this further implies

$$k \leq \frac{R^2 \|\boldsymbol{w}^*\|_2^2}{\rho^2} \tag{16}$$

Hence the algorithm converges in a finite number of steps, upper bounded by the expression above.

(c) (**SVM**) Recall the primal optimization problem of the soft-margin SVM with $\rho = \|\cdot\|_2^2$:

$$\begin{aligned} &\underset{\boldsymbol{\theta},\boldsymbol{\xi}}{\text{argmin}}\ \frac{1}{2}\|\boldsymbol{w}\|_2^2 + C\|\boldsymbol{\xi}\|_2^2 \\ &\text{subject to } y_j g_{\boldsymbol{\theta}}(\boldsymbol{x}_j) \geq 1 - \xi_j,\ \xi_j \geq 0,\ j = 1, ..., N \end{aligned} \tag{17}$$

Let us find the convex dual problem of it.

(i) The constraint $\xi_j \geq 0\,\forall j$ can be removed without affecting the solution to the optimization problem. Why?

(ii) Write out the Lagrangian of the problem, and show that for the optimal $\boldsymbol{\theta}^*$ and $\boldsymbol{\xi}^*$,

$$\boldsymbol{w}^* = \sum_{j=1}^N \lambda_j y_j \boldsymbol{x}_j,\ \sum_{j=1}^N \lambda_j y_j = 0,\ C\xi_j^* = \lambda_j\,\forall j \tag{18}$$

(iii) Prove that the convex dual problem is

$$\begin{aligned} &\max_{\boldsymbol{\lambda}\geq 0} \left\{ \sum_{j=1}^N \lambda_j - \frac{1}{2}\sum_{j=1}^N \sum_{k=1}^N \lambda_j \lambda_k y_j y_k \boldsymbol{x}_j^T \boldsymbol{x}_k - \frac{1}{2}\sum_{j=1}^N \frac{\lambda_j^2}{C} \right\} \\ &\text{subject to } \sum_{j=1}^N \lambda_j y_j = 0 \end{aligned} \tag{19}$$

(d) **(SVM)** In the following two problems, consider the hard-margin SVM as in equation 5

(i) Show that, if the 1 on the right-hand side of the hard-margin SVM (equation 5) constraint is replaced by some arbitrary constant $\gamma > 0$, the solution for the maximum margin hyperplane is unchanged.

Hint: Show that a scalar multiple of $\boldsymbol{\theta}$ does not change the decision boundary

(ii) Show that, irrespective of the dimensionality of the data space, a data set consisting of just two data points, one from each class, is sufficient to determine the location of the maximum-margin hyperplane.

Hint: Use Lagrange multiplier to directly solve the hard-margin SVM with 2 data points and show that the decision boundary is uniquely determined (note that the magnitude of $\boldsymbol{\theta}$ does not matter).

Exercise 2: Implementations

In this exercise, we will implement the three linear classification algorithms in the binary classification setting, visualize how the decision boundary changes throughout the iterates, and compare the performances of the algorithms.

We will use a 2D randomly generated linearly separable dataset for easier visualization. In the following exercises, choose the maximum number of iterations M that your algorithm runs and the learning rate α_k based on your judgement, as we will only provide rough suggestions.

Tasks:

(a) Import the data files: the file containing the 2D sample vectors `hw04_sample_vectors.csv` (1000 samples), and the file containing the corresponding labels `hw04_labels.csv` (1000 labels).

Caution: the labels for the two classes are 0 and 1 instead of +1 and −1.

(a) Based on (2), implement logistic regression on the dataset. Plot the decision boundary for every $0.2 \times M$ iterates. You can base your implementation on the following code snippet. Comment on your results. For example, how do the decision boundaries vary with respect to the number of iterations? What do you notice about the magnitude of θ when you vary the maximum number of iterates?

```
def logistic(X, labels, learning_rate=???, max_num_iterations=???)
    for m in range(max_num_iterations):
        theta -= learning_rate*cost_function_derivative(X, labels, ???)
    return theta
```

Remark: When α_k is too small, the number of iterations to achieve good separation of data could be unrealistically large; but when α_k is too big, the iterates simply might not land on a good $\theta^{(k)}$. To begin with, try α_k in the range of 10^{-1}.

(b) (i) Using (4) **online** mode, implement the perceptron algorithm on the dataset. Plot the decision boundary for every $0.2 \times M$ iterates. You can base your implementation on the following code snippet.

```
def perceptron(X, labels, learning_rate=???, max_num_iterations=???)
    for m in range(max_num_iterations):
        # Preprocessing. Randomly shuffle the data to potentially increase
        # convergence rate and accuracy in online mode
        shuffled_index = np.random.permutation(labels.size)
        X = X[:, shuffled_index]
        labels = labels[shuffled_index]
        for i, label in enumerate(labels):
            # Your code for updating theta
        if every sample classified correctly:
            break
    return theta
```

(ii) Using (4) **batch** mode, implement the perceptron algorithm on the dataset. Plot the decision boundary for every $0.2 \times M$ iterates. Compare the performance of batch mode to online mode, and explain your observations.

(c) (i) Based on (5), implement the **hard-margin** SVM on the dataset. Plot the final decision boundary.

Hint: There are many ways of solving the quadratic programming problem. For example, you can continue using cvxpy, or `scipy.optimize` from the Scipy library for a more manual implementation.

(ii) Based on (11) and choosing $C = 1$, implement the **soft-margin** SVM with l_1 norm on the dataset. You can use (11) directly, or just implement the original optimization problem if you prefer. Plot the final decision boundary. Comment on your solution. For example, does the final decision boundary perform better than the one from (i)? How does variation in C affect the final solution? Why?

(d) Comment on the performance of the four classifiers. For example, are all the samples correctly classified for each classifier? How long did it take for each of the methods to finish training? What happens when you increase/decrease α_k's for logistic regression and the perceptron? How robust are the classifiers? What metrics are you using to measure robustness? One metric you can use is the signed margin

$$\gamma_{\text{signed}} = \min_{j \in \{1,\ldots,N\}} y_j \frac{\boldsymbol{w}^{*T}\boldsymbol{x}_j + w_0^*}{\|\boldsymbol{w}^*\|_2} \tag{20}$$

but feel free to use additional metrics. Justify your answers.