CS M146 HW #3

Megan Pham 505 313 300

Problem 1 (Support Vector Machines)

$\min \frac{1}{2}\|w\|^2$ s.t. $y^{(i)} w^T x^{(i)} \geq 1, \quad i = 1, \dots, n$

a) $x = (1,1)^T \in \mathbb{R}^2$ label: $y = -1$

$y_n w^T x_n = 1$

$w^* = \frac{-x}{\|x\|^2}$

b) $x^{(1)} = (1,1)^T \in \mathbb{R}^2$ $y_1 = 1$

$x^{(2)} = (1,0)^T \in \mathbb{R}^2$ $y_2 = -1$

$y_1 w^T x_1 = 1$ $y_2 w^T x_2 = 1$

$(1) w^T (1,1) = 1$ $(-1) w^T (1,0) = 1$

$w^T(-1,0) = 1$

$w^* = [-1, 2]^T$

c) We now make the support vectors include the bias, such that

$y_1 w^T x_1 + b = 1$ $y_2 w^T x_2 + b = 1$

$w^T \binom{1}{1} + b = 1$ $w^T \binom{-1}{0} + b = 1$

$w^* = [0, 2]^T, \quad b^* = -1$

very similar to answer in part (b) but we need to include bias term when calculating w.

Problem 2 (Kernelized SVMs)

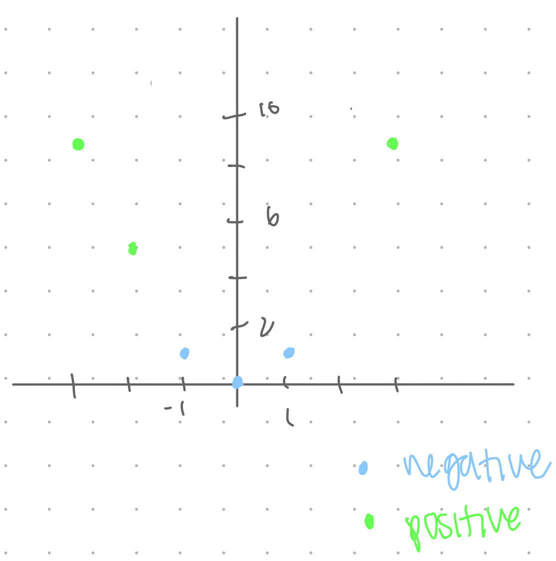
a)

	x	y
(1)	-1	-1
(2)	0	-1
(3)	1	-1
(4)	-3	+1
(5)	-2	+1
(6)	3	+1

$\phi(u) = (u, u^2)$

$K(u,v) = uv(1 + uv)$

look @ graph (hint)



$(x, x^2) = \phi(x) \rightarrow w_1 x + w_2 x^2 + w_0 = 0$

$-x + 3x^2 - 9 = 0$ decision boundary

hyperplane: $(-\frac{8}{5}, \frac{5}{2})$

$d = \sqrt{10}/2$

margin of hyperplane

$d = \sqrt{(-2-(-1))^2 + (4-1)^2}$

$= \sqrt{(1) + (3)^2}$

$= \sqrt{10}$

margin $= \frac{\sqrt{10}}{2}$

b) positive $(y = +1)$

4) $(-3, 9)$

5) $(-2, 4)$

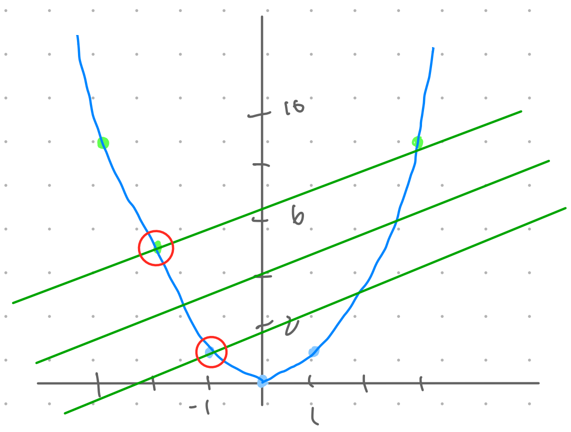
6) $(3, 9)$

negative $(y = -1)$

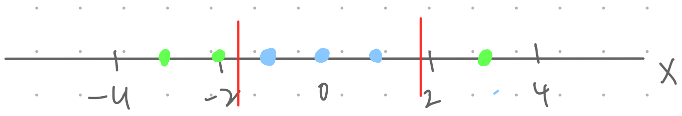
1) $(-1, 1)$

2) $(0, 0)$

3) $(1, 1)$



c)



