CS M146: Introduction to Machine Learning — UCLA Fall Quarter 2023
Prof. Aditya Grover — 30th October 2023

Homework #3

Due: 20th November 2023, Monday, before 11:59 pm

Problem 1 (Support Vector Machines [9 pts])

Suppose we are looking for a maximum-margin linear classifier *through the origin*, (i.e. bias $b = 0$) for the hard margin SVM formulation, (i.e., no slack variables). In other words,

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

(a) **(3 pts)** Given a single training vector $\mathbf{x} = (1, 1)^T \in \mathbb{R}^2$ with label $y = -1$, what is the $\mathbf{w}^*$ that satisfies the above constrained minimization?

Solution: The SVM with one negative data point orients $\mathbf{w}$ in the opposite direction of the single data point $\mathbf{x}$ in order to minimize the objective function while satisfying the constraint $y^{(1)}\mathbf{w}^T\mathbf{x}^{(1)} = 1$. The corresponding $\mathbf{w}$ is

$$\mathbf{w}^* = -\frac{\mathbf{x}}{||\mathbf{x}||^2} = (-\frac{1}{2}, -\frac{1}{2})^T.$$

(b) **(3 pts)** Suppose we have two training examples, $\mathbf{x}^{(1)} = (1, 1)^T \in \mathbb{R}^2$ and $\mathbf{x}^{(2)} = (1, 0)^T \in \mathbb{R}^2$ with labels $y^{(1)} = 1$ and $y^{(2)} = -1$. What is $\mathbf{w}^*$ in this case?

Solution: Observation: $\mathbf{w}^*$ must satisfy either $y^{(1)}\mathbf{w}^{*T}\mathbf{x}^{(1)} = 1$ or $y^{(2)}\mathbf{w}^{*T}\mathbf{x}^{(2)} = 1$ (at least one of the constraints is satisfied with equality).

Proof of observation: Assume, for contradiction, that the observation is false. Then, for some $c_1, c_2 > 1$, we have that $y^{(1)}\mathbf{w}^{*T}\mathbf{x}^{(1)} = c_1 > 1$ and $y^{(2)}\mathbf{w}^{*T}\mathbf{x}^{(2)} = c_2 > 1$ ($\mathbf{w}^*$ satisfies both constraints but satisfies neither constraint with equality). Then, we note that $\mathbf{w}^{'} = \frac{\mathbf{w}^*}{\min(c_1,c_2)}$ satisfies both constraints and at least one constraint with equality: $y^{(1)}\mathbf{w}^{'T}\mathbf{x}^{(1)} = \frac{c_1}{\min(c_1,c_2)}$ and $y^{(2)}\mathbf{w}^{'T}\mathbf{x}^{(2)} = \frac{c_2}{\min(c_1,c_2)}$ (at least one of these two equalities has right hand side equal to 1). However, $\frac{1}{2}||\mathbf{w}^{'}||^2 = \frac{1}{2}\frac{||\mathbf{w}^*||^2}{(\min(c_1,c_2))^2} < \frac{1}{2}||\mathbf{w}^*||^2$, and hence, $\mathbf{w}^*$ is not the optimal solution, a contradiction.

By the observation, we can divide the problem into two cases: $y^{(1)}\mathbf{w}^T\mathbf{x}^{(1)} = 1$ or $y^{(2)}\mathbf{w}^T\mathbf{x}^{(2)} = 1$.

