CS M146 Fall 2023 Homework 3 Solutions

UCLA ID: 605900342
Name: Caleb Traxler
Collaborators: N/A

By turning in this assignment, I agree by the UCLA honor code and declare that all of this is my own work.

Problem 1 (Support Vector Machines)

.

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,

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

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

   Given the constraint: $y^{(i)} w^T x^{(i)} \geq 1$, where $x = (1,1)^T, y = -1$.

   $\implies (-1)\begin{bmatrix} w_1 & w_2 \end{bmatrix}\begin{bmatrix} 1 \\ 1 \end{bmatrix} \geq 1 \implies$ -w$_1$ $- w_2 \geq 1$.

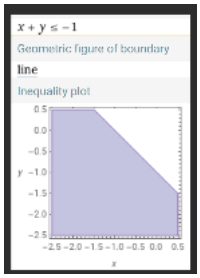
   $\implies -(w_1 + w_2) \geq 1$. Thus I see that $w_1 + w_2 \leq -1$.

   Because I am minimizing $\frac{1}{2}(w_1^2 + w_2^2)$. I want to find that smallest values of w_1 and w_2 that satisfy this inequality. This occurs when w_1 and w_2 are equal because we are in the first quadrant where both w_1 and w_2 are negative and the line $w_1 + w_2 = -1$ is symmetric w.r.t the line $w_1 = w_2$.

   Thus, upon solving for w_1 and w_2 using $w_1 = w_2$, I see that: $w_1 + w_1 = -1 \implies w_1 = -\frac{1}{2}$.

   Furthermore I can conclude that $w_1 = w_2 = -\frac{1}{2}$. Therefore the weight that satisfies the constrained minimization is: $w^* = (-\frac{1}{2}, -\frac{1}{2})$, such that $\frac{1}{2}||w^*||^2 = 0.25$.

   Geometric Justification:

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

   Given the constraint: $y^{(i)} w^T x^{(i)} \geq 1$, where $x^{(1)} = (1,1)^T$, $y^{(1)} = 1$, and $x^{(2)} = (1,0)^T$, $y^{(2)} = -1$.

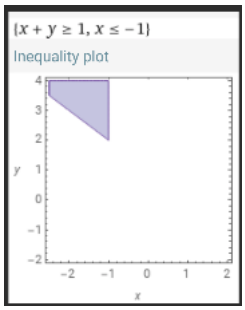
   $\implies \begin{bmatrix} 1 \\ -1 \end{bmatrix}\begin{bmatrix} w_1 & w_2 \end{bmatrix}\begin{bmatrix} 1 & 1 \\ 1 & 0 \end{bmatrix} \geq 1 \implies w_1 + w_2 \geq 1$ and $-w_1 \geq 1$ i.e. $w_1 \leq -1$.

   Thus the value of w_1 that minimizes should be $w_1 = -1$.

With $w_1 = -1$, I see that $-1 + w_2 \geq 1$. Thus, if I choose $w_2 = 2$, I obtain the smallest values of w_1 and w_2 to minimize the objective function.

$\implies w_1 = -1$ and $w_2 = 2$. Therefore the weight vector that minimizes the objective function subject to the constraints is $w^* = (-1, 2)^T$. Such that the value of the objective function is $\frac{1}{2}((-1)^2+(2)^2) = \frac{5}{2}$.

Geometric Justification:



3. 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 $w = \boldsymbol{\theta}_{1:d}$ are the parameters excluding the bias:

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

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

When I introduce a bias term b into the SVM formulation, the decision boundary is no longer forced to pass through the origin. This allows the hyperplane to be shifted up or down, which give potential to better generalize on unseen data.

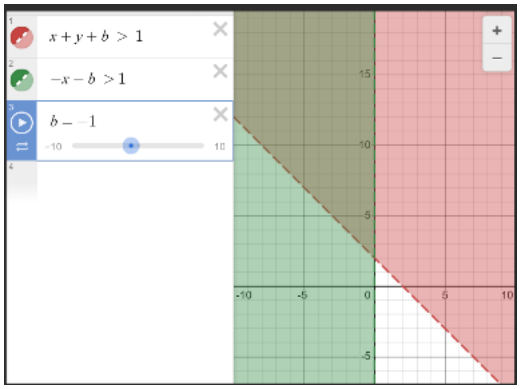
Recall from lecture the definition of Hard-Margin Support Vector Machines: The objective function is $min_\theta ||w||_2^2$, subject to the constraints $y^{(i)}\theta^T x^{(i)} \geq 1$ for all $i = 1, ...n$. Note that this is a convex quadratic objective, n linear constraints and can be solved using quadratic programming solvers.

