# Introduction

This assignment serves as pseudo pre-requisite for the course. For most, we expect it to be a refresher. Its primary purpose is to bring you up-to-speed on:

- Programming using Python and its numerical libraries, numpy and TensorFlow.
- Applying the chain rule through matrix calculus (i.e. backpropagation).
- Machine learning concepts, including support vector machines (SVMs) and Gaussian mixture models (GMMs).

In this assignment, you will implement stochastic learning algorithms for modified versions of SVMs and GMMs. You will be *directly optimizing* the objective functions of these models using the analytical gradients w.r.t. model parameters.

Your assignment must be uploaded to Vocareum (details TBD). You must upload the following:

- File **code.py**. You should fill the starter code at sections marked with `# TODO(student)`. Starter code can be downloaded from: http://sami.haija.org/cs699/hw1/code.py. Feel free to add new functions. However, do not change names or signatures of existing functions, or else auto-grader would fail to work.
- Files **q1_b.pdf** and **q2_b.pdf**, as described in the corresponding parts.

Every time you upload your code to Vocareum, your **code.py** will be automatically-graded, which should immediately print your score. Our automatic grading script will use python3, so please write python3-compatible code. All other files will be manually-graded.

**Bonus:** For people who submit (their last submission) before 10 PM on Friday September 13, they will get 1% bonus credit on the course.

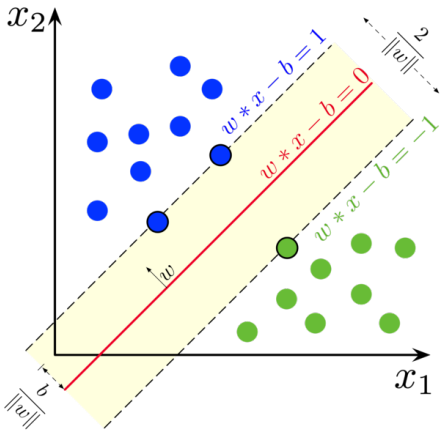
# [15 points] Asymmetric Margin Support Vector Machines

Support vector machines (SVMs) are known to be some of the best off-the-shelf classification algorithms. They are easy to train, and their objective function is convex.

Given data $n$ training examples $\left\{(\mathbf{x}^{(i)}, y^{(i)})\right\}_{i=1}^{n}$, with the $i$th example having feature vector $\mathbf{x}^{(i)} \in \mathbb{R}^m$, and binary label $y^{(i)} \in \{-1, 1\}$, an SVM with linear kernel solves the following problem:

$$\underset{\mathbf{w},b}{\text{minimize}}\, \lambda||\mathbf{w}||^2 + \frac{1}{n}\sum_{i=1}^{n} \max(0, 1 - y^{(i)}(\mathbf{w}^\top \mathbf{x}^{(i)} + b)), \tag{1}$$

where $\lambda \in \mathbb{R}$ is the L2-regularization coefficient, and $\mathbf{w} \in \mathbb{R}^m, b \in \mathbb{R}$ are the trainable model parameters. The objective in equation (1) is known as the hinge loss. It finds the maximum margin decision boundary between the positive and negative classes, which is *equidistant* from the classes. See the figure below taken from Wikipedia.




In this assignment your goal is to find a decision boundary that is *not necessarily equidistant* from the classes, i.e. when the distance from the positive class is proportional to $\alpha \in \mathbb{R}_+$, while the distance from the negative class is proportional to $\beta \in \mathbb{R}_+$. One can achieve this by modifying the objective of equation the following way:

$$\underset{\mathbf{w},b}{\text{minimize}}\, \lambda||\mathbf{w}||^2 + \frac{1}{n}\sum_{i=1}^{n} \max(0, g^{(i)} - y^{(i)}(\mathbf{w}^\top \mathbf{x}^{(i)} + b)), \tag{2}$$

with $g^{(i)} \triangleq \mathbf{1}[y^{(i)} = 1]\alpha + \mathbf{1}[y^{(i)} = -1]\beta$.

10 pts (Q1.a) **[Implementation]** Implement the function `stochastic_update` in class `AsymSVMModel`, such that it applies the learning rules:

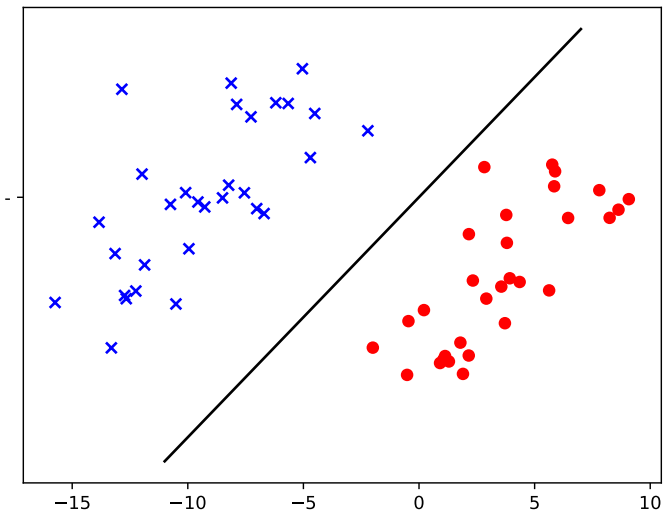
$$\mathbf{w} \leftarrow \mathbf{w} - \eta \frac{\partial}{\partial \mathbf{w}} \left( \lambda ||\mathbf{w}||^2 + \frac{1}{n} \sum_{i=1}^{n} L_i \right),$$
$$b \leftarrow b - \eta \frac{\partial}{\partial b} \left( \frac{1}{n} \sum_{i=1}^{n} L_i \right),$$

where $\eta \in \mathbb{R}$ is the learning rate, and the loss $L_i$ is defined as:

$$L_i = \max(0, g(y^{(i)}) - y^{(i)}(\mathbf{w}^\top \mathbf{x}^{(i)} + b)).$$

You must implement this part in numpy. We will deduct 5 points if you use tensorflow.

