

Assignment 2: Support Vector Machines

Due 17 Jul at 23:59 **Points** 35 **Questions** 7
Available 15 Jun at 0:00 - 20 Jul at 23:59 **Time limit** None
Allowed attempts Unlimited

Instructions

Weighting

This assessment is worth 35% of your overall grade.

Course Learning Outcomes

- CLO 1: Apply basic concepts of machine learning, and classic algorithms such as Support Vector Machines and Neural Networks, Deep Learning.
- CLO 2: Demonstrate an understanding of the basic principles and theory of machine learning necessary to develop algorithms.
- CLO 3: Devise algorithms to solve real-world problems.

Task Description

Purpose:

The purpose of this assignment is to examine your ability to apply basic concepts of machine learning, and classic algorithms, Support Vector Machines for this assignment. This is to demonstrate the outcomes of your learning and receive feedback on your progress.

Instructions:

This test comprises multiple-choice questions. You are welcome to use your textbook, readings and notes to assist you in answering the questions. You must complete the online test by the deadline.

Academic Integrity



It is your responsibility to ensure that any work you submit is your own. You can use the **Turnitin Originality Checker** before you submit your work.

Instructions and submission guidelines:

Please read the following instruction carefully before you complete assignment 2

- This is not a group assignment. Everyone is asked to complete the assignment individually.
- The assignment consists of a mixed set of tasks, including derivation, question answering, coding and result analysis. You need to provide your solutions to the corresponding sections in this quiz.
- You need to also upload your code. We will check the code to ensure that your results can be generated from your code. If your result does not match your code, you will get a 0 mark for the relevant sections.
- You should use Python to finish this assignment. Jupyter notebook is allowed and encouraged.

Data:

Please download the data that will be used in this assignment here: [**train.csv**](#) 
[**test.csv**](#) 

There are a "train.csv" file and a "test.csv" file. They are the training data and testing data, respectively. Inside the CSV file, the first column is the class label, the remaining columns are the features.

The label (y) can take one of two classes - 0, 1;

In total, there are 200 features (X);

Trainset consists of 8500 samples, testset has 1500;

Please use the first 4000 samples in the "train.csv" file as the training set and the remaining samples as validation set. All the samples in the "test.csv" file will be used as test samples.

Third-party Libraries,

You can use any third-party libraries to read and process data.

You can use numpy in any question.

Please follow the instruction for each question to complete the relevant parts.

This quiz is no longer available as the course has been concluded.

Attempt history

	Attempt	Time	Score
KEPT	<u>Attempt 2</u>	60 minutes	35 out of 35
LATEST	<u>Attempt 2</u>	60 minutes	35 out of 35
	<u>Attempt 1</u>	40,262 minutes	0 out of 35 *

* Some questions not yet graded

Score for this attempt: **35** out of 35

Submitted 17 Jul at 3:44

This attempt took 60 minutes.

Question 1

10 / 10 pts

Please implement the training and testing algorithms of soft-margin Linear Support Vector Machine from its primal form, that is,

— —

by using CVX. Your implementation should follow the following I/O format:

```
svm_model = svm_train_primal ( data_train , label_train ,
regularisation_para_C )
```

```
test_accuracy = svm_predict_primal ( data_test , label_test ,
svm_model )
```

Please copy the code snippet for the implementation of those two functions (You can also do this by attaching the screenshot of your code.).

By setting $C = 100$, run your implementation, please report the solution of and sum of all dimensions of solution, e.g., `np.sum(w)`. (For a quick check of the correctness of your code)

Your answer:

After running the code, the optimal solution was obtained.
The outputs from the code are shown below:

Status: optimal**The sum of W^* : -0.1452156803361282** **b^* : 1.779813717087077****Accuracy on the test set: 96.8%**