d) $h(x) = \text{sign}\left(\sum_{i=1}^{\#SV} \alpha_i y^{(i)} k(x^{(i)}, x) + b\right)$ $\qquad \alpha_1 = \alpha_2 = \alpha_i$

support vectors

$h(x) = \alpha_1 + \alpha_2 - \frac{1}{2}\left(\alpha_1^2 (u_1 \cdot u_2) - 2\alpha_1\alpha_2 (u_1 \cdot u_2) + \alpha_2^2 (u_2 \cdot u_2)\right)$

$= 2\alpha_i - \frac{1}{2}\left(\alpha_i^2 (u) + 2\alpha_i^2 (u) + \alpha_i^2 (u)\right)$

$= 2\alpha_i - \frac{1}{2}\left(4\alpha^2 (u)\right)$

$= 2\alpha - 2\alpha^2$

$\frac{dy}{d\alpha} = 2 - 4\alpha$

$0 = 2 - 4\alpha$

$4\alpha = 2$

$\alpha = \frac{1}{2}$

For b:

$1 = \sum \alpha y u + b$

$1 = 2\alpha - 2\alpha^2 + b$

$1 = 2\left(\frac{1}{2}\right) - 2\left(\frac{1}{4}\right) + b$

$1 = 1 - \frac{1}{2} + b$

$b = 3/2$

e) No because the point is outside margin and should be negatively labelled

Problem 3 : Boosting

a)

i	x_1	x_2	Label
1	0	8	−
2	1	4	−
3	3	7	+
4	-2	1	−
5	-1	13	−
6	9	11	−
7	12	7	+
8	-7	-1	−
9	-3	12	+
10	5	9	+

vertical $\quad$ horizontal $\quad$ vertical $\quad$ horizontal $\quad$ sixth column

i	Label	Hypothesis 1 (1st iteration)				Hypothesis 2 (2nd iteration)			
		$\mathbf{w}_0$	$f_1 \equiv$ sign($x_1 - 2$)	$f_2 \equiv$ sign($x_2 - 4$)	$h_1 \equiv f_1$	$\mathbf{w}_1$	$f'_1 \equiv$ sign($x_1 - 10$)	$f'_2 \equiv$ sign($x_2 - 11$)	$h_2 \equiv f_2$
(1)	(2)	(3)	(4)	(5)	(6)	(7)	(8)	(9)	(10)
1	−	0.1	−	+	−	0.0625	−	−	−
2	−	0.1	−	−	−	0.0625	−	−	−
3	+	0.1	+	+	+	0.0625	−	−	−
4	−	0.1	−	−	−	0.0625	−	−	−
5	−	0.1	−	+	−	0.0625	−	+	+
6	−	0.1	+	+	+	0.25	−	−	−
7	+	0.1	+	+	+	0.0625	+	−	−
8	−	0.1	−	−	−	0.0625	−	−	−
9	+	0.1	−	+	−	0.25	−	+	+
10	+	0.1	+	+	+	0.0625	−	−	−

Table 1: Table for Boosting results

$\beta_t = \frac{1}{2} \ln\left(\frac{1 - e_t}{e_t}\right)$ $\qquad H(x) = \text{sign}\left(\sum_{t=1}^{T} \beta_t h_t(x)\right)$

$w_{t+1,i} = w_{t,i} \exp(-\beta_t y_i h_t(x_i))$

b) $f_2 \equiv \text{sign}(x_2 - 11)$

$\beta_0 = \frac{1}{2} \ln \frac{1 - \epsilon}{\epsilon}$ $\qquad \epsilon = \frac{2}{10}$

$= \frac{1}{2} \ln\left(\frac{1 - 2/10}{2/10}\right)$

$= \frac{1}{2} \ln\left(\frac{8/10}{2/10}\right)$

$\beta_0 = \frac{1}{2} \ln 4 = \ln 2$

c) $w_{t+1,i} = w_{t,i} \times \begin{cases} e^{-\beta_t} & \text{if } h_t(x_i) = y_i \\ e^{\beta_t} & \text{if } h_t(x_i) \neq y_i \end{cases}$

