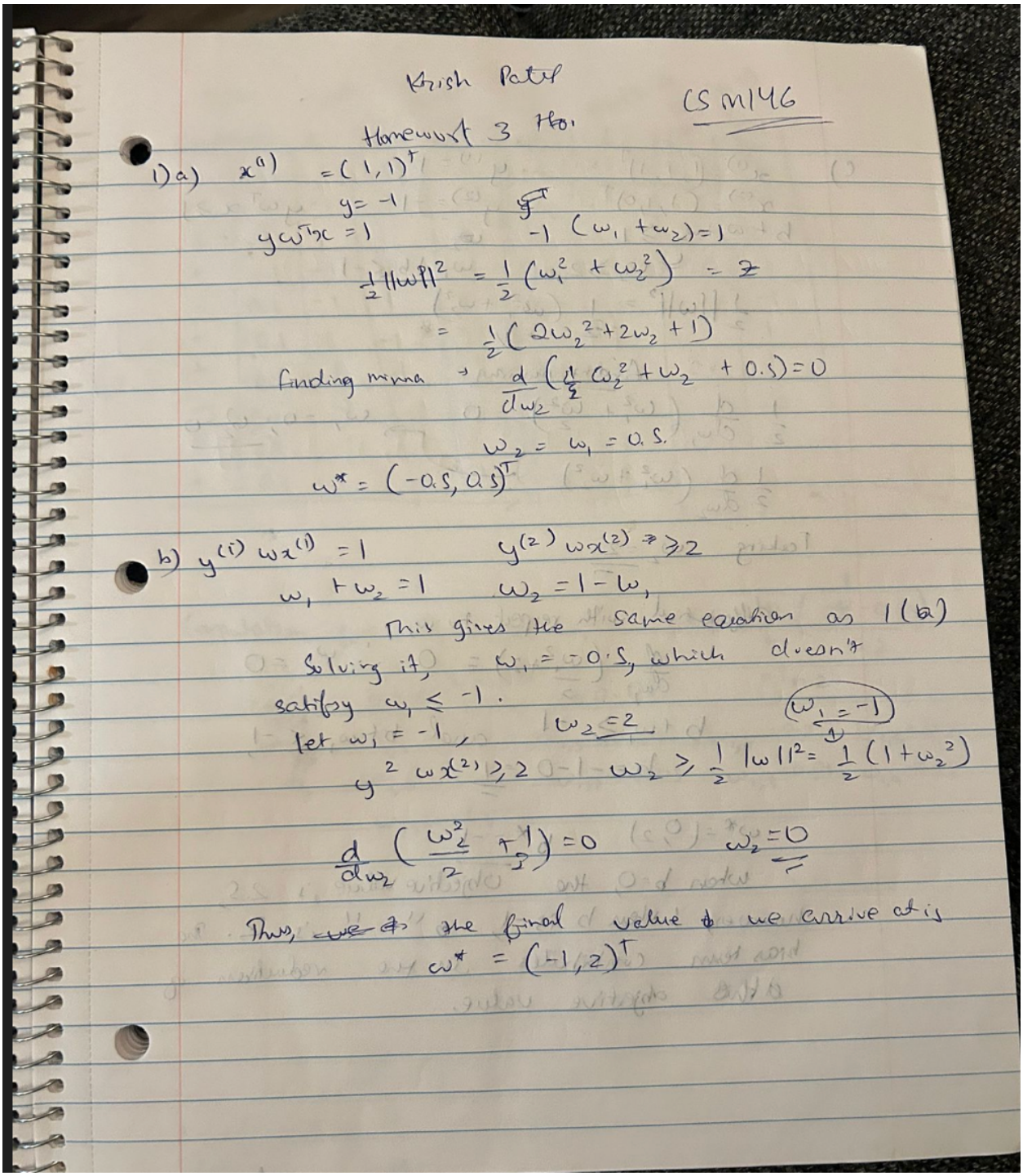
Homework 3:
CS M146:
Problem 1:



Krish Patel
CS M146
Homework 3 No.

