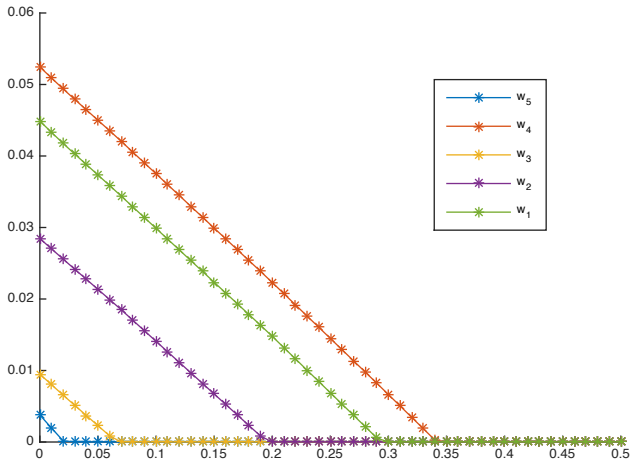

Figure 5: As λ grows, it penalizes w_i-s and drives them to 0, thus we get a sparse w. After λ grows above a certain threshold, w becomes a zero vector, which explains why training and validation errors become constant.

Problem 2 (Support Vector Machine) (10 pts) In this question, we consider an application of SVM in text classification for volatility prediction. Your answer should include figures and source code.

In `NewsData.mat`, you are given a 1470×971 matrix where each row corresponds to a published article and each column to the frequency of a word that appears in the article (i.e. our archive contains 1470 articles and our dictionary contains 971 keywords). Each article is about a certain company. You are also given a 1470×1 vector of labels. An article's label is $+1$ if the article caused an immediate and significant change (positive or negative) to the company's stock. Otherwise, the label is -1. The data has been divided into a training set which will be used to train you SVM and a validation set which will be used to test the SVM's prediction accuracy.

1. A linear SVM can be formulated as the following optimization problem:

$$\min_{w,b} \frac{1}{2} w^T w + C \sum_{i=1}^{1470} \max\{0, 1 - y_i(w^T x_i + b)\}$$

where y_i-s are labels, x_i-s are vectors of word frequencies in articles. w is the weight vector, b is the offset and C is a model parameter. This is a quadratic optimization problem. Please implement this SVM in Matlab/CVX (or other tools if you prefer), tune the parameter C from 0 to 100, fit w and b for each parameter C on the training data set. Plot training accuracy versus C curve and validation accuracy versus C curve. Briefly comment on the result.

2. Perform feature selection to reduce the problem scale. In this case, we want to find the keywords that are most important for classification. Let $C = 10$, perform SVM to fit w on the training set. Sort elements of w by their absolute value in descending order. Then perform SVM on part of features. Sweep the number of (the most significant) features to be used in the model from 10 to 200, with increments of 10. Evaluate the performances on the validation set. Comment on the result.

3. Try a different approach to do feature selection. Subsitute the ℓ_2-norm penalty with an ℓ_1-norm penalty as

$$\min_{w,b} \|w\|_1 + C \sum_{i=1}^{1470} \max\{0, 1 - y_i(w^T x_i + b)\}$$

Sweep the parameter C, plot the number of non-zero elemenets in w versus C curve. Note that due to limited numerical precision, zero elements are not exactly 0. So the criterion for non-zero element is $|w_i| > 10^{-6}$. Comment on the prediction quality after replacing the penalty term.

[solution]

1. The following code implements the linear SVM in Matlab/CVX and solves this problem.

```
% load data
load NewsData

% seperate training and validation set
nTrain = length(idxTrain);
nTest = length(idxTest);
n = size(features,2);

XTrain = features(idxTrain,:);
XTest = features(idxTest,:);
yTrain = labels(idxTrain);
yTest = labels(idxTest);

ATrain = [];
ATest = [];

% sweep C from 0 to 10 in log-scale, then sweep from 10-80 in ...
    linear scale
Cs = [20:10:80];
Cs = [0, logspace(0,1,6), Cs];

for C=Cs
    cvx_begin
```

```
        variables w(n, 1) b(1)
        minimize (0.5*w'*w + C*sum(max(0, 1-yTrain.*(XTrain*w + b))))
    cvx_end
    % Validation
    ATrain = [ATrain, length(find(yTrain.*(XTrain*w + b) >0))/nTrain];
    ATest = [ATest, length(find(yTest.*(XTest*w + b) >0))/nTest];
end

figure; hold on;
plot(Cs, ATrain*100);
plot(Cs, ATest*100);
xlabel('Parameter C'); ylabel('Accuracy %');
legend('Training accuracy', 'Validation accuracy');
```

