Derivation of Newton Algorithm

Exercise 1: Part 1: Newton Algorithm

In Newton algorithm we iteratively update weight vector, using the following equation:

$$\underset{\sim}{w}_{t+1} \leftarrow \underset{\sim}{w}_t - \eta_t \left[\nabla^2 f(\underset{\sim}{w}_t)\right]^{-1} \cdot \nabla f(\underset{\sim}{w}_t)$$

where $\nabla^2 f(\underset{\sim}{w}_t)$ is Hessian and $\nabla f(\underset{\sim}{w}_t)$ is gradient.

Given: $f(\underset{\sim}{w}) = \frac{1}{h_+} \sum_{i:y_i=1} \log(1+\exp(-y_i \underset{\sim}{w}^T \underset{\sim}{x}_i)) + \frac{1}{h_-} \sum_{j:y_j=-1} \log(1+\exp(-y_j \underset{\sim}{w}^T \underset{\sim}{x}_j)) + \lambda \|\underset{\sim}{w}\|_2^2$

$\boxed{p_i = \frac{1}{1+\exp(-\underset{\sim}{w}^T \underset{\sim}{x}_i)}}$ → sigmoid function

$$\nabla f(\underset{\sim}{w}) = \frac{df(\underset{\sim}{w})}{d\underset{\sim}{w}} = \frac{1}{h_+} \sum_{i:y_i=1} \frac{-\underset{\sim}{x}_i y_i \exp(-y_i \underset{\sim}{w}^T \underset{\sim}{x}_i)}{1+\exp(-y_i \underset{\sim}{w}^T \underset{\sim}{x}_i)} + \frac{1}{h_-} \sum_{j:y_j=-1} \frac{-\underset{\sim}{x}_j y_j \exp(-y_j \underset{\sim}{w}^T \underset{\sim}{x}_j)}{1+\exp(-y_j \underset{\sim}{w}^T \underset{\sim}{x}_j)} + 2\lambda \underset{\sim}{w} =$$

$$= \frac{1}{h_+} \sum_{i:i=1} \left(-y_i \underset{\sim}{x}_i + \frac{y_i \underset{\sim}{x}_i}{1+\exp(-y_i \underset{\sim}{w}^T \underset{\sim}{x}_i)}\right) + \frac{1}{h_-} \sum_{j:y_j=-1} \left(-y_j \underset{\sim}{x}_j + \frac{y_j \underset{\sim}{x}_j}{1+\exp(-y_j \underset{\sim}{w}^T \underset{\sim}{x}_j)}\right) + 2\lambda \underset{\sim}{w} =$$

$$= \frac{1}{h_+} \sum_{i:y_i=1} \left(-\underset{\sim}{x}_i + \frac{\underset{\sim}{x}_i}{1+\exp(-\underset{\sim}{w}^T \underset{\sim}{x}_i)}\right) + \frac{1}{h_-} \sum_{j:y_j=-1} \left(\underset{\sim}{x}_j + \frac{\underset{\sim}{x}_j}{1+\exp(\underset{\sim}{w}^T \underset{\sim}{x}_j)}\right) + 2\lambda \underset{\sim}{w} =$$

$$= \frac{1}{h_+} \sum_{i:y_i=1} \left(-\underset{\sim}{x}_i + \frac{\underset{\sim}{x}_i}{1+\exp(-\underset{\sim}{w}^T \underset{\sim}{x}_i)}\right) + \frac{1}{h_-} \sum_{j:y_j=-1} \left(\underset{\sim}{x}_j - \underset{\sim}{x}_j \left(1 - \frac{1}{1+\exp(-\underset{\sim}{w}^T \underset{\sim}{x}_j)}\right)\right) + 2\lambda \underset{\sim}{w} =$$

$$= \frac{1}{h_+} \sum_{i:y_i=1} \left(\underset{\sim}{x}_i \left(-1 + \frac{1}{1+\exp(-\underset{\sim}{w}^T \underset{\sim}{x}_i)}\right)\right) + \frac{1}{h_-} \sum_{j:y_j=-1} \left(\underset{\sim}{x}_j \frac{1}{1+\exp(-\underset{\sim}{w}^T \underset{\sim}{x}_j)}\right) + 2\lambda \underset{\sim}{w} =$$