1) a) $x^{(1)} = (1,1)^T$
$y = -1$
$y w^T x = 1$
$-1 (w_1 + w_2) = 1$
$\frac{1}{2}\|w\|^2 = \frac{1}{2}(w_1^2 + w_2^2)$
$= \frac{1}{2}(2w_2^2 + 2w_2 + 1)$
finding minima → $\frac{d}{dw_2}\left(\frac{1}{2}w_2^2 + w_2 + 0.5\right) = 0$
$w_2 = w_1 = 0.5$
$w^* = (-0.5, 0.5)^T$

b) $y^{(1)} w x^{(1)} = 1$
$w_1 + w_2 = 1$
$y^{(2)} w x^{(2)} \geq 2$
$w_2 = 1 - w_1$
This gives the same equation as 1(b)
Solving it, $w_1 = -0.5$, which doesn't satisfy $w_1 \leq -1$.
let $w_1 = -1$, $w_2 = 2$ ($w_1 = -1$)
$y^2 w x^{(2)} \geq 2$
$w_2 \geq \frac{1}{2}\|w\|^2 = \frac{1}{2}(1 + w_2^2)$
$\frac{d}{dw_2}\left(\frac{w_2^2}{2} + \frac{1}{2}\right) = 0$
$w_2 = 0$
Thus, the final value we arrive at is
$w^* = (-1, 2)^T$

c) $x^{(1)} = (1,1,1)^T$ $\quad y^{(1)} = 1$

$x^{(2)} = (1,1,0)^T$ $\quad y^{(2)} = -1$ $\quad y w^T x \geq 1$

$b + w_1 + w_2 \geq 1$ $\quad w_2$

$\therefore w_2 \geq 2,$ $\quad w_1 + b \leq -1$

$\frac{1}{2}\|w\|^2 = \frac{1}{2}(w_1^2 + w_2^2)$

finding minima

$\frac{1}{2}\frac{d}{dw_1}(w_1^2 + w_2^2) = 0 \rightarrow w_1 = 0, \; w_2 = 0$

$\frac{1}{2}\frac{d}{dw_2}(w_1^2 + w_2^2)$

Taking $w_2 = 2$

differentiating with respect to w_1,

$\frac{d}{dw_1}\left(\frac{w_1^2}{2} + 2\right) = 0 \quad \therefore w_1 = 0$

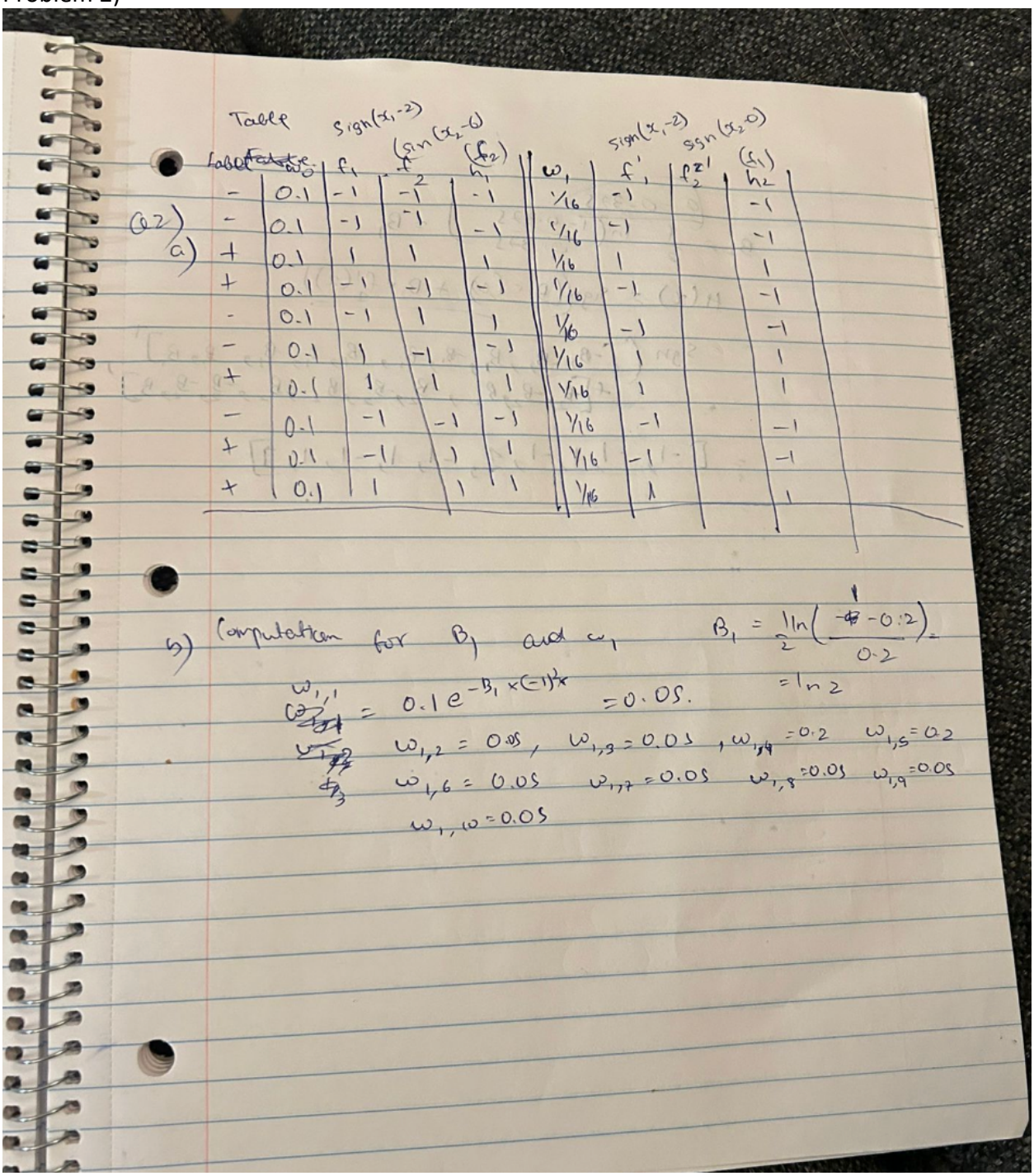
$b + w_1 \geq -1$ and $b + w_1 \leq -1$,