Case 1: $y^{(1)}\mathbf{w}^T\mathbf{x}^{(1)} = 1$
From $y^{(1)}\mathbf{w}^T\mathbf{x}^{(1)} = 1$ and $y^{(2)}\mathbf{w}^T\mathbf{x}^{(2)} \geq 1$, we obtain that $w_2 = 1 - w_1$ and that $w_1 \leq -1$. As $\frac{1}{2}\|\mathbf{w}\|^2 = \frac{1}{2}(w_1^2 + w_2^2) = \frac{1}{2}(w_1^2 + (1 - w_1)^2)$, we have the following optimization problem:

$$\min \frac{1}{2}(w_1^2 + (1 - w_1)^2) \ s.t. \ w_1 \leq -1.$$

The two possible solutions are $w_1 = -1$, $w_1 = \frac{1}{2}$ (take derivative of objective function and set it to 0), but we can exclude $w_1 = \frac{1}{2}$ as it does not satisfy the constraint $w_1 \leq -1$. $w_1 = -1$ corresponds to $\mathbf{w} = (-1, 2)^T$ and an objective value of $\frac{1}{2}\|\mathbf{w}\|^2 = \frac{5}{2}$.

Case 2: $y^{(2)}\mathbf{w}^T\mathbf{x}^{(2)} = 1$
From $y^{(2)}\mathbf{w}^T\mathbf{x}^{(2)} = 1$ and $y^{(1)}\mathbf{w}^T\mathbf{x}^{(1)} \geq 1$, we obtain that $w_1 = -1$ and that $w_2 \geq 2$. As $\frac{1}{2}\|\mathbf{w}\|^2 = \frac{1}{2}(w_1^2 + w_2^2) = \frac{1}{2}(1 + w_2^2)$, we have the following optimization problem:

$$\min \frac{1}{2}(1 + w_2^2) \; s.t. \; w_2 \geq 2.$$

We observe that $w_2 = 2$ is the solution to this optimization problem, and we have that $\mathbf{w} = (-1, 2)^T$ (same as case 1).

Hence,

$$\mathbf{w}^* = [-1, 2]^T.$$

(c) **(3 pts)** Suppose we now allow the bias b to be non-zero. In other words, we now adopt the hard margin SVM formulation from lecture, where $\mathbf{w} = \boldsymbol{\theta}_{1:d}$ are the parameters excluding the bias:

$$\min_{\boldsymbol{\theta}} \frac{1}{2}\|\mathbf{w}\|^2 \; s.t. \; y^{(i)}\boldsymbol{\theta}^T\mathbf{x}^{(i)} \geq 1, i = 1, \ldots, n.$$

How would the classifier and the margin change in the previous question? What are $(\mathbf{w}^*, b^*)$? Compare your solutions with and without bias.

Solution: In this case, the SVM uses both data points as support vectors such that $y^{(1)}(\mathbf{w}^T\mathbf{x}^{(1)} + b) = 1$ and $y^{(2)}(\mathbf{w}^T\mathbf{x}^{(2)} + b) = 1$. The corresponding $\mathbf{w}$, and b are

$$\mathbf{w}^* = [0, 2]^T, b^* = -1$$

The margin for the classifier with bias is larger than the margin for the classifier without bias.

Problem 2 (Boosting [24 pts])

Consider the following examples $(x_1, x_2) \in \mathbb{R}^2$ in Table 1 (i is the example index):

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

Table 1: Dataset for Boosting Problem

In this problem, you will use Boosting to learn a hidden Boolean function from this set of examples. We will use two rounds of AdaBoost to learn a hypothesis for this data set. In each round, AdaBoost chooses a weak learner that minimizes the weighted error ϵ. As weak learners, use hypotheses of the form either (a) $f_1(x_1, x_2) = \text{sign}(x_1 - j_1)$ or (b) $f_2(x_1, x_2) = \text{sign}(x_2 - j_2)$, for some integers $j_1 \in \{-4, 2, 4, 6\}, j_2 \in \{0, 2, 6, 8\}$. Note that values of j_1, j_2 may be different for each round of AdaBoost. When using log, use base e.