$= 0.1 \times \begin{cases} e^{-\beta_t} \rightarrow e^{-\ln 2} \longrightarrow \frac{1}{2} \\ e^{\beta_t} \rightarrow e^{\ln 2} \rightarrow 2 \end{cases}$

$= \begin{cases} 0.05 & \text{if they match} \\ 0.2 & \text{if they don't} \end{cases}$

normalize weights

$\frac{0.05}{0.8} = 0.0625$

$\frac{0.2}{0.8} = 0.25$

$\epsilon = \sum w_{t,i}$

$\beta_t = \frac{1}{2} \ln\left(\frac{1-\epsilon_t}{\epsilon_t}\right)$

d) $H(x) = \text{sign}\left(\sum_{t=1}^{T} \beta_t h_t(x)\right)$

$H(x) = \text{sign}\left(\ln 2 \cdot \text{sign}(x_1 - 2) + \ln(3)^{1/2} \, \text{sign}(x_2 - 11)\right)$

Problem 4: Twitter Analysis using SVM

1 Feature Extraction

a) code

b) code

c) code

2 Hyperparameter selection for a linear kernel SVM

a) code

b) We want each fold to be representative of all splits in the data, which is why it is best to maintain class proportions across folds.

code

c) code

d) code - for some reason, select_param_linears says that U mode is deprecated so could not compare values.

3 hyperparameter selection for an RBF kernel SVM

a) The gamma parameter controls the influnce of a single training point. Larger values of gamma means that more points need to be grouped together to be considered in the same group. A smaller value of gamma will generalize the data.

b) used a logarithmic grid in order to show the very small numbers of gamma and the larger numbers of C

c) code
values are very similar from each other

4 test set performance

a) code

b) code

c) code

values between the linear and RBF kernel are very similar. However, the RBF model did slightly better in accuracy, F1 score, auroc, precision, sensitivity, and specificity.

Problem 5 : Random Forest vs Decision Tree

a) code

b) code

c) code

```
import os
import sys

# To add your own Drive Run this cell.
from google.colab import drive
drive.mount('/content/drive')
```

```
Drive already mounted at /content/drive; to attempt to forcibly
remount, call drive.mount("/content/drive", force_remount=True).
```

```
# Please append your own directory after '/content/drive/My Drive/'
### ========== TODO : START ========== ###
sys.path += ['/content/drive/My Drive/CSM146-S22-HW3-code']
### ========== TODO : END ========== ###

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

from string import punctuation

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

Problem 4: Twitter Analysis Using SVM

```
######################################################################
# 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)


def write_label_answer(vec, outfile):
    """
    Writes your label vector to the given file.

    Parameters
    --------------------
        vec     -- numpy array of shape (n,) or (n,1), predicted
scores
        outfile -- string, output filename
    """

    # for this project, you should predict 70 labels
    if(vec.shape[0] != 70):
        print("Error - output vector should have 70 rows.")
        print("Aborting write.")
        return

    np.savetxt(outfile, vec)


######################################################################
# 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()
```

```python
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 = {}
    idx = 0
    with open(infile, 'r') as fid :
        ### ========== TODO : START ========== ###
        # part 1a: process each line to populate word_list
        count =0
        for line in fid:
            ext_words = extract_words(line)
            for word in ext_words:
                if word not in word_list:
                    word_list[word] = count
                    count +=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, 'r') as fid :
        ### ========== TODO : START ========== ###
        # part 1b: process each line to populate feature_matrix
        count =0
        for line in fid:
            ext_words = set(extract_words(line))
            for word in word_list:
                if word in word_list:
                    if word in ext_words:
                      feature_matrix[count, word_list[word]] = 1
            count+=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',
'precision',
                           'sensitivity', 'specificity'