($\frac{1}{1+\exp(-\underset{\sim}{w}^T \underset{\sim}{x}_i)} \rightarrow \underset{\sim}{p}_i$, $\frac{1}{1+\exp(-\underset{\sim}{w}^T \underset{\sim}{x}_j)} \rightarrow \underset{\sim}{p}_j$)

$$= \boxed{\frac{1}{h_+} \sum_{i:y_i=1} \underset{\sim}{x}_i (p_i - 1) + \frac{1}{h_-} \sum_{j:y_j=-1} \underset{\sim}{x}_j p_j + 2\lambda \underset{\sim}{w}}$$

In a vectorized form it can be expressed as:

$$\boxed{\nabla f(\underset{\sim}{w}) = \frac{1}{h_+} X_{pos}^T (\underset{\sim}{p}_{pos} - 1) + \frac{1}{h_-} X_{neg}^T (\underset{\sim}{p}_{neg}) + 2\lambda \underset{\sim}{w}}$$

→ Implementation attached

where $X_{pos} = X[y=1]$ $\quad \underset{\sim}{p}_{pos} = \underset{\sim}{p}[y=1]$

$X_{neg} = X[y=-1]$ $\quad \underset{\sim}{p}_{neg} = \underset{\sim}{p}[y=-1]$

Alternatively, gradient can be expressed as follows:

$$\nabla f(\underset{\sim}{w}) = \frac{1}{n_+}\left(\frac{y+1}{2}\right) X (\underset{\sim}{p} - 1) + \frac{1}{n_+}\left(\frac{1-y}{2}\right) X \underset{\sim}{p} + 2\lambda \underset{\sim}{w}$$

Hessian

$$\nabla^2 f(\underset{\sim}{w}) = \frac{d\,\nabla f(\underset{\sim}{w})}{d\,\underset{\sim}{w}} = \frac{d}{d\,\underset{\sim}{w}}\left(\frac{1}{n_+}\sum_{i:y_i=1} \underset{\sim}{x_i}(p_i - 1) + \frac{1}{n_-}\sum_{j:y_j=-1} \underset{\sim}{x_j} p_j + 2\lambda \underset{\sim}{w}\right) =$$

$$= \frac{1}{n_+}\sum_{i:y_i=1} \underset{\sim}{x_i}\,\underset{\sim}{x_i}^T p_i(1-p_i) + \frac{1}{n_-}\sum_{j:y_j=-1} \underset{\sim}{x_j}\,\underset{\sim}{x_j}^T p_j(1-p_j) + 2\lambda I =$$

$$= \frac{1}{n_+} X_{pos}^T \left(\underset{\sim}{p}_{pos}(1-\underset{\sim}{p}_{pos})\right) \cdot X_p + \frac{1}{n_-} X_{neg}^T (\underset{\sim}{p}_{neg}(1-\underset{\sim}{p}_{neg})) X_{neg} + 2\lambda I$$

where $X_{pos} = X[y=1]$ → rows of matrix X corresponding to $y=1$

$X_{neg} = X[y=-1]$ → rows of matrix X corresponding to $y=-1$

$$\underset{\sim}{p} = \begin{bmatrix} p_1 \\ p_2 \\ \vdots \\ p_n \end{bmatrix} ; \quad \underset{\sim}{p}_{pos} = \underset{\sim}{p}[y=1] ; \quad \underset{\sim}{p}_{neg} = \underset{\sim}{p}[y=-1]$$

Logistic Regression on CIFAR-10 data set

1. One-vs-All strategy for multiclass classification

The one-vs-all strategy trains 10 classifiers, each time with some c-th class as positive and the rest as negative. For prediction, we run all 10 classifiers and predict with the classifier that has the highest dot product (probability/confidence).