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

Table 2: Table for Boosting results

(a) **[6 points]** Start the first round with a uniform distribution $\mathbf{w}_0$, i.e., $w_{0,i} = 0.1$. Place the value for $\mathbf{w}_0$ for each example in the third column of Table 2. Pick an appropriate value of j_1 for $f_1 = \text{sign}(x_1 - j_1)$, i.e. the value that minimizes the error under the uniform distribution $\mathbf{w}_0$, provide the selected value of j_1 in the heading to the fourth column of Table 2, and then write down the value of $f_1(x_1, x_2) = \text{sign}(x_1 - j_1)$ for each example in the fourth column. Repeat this process for j_2 and $f_2(x_1, x_2) = \text{sign}(x_2 - j_2)$ using the fifth column of Table 2. You should not need to consider the value of sign(0). You are permitted to write a script to find the optimal j_1, j_2, though it is not necessary or required.

Solution: We note that $w_{0,i} = 0.1$ for all ten examples. The best learner aligned to the y-axis is $f_1 \equiv \text{sign}(x_1 - 2)$ and the best learner aligned to the x-axis is $f_2 \equiv \text{sign}(x_2 - 6)$.

(b) **[6 points]** Find the candidate hypothesis (i.e., one of f_1 or f_2) given by the weak learner that minimizes the training error ϵ for the uniform distribution. Place this chosen hypothesis as the heading to the sixth column of Table 2, and fill its prediction for each example in that column.

Solution: We choose the latter as h_1 since

$$\epsilon_{f1} = [\text{weighted sum of mistakes if } h_1 = f_1] = \frac{3}{10}$$

$$\epsilon_{f2} = [\text{weighted sum of mistakes if } h_1 = f_2] = \frac{2}{10}$$

.

		Hypothesis 1 (1st iteration)				Hypothesis 2 (2nd iteration)			
i	Label	$\mathbf{w}_0$	$f_1 \equiv$ sign$(x_1 - 2)$	$f_2 \equiv$ sign$(x_2 - 6)$	$h_1 \equiv f_2$	$\mathbf{w}_1$	$f_1' \equiv$ sign$(x_1 - 2)$	$f_2' \equiv$ sign$(x_2 - 0)$	$h_2 \equiv f_1'$
(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	-	-	-	0.25	-	+	-
5	−	0.1	-	+	+	0.25	-	+	-
6	−	0.1	+	-	-	0.0625	+	+	+
7	+	0.1	+	+	+	0.0625	+	+	+
8	−	0.1	-	-	-	0.0625	-	-	-
9	+	0.1	-	+	+	0.0625	-	+	-
10	+	0.1	+	+	+	0.0625	+	+	+

Table 3: **Solution** Table for Boosting results

(c) **[6 points]** Now compute $\mathbf{w}_1$ for each example using h_1, find the new best weak learners f_1' and f_2' given these weights (i.e. find weak learners that minimize the weighted error given weights $\mathbf{w}_1$), and select hypothesis h_2 that minimizes error on the distribution given by $\mathbf{w}_1$, placing the relevant values and predictions in the seventh to tenth columns of Table 2 (similar to parts a and b). Similar to part (a), you should not need to consider the value of sign(0).

Solution: If base e is used, we get:

$$\beta_1 = \frac{1}{2}\ln\frac{1-\epsilon_{f2}}{\epsilon_{f2}} = \frac{1}{2}\ln\frac{0.8}{0.2} = \ln 2 \approx 0.693$$

. Using β_1 (base e) to compute the new distribution, we get:

$$w_{1,i} = \begin{cases} \frac{1}{Z_0} w_{0,i} e^{-\beta_1} & \text{if } h_1(x^{(i)}) = y^{(i)} \\ \frac{1}{Z_0} w_{0,i} e^{\beta_1} & \text{if } h_1(x^{(i)}) \neq y^{(i)} \end{cases}$$

$$w_{1,i} = \begin{cases} \frac{1}{20 Z_0} & \text{if } h_1(x^{(i)}) = y^{(i)} \\ \frac{1}{5 Z_0} & \text{if } h_1(x^{(i)}) \neq y^{(i)} \end{cases}$$

