CSM146 Homework 3

Zipeng Fu

February 28, 2018

1 VC Dimension

H makes prediction in one dimension feature space. Let f be the correct concept. Let $h(x) = ax^2 + bx + c$.

Randomly pick 3 distinct points in R, denoted as x_1, x_2 and x_3, with the relation $x_1 < x_2 < x_3$. There $2^3 = 8$ possible labelings on these 3 points. Let's denote x_a, x_b, x_c and x_d as two points with the condition that $x_a < x_1 < x_b < x_2 < x_c < x_3 < x_d$.

Case 0 $f(x_1) = f(x_2) = f(x_3) = 0$. $a < 0$. h does not intersect with x-axis.

Case 1 $f(x_1) = 1, f(x_2) = f(x_3) = 0$. $a > 0$. h passes through x_b and x_d.

Case 2 $f(x_2) = 1, f(x_1) = f(x_3) = 0$. $a < 0$. h passes through x_b and x_c.

Case 3 $f(x_3) = 1, f(x_1) = f(x_2) = 0$. $a > 0$. h passes through x_a and x_c.

Case 4 $f(x_1) = 0, f(x_2) = f(x_3) = 1$. $a < 0$. h passes through x_b and x_d.

Case 5 $f(x_2) = 0, f(x_1) = f(x_3) = 1$. $a > 0$. h passes through x_b and x_c.

Case 6 $f(x_3) = 0, f(x_1) = f(x_2) = 1$. $a < 0$. h passes through x_a and x_c.

Case 7 $f(x_3) = f(x_1) = f(x_2) = 1$. $a > 0$. h does not intersect with x-axis.

For randomly picked 4 points, which satisfy the relation, $x_1 \leq x_2 \leq x_3 \leq x_4$. Any $h \in H$ will not correctly predict the labeling with $f(x_1) = f(x_3) = 1$ and $f(x_2) = f(x_4) = 0$.

Hence, $3 \leq \mathrm{VC}(H) < 4$, which means $\mathrm{VC}(H) = 3$.

2 Kernels

$$K_\beta(x,z) = \beta^3(x^Tz)^3 + 3\beta^2(x^Tz)^2 + 3\beta(x^Tz) + 1$$
$$= \beta^3(x_1z_1 + ... + x_nz_n)^3 + 3\beta^2\left(\sum_i x_i^2z_i^2 + 2\sum_i\sum_{j>i} x_iz_ix_jz_j\right) + 3\beta\sum_i x_1z_i + 1$$

$$(x_1z_1 + ... + x_nz_n)^3 = \left(\sum_i x_i^3z_i^3 + 6\sum_i\sum_{j>i}\sum_{k>j>i} x_ix_jx_kz_iz_jz_k + 3\sum_i\sum_{j\neq i} x_i^2x_jz_i^2z_j\right)$$

$$\phi_\beta(x) = (1, (\beta^{\frac{1}{2}}x_1)^3, (\beta^{\frac{1}{2}}x_2)^3, ..., (\beta^{\frac{1}{2}}x_n)^3, \sqrt{6}\beta^{\frac{3}{2}}x_1x_2x_3, \sqrt{6}\beta^{\frac{3}{2}}x_1x_2x_4, ..., \sqrt{6}\beta^{\frac{3}{2}}x_{n-2}x_{n-1}x_n,$$
$$\sqrt{3}\beta^{\frac{3}{2}}x_1^2x_2, \sqrt{3}\beta^{\frac{3}{2}}x_1^2x_3, ..., \sqrt{3}\beta^{\frac{3}{2}}x_n^2x_{n-1}, \sqrt{3}\beta x_1^2, \sqrt{3}\beta x_2^2, ..., \sqrt{3}\beta x_n^2, \sqrt{6}\beta x_1x_2, \sqrt{6}\beta x_1x_3, ..., \sqrt{6}\beta x_{n-1}x_n,$$
$$\sqrt{3\beta}x_1, \sqrt{3\beta}x_2, ..., \sqrt{3\beta}x_n)$$

When x and z are in 2D dimension,

$$\phi_\beta(x) = (1, \sqrt{3\beta}x_1, \sqrt{3\beta}x_2, \sqrt{3}\beta x_1^2, \sqrt{3}\beta x_2^2, \sqrt{6}\beta x_1x_2, \sqrt{3\beta^3}x_1^2x_2, \sqrt{3\beta^3}x_2^2x_1, \sqrt{\beta^3}x_1^3, \sqrt{\beta^3}x_2^3).$$

β acts like the coefficents for the entries of the feature map. When β is 1, the feature map $\phi_\beta(x)$ behaves as the kernal for $K(x,z) = \phi_\beta(x) * \phi_\beta(z)$. Thus, the coefficents provide more flexibility during training and maps the linear model to larger sets of non-linear models or high dimensional models.

