# DATA 609 - Homework 6: Applications to Stats and Machine Learning

Eddie Xu

## Instructions

Please submit a .qmd file along with a rendered pdf to the Brightspace page for this assignment. You may use whatever language you like within your qmd file, I recommend python, julia, or R.

## Problem 1: Multi-Label Support Vector Machine (CVX Additional Exercises 6.18)

The basic SVM described in chapter 8 of the book is used for classification of data with two labels. In this problem we explore an extension of SVM that can be used to carry out classification of data with more than two labels. Our data consists of pairs:

$(\mathbf{x}_i, y_i) \in \mathbf{R}^n \times \{1, \dots, K\}$, $i = 1, \dots, m$

where $\mathbf{x}_i$ is the feature vector and $y_i$ is the label of the $i$th data point. (So the labels can take the values $1, \dots, K$.)

Our classifier will use $K$ affine functions, $f_k(\mathbf{x}) = \mathbf{a}_k^T\mathbf{x} + \mathbf{b}_k$, $k = 1, ..., K$, which we also collect into affine function from $\mathbb{R}^n$ into $\mathbb{R}^K$ as $f(\mathbf{x}) = A\mathbf{x} + \mathbf{b}$. (Therows of $A$ are $\mathbf{a}_k^T$ .) Given the feature vector $\mathbf{x}$, our model predicts the label $\hat{y} = \text{argmax}_k f_k(\mathbf{x})$, i.e. the predicted label is given by the index of the largest value of the $f_k$ functions evaluated at the data point.

We assume that exact ties never occur, or if they do, an arbitrary choice can be made. Note that if a multiple of 1 is added to $\mathbf{b}$, the classifier does not change. Thus, without loss of generality, we can assume that

$\mathbf{1}^T\mathbf{b} = 0$.

To correctly classify all the data examples perfectly, we would need $f_{y_i}(\mathbf{x}_i) > \max_{k \neq y_i} f_k(\mathbf{x}_i)$ for all $i$. This set of inequalities in $a_k$ and $b_k$, are feasible if and only if the set of inequalities $f_{y_i}(\mathbf{x}_i) \geq 1 + \max_{k \neq y_i} f_k(\mathbf{x}_i)$ are feasible. This motivates the loss function:

$$L(A, \mathbf{b}) = \sum_{i=1}^{m} \left(1 + \max_{k \neq y_i} f_k(\mathbf{x}_i) - f_{y_i}(\mathbf{x}_i)\right)_+$$

where $(u)_+ = \max\{u, 0\}$. The multi-label SVM chooses $A$ and $\mathbf{b}$ to minimize $L(A, b) + \mu\|A\|_F^2$, subject to $\mathbf{1}^T\mathbf{b} = 0$, where $\mu > 0$ is a regularization parameter. (Several variations on this are possible, such as regularizing b as well, or replacing the Frobenius norm squared with the sum of norms of the columns of A.). The Frobenius norm is a generalization of the 2-norm from vectors to matrices, it is defined as $\|A\|_F = \left(\sum_{ij} A_{ij}^2\right)^{1/2}$ and implemented in `CVX` using `norm(A,'fro')`.

(a) Show how to find $A$ and $\mathbf{b}$ using convex optimization. Be sure to justify any changes of variables or reformulation (if needed), and convexity of the objective and constraints in your formulation.

(b) Carry out multi-label SVM on the data given in multi_label_svm_data.csv.

Use the data given in $\mathbf{X}$ and $y$ to fit the SVM model, for a range of values of $\mu$. Use the data given in multi_label_svm_test.csv to test the SVM models. Plot the test set classification error rate (i.e., the fraction of data examples in the test set for which $\hat{y} \neq y$) versus $\mu$.