Given the two training examples $x^{(1)} = (1, 1)^T \in \mathbb{R}^2$ and $x^{(2)} = (1, 0)^T \in \mathbb{R}^2$ with labels $y^{(1)} = 1$ and $y^{(2)} = -1$, we can set up the following constraints for the SVM with a non-zero bias term:

$y^{(1)}(w_1 + w_2 + b) \geq 1 \implies w_1 + w_2 + b \geq 1.$

$y^{(2)}(w_1 + b) \geq 1 \implies -w_1 - b \geq 1.$

Geometric Intuition:



Geometrically, as b varies from $-\infty \rightarrow \infty$, the intersection point of the two constraints in the w_1, w_2 plane always occurs at $w_2 = 2$, which indicates that w_2 is fixed. The value of w_1 changes linearly with b such that $w_1 = b - 1$. For any chosen , we can solve for the corresponding w_1 to meet the constraints.

If I choose $b = -1$, then based on geometric interpretation, the solution for w^* and b^* is : $w^* = (0, 2)$, because at $b = -1$, w_1 needs to be 0 to satisfy $w_1 = b - 1$. Thus, $b^* = -1$.

Thus the objective function has a value of $\frac{1}{2}||w||^2 = 2$.

With Bias:

Allowing a bias term for a specific value of $b = -1$, I found that $w^* = (0, 2)$, with an objective function value of 2.

Without Bias:

Without a bias, i.e. b=0. For the second problem, I found that the optimal solution was $w^* = (-1, 2)$ with an objective function value of 2.5.

Thus, allowing a bias reduced the value of the objective function, hence resulting in a successful bias selection and implementation.

Problem 2 (Boosting)

. 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

$x_1 = [0, 1, 3, -2, -1, 10, 12, -7, -3, 5]$
$x_2 = [5, 4, 7, 1, 13, 3, 7, -1, 12, 9]$
$Labels = [-, -, +, +, -, -, +, -, +, +]$

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. Note that $j_{1,1} = -4, j_{1,2} = 2, j_{1,3} = 4, j_{1,4} = 6$ and $j_{2,1} = 0, j_{2,2} = 2, j_{2,3} = 6, j_{2,4} = 8$

		Hypothesis 1 (1st iteration)				Hypothesis 2 (2nd iteration)			
i (1)	Label (2)	$\mathbf{w}_0$ (3)	$f_1 \equiv$ sign(x_1−j$_{1,2}$) (4)	$f_2 \equiv$ sign($x_2 - j_{2,3}$) (5)	$h_1 \equiv$ f$_2$ (6)	$\mathbf{w}_1$ (7)	$f_1' \equiv$ sign($x_1 - j_{1,2}$) (8)	$f_2' \equiv$ sign($x_2 - j_{2,1}$) (9)	$h_2 \equiv$ f'$_1$ (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 2: Table for Boosting results

Recall the AdaBoost Algorithm:
1: Initialize a vector of n uniform weights w_1.
2: for $t = 1, ..., T$
3: Train model h_T on X, y with weights w_t
4: Compute the weighted training error of h_t.
5: Choose $\beta_t = \frac{1}{2} ln(\frac{1-\epsilon_t}{\epsilon_t})$
6: Update all instance weights: $w_{t+1,i} = w_{t,i} exp(-\beta_t y_i h_t(x_i))$
7: Normalize w_{t+1} to be a distribution
8: end for
9: Return the hypothesis $H(x) = sign(\sum_{t=1}^{T} \beta_t h_t(x))$

1. **[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 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.

   ANSWER:

   The hypothesis are of the form: $f_1(x_1, x_2) = \text{sign}(x_1 - j_1)$ or $f_2(x_1, x_2) = \text{sign}(x_2 - j_2)$

   Given that$j_1 \in \{-4, 2, 4, 6\}, j_2 \in \{0, 2, 6, 8\}$ and $w_{0,i} = 0.1$, I want to find the smallest weighted error:

j_1										
x_1	0	1	3	-2	-1	10	12	-7	-3	5
x_1 - j_1,1	4	5	7	2	3	14	16	-3	1	9
x_1 - j_1,2	-2	-1	1	-4	-3	8	10	-9	-5	3
x_1 - j_1,3	-4	-3	-1	-6	-5	6	8	-11	-7	1
x_1 - j_1,4	-6	-5	-3	-8	-7	4	6	-13	-9	-1
j_2										
x_2	5	4	7	1	13	3	7	-1	12	9
x_2- j_2,1	5	4	7	1	13	3	7	-1	12	9
x_2 - j_2,2	3	2	5	-1	11	1	5	-3	10	7
x_2 - j_2,3	-1	-2	1	-5	7	-3	1	-7	6	3
x_2 - j_2,4	-3	-4	-1	-7	5	-5	-1	-9	4	1