The training and validation accuracy versus C curve is plotted in Fig. 6.

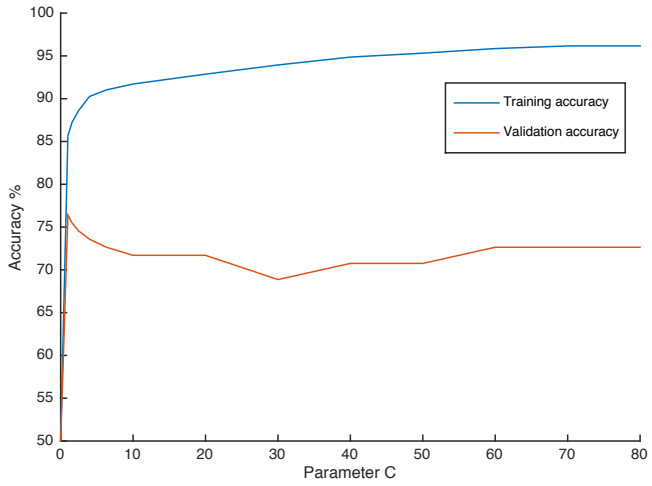

Figure 6: Training accuracy and validation accuracy versus parameter C. When C is 0, both training and validation accuracy are 50%, which is simply a random guess. As C increases, the training accuracy quickly rises to above 90%, but validation accuracy first jumps to 76% and then decreases and stays around 70%.

2. The following code is a solution of this problem.

```
% load data
load NewsData

% seperate training and validation set
nTrain = length(idxTrain);
nTest = length(idxTest);
n = size(features,2);

XTrain = features(idxTrain,:);
XTest = features(idxTest,:);
yTrain = labels(idxTrain);
yTest = labels(idxTest);


% Solve w, sort w_i by their asbolute value.
C = 10;
cvx_begin
    variables w(n, 1) b(1)
    minimize (0.5*w'*w + C*sum(max(0, 1-yTrain.*(XTrain*w + b))))
cvx_end

[ws, idx] = sort(abs(w), 'descend');

ATrain = [];
ATest = [];
```

```
26
27 % Sweep number of features used in SVM
28 nums = [10:10:200];
29 for num = nums
30     XTrainhat = sparse(nTrain, n);
31     XTrainhat(:, idx(1:num)) = XTrain(:, idx(1:num));
32     cvx_begin
33         variables w(n, 1) b(1)
34         minimize (0.5*w'*w + C*sum(max(0, 1-yTrain.*(XTrainhat*w + ...
               b))))
35     cvx_end
36     ATrain = [ATrain, length(find(yTrain.*(XTrain*w + b) >0))/nTrain];
37     ATest = [ATest, length(find(yTest.*(XTest*w + b) >0))/nTest];
38 end
39
40 % Plot
41 figure; hold on;
42 plot(nums, ATrain*100);
43 plot(nums, ATest*100);
44 xlabel('Number of features'); ylabel('Accuracy %');
45 legend('Training accuracy', 'Validation accuracy');
```

The accuracy versus number of features curves are plotted in Fig. 7.