To calculate Z_0,

$$\frac{8}{20 Z_0} + \frac{2}{5 Z_0} = 1 \Rightarrow Z_0 = \frac{4}{5}$$

The new distribution D_1 is thus

$$w_{1,i} = \begin{cases} 0.0625 & \text{if } h_1(x^{(i)}) = y^{(i)} \\ 0.25 & \text{if } h_1(x^{(i)}) \neq y^{(i)} \end{cases}$$

The best learner aligned to the y-axis is $f_1' \equiv \text{sign}(x_1 - 2)$ and the best learner aligned to the x-axis is $f_2' \equiv \text{sign}(x_2 - 0)$.

$$\epsilon_{f1'} = [\text{weighted sum of mistakes if } h_2 = f_1'] = 1 \times 0.25 + 2 \times 0.0625 = 0.375$$

$$\epsilon_{f2'} = [\text{weighted sum of mistakes if } h_2 = f_2'] = 1 \times 0.25 + 3 \times 0.0625 = 0.4375$$

Clearly f_1' is the winner. Then we have: using base e:

$$\beta_2 = \frac{1}{2} \ln \frac{1 - \epsilon_{f1'}}{\epsilon_{f1'}} = \frac{1}{2} \ln \frac{0.625}{0.375} \approx 0.255$$

(d) **[6 points]** What is the final hypothesis produced by AdaBoost?

Solution: The final hypothesis is just a scaled equivalent:

$$H(\mathbf{x}) = \text{sign}(0.693 \times \text{sign}(x_2 - 6) + 0.255 \times \text{sign}(x_1 - 2))$$

What to submit: Fill out Table 2 as explained, show computation of $\mathbf{w}_1$, β_1, β_2 for the chosen hypothesis at each round, and give the final hypothesis, $H(\mathbf{x})$.

Problem 3 (Twitter analysis using SVM [32 pts])

In this project, you will be working with Twitter data. Specifically, we have supplied you with a number of tweets that are reviews/reactions to movies[1],

e.g., *"@nickjfrost just saw The Boat That Rocked/Pirate Radio and I thought it was brilliant! You and the rest of the cast were fantastic! < 3"*.

You will learn to automatically classify such tweets as either positive or negative reviews. To do this, you will employ Support Vector Machines (SVMs), a popular choice for a large number of classification problems.

Starter Files

Code and Data

- `HW3_release.ipynb`. Notebook for the assignment. [2].
- `tweets.txt` contains 630 tweets about movies. Each line in the file contains exactly one tweet, so there are 630 lines in total. The first 560 tweets will be used for training and the last 70 tweets will be used for testing.
- `labels.txt` contains the corresponding labels. If a tweet praises or recommends a movie, it is classified as a positive review and labeled +1; otherwise it is classified as a negative review and labeled −1. These labels are ordered, i.e. the label for the i^{th} tweet in `tweets.txt` corresponds to the i^{th} number in `labels.txt`.

Documentation

- LinearSVC (linear SVM classifier):
  https://scikit-learn.org/stable/modules/generated/sklearn.svm.LinearSVC.html

[1] Please note that these data were selected at random and thus the content of these tweets do not reflect the views of the course staff. :-)

[2] To run the notebook on Google Colab, check the first 3 cells in `HW3_release.ipynb`; otherwise, delete the first 3 cells.

- Cross-Validation:
  https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.StratifiedKFold.html
- Metrics:
  Accuracy: https://scikit-learn.org/stable/modules/generated/sklearn.metrics.accuracy_score.html
  F1-Score: https://scikit-learn.org/stable/modules/generated/sklearn.metrics.f1_score.html
  AUROC: https://scikit-learn.org/stable/modules/generated/sklearn.metrics.roc_auc_score.html
  Precision: https://scikit-learn.org/stable/modules/generated/sklearn.metrics.precision_score.html
  Sensitivity (recall): https://scikit-learn.org/stable/modules/generated/sklearn.metrics.recall_score.html
  Confusion Matrix: https://scikit-learn.org/stable/modules/generated/sklearn.metrics.confusion_matrix.html

---

Skim through the tweets to get a sense of the data and skim through the code to understand its structure.