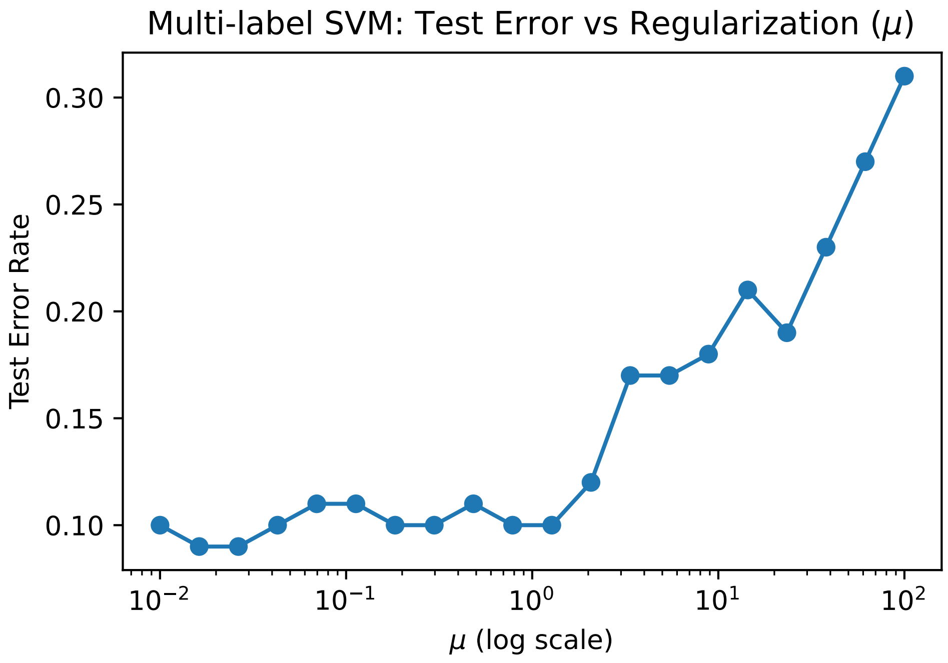
You don't need to try more than 10 or 20 values of $\mu$, and we suggest choosing them uniformly on a `log` scale, from (say) $10^{-2}$ to $10^2$.

## Problem 1 Solution

The problem can be solved using convex optimization because the constraint is affline, therfore it is convex. The hinge loss term is convex and the regulation term, robenius norm squared term is also convex in A.

```
# load dependencies
import cvxpy as cp
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt

# load training and test data
svm_train_url = 'https://media.githubusercontent.com/media/georgehagstrom/DATA609Spring2025/r
svm_test_url = 'https://media.githubusercontent.com/media/georgehagstrom/DATA609Spring2025/r
svm_train_data = pd.read_csv(svm_train_url)
svm_test_data  = pd.read_csv(svm_test_url)

# split both data
```

```
y_train = svm_train_data.iloc[:, 0].values.astype(int)
X_train = svm_train_data.iloc[:, 1:].values

y_test = svm_test_data.iloc[:, 0].values.astype(int)
X_test = svm_test_data.iloc[:, 1:].values

m, n = X_train.shape
K = len(np.unique(y_train))

# define the range of mu values (log spaced)
mu_values = np.logspace(-2, 2, 20)
test_errors = []

# Run the problem for every mu value
for mu in mu_values:
    # Define optimization variables
    A = cp.Variable((K, n))
    b = cp.Variable(K)
    xi = cp.Variable(m)

    # Constraints
    constraints = [xi >= 0, cp.sum(b) == 0]

    for i in range(m):
        x_i = cp.Constant(X_train[i, :])
        for k in range(K):
            if k != (y_train[i] - 1):
                constraints.append(
                    xi[i] >= 1 + cp.matmul(A[k, :], x_i) + b[k] - cp.matmul(A[y_train[i] - 1
                )

    # define the bjective function
    objective = cp.Minimize(cp.sum(xi) + mu * cp.norm(A, 'fro')**2)

    # solve problem
    prob = cp.Problem(objective, constraints)
    prob.solve(solver=cp.SCS, verbose=False)  \

    # predict on test set
    f_test = A.value @ X_test.T + b.value[:, np.newaxis]
    y_pred = np.argmax(f_test, axis=0) + 1  # Adjust back to 1-based labels
    # calculate test error
    test_error = np.mean(y_pred != y_test)
    test_errors.append(test_error)

# plot result
plt.semilogx(mu_values, test_errors, marker='o')
plt.xlabel(r'$\mu$ (log scale)')
plt.ylabel('Test Error Rate')
plt.title('Multi-label SVM: Test Error vs Regularization ($\mu$)')
plt.show()
```



