M146 HW3

Zheyi Wang

November 27 2018

1 VC

(a)If $b^2 - 4ac > 0$ and $a > 0$,
when x increase from -$\infty to \infty, h(x)$ change from +1 to -1 to +1
(b)If $b^2 - 4ac > 0$ and $a < 0$,
when x increase from -$\infty to \infty, h(x)$ change from -1 to +1 to -1
(c)If $b^2 - 4ac < 0$ and $a > 0$,
when x increase from -$\infty to \infty, h(x) = 1$
(d)If $b^2 - 4ac > 0$ and $a < 0$,
when x increase from -$\infty to \infty, h(x) = -1$
If we sort the data with increasing x, make in to a list$L = [h(x_1), h(x_2), ..., h(x_n)]$.
If VC = 1, $L = [1]$ and $L = [-1]$ can be shattered by (a), (b), (c) or (d).
If VC = 2, $L = [1, -1]$,$L = [-1, 1]$,$L = [-1, -1]$ and $L = [1, 1]$ can be shattered by (a), (b).
If VC = 3, similarly, all possibilities can be shattered by (a) or (b).
If VC = 4, counterexample : $L = [1, -1, 1, -1]$ cannot be shattered.
Therefore, VC = 3.

2 Kernel

$(1 + \beta x^T z)^3 = 1 + 3\beta x^T z + 3\beta^2 (x^T z)^2 + \beta^3 (x^T z)^3$
$= 1 + 3\beta \sum_1^D x_i z_i + 3\beta^2 \sum_1^D x_i z_i x_j z_j + \beta^3 \sum_1^D x_i z_i x_j z_j x_k z_k$
$\phi_\beta(x) = (1, \sqrt{3\beta}x_1, ..., \sqrt{3\beta}x_D, \sqrt{3\beta^2}x_1^2, ..., \sqrt{3\beta^2}x_1 x_D, \sqrt{3\beta^2}x_2 x_1 ..., \sqrt{3\beta^2}x_D x_1, ... \sqrt{3\beta^2}x_D^2,$
$\sqrt{\beta^3}x_1^3, ..., \sqrt{\beta^3}x_1^2 x_D, \sqrt{\beta^3}x_1 x_2 x_1, ..., \sqrt{\beta^3}x_1 x_D^2, \sqrt{\beta^3}x_2 x_1^2, ... \sqrt{\beta^3}x_D^3)$
For $D = 2$, $\phi_\beta(x) = (1, \sqrt{3\beta}x_1, \sqrt{3\beta}x_2, \sqrt{3\beta^2}x_1^2, \sqrt{3\beta^2}x_1 x_2, \sqrt{3\beta^2}x_2 x_1, \sqrt{3\beta^2}x_2^2,$
$\sqrt{\beta^3}x_1^3, \sqrt{\beta^3}x_1^2 x_2, \sqrt{\beta^3}x_1 x_2 x_1, \sqrt{\beta^3}x_1 x_2^2, \sqrt{\beta^3}x_2 x_1^2, \sqrt{\beta^3}x_2 x_1 x_2, \sqrt{\beta^3}x_2^2 x_1, \sqrt{\beta^3}x_2^2)$
Similarity: $K_{\beta=1}(x, z) = K(x, z)$
Difference: $K(x, z)$ is a subset of $K_\beta(x, z)$
Introducing β can let us pick up a best fit β from a set of β value, it can help the model fitting the training data better.

3 SVM

(a)
Assume $\omega = (i, j)$
$\omega^T x_1 + \omega^T x_2 = i + j + i = 0$, therefore $j = -2i$
$\omega^T x_1 = i + j = -i = 1$, therefore $\omega = (-1, 2)$
(b)
With offset, max margin will make $\frac{\omega}{|\omega|} = \frac{x_2 - x_1}{|x_2 - x_1|}$
Therefore, $\frac{\omega}{|\omega|} = (0, 1)$,
Since $\omega^T x_1 + b + \omega^T x_2 + b = 0$ and $\omega^T x_1 + b = 1$
$\omega = (0, 2), b = -1$
$(\omega^*, b^*) = (0, 2, -1)$ Comparing the result with the first part, we can see that the ω changed but the ω_2 didn't change.

