```
np.random.seed(0)
```

Question 1.1: Implement the cost function cost/objective function:

$$J(\mathbf{w}) = \frac{1}{2}\|\mathbf{w}\|^2 + C\left[\frac{1}{N}\sum_{i}^{n} \max\left(0,\ 1 - y_i * (\mathbf{w} \cdot x_i + b)\right)\right] \quad (1)$$

In [2]:
```
def compute_cost(W, X, Y, reg_strength=1000):
    # TODO calculate cost function
    w_norm = 0.5*np.dot(W, W)**2

    offsets = np.dot(W, X.T).flatten()*Y
    smax = np.clip(a=1-offsets, a_min=0.0, a_max=np.inf)
    hinge_loss = reg_strength*np.sum(smax)/len(smax)

    cost = w_norm+hinge_loss
    return cost
```

Question 1.2: Write a method that calculate the cost gradient:

$$J(\mathbf{w}) = \frac{1}{N}\sum_{i}^{n}\left[\frac{1}{2}\|\mathbf{w}\|^2 + C\max\left(0,\ 1 - y_i * (\mathbf{w} \cdot x_i)\right)\right] \quad (4)$$

$$\nabla_w J(\mathbf{w}) = \frac{1}{N}\sum_{i}^{n}\begin{cases}\mathbf{w} & \text{if } \max\left(0,\ 1 - y_i * (\mathbf{w} \cdot x_i)\right) = 0 \\ \mathbf{w} - C y_i\, x_i & \text{otherwise}\end{cases} \quad (5)$$

In [3]:
```
def calculate_cost_gradient(W, X_batch, Y_batch, reg_strength=1000):
    # distance = 0
    dw = np.zeros(W.shape[0])
    for x, y in zip(X_batch, Y_batch):
        hinge_loss = np.max((0.0, 1.0-np.dot(W.T, x)*y))
        if hinge_loss == 0:
            dw += W

        else:
            dw += W - reg_strength*x*y

    return dw/len(Y_batch)
```

Question 1.3: Write a method that performs stochastic Gradient descent as follows:

- Caluclate the gradient of cost function i.e. ∇J(w)

- Caluclate the gradient of cost function i.e. ∇J(w)
- Update the weights in the opposite direction to the gradient: w = w — ∝(∇J(w))
- Repeat until conversion or until 5000 epochs are reached

In [4]:
```
def sgd(data, outputs, learning_rate=0.0001, max_epochs=5000):
    weights = np.random.uniform(-1.0, 1.0, size=data.shape[1])
    nth = 1
    prev_cost = 0.0
    cost_threshold = 0.01  # in percent
    # stochastic gradient descent
    for epoch in range(1, max_epochs):
        # shuffle to prevent repeating update cycles
        X, Y = shuffle(data, outputs)
        for x_chunk, y_chunk in zip(np.array_split(X, 100), np.array_split(Y, 100)):
            ascent = calculate_cost_gradient(weights, x_chunk, y_chunk)
            weights -= learning_rate*ascent

        # convergence check on 2^nth epoch
        if epoch == 2**nth:
            cost = compute_cost(weights, X, Y)
            print("Epoch is:{} and Cost is: {}".format(epoch, cost))
            # stoppage criterion
            if epoch == max_epochs: # or np.abs(prev_cost - cost) < converge:
                return weights
            prev_cost = cost
            nth += 1

    return weights
```

Dataset

In [5]:
```
data = pd.read_csv('data_banknote_authentication.csv')

Y = data.iloc[:, -1]*2-1
X = data.iloc[:, 1:4]
X.insert(loc=len(X.columns), column='intercept', value=1)

X_train, X_test, y_train, y_test = train_test_split(X, Y, test_size=0.4, random_state=42)
```