**Problem 2: Maximum Likelihood Prediction of Team Abilities (Adapted from Exercise 7.4 in Convex Optimization Extended Exercises)**

A set of $n$ teams compete in a tournament. We model each team's ability by a number $a_j$, $j = 1, \cdots, n$. When teams $j$ and $k$ play each other, the probability that team $j$ wins is equal to:

$$\text{prob}(a_j - a_k + v > 0)$$

where $v \sim \text{Normal}(0, \sigma^2)$. This means we can also write the probability as $p(\text{i beats j}) == \Phi\left(\frac{a_j - a_k}{\sigma}\right)$, where $\Phi$ is the cumulative distribution function of the standard normal distribution.

You are given the outcome of $m$ past games. These are organized as in a game incidence matrix $A$, where the $l$th row of $A$ corresponds to game $l$ and where:

$$A_{il} = \begin{cases} 1 & \text{if team i played in game l and won} \\ -1 & \text{if team i played in game l and lost} l = k^{(i)} \\ 0 & \text{otherwise} \end{cases},$$

This means that each row of $A$ has exactly two non-zero entries, with a 1 in the column of the team that played and won, and a $-1$ in the column of the team that played and lost.

(a) Formulate the problem of finding the maximum likelihood estimate of team abilities, $\hat{a} \in \mathbb{R}^n$, given the outcomes, as a convex optimization problem. Because the optimal solution can be shifted by a constant, you should specify a prior constraint on the first variable $\hat{a}_0 = 0$.

In order to keep the estimates bounded, an additional set of prior constraints $\hat{a}_i \in [-3, 3]$ should be included in the problem formulation, and you should take $\sigma = 0.25$ to be a constant value rather than a variable. Also, we note that if a constant is added to all team abilities, there is no change in the probabilities of game outcomes.

This means that $\hat{a}$ is determined only up to a constant, like a potential. But this doesn't affect the ML estimation problem, or any subsequent predictions made using the estimated parameters.

(b) Find $\hat{a}$ for the team data by the game incidence matrix AMat_train.csv. (This matrix gives the outcomes for a tournament in which each team plays each other team once.)

You may find the CVX function `log_normcdf` helpful for this problem.

Remember that the cumulative distribution function of a log-concave distribution is log-concave, and also that it is vectorized. Hint: the $l$th row of $A\text{a} = a_{\text{win},l} - a_{\text{lose},l}$.

(c) Use the maximum likelihood estimate $\hat{a}$ found in part (b) to predict the outcomes of next year's tournament games, given in the file team_data_test.csv, using

$$\hat{y}^{(i)} = \text{sign}(\hat{a}_{j^{(i)}} - \hat{a}_{k^{(i)}})$$

The first two rows of this file contain the indices of the two teams playing, and the third column is 1 if the first team won and $-1$ otherwise. Compare the predictions predictions based on $\hat{a}$ with the actual outcomes, given in the third column of test. Give the fraction of correctly predicted outcomes.

The games played in train and test are the same, so another, simpler method for predicting the outcomes in test it to just assume the team that won last year's match will also win this year's match. You can find a similarly structured matrix in the file team_data_test.csv, or you can construct it from the game incidence matrix. Give the percentage of correctly predicted outcomes using this simple method.

