# Exercise

1. What is the relationship between the conditions of SVM updates and log-linear models updates? The SVM counterpart for the weights $P(y = +1|x_i) - 1$ and $P(y = +1|x_i)$ for log-linear models in Algorithm 3 and Algorithm 4 is a condition that parameters are updated only if $\cdot\big((x_i, y_i) - (x_i, c)\big) < 1$, which can be viewed as a *hard condition* alternative to the *soft* weights of log-linear models for controlling parameter updates according to violations.

2. Consider the *indicator loss function* for multi-class SVM, which considers all violations $L = \sum_{i=1}^{N} \sum_{c} \max\Big(0, 1 - \cdot(x_i, y_i) + \cdot(x_i, c)\Big) + \frac{1}{2}\lambda||\vec{\theta}||^2$.

   (a) Calculate the derivative $\frac{\partial L}{\partial}$.

   (b) Based on $\frac{\partial L}{\partial}$, give the SGD training algorithm, comparing its update rules with log-linear models and perceptrons.

3. Define a log-linear model training objective with $L2$ regularization. What is the parameter update value for this training objective in SGD training?

4. Some researchers say that regularised perceptron is as good as SVM. Can you give some theoretical justification to this claim?

5. The generalized linear model allows flexible loss functions to be defined. With SGD training, define the parameter updates for the models below. Can each be regarded as a log-linear model, SVM or perceptron?

   (a) a discriminative linear model with a large-margin objective and $L1$-regularization, where the loss function is: $\text{loss}(\vec{\theta}) = \frac{1}{N}\sum_{i=1}^{N} \ell\Big(x_i, y_i, \vec{\theta}\Big) + \lambda||\vec{\theta}||$ where $\ell(x_i, y_i, \vec{\theta}) = \max_{y \neq y_i}\Big(\vec{\theta} \cdot (x_i, y) + 1\Big) - \vec{\theta} \cdot \vec{\phi}(x_i, y_i)$

   (b) a discriminative linear model with a maximum log-likelihood objective and $L2$-regularization, where the loss function is: $\text{loss}(\vec{\theta}) = -\sum_{i=1}^{N} \log P(y_i|x_i) + \frac{\lambda}{2}||\theta||^2$

6. For some classification tasks, the output classes can be **unbalanced**, with one text class being much more frequent compared with other classes. Take spam filters for example, the number of spam emails can be much larger than that of non-spam emails for some accounts. If the training data are highly unbalanced, a classifier may decide to predict the majority class label for every test input. For example, if the class $+1$ is 9 times the class $-1$, then a fixed guess of $+1$ for all test samples can give a 90% accuracy. One way to address this issue is to *under sample* training instances, so that the number of training samples for all classes are balanced. This method has been shown to be effective empirically. However, this *data-level* method can fail to utilize all gold-standard training samples. Let us

consider some *algorithm-level methods* instead. For simplicity, only binary classification is discussed.

(a) For probabilistic models (e.g., log-linear models), one method to alleviate our prediction of the majority class is to adjust the *decision threshold*. In particular, we have assumed that $P(y = +1|x) = 0.5$ is the threshold so that instances with $P(y = +1|x)0.5$ is assigned class $+1$, and others class $-1$. If the datasets are unbalanced, discuss how the threshold 0.5 can be adjusted.

(b) For large-margin models (e.g SVMs and perceptrons), the score $\theta \cdot \phi(x)$ cannot be interpreted directly. However, we can adjust the training objective so that the minority class has a better chance being considered. One example is *cost-sensitive training*, where the training objective becomes to minimize $1\frac{}{2||\vec{w}||^2 + C^+ \sum_{i+} \xi_i + C^- \sum_{i-} \xi_i}$ so that $\mathrm{y}_i \times \left(\vec{w}^T \vec{v}(x_i) + b\right) 1 - \xi_i, \qquad \xi_i 0$ This can translate to $(\vec{w}, b) =_{(\vec{w},b)} \left(C^+ \sum_{i+} \max\left(0, 1 - (\vec{w}^T \vec{v}(x_i) + b)\right)\right.$ $\left. + C^- \sum_{i-} \max\left(0, 1 + (\vec{w}^T \vec{v}(x_i) + b)\right) + \frac{1}{2}\vec{w}^2\right)$ Discuss how to set the costs $C^+$ and $C^-$.

(c) Empirically verify your methods.

7. Implement SGD training for log-linear models and SVM.

(a) Consider adding L1 regularisation to log-linear models. The regularisation term is not differentiable. Specify a sub-gradient for the weight update, following examples from SVM training.

(b) Unlike feature vectors, which are highly sparse, with only a few non-zero elements for each training instances, (sub-)gradients for L1 and L2 regularisation can have non-zero values for most elements in $\vec{\theta}$. As a result, SGD training can be significantly slower at each iterative step. One way to expedite training is **lazy update**. The main idea is not to update a weight value if the corresponding feature is not used in the current training instance. Instead, a record can be used to remember the last iteration at which the weight is updated. For most weight values, there is a constant incremental change through a large number of iterations. When a weight value is used or updated, the constant incremental value per iteration can be cumulatively added to the weight, by multiplication with the number of iterations from the last update to the current iteration. Use the lazy update strategy to optimise L1 and L2 training, comparing the resulting efficiency with naive parameter update.

8. Generalize self-training, co-training and tri-training into one unified semi-supervised algorithm, and write its pseudocode.