$b = -1 - 0 = -1$

$w^* = (0, 2) \quad b^* = -1$

when $b = 0$, the objective value is 2.5, however, when b is -1, " " is 2. The bias term contributes in the reduction of the objective value.

Problem 2)



Table

Label	w_0	f_1	$sign(x_1-2)$ f_1	$sign(x_2-6)$ f_2	h_1	w_1	$sign(x_1-2)$ f_1'	f_2'	$sgn(x_2-0)$ h_2
-	0.1	-1	-1	-1	-1	1/16	-1		-1
-	0.1	-1	-1	-1	-1	1/16	-1		-1
+	0.1	1	1	1	1	1/16	1		1
+	0.1	-1	-1	-1	-1	1/16	-1		-1
-	0.1	-1	1	1	1	1/16	-1		-1
-	0.1	1	-1	-1	-1	1/16	1		1
+	0.1	1	1	1	1	1/16	1		1
-	0.1	-1	-1	-1	-1	1/16	-1		-1
+	0.1	-1	1	1	1	1/16	-1		-1
+	0.1	1	1	1	1	1/16	1		1

(Q2)

a)

b) Computation for B_1 and ω_1

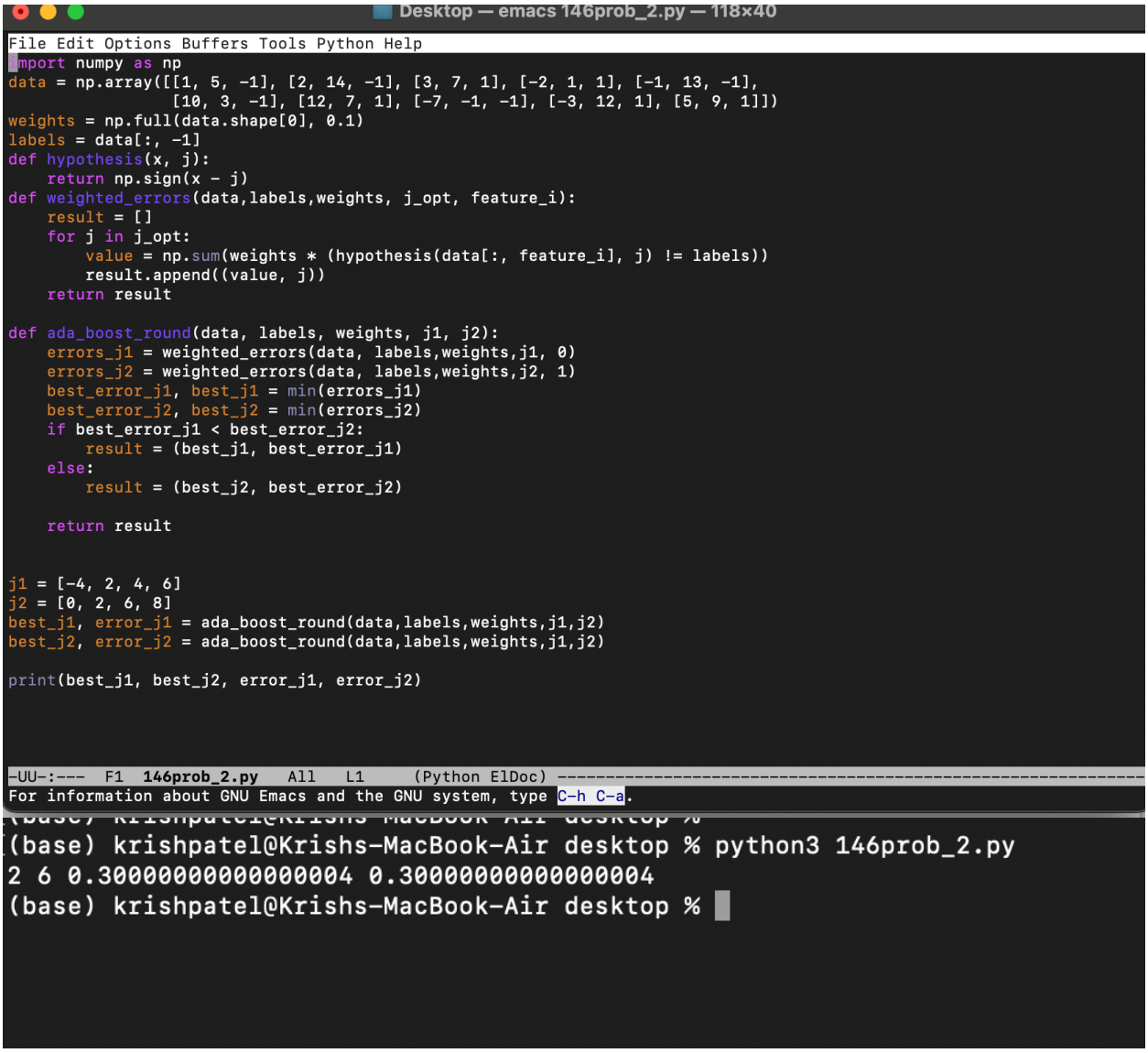
$$B_1 = \frac{1}{2}\ln\left(\frac{1-0.2}{0.2}\right) = \ln 2$$

$$\omega_{1,1} = 0.1 e^{-B_1 \times (-1)^1} = 0.05.$$

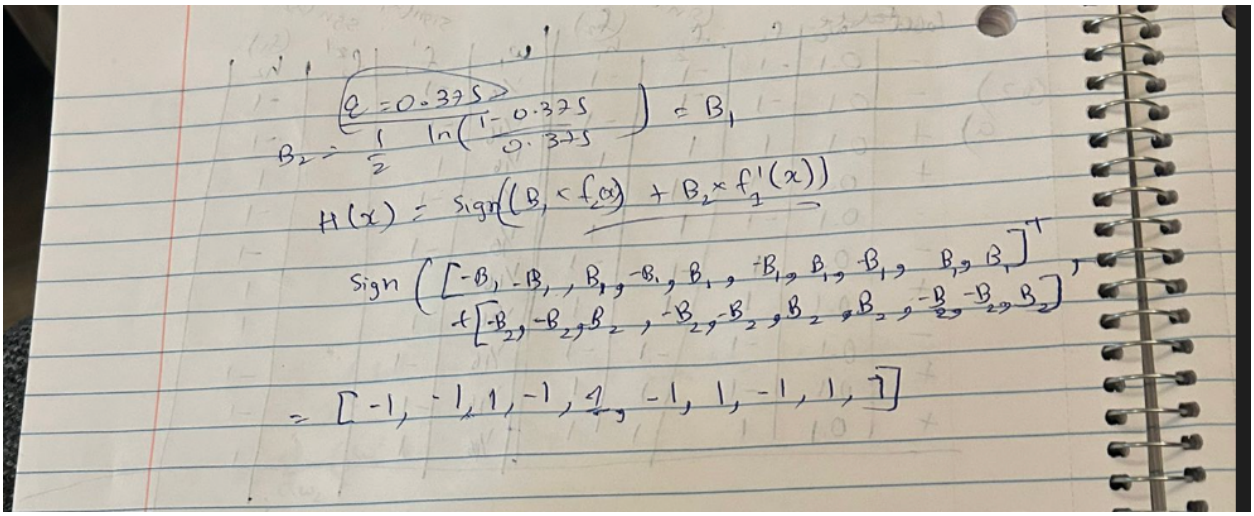
$\omega_{1,2} = 0.05$, $\omega_{1,3} = 0.05$, $\omega_{1,4} = 0.2$ $\omega_{1,5} = 0.2$