There was one thing that needed to be careful with, which is the 'N' in the formula provided is the number of samples, not the number of the features.

```
def svm_train_primal(data_train, label_train, regularisation_para_C):
    X_train = data_train
    Y_train = label_train

    m, n = X_train.shape #m: number of samples, n: number of features
    W = cp.Variable(n) # w: n*1
    Psi = cp.Variable(m) # Psi: m*1, hinge loss, slack variable
    b = cp.Variable() # b is a scalar parameter
    C = regularisation_para_C # Hyper-parameter, control softness trade-off of the margin

    # Objective in primal problem:
    objective = cp.Minimize(0.5 * cp.sum_squares(cp.norm(W)) + (C/m) * cp.sum(Psi)) ###c/n

    # Constraints in primal problem:
    constraints = [cp.multiply(Y_train, X_train @ W + b) - 1 + Psi >= 0, Psi >= 0]

    # Solve constrained convex optimization
    prob = cp.Problem(objective, constraints)
    prob.solve()

    # Show results
    print('Status: ', prob.status)
    #print('W*: ', W.value)
    print('The sum of W*: ', np.sum(W.value))
    print('b*: ', b.value, '\n')

    # Create svm prime model:
    prime_model = svm_model(W.value, b.value, loss = Psi.value)
    return prime_model

# The I/O format requested in the assignment description
prime_model = svm_train_primal(X_train, Y_train, 100)
test_accuracy = svm_predict_primal(X_test, Y_test, prime_model)
print(f'Accuracy on the test set: {test_accuracy*100}%')
```

good work

Question 2**8 / 8 pts**

Please implement the training algorithm of the soft-margin Linear Support Vector Machine from its dual form, that is,

by using CVX. Your implementation should follow the following I/O format:

**svm_model = svm_train_dual (data_train , label_train ,
regularisation_para_C)**

Please copy the code snippet for the implementation of this function (You can also do this by attaching the screenshot of your code.).

By setting $C = 100$, run your implementation, please report the sum of all dimensions of the optimal α , e.g., `np.sum(alpha)`. (For a quick check of the correctness of your code)

Your answer:

The outputs from the code for dual:

Status: optimal

The sum of Alpha*: 7.28163240568506

The sum of dual W*: -0.1452157032864848

Dual b*: 1.7942996486309701

Accuracy on the test set: 96.8%

I also calculated the W^* and b^* from the dual solution, to check the correctness of the dual solution.

Dual derived W^* and b^* are the same as the prime solution, but with tiny precision rounding errors.

```
def svm_train_dual(data_train, label_train, regularisation_para_C):
    X_train = data_train
    Y_train = label_train
    C = regularisation_para_C

    m, n = X_train.shape #m: number of samples, n: number of features
    Alpha = cp.Variable(m) # Alpha: size m, Lagrange multiplier

    # Objective in dual problem:
    dual_objective = cp.Maximize(cp.sum(Alpha) - 0.5*cp.sum_squares(cp.multiply(Alpha, Y_train) @ X_train))

    # Constraints in dual problem:
    dual_constraints = [Alpha >= 0, Alpha <= (C/m), Alpha.T @ Y_train == 0] #C/sample num

    # Solve constrained convex optimization
    dual_prob = cp.Problem(dual_objective, dual_constraints)
    # dual_prob.solve(verbose=True)

    # solvers will fail when the numerical data is very large or very small, which can lead to what's known as poorly conditioned problem data
    dual_prob.solve(solver=cp.ECOS) #using the ECOS solver solved the OSQP Error

    # Show results
    print('Status: ', dual_prob.status)
    print('The sum of Alpha*: ', np.sum(Alpha.value))

    # Calculate W* and b* from dual problem
    dual_W = X_train.T @ np.multiply(Alpha.value, Y_train)
    dual_b = np.mean(Y_train - X_train @ dual_W)
    print('The sum of dual W*: ', np.sum(dual_W))
    print('Dual b*: ', dual_b, '\n')

    # Create svm dual model:
    dual_model = svm_model(dual_W, dual_b, loss = None, A = Alpha.value)
    return dual_model
```

Question 3

2 / 2 pts

Write code to find the support vectors from the primal problem solutions. Please attach the code snippet for the implementation.

Your answer:

The conditions of being soft-margin support vectors:
 support vectors are the ones that satisfy: $y_i(W^T @ X_i + b) = 1 - \Psi_i$
 There are **392** support vectors found, their details are in the code.

```
#get the values from prime model
W = prime_model.W
b = prime_model.b
Psi = prime_model.Psi

prime_sv_idx = []
for i in range(X_train.shape[0]): #iterate through each sample
    X_i = X_train[i]
    y_i = Y_train[i]
    loss_i = Psi[i]

    res = y_i * (X_i @ W + b) - 1 + loss_i
    # to solve the precision problem, changed '== 0' to a range around +-1e-4
    if res < 1e-4 and res > -1e-4:
        prime_sv_idx.append(i)

prime_sv = X_train[prime_sv_idx]
print(f'There are {len(prime_sv_idx)} support vectors:\n')
print(prime_sv, '\n')
print(f'The index of support vectors in prime: \n {prime_sv_idx}')
```