4 Twitter Analysis

1.
(a)

```
word_list = {}
with open(infile, 'rU') as fid :
    ### ========== TODO : START ========== ###
    # part 1a: process each line to populate word_list
    index = 0
    for line in fid:
        for word in extract_words(line):
            if word not in word_list:
                word_list[word] = index
                index = index + 1
    ### ========== TODO : END ========== ###

return word_list
```

(b)

```
with open(infile, 'rU') as fid :
    ### ========== TODO : START ========== ###
    # part 1b: process each line to populate feature_matrix
    line_num = 0
    for line in fid:
        for word in extract_words(line):
            if word in word_list:
                feature_matrix[line_num, word_list[word]] = 1
        line_num = line_num + 1
    ### ========== TODO : END ========== ###

return feature_matrix
```

(c)

```
### ========== TODO : START ========== ###
# part 1: split data into training (training + cross-validation) and testing set
X_train = X[0:560,]
y_train = y[0:560]
X_test = X[560:630,]
y_test = y[560:630]
print(X_train.shape, y_train.shape, X_test.shape, y_test.shape)
```

(d)

```
(560, 1811) (560,) (70, 1811) (70,)
```

The output shows that the shape of X train is 560 * 1811, the shape of X test is 70*1811, the shape of y train is 560 * 1, the shape of y test is 70 * 1. Therefore, it does split the training and test data.

2.

(a)

```
### ========== TODO : START ========== ###
# part 2a: compute classifier performance
if metric == 'accuracy':
    return metrics.accuracy_score(y_true, y_label)
elif metric == 'f1_score':
    return metrics.f1_score(y_true, y_label)
elif metric == 'auroc':
    return metrics.roc_auc_score(y_true, y_label)
return 0
### ========== TODO : END ========== ###
```

(b)

```
### ========== TODO : START ========== ###
# part 2b: compute average cross-validation performance
counter = 0
sum = 0
for train_index,  test_index in kf.split(X,y):
    X_train, X_test = X[train_index], X[test_index]
    y_train, y_test = y[train_index], y[test_index]
    clf.fit(X_train,y_train)
    sum = sum + performance(y_test, clf.decision_function(X_test), metric)
    counter = counter + 1
return (sum/counter)
### ========== TODO : END ========== ###


# part 2: create stratified folds (5-fold CV)
s5f = StratifiedKFold(n_splits=5)
```

If we don't keep the proportion of positive and negative class, it may create some folds with every few positive data or negative data. In this case, model trained by these folds will have a great error rate. Therefore, it is better to keep the positive and negative classes roughly even.

(c)

```
### ========== TODO : START ========== ###
# part 2: select optimal hyperparameter using cross-validation
performance_list = []
for c in C_range:
    performance_list.append(cv_performance(SVC(kernel='linear', C=c),X,y,kf,metric))

return performance_list
### ========== TODO : END ========== ###
```

(d)

```python
# part 2: for each metric, select optimal hyperparameter for linear-kernel SVM using CV
for metric in metric_list:
    print(select_param_linear(X_train, y_train, s5f, metric))
```

```
Linear SVM Hyperparameter Selection based on accuracy:
[0.7089419539640778, 0.7107437557658796, 0.8060326761654195, 0.8146271113085273, 0.8181827370986664, 0.8181827370986664]
Linear SVM Hyperparameter Selection based on f1_score:
[0.8296828227419593, 0.8305628004640422, 0.875472682955829, 0.8748648327495685, 0.876562152886752, 0.876562152886752]
Linear SVM Hyperparameter Selection based on auroc:
[0.5, 0.503125, 0.7187871595703873, 0.753111334867664, 0.75917194092827, 0.75917194092827]
```

C	accuracy	F1-score	AUROC
10^{-3}	0.7089	0.8297	0.5
10^{-2}	0.7107	0.8306	0.5031
10^{-1}	0.8060	0.8755	0.7188
10^{0}	0.8146	0.8749	0.7531
10^{1}	0.8182	0.8766	0.7592
10^{2}	0.8182	0.8766	0.7592
Best C=100	0.8182	0.8766	0.7592

Both c=10 and c=100 are the best C which have the largest accuracy, f1-score and auroc. In almost all cases, accuracy, f1-score and auroc increases when C increases. The increases rate becomes slower when C increases.