    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)
    elif metric == "precision":
        return metrics.precision_score(y_true, y_label)
    elif metric == 'sensitivity':
        tn, fp, fn, tp = metrics.confusion_matrix(y_true, 
y_label).ravel()
        return tp/(tp + fp)
    elif metric == 'specificity':
        tn, fp, fn, tp = metrics.confusion_matrix(y_true, 
y_label).ravel()
        return tn/(tn + fp)
    ### ========== 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
    result = 0
    for i, j in kf.split(X, y):
      X_train, X_test =  X[i], X[j]
      y_train, y_test = y[i], y[j]
      clf.fit(X_train, y_train)
      y_pred = clf.decision_function(X_test)
      result += performance(y_test, y_pred, metric)
    return np.mean(np.array(result))
    ### ========== 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 2c: select optimal hyperparameter using cross-validation
    result = {}
    for c in C_range:
        sclf = SVC(kernel="linear", C=c)
        result[c] = cv_performance(sclf, X, y, kf, metric)
```

```
    optimal_c = -1
    perf = -1
    for i, j in result.items():
      if j > perf:
          optimal_c = i
          perf = j

    return optimal_c, result
    ### ========== TODO : END ========== ###


def select_param_rbf(X, y, kf, metric="accuracy"):
    """
    Sweeps different settings for the hyperparameters of an RBF-kernel
SVM,
    calculating the k-fold CV performance for each setting, then
selecting the
    hyperparameters 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
    --------------------
        gamma, C -- tuple of floats, optimal parameter values for an
RBF-kernel SVM
    """

    print('RBF SVM Hyperparameter Selection based on ' + str(metric) +
':')

    ### ========== TODO : START ========== ###
    # part 3b: create grid, then select optimal hyperparameters using
cross-validation
    gamma = 10.0 ** np.arange(-3, 3)
    C = 10.0 ** np.arange(-3, 3)
    result = np.zeros((C.shape[0], gamma.shape[0]))
    for i in np.arange(C.shape[0]):
      for j in np.arange(gamma.shape[0]):
          sclf = SVC(kernel="rbf", C = C[i], gamma = gamma[j])
          result[i, j] = cv_performance(sclf, X, y, kf, metric)
    best = np.unravel_index(np.argmax(result), result.shape)
```

```
    return C[best[0]], gamma[best[1]], result[best[0], best[1]]


    ### ========== 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
        lower, upper -- tuple of floats, confidence interval
    """


    ### ========== TODO : START ========== ###
    # part 4b: return the values of test results under a metric.
    y_pred = clf.decision_function(X)
    return performance(y, y_pred, metric)
    ### ========== TODO : END ========== ###

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

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

    # read the tweets and its labels, change the following two lines
to your own path.
    file_path = '/content/drive/My
Drive/CSM146-S22-HW3-code/data/tweets.txt'
    label_path = '/content/drive/My
Drive/CSM146-S22-HW3-code/data/labels.txt'
```

```python
    dictionary = extract_dictionary(file_path)
    print(len(dictionary))
    X = extract_feature_vectors(file_path, dictionary)
    y = read_vector_file(label_path)


    metric_list = ["accuracy", "f1_score", "auroc", "precision",
"sensitivity", "specificity"]

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

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

    # part 2d: for each metric, select optimal hyperparameter for
linear-kernel SVM using CV
    best_c = {}
    for metric in metric_list:
        best_c[metric] = select_param_linear(X, y, fiveFold, metric)
    for metric in best_c:
        print(metric)
        print("Best C:", best_c[metric][0])
    best_cv = best_c[metric][0]

    # part 3c: for each metric, select optimal hyperparameter for RBF-
SVM using CV
    best_rbf = {}
    for metric in metric_list:
        best_rbf[metric] = select_param_rbf(X, y, fiveFold, metric)
    print(best_rbf)
    for metric in best_rbf:
        best_cf, best_gamma, score = best_rbf[metric]
        print("Metric:", metric)
        print ("Best C for RBF:", best_cf)
        print ("Best Gamma:", best_gamma)
        print ("Score:", score)

    # part 4a: train linear- and RBF-kernel SVMs with selected
hyperparameters
    linear = SVC(C=best_cv, kernel = "linear")
    linear.fit(X_train, y_train)

    rbf = SVC(kernel = "rbf", C = best_cf, gamma = best_gamma)
    rbf.fit(X_train, y_train)
```

```
    # part 4c: test the performance of your two classifiers.
    for metric in metric_list:
        print("%s linear kernel: " % metric, performance_test(linear,
X_test, y_test, metric))
        print("%s rbf kernel: " % metric, performance_test(rbf,
X_test, y_test, metric))