$\omega_{1,6} = 0.05$ $\omega_{1,7} = 0.05$ $\omega_{1,8} = 0.05$ $\omega_{1,9} = 0.05$

$\omega_{1,10} = 0.05$



Best j1 = 2
Best j2 = 6
J1 error ~= 0.3
J2 error ~= 0.3



Problem 3 code snippets:

```
##################################################################
from sklearn.model_selection import cross_val_score
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 1a: compute classifier performance

    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_pred)
    elif metric == "precision":
        score = metrics.precision_score(y_true, y_label)
    elif metric == "sensitivity":
        score = metrics.recall_score(y_true, y_label)
    elif metric == "specificity":
        tn, fp, fn, tp = metrics.confusion_matrix(y_true, y_pred).ravel()

        return tn / (tn + fp)
    else:
        raise ValueError("Invalid metric. Supported metrics: 'accuracy', 'f1-score', 'auroc', 'precision', 'sensitivity', 'specificity'.")
    return score
```

```python
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 LinearSVC)
        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     -- model_selection.StratifiedKFold
        metric -- string, option used to select performance measure

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

    ### ========== TODO : START ========== ###
    # part 1b: compute average cross-validation performance

    scores = []
    for train_i, test_i in kf.split(X, y):
        X_train,X_test = X[train_i], X[test_i]
        y_train,y_test = y[train_i], y[test_i]
        clf.fit(X_train, y_train)
        y_pred = clf.decision_function(X_test)
        score = performance(y_test, y_pred, metric = metric)
        scores.append(score)
    return np.mean(scores)


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

```python
def select_param_linear(X, y, kf, metric="accuracy"):
    """
    Sweeps different settings for the hyperparameter of a linear 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     -- model_selection.StratifiedKFold
        metric -- string, option used to select performance measure

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

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

    ### ========== TODO : START ========== ###
    # part 1c: select optimal hyperparameter using cross-validation
    best_C  = None
    best_score = -np.inf
    kf =  StratifiedKFold(n_splits=5, shuffle=True, random_state=0)
    for C in C_range:
        clf = LinearSVC(loss= 'hinge', random_state=0, C=C)
        score = cv_performance(clf, X, y, kf, metric)
        if score > best_score:
            best_score = score
            best_C = C
    return best_C
    ### ========== TODO : END ========== ###
```

```python
def performance_test(clf, X, y, metric="accuracy"):
    """
    Estimates the performance of the classifier.

    Parameters
    --------------------
        clf          -- classifier (instance of LinearSVC)
                        [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 ========== ###
    y_pred = clf.predict(X)
    return performance(y, y_pred, metric)

    # part 2b: return performance on test data under a metric.

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

✓ 0.0s

```python
import numpy as np
def main() :
    np.random.seed(1234)

    # read the tweets and its labels, change the following two lines to your own path.
    ### ========== TODO : START ========== ###
    file_path = '../data/tweets.txt'
    label_path = '../data/labels.txt'
    ### ========== TODO : END ========== ###
    dictionary = extract_dictionary(file_path)
    print(len(dictionary))
    X = extract_feature_vectors(file_path, dictionary)
    y = read_vector_file(label_path)
    # 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:]

    metric_list = ["accuracy", "f1-score", "auroc", "precision", "sensitivity", "specificity"]
    # part 1b: create stratified folds (5-fold CV)
    kf = StratifiedKFold(n_splits=5, shuffle=True, random_state=0)

    # part 1c: for each metric, select optimal hyperparameter for linear SVM using CV
    best_C_values = {}
    for metric in metric_list:
        best_C_values[metric] = select_param_linear(X_train, y_train, kf, metric)
    # part 2a: train Linear SVMs with selected hyperparameters
    LinearSVCs = {}
    for metric in metric_list:
        clf = LinearSVC(loss='hinge', random_state=0, C=best_C_values[metric])
        clf.fit(X_train, y_train)
        LinearSVCs[metric] = clf
    # part 2b: test the performance of your classifiers.
    for metric in metric_list:
        score = performance_test(LinearSVCs[metric], X_test, y_test, metric)
        print(f"Test performance based on {metric}: {score}")

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


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

✓ 3.9s