3. I choose to use C=100 as the best C for the following steps

(a)

```python
# part 3: train linear-kernel SVMs with selected hyperparameters
clf = SVC(kernel='linear', C=100)
clf.fit(X_train, y_train)
```

(b)

```python
score = performance(y,clf.decision_function(X),metric)
return score
```

(c)

```python
# part 3: report performance on test data
for metric in metric_list:
    print ('Linear SVM Hyperparameter Selection based on ' + str(metric) + ':')
    print(performance_test(clf,X_test, y_test, metric))
```

```
Linear SVM Hyperparameter Selection based on accuracy:
0.7428571428571429
Linear SVM Hyperparameter Selection based on f1_score:
0.43749999999999994
Linear SVM Hyperparameter Selection based on auroc:
0.6258503401360545
```

C	accuracy	F1-score	AUROC
10^2	0.7429	0.4375	0.6259

twitter.py

```
"""
Author      : Yi-Chieh Wu, Sriram Sankararman
Description : Twitter
"""

from string import punctuation

import numpy as np

# !!! MAKE SURE TO USE SVC.decision_function(X), NOT SVC.predict(X) !!!
# (this makes ``continuous-valued'' predictions)
from sklearn.svm import SVC
from sklearn.model_selection import StratifiedKFold
from sklearn import metrics

######################################################################
# functions -- input/output
######################################################################

def read_vector_file(fname):
    """
    Reads and returns a vector from a file.

    Parameters
    --------------------
        fname  -- string, filename

    Returns
    --------------------
        labels -- numpy array of shape (n,)
                    n is the number of non-blank lines in the text file
    """
    return np.genfromtxt(fname)
```

```
######################################################################
# functions -- feature extraction
######################################################################

def extract_words(input_string):
    """
    Processes the input_string, separating it into "words" based on the presence
    of spaces, and separating punctuation marks into their own words.

    Parameters
    --------------------
        input_string -- string of characters

    Returns
    --------------------
        words        -- list of lowercase "words"
    """

    for c in punctuation :
        input_string = input_string.replace(c, ' ' + c + ' ')
    return input_string.lower().split()


def extract_dictionary(infile):
    """
    Given a filename, reads the text file and builds a dictionary of unique
    words/punctuations.

    Parameters
    --------------------
        infile    -- string, filename

    Returns
    --------------------
        word_list -- dictionary, (key, value) pairs are (word, index)
    """

    word_list = {}
    with open(infile, 'rU') as fid :
        ### ========== TODO : START ========== ###
        # part 1a: process each line to populate word_list
        index = 0
        for line in fid:
            for word in extract_words(line):
                if word not in word_list:
```

```
                word_list[word] = index
                index = index + 1
        ### ========== TODO : END ========== ###

    return word_list


def extract_feature_vectors(infile, word_list):
    """
    Produces a bag-of-words representation of a text file specified by the
    filename infile based on the dictionary word_list.

    Parameters
    --------------------
        infile         -- string, filename
        word_list      -- dictionary, (key, value) pairs are (word, index)

    Returns
    --------------------
        feature_matrix -- numpy array of shape (n,d)
                          boolean (0,1) array indicating word presence in a string
                            n is the number of non-blank lines in the text file
                            d is the number of unique words in the text file
    """

    num_lines = sum(1 for line in open(infile,'rU'))
    num_words = len(word_list)
    feature_matrix = np.zeros((num_lines, num_words))

    with open(infile, 'rU') as fid :
        ### ========== TODO : START ========== ###
        # part 1b: process each line to populate feature_matrix
        line_num = 0
        for line in fid:
            for word in extract_words(line):
                if word in word_list:
                    feature_matrix[line_num, word_list[word]] = 1
            line_num = line_num + 1
        ### ========== TODO : END ========== ###

    return feature_matrix


######################################################################
# functions -- evaluation
######################################################################
```

```
def performance(y_true, y_pred, metric="accuracy"):
    """
    Calculates the performance metric based on the agreement between the
    true labels and the predicted labels.

    Parameters
    --------------------
        y_true -- numpy array of shape (n,), known labels
        y_pred -- numpy array of shape (n,), (continuous-valued) predictions
        metric -- string, option used to select the performance measure
                  options: 'accuracy', 'f1-score', 'auroc'

    Returns
    --------------------
        score  -- float, performance score
    """
    # map continuous-valued predictions to binary labels
    y_label = np.sign(y_pred)
    y_label[y_label==0] = 1

    ### ========== TODO : START ========== ###
    # part 2a: compute classifier performance