5 pts (Q1.b) **[Visualization]** Download data file: http://sami.haija.org/cs699/hw1/q1.npz. On which, train your model using $\alpha = 3, \beta = 1$, and produce a scatter plot where every positive example is shown as a blue-cross and every negative example as a red circle. In addition, plot the decision boundary as black solid line. The code for this visualization can live in code.py and you should call call it within the `main()` block. Feel free to comment-out your code, visualization as long as you clearly indicate in a comment steps for teaching staff to replicate your plot. The plot must be saved as `q1_b.pdf` and submitted via vocareum. Your figure should look somewhat like:




[Note: this data is slightly different from what your file]

# [25 points] Parametrized-Mean Gaussian Mixture Model

The vanilla Gaussian mixture model (GMM) is a generative model. Given a dataset $\mathcal{D} = \{\mathbf{x}^{(i)}\}_{i=1}^n$ with $\mathbf{x}^{(i)} \in \mathbb{R}^m$, it models $p(\mathbf{x})$ as a mixture of $K$ Gaussians:

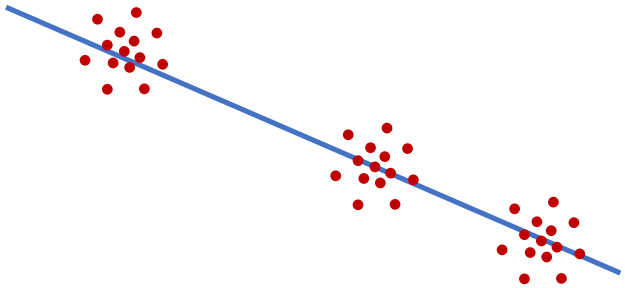
$$p(\mathbf{x}) = \sum_{j=1}^{K} p(z=j)p(\mathbf{x}|z=j),$$

with

$$p(z=j) = \pi_j, \quad p(\mathbf{x}|z=j) = \mathcal{N}(\mathbf{x}; \mu_j, \Sigma_j),$$

where mean vector $\mu_j \in \mathbb{R}^m$ and covariance matrix $\Sigma_j \in \mathbb{R}^{m \times m}$ are the parameters of the $j$'th mixture, and $0 \leq \pi_j \leq 1$, $\sum_{j=1}^K \pi_j = 1$. Even though one can use the expectation-maximization (EM) algorithm to learn the GMM parameters $(\pi_j, \mu_j, \Sigma_j)$, we would like to learn them using stochastic gradient descent (SGD).

Consider a simple problem: assume a truck is transporting water in gallon containers. The truck is moving in a straight line, through an empty field, like:



The path of truck is depicted in blue. The red dots indicate spilled water concentrations. Specifically, in the depiction, 3 gallon containers fell from the truck and upon their impact, the gallons broke, each creating a spill of water that can be modelled as a 2-dimensional Gaussian distribution.

Unfortunately, we do not know exactly the path of the truck (i.e. the equation of the straight line). All we observe are a 2-dimensional top-view scan of where the $n$ water drops are (i.e. $\mathcal{D} = \{\mathbf{x}^{(i)}\}_{i=1}^n$ with $\mathbf{x}^{(i)} \in \mathbb{R}^2$). Further, we can count how many gallons are missing (i.e. $K$). Nonetheless, we would like to inject our knowledge into our model. Specifically, we want to fit a GMM where:

- $\pi_1 = \pi_2 = \cdots = \pi_K$, since each container has the same amount of water.
- $\Sigma_j = I_{2\times 2}$; covariance matrix is identity[1].

[1] Makes training via SGD earlier. If was it was a diagonal matrix, one would have to use another training algorithm such as one with different learning rates per parameter, as will be taught later in the course.

- Recall straight-line equation in Cartesian coordinates $(x, y) \in \mathbb{R}^2$ as: $y = mx + b$. Therefore, we can model the center of each Gaussian along the line as: $\mu_j = (t_j, mt_j + b)$, with $t_j \in \mathbb{R}$. We can override notation and define $\mu$ as a function: $\mu(t; m, b) = (t, mt + b)$

Now, the parameters of our modified GMM become: $m, b$, and $t_j$, for $j = 1, \ldots, K$.

18 pts (Q2.a) [**Implementation**] Implement the function `fit` class `ParametricGMM`, such that it trains the model. The auto-grader will only check your estimates for $m, b, t_1, \ldots t_K$ are correct. **Use TensorFlow for this question**[2].

Hint: tf.math.logsumexp is numerically stable.

7 pts (Q2.b) [**Visualization**] Download data from: `http://sami.haija.org/cs699/hw1/q2.npy`. Run your model with K=5. Create visualization showing the straight line taken by the truck, and depict the 5 Gaussians on the plot. Your plot should also show the values of $m$ and $b$. Save it as `q2_b.pdf` and submit it via vocareum.

[2]For fun, try to derive the analytical gradient for one of the parameters by hand, just to appreciate the beauty of automatic differentiation of TensorFlow.