**Problem 2(b) and (c) Solution**

```
# load dependencies
import numpy as np
import cvxpy as cp
import pandas as pd

# load data
train_url = 'https://media.githubusercontent.com/media/georgehagstrom/DATA609Spring2025/refs/
A = pd.read_csv(train_url).values

# define variables
m, n = A.shape
a = cp.Variable(n)
sigma = 0.25

# define and calculate residual
residual = A @ a

# define the objective
objective = cp.sum(cp.log_normcdf(residual / sigma))

# define the constraint
constraints = [a[0] == 0, a >= -3, a <= 3]

# define the problem
prob = cp.Problem(cp.Maximize(objective), constraints)

# solve the problem
prob.solve(solver='ECOS', abstol=1e-5, reltol=1e-5, max_iters=10000)

# print result
ahat = a.value
print("\nEstimated team abilities (ahat):\n", ahat)
```

```python
# load the test data
test_data_url = 'https://media.githubusercontent.com/media/georgehagstrom/DATA609Spring2025/
test_data = pd.read_csv(test_data_url).values

# split the test data into team 1 and team 2
team1_indices = test_data[:, 0].astype(int)
team2_indices = test_data[:, 1].astype(int)
actual_outcomes = test_data[:, 2]

# calculate the predicted value based on a_hat

predictions = np.sign(ahat[team1_indices] - ahat[team2_indices])

# define and calculate accuracy
correct = np.sum(predictions == actual_outcomes)
total = len(actual_outcomes)
accuracy = correct / total

# apply the prediction
last_year_winners = np.argmax(A == 1, axis=1)
last_year_losers = np.argmax(A == -1, axis=1)
simple_predictions = np.where(
    team1_indices == last_year_winners, 1,
    np.where(team2_indices == last_year_winners, -1, 0)
)

# compare the predicted value with the actual outcome
correct_simple = np.sum(simple_predictions == actual_outcomes)
accuracy_simple = correct_simple / total

print(f"\nPrediction accuracy assuming last year's winner wins again: {accuracy_simple:.4f}
```

```
Estimated team abilities (ahat):
 [-8.36354694e-11  2.90412993e+00 -1.12088515e+00  1.82166813e+00
 -2.00997238e+00  1.99242627e+00  1.98096517e+00  5.89411388e-01
 -2.92179720e+00  1.81020704e+00 -2.82541274e-01]

Prediction accuracy assuming last year's winner wins again: 0.9778 (97.78%)
```

## Problem 3: Flux balance analysis in systems biology. (Exercise 21.3 in CVX Additional Exercises)

Flux balance analysis is based on a very simple model of the reactions going on in a cell, keeping track only of the gross rate of consumption and production of various chemical species within the cell. Based on the known stoichiometry of the reactions, and known upper bounds on some of the reaction rates, we can compute bounds on the other reaction rates, or cell growth, for example.

We focus on $m$ metabolites in a cell, labeled $M_1$ , . . . , $M_m$ . There are $n$ reactions going on, labeled $R_1$ , . . . , $R_n$ , with nonnegative reaction rates $v_1$ , . . . , $v_n$.

In our particular case, we will be working with a simplified model of cell metabolism having 9 reactions and 6 metabolites. Each reaction has a (known) stoichiometry, which tells us the rate of consumption and production of the metabolites per unit of reaction rate.

The stoichiometry data is given by the stoichiometry matrix $S \in \mathbb{R}^{m\times n}$ , defined as follows: $S_{ij}$ is the rate of production of $M_i$ due to unit reaction rate $v_j = 1$.

Here we consider consumption of a metabolite as negative production; so $S_{ij} = -2$, for example, means that reaction $\mathbb{R}^j$ causes metabolite $M_i$ to be consumed at a rate $2v_j$ .

As an example, suppose reaction $R_1$ has the form $M_1 \to M_2 + 2M_3$ . The consumption rate of $M_1$ , due to this reaction, is $v_1$ ; the production rate of $M_2$ is $v_1$ ; and the production rate of $M_3$ is $2v_1$ . (The reaction $R_1$ has no effect on metabolites $M_4$ , . . . , $M_m$ .)

This corresponds to a first column of $S$ of the form $(-1, 1, 2, 0, \dots, 0)$.

Reactions are also used to model flow of metabolites into and out of the cell. For example, suppose that reaction $R_2$ corresponds to the flow of metabolite $M_1$ into the cell, with $v_2$ giving the flow rate. This corresponds to a second column of $S$ of the form $(1, 0, ..., 0)$.

The last reaction, $R_n$ , corresponds to biomass creation, or cell growth, so the reaction rate $v_n$ is the cell growth rate. The last column of $S$ gives the amounts of metabolites used or created per unit of cell growth rate.

Since our reactions include metabolites entering or leaving the cell, as well as those converted to biomass within the cell, we have conservation of the metabolites, which can be expressed as $Sv = 0$. In addition, we are given upper limits on some of the reaction rates, which we express as $v \preceq v^{\max}$ , where we set $v_j^{\max} = \infty$ if no upper limit on reaction rate $j$ is known.