The implemented Newton Algorithm was tested on the CIFAR-10 dataset, using the one-vs-all strategy. Only the first training batch was used for training. The table below summarizes the results of this test.

Regression constant λ	**Percent error**
0.1	0.7405
1	0.7103
10	0.6815

The results of all tests are significantly better than random prediction, which would give the percent error of 0.9. Out of three values of regularization constant that was tested, $\lambda = 10$ yielded the best performance with 0.6815 percent error, or the accuracy rate of 0.3185.

The implementation of the algorithm is attached to the assignment (in ipython notebook).

2. One-vs-One strategy for multiclass classification

The one-vs-one strategy trains 45 classifiers, each time with some c-th class as positive and some c'-th class as negative. For prediction, we run all 45 classifiers, each of which will vote either for some class c or some class c'. Predict with the class that has most votes (break ties arbitrarily).

The implemented Newton Algorithm was tested on the CIFAR-10 dataset, using the one-vs-one strategy. Only the first training batch was used for training. The table below summarizes the results of this test.

Regression constant λ	**Percent error**
0.1	0.692
1	0.686
10	0.6668

The results of all tests are significantly better than random prediction, which would give the percent error of 0.9. Similarly to the one-vs-all strategy, out of three

values of regularization constant that was tested, $\lambda = 10$ yielded the best performance with 0.6668 percent error, or the accuracy rate of 0.3332.

The implementation of the algorithm is attached to the assignment (in ipython notebook).

Kernel Logistic Regression on CIFAR-10 data set

The implementation of the kernel binary algorithm is attached in ipython notebook. The tables below summarize the results of the test with three different kernels used.

Regression constant λ	**Percent error**
0.1	0.438
1	0.438
10	0.438

Table 3.1: Linear kernel

For linear kernel the results of all tests are better than random prediction, which would give the percent error of 0.5. The percent error turned out to be 0.438 for all values of λ that were tested.

Regression constant λ	**Percent error**
0.1	0.4395
1	0.439
10	0.4365

Table 3.2: Polynomial kernel

For polynomial kernel the results of all tests are better than random prediction, which would give the percent error of 0.5. Out of all tests, the smallest percent error of 0.4365 was obtained when λ=10.

Regression constant λ	**Percent error $\sigma = 1$**	**Percent error $\sigma = 5$**	**Percent error $\sigma = 10$**
0.1	0.4305	0.4305	0.4305
1	0.453	0.453	0.453
10	0.4625	0.4625	0.4625

Table 3.3: Gaussian kernel

For Gaussian kernel the results of all tests are better than random prediction, which would give the percent error of 0.5. The results were the same for different sigma values. Out of all tests, the smallest percent error of 0.4305 was obtained when λ=0.1.

Derivation of Kernel Logistic Regression

Part 4: Kernel Logistic Regression

Kernelize: $\underset{\sim}{x}_i \rightarrow \phi(\underset{\sim}{x}_i)$.

Then the objective function becomes:

$$f(\underset{\sim}{w}) = \frac{1}{n_+}\sum_{i:y_i=1} \log\left(1+\exp\left(-y_i\, \underset{\sim}{w}^T\phi(\underset{\sim}{x}_i)\right)\right) + \frac{1}{n_-}\sum_{j:y_j=-1} \log\left(1+\exp\left(y_j\, \underset{\sim}{w}^T\phi(\underset{\sim}{x}_j)\right)\right) + \lambda\|\underset{\sim}{w}\|_2^2$$