There are 392 support vectors:

```
[[-0.36 -0.91 -0.99 ... 0.3 2.44 -1.26]
 [ 1.05 -1.79 0.9 ... 0.39 0.6 -1.66]
 [ 1.01 -1.13 1.49 ... 0.23 -0.3 -0.01]
 ...
 [ 2.16 -0.78 -0.78 ... -0.38 1.1 0.39]
 [ 0.36 -0.19 -1.06 ... -0.83 -0.2 0.12]
 [-0.73 -1.19 -0.24 ... 1.46 -1.36 1.21]]
```

Nice work

Question 4

2 / 2 pts

Write code to find the support vectors from the dual problem solutions. Please copy the code snippet for the implementation

Your answer:

In dual problems, only the samples that are associated with a positive Lagrangian coefficient (i.e. $\text{Alpha}[i] > 0$) can have effects on the decision boundary. These samples are the support vectors.

From this condition, there are **392** support vectors found, they are the same as the ones found in the prime problem. Instead of checking every support vector, this can be easily checked by their index in the training set (can see them in my code).

Support Vectors Dual Problem

```
: # In dual problem, only the samples that associated with a positive Lagranian coefficient
# (i.e. Alpha[i] > 0) can have effect on the decision boundary. These samples are the support vectors.
|
Alpha = dual_model.Alpha

dual_sv_idx = []
for i in range(X_train.shape[0]):
    if Alpha[i] > 1e-4:
        dual_sv_idx.append(i)

dual_sv = X_train[dual_sv_idx]
print(f'There are {len(dual_sv_idx)} support vectors:\n')
print(dual_sv, '\n')
print(f'The index of support vectors in dual: \n {dual_sv_idx}')
```

There are 392 support vectors:

```
[[-0.36 -0.91 -0.99 ...  0.3  2.44 -1.26]
 [ 1.05 -1.79  0.9 ...  0.39  0.6 -1.66]
 [ 1.01 -1.13  1.49 ...  0.23 -0.3 -0.01]
 ...
 [ 2.16 -0.78 -0.78 ... -0.38  1.1  0.39]
 [ 0.36 -0.19 -1.06 ... -0.83 -0.2  0.12]
 [-0.73 -1.19 -0.24 ...  1.46 -1.36  1.21]]
```

Question 5

4 / 4 pts

Write code to choose C by using the validation set. Please copy the code snippet for the implementation. Report the test accuracy you get by using the optimal C found in the validation set.

Hint: you can search C from the range

Your answer:

The optimal C is 2^2 (i.e. **4**).

The test accuracy is :