The goal is to find the maximum possible cell growth rate (i.e., largest possible value of $v_n$ ) consistent with the constraints:

$$\max_v v_9 Sv = 0 v \succeq 0 v \preceq v^{\max}$$

The questions below pertain to the data found in fba_S.csv and fba_vmax.csv, which contain the Stoichiometric Matrix and the upper bounds on the reaction fluxes, respectively. This exercise was inspired by the following paper: Segre et all 2003

(a) Find the maximum possible cell growth rate $G^\star$ , as well as optimal Lagrange multipliers for the reaction rate limits. How sensitive is the maximum growth rate to the various reaction rate limits?

(b) Essential genes and synthetic lethals. For simplicity, we'll assume that each reaction is controlled by an associated gene, i.e., gene $G_i$ controls reaction $R_i$ .

Knocking out a set of genes associated with some reactions has the effect of setting the reaction rates (or equivalently, the associated v max entries) to zero, which of course reduces the maximum possible growth rate.

If the maximum growth rate becomes small enough or zero, it is reasonable to guess that knocking out the set of genes will kill the cell. An essential gene is one that when knocked out reduces the maximum growth rate below a given threshold $G^{\min}$ . (Note that $G_n$ is always an essential gene.)

A synthetic lethal is a pair of non-essential genes that when knocked out reduces the maximum growth rate below the threshold. Find all essential genes and synthetic lethals for the given problem instance, using the threshold $G^{\min} = 0.2G^\star$ .

## Problem 3(a) and (b) Solution

```
# load dependencies
import numpy as np
import pandas as pd
from scipy.optimize import linprog
from itertools import combinations

# load data
fba_s_url = 'https://media.githubusercontent.com/media/georgehagstrom/DATA609Spring2025/refs/
fba_vmax_url = 'https://media.githubusercontent.com/media/georgehagstrom/DATA609Spring2025/r

# define the data
S = pd.read_csv(fba_s_url, index_col = 0).values
vmax = pd.read_csv(fba_vmax_url, index_col= 0).values.flatten()
S = np.array(S, dtype=float).T
vmax = np.array(vmax, dtype=float)

# define variables
```

```python
c = np.zeros(9)
c[8] = -1
A_eq = S
b_eq = np.zeros(S.shape[0])
bounds = [(0, vmax[i]) for i in range(9)]

# define and solve the max growth
res = linprog(c, A_eq=A_eq, b_eq=b_eq, bounds=bounds, method='highs')
G_star = res.x[8]

# define the duality of essential and nonessential

dual_vmax = res.upper
G_min = 0.2 * G_star
essential = []
for i in range(9):
    vmax_ko = vmax.copy()
    vmax_ko[i] = 0
    bounds_ko = [(0, vmax_ko[j]) for j in range(9)]
    res_ko = linprog(c, A_eq=A_eq, b_eq=b_eq, bounds=bounds_ko, method='highs')
    if not res_ko.success or res_ko.x[8] < G_min:
        essential.append(i+1)

nonessential = [i for i in range(9) if (i+1) not in essential]
synthetic = []
for i, j in combinations(nonessential, 2):
    vmax_ko = vmax.copy()
    vmax_ko[i] = vmax_ko[j] = 0
    bounds_ko = [(0, vmax_ko[k]) for k in range(9)]
    res_ko = linprog(c, A_eq=A_eq, b_eq=b_eq, bounds=bounds_ko, method='highs')
    if not res_ko.success or res_ko.x[8] < G_min:
        synthetic.append((i+1, j+1))

# print result
print(f"Maximum growth rate G* = {G_star:.4f}")
print("Essential genes (reactions):", essential)
print("Synthetic lethal pairs:", synthetic)
print("Dual values for reaction bounds:", dual_vmax)
```

```
Maximum growth rate G* = 13.5500
Essential genes (reactions): [1, 9]
Synthetic lethal pairs: [(2, 3), (2, 7), (4, 7), (5, 7)]
```

```
Dual values for reaction bounds:  marginals: array([-0.5,  0. , -0.5,  0. , -1.5,  0. ,  0.
  residual: array([ 0.  , 95.8 ,  0.  , 96.3 ,  0.  , 99.75, 93.85, 99.75, 86.45])
```