We use a bag-of-words model to convert each tweet into a feature vector. A bag-of-words model treats a text file as a collection of words, disregarding word order. The first step in building a bag-of-words model involves building a "dictionary". A dictionary contains all of the unique words in the text file. For this project, we will be including punctuations in the dictionary too. For example, a text file containing *"John likes movies. Mary likes movies2!!"* will have a dictionary `{'John':0, 'Mary':1, 'likes':2, 'movies':3, 'movies2':4, '.':5, '!':6}`. Note that the `(key,value)` pairs are `(word, index)`, where the index keeps track of the number of unique words (size of the dictionary).

Given a dictionary containing d unique words, we can transform the n variable-length tweets into n feature vectors of length d (bag of words representation) by setting the i^{th} element of the j^{th} feature vector to 1 if the i^{th} dictionary word is in the j^{th} tweet, and 0 otherwise. We save the feature vectors in a feature matrix, where the rows correspond to tweets (examples) and the columns correspond to words (features).

1 Hyperparameter Selection for a Linear SVM [22 pts]

Next, we will learn a classifier to separate the training data into positive and negative tweets. For the classifier, we will use linear SVMs. We will use the `sklearn.svm.LinearSVC` class[3] and explicitly set the following initialization parameters (and only these initialization parameters): set `loss` to 'hinge', `random_state` to 0, and `C` to various values per the instructions. As usual, we will use `LinearSVC.fit(X,y)` to train our SVM, but in lieu of using `LinearSVC.predict(X)` to make predictions, we will use `LinearSVC.decision_function(X)`, which returns a confidence score proportional to the (signed) distance of the samples to the hyperplane.

SVMs have hyperparameters that must be set by the user. We will select the hyperparameters using 5-fold cross-validation (CV). Using 5-fold CV, we will select the hyperparameters that lead to the 'best' mean performance across all 5 folds.

[3]Note that when using SVMs with the linear kernel (linear SVMs), it is recommended to use sklearn.svm.LinearSVC instead of sklearn.svm.SVC because the backbone of sklearn.svm.LinearSVC is the LIBLINEAR library, which is specifically designed for the linear kernel.

(a) **(6 pts)** The result of a hyperparameter selection often depends upon the choice of performance measure. Here, we will consider the following performance measures: **accuracy**, **F1-Score**, **AUROC**, **precision**, **sensitivity** (i.e. recall), and **specificity**. [4]

Implement `performance(...)`. All measures except specificity are implemented in `sklearn.metrics` library. You can use `sklearn.metrics.confusion_matrix(...)` to calculate specificity. Include a screenshot of your code in the writeup.

Solution: See Figure 1.

```
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)
    else:
        conf_matrix = metrics.confusion_matrix(y_true, y_label, labels=[-1, 1])
        score = np.divide(conf_matrix[0, 0], (conf_matrix[0, 0] + conf_matrix[0, 1] + 0.0))
    return score
    ### ========== TODO : END ========== ###
```

Figure 1: Implementation of performance(...)

(b) **(4 pts)** Next, implement `cv_performance(...)` to return the mean k-fold CV performance for the performance metric passed into the function. Here, you will make use of `LinearSVC.fit(X,y)` and `LinearSVC.decision_function(X)`, as well as your `performance(...)` function.

You may have noticed that the proportion of the two classes (positive and negative) are not equal in the training data. When dividing the data into folds for CV, you should try to keep the class proportions roughly the same across folds. In your write-up, briefly describe why it might be beneficial to maintain class proportions across folds. Then, in `main(...)`, use

[4] Read menu link to understand the meaning of these evaluation metrics.

`sklearn.model_selection.StratifiedKFold(...)` to split the data for 5-fold CV, making sure to stratify using only the training labels.