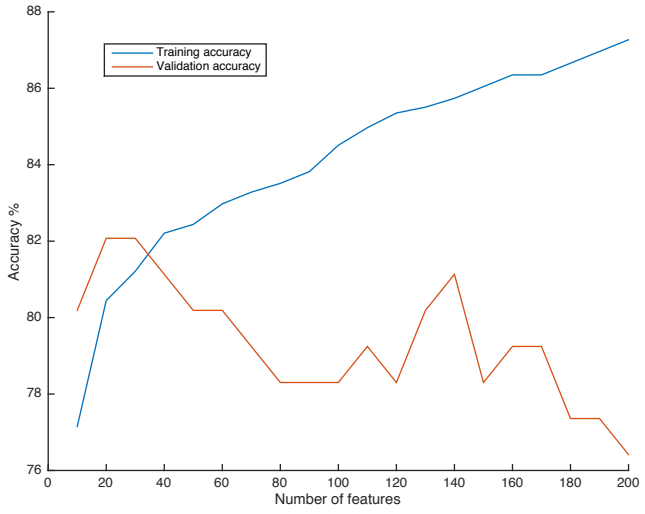

Figure 7: Training accuracy and validation accuracy versus number of features. As number of features increases, training accuracy also increases, but prediction accuracy decreases. Using all the features, the training accuracy is 92% and the prediction accuracy is 72%. This result is somehow surprising that using less features, we can achieve better prediction quality.

3. The following code is an implementation of SVM with ℓ_1-norm penalty.

```
% load data
load NewsData

% seperate data into training set and validation set
nTrain = length(idxTrain);
nTest = length(idxTest);
n = size(features,2);

XTrain = features(idxTrain,:);
XTest = features(idxTest,:);
yTrain = labels(idxTrain);
yTest = labels(idxTest);

ATrain = [];
ATest = [];
nnzs = [];

% sweep parameter C, solve for w
Cs = logspace(-1, 1, 10);
for C=Cs
    cvx_begin
        variables w(n, 1) b(1)
        minimize (norm(w,1) + C*sum(max(0, 1-yTrain.*(XTrain*w + b))))
    cvx_end
    % Validation
    ATrain = [ATrain, length(find(yTrain.*(XTrain*w + b) >0))/nTrain];
    ATest = [ATest, length(find(yTest.*(XTest*w + b) >0))/nTest];
    nnzs = [nnzs, length(find(w>1e-6))];
end

% plot
figure; hold on;
plot(Cs, ATrain*100);
plot(Cs, ATest*100);
xlabel('Parameter C'); ylabel('Accuracy %');
legend('Training accuracy', 'Validation accuracy');

figure, stem(Cs, nnzs);
xlabel('Parameter C'); ylabel('Number of non-zero elements in w');
```

The accuracy versus parameter C curve and number of non-zero elements versus C curve are plotted in Fig. 8.

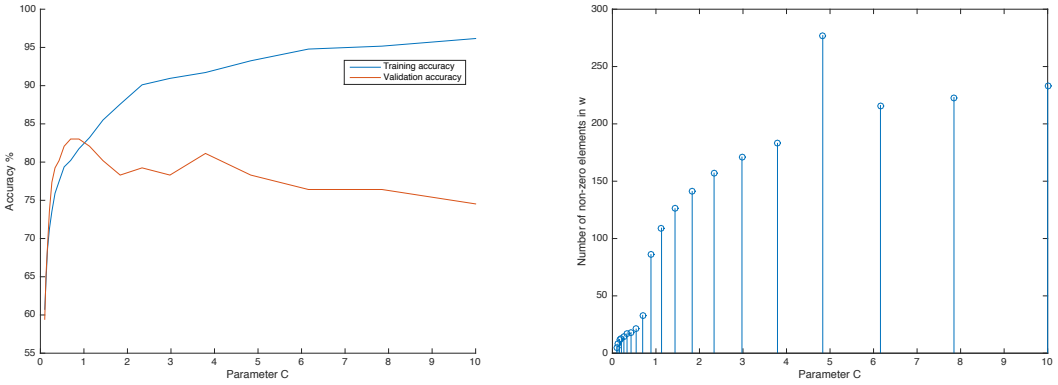

Figure 8: As parameter C increases, training accuracy rises monotonically, but prediction accuracy first increases and then drops. When C is small, the ℓ_1-norm penalty is relatively large, so we obtain a sparse w with a few non-zero elements. We also notice that the number of non-zero elements grows as C increases, but not monotonically