$\underset{\sim}{w} \in \mathbb{R}^n$

$$\underset{\sim}{w} = \sum_i \alpha_i \phi(\underset{\sim}{x}_i) \Rightarrow \underset{\sim}{w}^T\phi(\underset{\sim}{x}_i) = \sum \alpha_i \underbrace{\phi^T(\underset{\sim}{x}_i)\,\phi(\underset{\sim}{x}_i)}_{K} = \underset{\sim}{\alpha}^T K_{:i} \Rightarrow$$

$$\Rightarrow \boxed{f(\underset{\sim}{\alpha}) = \frac{1}{n_+}\sum_{i:y_i=1} \log\left(1+\exp\left(-y_i\, \underset{\sim}{\alpha}^T K_{:i}\right)\right) + \frac{1}{n_-}\sum_{j:y_j=-1} \log\left(1+\exp\left(-y_j\, \underset{\sim}{\alpha}^T K_{:i}\right)\right) + \lambda\, \underset{\sim}{\alpha}^T K\, \underset{\sim}{\alpha}}$$

This expression has exactly the same form, as the objective function in part 1. Here we have $\underset{\sim}{\alpha}$ instead of $\underset{\sim}{w}$, $K_{:i}$ instead of $\underset{\sim}{x}_i$ and K instead of X.

$\Rightarrow$ Newton algorithm can be applied to solve the minimization problem $\Rightarrow$

$$\Rightarrow \underset{\sim}{\alpha}_{t+1} = \underset{\sim}{\alpha}_t - \left[\nabla^2 f(\underset{\sim}{\alpha}_t)\right]^{-1} \cdot \nabla f(\underset{\sim}{\alpha}_t)$$

$\nabla^2 f(\underset{\sim}{\alpha}_t)$ and $\nabla f(\underset{\sim}{\alpha}_t)$ have the same form as $\nabla f^2(\underset{\sim}{w}_t)$ and $\nabla f(\underset{\sim}{w}_t)$ in part 1. $\Rightarrow$ following the same derivations, we can show that:

$$\begin{cases} \nabla f(\underset{\sim}{\alpha}_t) = \frac{1}{n_+} K_{pos}^T(\underset{\sim}{p}_{pos} - 1) + \frac{1}{n_-} X_{neg}^T(\underset{\sim}{p}_{neg} - 1) + 2\lambda K \underset{\sim}{\alpha}_t \\ \nabla^2 f(\underset{\sim}{\alpha}_t) = \frac{1}{n_+} K_{pos}^T(\underset{\sim}{p}_{pos}(1-\underset{\sim}{p}_{pos}))K_{pos} + \frac{1}{n_-} K_{neg}^T(\underset{\sim}{p}_{neg}(1-\underset{\sim}{p}_{neg}))K_{neg} + 2\lambda K \end{cases}$$

Implementation attached

$K_{pos} = K[y=1]$, $K_{neg} = K[y=-1]$ } matrices composed of rows of matrix K corresponding to $y=1$ and $y=-1$

Now $p_i = \frac{1}{1+\exp(-\underset{\sim}{\alpha}^T k_{:i})} \Rightarrow \underset{\sim}{p} = \begin{bmatrix} p_1 \\ \vdots \\ p_n \end{bmatrix}$

$\underset{\sim}{p}_{pos} = \underset{\sim}{p}[y=1]$; $\underset{\sim}{p}_{neg} = \underset{\sim}{p}[y=-1]$

Time complexity of training is $O(\underbrace{n^2 d}_{\text{calculating kernel matrix}})$

Testing

for a new test point $\underset{\sim}{x}'$:

$$p = \frac{1}{1+\exp(-\underset{\sim}{w}^T \phi(\underset{\sim}{x}'))} = \boxed{\frac{1}{1+\exp(-\underset{\sim}{\alpha}^T k(\underset{\sim}{x}'))}}$$

where $k(\underset{\sim}{x}') = K(X, x')$

↳ training samples

Time complexity for testing is $\underbrace{O(nd)}_{\text{calculating kernel vector } k(\underset{\sim}{x}')}$