    if metric == 'accuracy':
        return metrics.accuracy_score(y_true, y_label)
    elif metric == 'f1_score':
        return metrics.f1_score(y_true, y_label)
    elif metric == 'auroc':
        return metrics.roc_auc_score(y_true, y_label)
    return 0
    ### ========== TODO : END ========== ###


def cv_performance(clf, X, y, kf, metric="accuracy"):
    """
    Splits the data, X and y, into k-folds and runs k-fold cross-validation.
    Trains classifier on k-1 folds and tests on the remaining fold.
    Calculates the k-fold cross-validation performance metric for classifier
    by averaging the performance across folds.

    Parameters
    --------------------
        clf    -- classifier (instance of SVC)
        X      -- numpy array of shape (n,d), feature vectors
                    n = number of examples
                    d = number of features
```

```
        y      -- numpy array of shape (n,), binary labels {1,-1}
        kf     -- cross_validation.KFold or cross_validation.StratifiedKFold
        metric -- string, option used to select performance measure

    Returns
    --------------------
        score   -- float, average cross-validation performance across k folds
    """

    ### ========== TODO : START ========== ###
    # part 2b: compute average cross-validation performance
    counter = 0
    sum = 0
    for train_index,  test_index in kf.split(X,y):
        X_train, X_test = X[train_index], X[test_index]
        y_train, y_test = y[train_index], y[test_index]
        clf.fit(X_train,y_train)
        sum = sum + performance(y_test, clf.decision_function(X_test), metric)
        counter = counter + 1
    return (sum/counter)
    ### ========== TODO : END ========== ###


def select_param_linear(X, y, kf, metric="accuracy"):
    """
    Sweeps different settings for the hyperparameter of a linear-kernel SVM,
    calculating the k-fold CV performance for each setting, then selecting the
    hyperparameter that 'maximize' the average k-fold CV performance.

    Parameters
    --------------------
        X      -- numpy array of shape (n,d), feature vectors
                    n = number of examples
                    d = number of features
        y      -- numpy array of shape (n,), binary labels {1,-1}
        kf     -- cross_validation.KFold or cross_validation.StratifiedKFold
        metric -- string, option used to select performance measure

    Returns
    --------------------
        C -- float, optimal parameter value for linear-kernel SVM
    """

    print ('Linear SVM Hyperparameter Selection based on ' + str(metric) + ':')
    C_range = 10.0 ** np.arange(-3, 3)
```

```
    ### ========== TODO : START ========== ###
    # part 2: select optimal hyperparameter using cross-validation
    performance_list = []
    for c in C_range:
        performance_list.append(cv_performance(SVC(kernel='linear', C=c),X,y,kf,metric))

    return performance_list
    ### ========== TODO : END ========== ###



def performance_test(clf, X, y, metric="accuracy"):
    """
    Estimates the performance of the classifier using the 95% CI.

    Parameters
    --------------------
        clf          -- classifier (instance of SVC)
                          [already fit to data]
        X            -- numpy array of shape (n,d), feature vectors of test set
                          n = number of examples
                          d = number of features
        y            -- numpy array of shape (n,), binary labels {1,-1} of test set
        metric       -- string, option used to select performance measure

    Returns
    --------------------
        score        -- float, classifier performance
    """

    ### ========== TODO : START ========== ###
    # part 3: return performance on test data by first computing predictions and then callin

    score = performance(y,clf.decision_function(X),metric)
    return score
    ### ========== TODO : END ========== ###


######################################################################
# main
######################################################################

def main() :
    np.random.seed(1234)

    # read the tweets and its labels
```

```
    dictionary = extract_dictionary('../data/tweets.txt')
    X = extract_feature_vectors('../data/tweets.txt', dictionary)
    y = read_vector_file('../data/labels.txt')

    metric_list = ["accuracy", "f1_score", "auroc"]

    ### ========== TODO : START ========== ###
    # part 1: split data into training (training + cross-validation) and testing set
    X_train = X[0:560,]
    y_train = y[0:560]
    X_test = X[560:630,]
    y_test = y[560:630]
    print(X_train.shape, y_train.shape, X_test.shape, y_test.shape)

    # part 2: create stratified folds (5-fold CV)
    s5f = StratifiedKFold(n_splits=5)

    # part 2: for each metric, select optimal hyperparameter for linear-kernel SVM using CV
    for metric in metric_list:
        print(select_param_linear(X_train, y_train, s5f, metric))

    # part 3: train linear-kernel SVMs with selected hyperparameters
    clf = SVC(kernel='linear', C=100)
    clf.fit(X_train, y_train)
    # part 3: report performance on test data
    for metric in metric_list:
        print ('Linear SVM Hyperparameter Selection based on ' + str(metric) + ':')
        print(performance_test(clf,X_test, y_test, metric))
    ### ========== TODO : END ========== ###


if __name__ == "__main__" :
    main()
```