```
1811
Linear SVM Hyperparameter Selection based on accuracy:
/opt/homebrew/lib/python3.11/site-packages/sklearn/svm/_classes.py:32: FutureWarning: The defa
  warnings.warn(
Output is truncated. View as a scrollable element or open in a text editor. Adjust cell output settings...
Test performance based on accuracy: 0.7428571428571429
Test performance based on f1-score: 0.43749999999999994
Test performance based on auroc: 0.6258503401360545
Test performance based on precision: 0.6363636363636364
Test performance based on sensitivity: 1.0
Test performance based on specificity: 0.9183673469387755
/opt/homebrew/lib/python3.11/site-packages/sklearn/svm/_classes.py:32: FutureWarning: The defa
```

```
    # Change the path to your own data directory

    ### ========== TODO : START ========== ###
    titanic = load_data("../data/titanic_train.csv", header=1, predict_col=0)
    ### ========== TODO : END ========== ###
    X = titanic.X; Xnames = titanic.Xnames
    y = titanic.y; yname = titanic.yname
    n,d = X.shape  # n = number of examples, d =  number of features
```

[39] ✓ 0.0s

```
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
        test_size   -- proportion of data used for evaluation

    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)

        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
```

✓ 0.0s

```
    ### ========== TODO : START ========== ###

    from sklearn.tree import DecisionTreeClassifier
    from sklearn.metrics import accuracy_score
    # Part 4(a): Implement the decision tree classifier and report the training error.
    print('Classifying using Decision Tree...')
    decision_tree = DecisionTreeClassifier()
    decision_tree.fit(X, y)

    y_train_pred = decision_tree.predict(X)

    training_error = 1 - accuracy_score(y, y_train_pred)
    print(f"Training error: {training_error}")

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

[44] ✓ 0.0s

```
Classifying using Decision Tree...
Training error: 0.014044943820224698
```

```
    ### ========== TODO : START ========== ###
    # Part 4(b): Implement the random forest classifier and adjust the number of samples used in bootstrap sampling.
    # Part 4(c): Implement the random forest classifier and adjust the number of samples used in bootstrap sampling.
    print('Classifying using Random Forest...')
    max_samples_values = [0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8]
    best_max_samples = None
    best_train_error = float('inf')
    best_test_error = float('inf')
    for max_samples in max_samples_values:
        rf_clf = RandomForestClassifier(criterion='entropy', random_state=0, max_samples=max_samples)
        rf_clf.fit(X, y)
        train_error, test_error = error(rf_clf, X, y)
        if test_error < best_test_error:
            best_test_error = test_error
            best_train_error = train_error
            best_max_samples = max_samples
    print("Best max samples: " + str(best_max_samples))
    print("Corresponding training error: " + str(best_train_error))
    print("Corresponding test error: " + str(best_test_error))

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

[50] ✓ 1m 25.5s

```
Classifying using Random Forest...
Best max samples: 0.3
Corresponding training error: 0.09427065026362039
Corresponding test error: 0.1874825174825175
```

```python
### ========== TODO : START ========== ###
# Part 4(c): Implement the random forest classifier and adjust the number of features for each decision tree.
print('Classifying using Random Forest...')

max_features_val = [1, 2, 3, 4, 5, 6, 7]
best_max_features = None
best_train_error = float('inf')
best_test_error = float('inf')

for max_features in max_features_val:
    rf_clf = RandomForestClassifier(criterion='entropy', random_state=0, max_samples=best_max_samples, max_features=max_features)
    rf_clf.fit(X, y)
    train_err, test_err = error(rf_clf, X, y)

    if test_error < best_test_error:
        best_test_error = test_err
        best_train_error = train_err
        best_max_features = max_features

print("Best max features: " + str(best_max_features))
print("Corresponding training error: " + str(best_train_error))
print("Corresponding test error: " + str(best_test_error))
### ========== TODO : END ========== ###
```

51] ✓ 1m 5.4s

```
Classifying using Random Forest...
Best max features: 1
Corresponding training error: 0.09165202108963091
Corresponding test error: 0.1904195804195804
```