$C = 2^2$ gives accuracy: **97.46666666666667%** on the **test** set.


```

import matplotlib.pyplot as plt

# Create hyper-parameter search space c = [2^-10, 2^-8, ..., 2^8, 2^10]
expo = np.linspace(-10, 10, num = 11)
C_lst = [2**(ex) for ex in expo]
str_lst = ['2^(' + str(int(ex)) + ')'] for ex in expo]
C_dict = dict(zip(str_lst, C_lst))

#initializing
best_score = 0
best_C_idx = 0
acc_lst = []

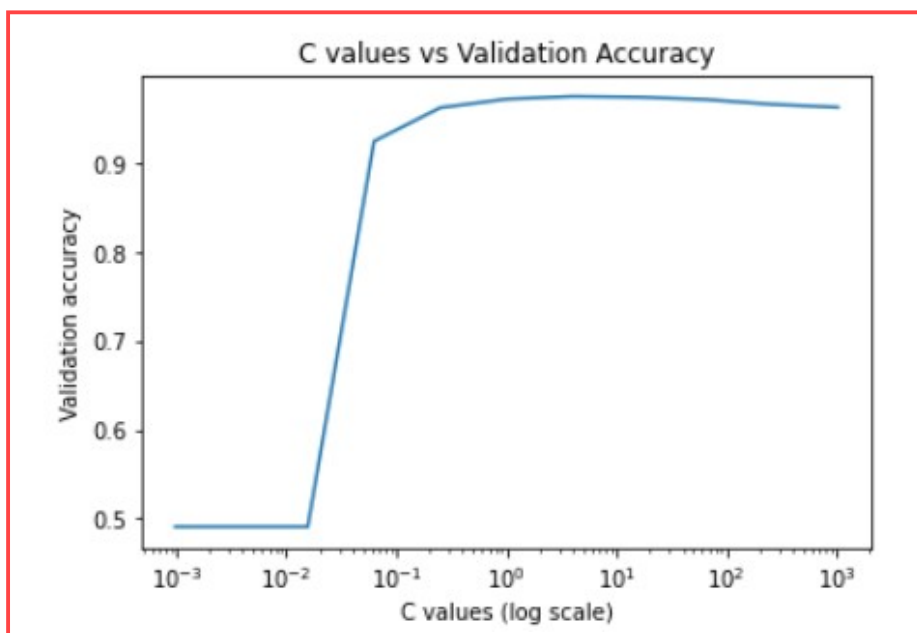
#find best C on validation set
for i, (C_str, C) in enumerate(C_dict.items()):
    p_model = svm_train_primal(X_train, Y_train, C) #create svm model for new C
    val_accuracy = svm_predict_primal(X_val, Y_val, p_model)
    print(f'C = {C_str} validation accuracy: {val_accuracy*100}%')
    acc_lst.append(val_accuracy)
    if val_accuracy > best_score:
        best_score = val_accuracy
        best_C_idx = i #keep the best one
    print('-----')

print('Best C is ', str_lst[best_C_idx], f', which gives {best_score*100}% accuracy.\n')
plt.plot(C_lst, acc_lst)
plt.xlabel('C values')
plt.ylabel('Validation accuracy')
plt.title('C values vs Validation Accuracy')
plt.show()

# Optimal C performance on the test set
print('Test set performance:')
test_model = svm_train_primal(X_train, Y_train, C_lst[best_C_idx])
test_accuracy = svm_predict_primal(X_test, Y_test, test_model)
print(f'C = {str_lst[best_C_idx]} gives accuracy: {test_accuracy*100}% on the test set.')

```

The C value VS validation accuracy plot (C in log scale):



Question 6

6 / 6 pts

Please study one of the following packages and perform classification with linear SVM (with optimal C searched in the validation set) on the assignment dataset

1. Libsvm

<https://www.csie.ntu.edu.tw/~cjlin/libsvm/>

2. Scikit-learn SVM

<https://scikit-learn.org/stable/modules/generated/sklearn.svm.LinearSVC.html#sk>

Please copy your code snippet and report your test accuracy.



Your answer:

1. Grid search using 'accuracy' as the scoring metric:

Best Para in Sklearn model: {'C': 0.004}, the corresponding C in last question: 16.0

Test accuracy: 96.8%
sum of W*: -0.08312181246869857
b*: 0.5394102610170345

2. Grid search using 'roc_auc' as the scoring metric:

Best Para in Sklearn model: {'C': 0.001}, the corresponding C in last question: 4.0

Test accuracy: 96.66666666666667%
sum of W*: -0.038602420901886494
b*: 0.29306626124222945

Method2 ROC_AUC is more recommended because our datasets were not split in a well-balanced way in the first place.

```
# Using the grid search to find optimal C for SKlearn model
from sklearn.model_selection import GridSearchCV

C_grid_lst = [ele/X_train.shape[0] for ele in C_lst]
# Create the parameter grid based on the results of random search
param_grid = {
    'C': C_grid_lst,
}

# Create a based model
clf = svm.LinearSVC()

# Instantiate the grid search model
grid_search = GridSearchCV(estimator = clf, param_grid = param_grid, scoring = 'accuracy',
cv = 5, n_jobs = -1, verbose = 2) #to be consistent as last question, using accuracy as the metric
grid_search.fit(X_val, Y_val)

print(f'Best Para in Sklearn model: {grid_search.best_params_}, the corresponding C in last question: {list(grid_search.best_params_.values)}')

#creat svm model using optimal paras
svm_star = svm.LinearSVC(C=grid_search.best_params_['C'])
# fit train set
svm_star.fit(X_train, Y_train)

# Performance
score = svm_star.score(X_test, Y_test)
print(f'Test accuracy: {score*100}%')
print('sum of W*: ', np.sum(svm_star.coef_))
print('b*: ', svm_star.intercept_[0])
```

good work

Question 7**3 / 3 pts**

Please upload the entire code of your assignment clearly indicating which part of the code is for which question.

SVM.ipynb

clear code