    ### ========== TODO : END ========== ###

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

```
1811
Linear SVM Hyperparameter Selection based on accuracy:

/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:74:
DeprecationWarning: 'U' mode is deprecated

Linear SVM Hyperparameter Selection based on f1_score:
Linear SVM Hyperparameter Selection based on auroc:
Linear SVM Hyperparameter Selection based on precision:
Linear SVM Hyperparameter Selection based on sensitivity:
Linear SVM Hyperparameter Selection based on specificity:
accuracy
Best C: 1.0
f1_score
Best C: 1.0
auroc
Best C: 10.0
precision
Best C: 10.0
sensitivity
Best C: 10.0
specificity
Best C: 10.0
RBF SVM Hyperparameter Selection based on accuracy:
RBF SVM Hyperparameter Selection based on f1_score:
RBF SVM Hyperparameter Selection based on auroc:
RBF SVM Hyperparameter Selection based on precision:
RBF SVM Hyperparameter Selection based on sensitivity:
RBF SVM Hyperparameter Selection based on specificity:
{'accuracy': (100.0, 0.001, 3.9444444444444446), 'f1_score': (10.0,
0.001, 4.232889019342606), 'auroc': (100.0, 0.001,
3.7493662357069475), 'precision': (100.0, 0.001, 4.215620884859476),
'sensitivity': (100.0, 0.001, 4.215620884859476), 'specificity':
(100.0, 0.01, 3.1954595791805094)}
Metric: accuracy
Best C for RBF: 100.0
Best Gamma: 0.001
Score: 3.9444444444444446
Metric: f1_score
```

```
Best C for RBF: 10.0
Best Gamma: 0.001
Score: 4.232889019342606
Metric: auroc
Best C for RBF: 100.0
Best Gamma: 0.001
Score: 3.7493662357069475
Metric: precision
Best C for RBF: 100.0
Best Gamma: 0.001
Score: 4.215620884859476
Metric: sensitivity
Best C for RBF: 100.0
Best Gamma: 0.001
Score: 4.215620884859476
Metric: specificity
Best C for RBF: 100.0
Best Gamma: 0.01
Score: 3.1954595791805094
accuracy linear kernel:  0.7428571428571429
accuracy rbf kernel:  0.7571428571428571
f1_score linear kernel:  0.43749999999999994
f1_score rbf kernel:  0.45161290322580644
auroc linear kernel:  0.6258503401360545
auroc rbf kernel:  0.6360544217687075
precision linear kernel:  0.6363636363636364
precision rbf kernel:  0.7
sensitivity linear kernel:  0.6363636363636364
sensitivity rbf kernel:  0.7
specificity linear kernel:  0.9183673469387755
specificity rbf kernel:  0.9387755102040817
```