Solution: Stratified splits are important, since the fundamental assumption of most ML algorithms is that the training set is a representative sample of the test set i.e., the training and test data are drawn from the same underlying distributions. If the ratio of positive to negative examples (the class balance) differs significantly between the training and test sets (across folds) this assumption will not hold.

(c) **(12 pts)** Now, implement `select_param_linear(...)` to choose a setting for C for a linear SVM based on the training data and the specified metric. Your function should call `cv_performance(...)`, passing in instances of `LinearSVC(loss='hinge', random_state=0, C=c)` with different values for `C`, e.g., $C = 10^{-3}, 10^{-2}, \ldots, 10^{2}$. Include a screenshot of your code for the `select_param_linear(...)` function in the writeup. Using the training data and the functions implemented here, find the best setting for C for each performance measure mentioned above. Report the best C for each performance measure.

Solution: See Figure 2 for code implementation.

```
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_score = 0.0
    best_C = 10.0 ** -3
    for c in C_range:
        clf = LinearSVC(loss='hinge', C=c, random_state=0)
        score = cv_performance(clf, X, y, kf, metric)
        if not np.isnan(score) and score > best_score:
            best_score = score
            best_C = c
    return best_C
    ### ========== TODO : END ========== ###
```

Figure 2: Implementation of select_param_linear(...)

Note that for some of the metrics, there may be multiple optimal C values. Only one optimal C value needs to be reported here and used for part 2.
Accuracy: $C = 1, 10, 100$

F1 Score: $C = 1$
AUROC: $C = 1$
Precision: $C = 10, 100$
Sensitivity: $C = 0.001$
Specificity: $C = 1, 10, 100$

2 Test Set Performance [10 pts]

In this section, you will apply the linear SVM classifiers learned in the previous section to the test data. Once you have predicted labels for the test data, you will measure performance.

(a) **(4 pts)** In `main(...)`, using the full training set and `LinearSVC.fit(...)`, train a linear SVM for each performance metric with your best settings of C (use the best setting for each metric; train a total of 6 linear SVMs, each with its own setting of C) and the initialization settings `loss='hinge'` and `random_state=0`. Include a screenshot of your code in the writeup.

Solution: See Figure 3.

```
# part 2a: train linear SVMs with selected hyperparameters
clfs = {}
for metric, best_C in zip(metric_list, best_C_list):
    clf = LinearSVC(loss='hinge', random_state=0, C=best_C)
    clf.fit(X_train, y_train)
    clfs[metric] = clf
```

Figure 3: Implementation of training linear SVM with best C values.

(b) **(6 pts)** Implement `performance_test(...)` which returns the value of a performance measure, given the test data and a trained classifier. Then, for each performance metric, use `performance_test(...)` and the corresponding trained linear-SVM classifier to measure performance on the test data. Include a screenshot of your code for the `performance_test(...)` function in the writeup and report the results. Be sure to include the name of the performance metric employed, and the performance on the test data.

Solution: See Figure 4 for code implementation.

Accuracy: 0.743
F1 Score: 0.471
AUROC: 0.742
Precision: 0.636
Sensitivity: 1.0
Specificity: 0.898 (if $C = 1$ was reported in part 1c for the specificity metric), 0.918 (if $C = 10, 100$ was reported in part 1c for the specificity metric)

```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 ========== ###
    # part 2b: return performance on test data under a metric.
    y_pred = clf.decision_function(X)
    score = performance(y, y_pred, metric)
    return score
    ### ========== TODO : END ========== ###
```

Figure 4: Implementation of performance_test(...).

Problem 4 (Random Forest versus Decision Tree [6 pts])

In this exercise, we will compare Decision Tree (DT) to Random Forest, i.e., ensemble of different DTs on different features. We will explore the effect of two hyper parameters on ensemble performance: i) the number of samples in bootstrap sampling; 2) the maximum number of features to consider for every split when training each DT.