3 SVM

(a) Since there are only 2 examples, they act like support vectors on the margins. The separation line should cross the origin and the middle point of x_1 and x_2, which is (1, 0.5). The gradient is 0.5. In order for $w^T x_1 = 1$ and $w^T x_2 = -1$, $w = (-1, 2)$.

(b) If offset is allowed, the separating line is horizontal so that it is the perpendicular bisector of the line connecting (1, 1) and (1, 0) to ensure the maximum margin with minimal $||w||$. Hence, $(w^*, b^*) = (w_1, w_2, b) = (0, 2, -1)$. We can see that the solutions with and without offset have the same w_2 but different w_1.

4 Twitter analysis using SVMs

4.1 (d) Dimensionalities for *X_train* is (560, 1811), for *y_train* is (560,), for *X_test* is (70, 1811) and for *y_test* is (70,).

4.2 (b) If the class proportions across folds are not maintained, extreme cases may happen, where some folds have nearly all positive labels and almost no negative labels. Such kind of folds are meaningless either for training or for validation due to lack of bipartite data to correctly form or test the separating hyperplane.

4.2 (d)

C	accuracy	F1-Score	AUROC
0.001	0.70894195	0.82968282	0.5000
0.01	0.71074376	0.8305628	0.503125
0.1	0.80603268	0.87547268	0.71878716
1	0.81462711	0.87486483	0.75311133
10	0.81818274	0.87656215	0.75917194
100	0.81818274	0.87656215	0.75917194

As C increases, the accracy rate is improving, which means the empirical loss on the validation fold keeps decreasing. However, the rate of increase in accuracy is decreasing. For $C = 100$, the best setting for C, all 3 metrics give the better performance.

4.3 (c) Pick the hyperparameter, C, 100.
Linear SVM Test based on accuracy: 0.742857142857.
Linear SVM Test based on F1-Score: 0.4375.
Linear SVM Test based on AUROC: 0.625850340136.

5 Appendix: 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, 'r', encoding="utf8") as fid :
        ### ========== TODO : START ========== ###
        # part 1a: process each line to populate word_list
        i = 0
        for line in fid:
            for word in extract_words(line):
                if word not in word_list:
                    word_list[word] = i
                    i += 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,'r', encoding="utf8"))
    num_words = len(word_list)
    feature_matrix = np.zeros((num_lines, num_words))

    with open(infile, 'r', encoding="utf8") as fid :
        ### ========== TODO : START ========== ###
        # part 1b: process each line to populate feature_matrix
        lCount = 0
        for line in fid:
            for word in extract_words(line):
                if word in word_list:
                    feature_matrix[lCount, word_list[word]] = 1
            lCount += 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
    score = 0
    if metric == "accuracy":
        score = metrics.accuracy_score(y_true, y_label)
    elif metric == "F1-Score":
        score = metrics.f1_score(y_true, y_label)
    elif metric == "AUROC":
        score = metrics.roc_auc_score(y_true, y_label)
    return score
    ### ========== 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
    sum = 0
    i = 0
    for train_index, valid_index in kf.split(X, y):
        X_train, X_valid = X[train_index], X[valid_index]
        y_train, y_valid = y[train_index], y[valid_index]
        clf.fit(X_train, y_train)
        sum += performance(y_valid, clf.decision_function(X_valid), metric)
```

```
        i += 1
    average = sum/i
    return average
    ### ========== 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
    arr = np.array([cv_performance(SVC(C=c, kernel='linear'), X, y, kf, metric)
                for c in C_range])
    return arr
    # argmin = np.argmin(arr)
    # return C_range[argmin]
    ### ========== 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 calling p
    print('Linear SVM Test based on ' + str(metric) + ':', end='')
    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('./code/data/tweets.txt')
    X = extract_feature_vectors('./code/data/tweets.txt', dictionary)
    y = read_vector_file('./code/data/labels.txt')

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

    ### ========== TODO : START ========== ###
    # part 1: split data into training (training + cross-validation) and testing set
    trainX = X[ :560, : ]
    trainy = y[:560]
    testX = X[560: , : ]
    testy = y[560: ]
    print(trainX.shape, trainy.shape, testX.shape, testy.shape)
    # part 2: create stratified folds (5-fold CV)
    skf = StratifiedKFold(n_splits=5, random_state=1234)

    # part 2: for each metric, select optimal hyperparameter for linear-kernel SVM using CV
    for metric in ['accuracy', 'F1-Score', 'AUROC']:
        lst = select_param_linear(trainX, trainy, skf, metric)
```

```
        print(lst)

    # part 3: train linear-kernel SVMs with selected hyperparameters
    clf = SVC(C=10**(2), kernel='linear')
    clf.fit(trainX, trainy)
    # part 3: report performance on test data


    ### ========== TODO : END ========== ###
    for metric in ['accuracy', 'F1-Score', 'AUROC']:
        print(performance_test(clf, testX, testy, metric))

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