Problem 5: Boosting vs. Decision Tree

```
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn import metrics
from sklearn.model_selection import cross_val_score, train_test_split

class Data :

    def __init__(self) :
        """
        Data class.

        Attributes
        --------------------
            X -- numpy array of shape (n,d), features
```

```
            y -- numpy array of shape (n,), targets
        """

        # n = number of examples, d = dimensionality
        self.X = None
        self.y = None

        self.Xnames = None
        self.yname = None

    def load(self, filename, header=0, predict_col=-1) :
        """Load csv file into X array of features and y array of
labels."""

        # determine filename
        f = filename

        # load data
        with open(f, 'r') as fid :
            data = np.loadtxt(fid, delimiter=",", skiprows=header)

        # separate features and labels
        if predict_col is None :
            self.X = data[:,:]
            self.y = None
        else :
            if data.ndim > 1 :
                self.X = np.delete(data, predict_col, axis=1)
                self.y = data[:,predict_col]
            else :
                self.X = None
                self.y = data[:]

        # load feature and label names
        if header != 0:
            with open(f, 'r') as fid :
                header = fid.readline().rstrip().split(",")

            if predict_col is None :
                self.Xnames = header[:]
                self.yname = None
            else :
                if len(header) > 1 :
                    self.Xnames = np.delete(header, predict_col)
                    self.yname = header[predict_col]
                else :
                    self.Xnames = None
                    self.yname = header[0]
        else:
            self.Xnames = None
```

```
            self.yname = None


# helper functions
def load_data(filename, header=0, predict_col=-1) :
    """Load csv file into Data class."""
    data = Data()
    data.load(filename, header=header, predict_col=predict_col)
    return data

# Change the path to your own data directory
titanic = load_data("/content/drive/My
Drive/CSM146-S22-HW3-code/data/titanic_train.csv", header=1,
predict_col=0)
X = titanic.X; Xnames = titanic.Xnames
y = titanic.y; yname = titanic.yname
n,d = X.shape  # n = number of examples, d =  number of features

def error(clf, X, y, ntrials=100, test_size=0.2) :
    """
    Computes the classifier error over a random split of the data,
    averaged over ntrials runs.

    Parameters
    --------------------
        clf         -- classifier
        X           -- numpy array of shape (n,d), features values
        y           -- numpy array of shape (n,), target classes
        ntrials     -- integer, number of trials

    Returns
    --------------------
        train_error -- float, training error
        test_error  -- float, test error
    """

    train_error = 0
    test_error = 0

    train_scores = []; test_scores = [];
    for i in range(ntrials):
        xtrain, xtest, ytrain, ytest = train_test_split (X,y,
test_size = test_size, random_state = i)
        clf.fit (xtrain, ytrain)

        ypred = clf.predict (xtrain)
        err = 1 - metrics.accuracy_score (ytrain, ypred, normalize =
True)
        train_scores.append (err)
```

```python
        ypred = clf.predict (xtest)
        err = 1 - metrics.accuracy_score (ytest, ypred, normalize =
True)
        test_scores.append (err)

    train_error =  np.mean (train_scores)
    test_error = np.mean (test_scores)
    return train_error, test_error

### ========== TODO : START ========== ###
# Part 5(a): Implement the decision tree classifier and report the
training error.
print('Classifying using Decision Tree...')

X_train, X_test, y_train, y_test = train_test_split (X,y, test_size =
0.2)

dclf = DecisionTreeClassifier(criterion='entropy')
dclf.fit(X_train, y_train)
y_pred = dclf.predict(X_train)
train_err = 1 - metrics.accuracy_score(y_train, y_pred,
normalize=True)
print('training error: %.3f' % train_err)

### ========== TODO : END ========== ###
```

```
Classifying using Decision Tree...
training error: 0.011
```

```python
train_error, test_error = error (DecisionTreeClassifier (criterion =
'entropy'), X, y)
print('\tDecision Tree\t-- avg train error : %.3f\tavg test error :
%.3f' %(train_error, test_error))
```

```
     Decision Tree   -- avg train error : 0.012 avg test error :
0.241
```

```python
### ========== TODO : START ========== ###
# Part 5(b): Implement the random forest classifier and adjust the
number of samples used in bootstrap sampling.

best_train_err = 1000.0
best_test_err = 1000.0
best_err = 1000.0

best_train_sample = 0.0
best_test_sample = 0.0
best_sample = 0.0

for i in range(1, 9):
  sample = i/10 #max_samples must be from 0.0 to 1.0
```

```
  rclf = RandomForestClassifier(criterion='entropy',
max_samples=sample, bootstrap=True)

  train_err, test_err = error(rclf, X, y)

  if (train_err < best_train_err):
    best_train_err = train_err
    best_train_sample = sample

  if (test_err < best_test_err):
    best_test_err = test_err
    best_test_sample = sample

  err = np.abs(train_err - test_err)
  if (err < best_err):
    best_err = err
    best_sample = sample

print('best sample: ', best_sample)

### ========== TODO : END ========== ###
```

```
best sample:  0.1
```

```
### ========== TODO : START ========== ###
# Part 5(c): Implement the random forest classifier and adjust the
number of features for each decision tree.

best_train_err = 1000.0
best_test_err = 1000.0
best_err = 1000.0

best_feature=1;

for i in range(1, 8):
  rclf = RandomForestClassifier(criterion='entropy', max_features=i,
bootstrap=True)

  train_err, test_err = error(rclf, X, y)

  if (train_err < best_train_err):
    best_train_err = train_err
    best_train_sample = sample

  if (test_err < best_test_err):
    best_test_err = test_err
    best_test_sample = sample

  err = np.abs(train_err - test_err)
```

```
  if (err < best_err):
    best_err = err
    best_feature = i

print('best number of features: ', best_feature)
```

```
best number of features:  4
```