Starter Files

Code and Data

- `HW3_release.ipynb`. Notebook for the assignment.
- `titanic_train.csv`. Toy dataset.

Documentation

- DecisionTreeClassifier:
  https://scikit-learn.org/stable/modules/generated/sklearn.tree.DecisionTreeClassifier.html
- RandomForestClassifier:
  https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestClassifier.html
- Accuracy: https://scikit-learn.org/stable/modules/generated/sklearn.metrics.accuracy_score.html

---

(a) **(2 pts)** Implement the DT algorithm using sklearn.tree.DecisionTreeClassifier with `criterion` set to 'entropy' and `random_state` set to 0. Train and report the training error on the whole dataset. Then use the `error(...)` function provided to report test error. Include the screen-

shot of your code.

Solution: See Figure 5 for code implementation.

Training error on whole dataset: 0.014

```
### ========== TODO : START ========== ###
# Part 4(a): Implement the decision tree classifier and report the training error.
print('Classifying using Decision Tree...')
clf = DecisionTreeClassifier(criterion='entropy', random_state=0)
clf.fit(X, y)
y_pred = clf.predict(X)
train_err = 1 - metrics.accuracy_score(y, y_pred)
print('\tDecision Tree\t-- train error on whole dataset : %.3f' % train_err)
train_error, test_error = error(DecisionTreeClassifier(criterion = 'entropy', random_state=0), X, y)
print('\tDecision Tree\t-- avg train error : %.3f\tavg test error : %.3f' %(train_error, test_error))
### ========== TODO : END ========== ###
```

Figure 5: Implementation of problem 4a.

Test error: 0.241

(b) **(2 pts)** Implement a random forest using sklearn.ensemble.RandomForestClassifier with `criterion` set to 'entropy' and `random_state` set to 0. Adjust the maximum number of samples among 10%, 20%, ..., 80% of the whole data (set `max_samples`), and report, using the `error(...)` function, the training and test error for the best setting and the corresponding choice of hyperparameter. Include the screenshot of your code.

Solution: See Figure 6 for code implementation.

Number of samples: 30% of the dataset

```
### ========== TODO : START ========== ###
# Part 4(b): Implement the random forest classifier and adjust the number of samples used in bootstrap sampling.
print('Classifying using Random Forest...')
best_max_samples = 0.0
best_test_error = np.inf
for max_samples in (0.1 * np.arange(1, 9)):
    clf = RandomForestClassifier(criterion='entropy', max_samples=max_samples, random_state=0)
    train_error, test_error = error(clf, X, y)
    print('\tRandom Forest\t-- max samples: %f\tavg train error : %.3f\tavg test error : %.3f' %(max_samples, train_error, test_error))
    if test_error < best_test_error:
        best_test_error = test_error
        best_max_samples = max_samples
### ========== TODO : END ========== ###
```

Figure 6: Implementation of problem 4b.

Training error: 0.094

Test error: 0.187

(c) **(2 pts)** Implement a random forest with `criterion` set to 'entropy' and `random_state` set to 0 and adjust the maximum number of features among 1, 2, ..., 7 (set `max_features`) and report, using the `error(...)` function, the training and test error for the best setting and the corresponding choice of hyperparameter. For the maximum number of samples, use the one that performed the best in Part b. Include the screenshot of your code.

Solution: See Figure 7 for code implementation.

Number of features: 2 or 3 features

Training error: 0.094 or 0.095, respectively

```
### ========== 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...')
for max_features in range(1, 8):
    clf = RandomForestClassifier(criterion='entropy', max_features=max_features, max_samples=best_max_samples, random_state=0)
    train_error, test_error = error(clf, X, y)
    print('\tRandom Forest\t-- max features: %f\tavg train error : %.3f\tavg test error : %.3f' %(max_features, train_error, test_error))
### ========== TODO : END ========== ###
```

Figure 7: Implementation of problem 4c.

Test error: 0.187