j_1											Errors:
Actual	n	n	p	p	n	n	p	n	p	p	
x_1 - j_1,1	p	p	p	p	p	p	p	n	p	p	0.4
x_1 - j_1,2	n	n	p	n	n	p	p	n	n	p	0.3
x_1 - j_1,3	n	n	n	n	n	p	p	n	n	p	0.4
x_1 - j_1,4	n	n	n	n	n	p	p	n	n	n	0.5
Min Error = 0.3											
j_2											Errors:
Actual	n	n	p	p	n	n	p	n	p	p	
x_2- j_2,1	p	p	p	p	p	p	p	n	p	p	0.4
x_2 - j_2,2	p	p	p	n	p	p	p	n	p	p	0.5
x_2 - j_2,3	n	n	p	n	p	n	p	n	p	p	0.2
x_2 - j_2,4	n	n	n	n	p	n	n	n	p	p	0.4
Min Error = 0.2											

Thus, I see that for x_1, the j_1 which produces the smallest error will be $j_{1,2} = 2$. For $x_1 - j_{1,2}$ gives an error of 0.3, which is the smallest. Furthermore, I also see that for x_2, the j_2 which produces the smallest error will be $j_{2,3} = 6$. For $x_2 - j_{2,3}$ gives an error of 0.2, which is the smallest. Choosing only one, I will select $x_1 - j_{2,3} = 0.2$.

2. **[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.

   I found that the candidate hypothesis, f_2 given by the weak learner minimizes the training error ϵ for the uniform distribution.

   Thus, $h_1 \equiv f_2 = [-, -, +, -, +, -, +, -, +, +]$

   Note that $\epsilon = 0.2$

   $\beta_1 = \frac{1}{2} ln(\frac{1-0.2}{0.2}) = 0.69315$

3. **[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).

   Now I want to compute w_1. Recall from the algorithm that $w_{t+1,i} = w_{t,i} exp(-\beta_t y_i h_t(x_i))$.

   Note that $y_i = [-1, -1, 1, 1, -1, -1, 1, -1, 1, 1]$ and $h_1(x_i) = [-1, -1, 1, -1, 1, -1, 1, -1, 1, 1]$

   I can first compute $([-1, -1, 1, 1, -1, -1, 1, -1, 1, 1])([-1, -1, 1, -1, 1, -1, 1, -1, 1, 1]) = [1, 1, 1, -1, -1, -1, 1, 1, 1, 1]$

   Thus, $w_1 = w_0 exp(-\beta_1 y_i h_1(x_i)) = (0.1) exp(-(0.69315)([1, 1, 1, -1, -1, 1, 1, 1, 1, 1)$

   $w_{1,1} = (0.1) exp(-(0.69315)(1)) = 0.0499999$

   $w_{1,2} = (0.1) exp(-(0.69315)(1)) = 0.0499999$

   $w_{1,3} = (0.1) exp(-(0.69315)(1)) = 0.0499999$

   $w_{1,4} = (0.1) exp(-(0.69315)(-1)) = 0.20$

   $w_{1,5} = (0.1) exp(-(0.69315)(-1)) = 0.20$

   $w_{1,6} = (0.1) exp(-(0.69315)(1)) = 0.0499999$

   $w_{1,7} = (0.1) exp(-(0.69315)(1)) = 0.0499999$

   $w_{1,8} = (0.1) exp(-(0.69315)(1)) = 0.0499999$

   $w_{1,9} = (0.1) exp(-(0.69315)(1)) = 0.0499999$

$w_{1,10} = (0.1)exp(-(0.69315)(1)) = 0.0499999$

Now, I want to normalize $w_{1,i}, i = 1, ..., 10$ to be a distribution.

In general, $w_{1,i}^{normalized} = \frac{w_{1,i}}{\sum_{j=1}^{N} w_{1,j}}$, such that $\sum_{j=1}^{N} w_{1,j} = 0.8$.

Thus, if $w_{1,i} = 0.0499999$, then $w_{1,i}^{normalized} = \frac{0.0499999}{0.8} = 0.0625$.

If $w_{1,i} = 0.2$, then $w_{1,i}^{normalized} = \frac{0.2}{0.8} = 0.25$. Also note that $(0.0625)(8) + (0.25)(2) = 1$, hence normalized.

4. **[6 points]** What is the final hypothesis produced by AdaBoost?

2nd Iteration											
j_1											
Weight:	0.0625	0.0625	0.0625	0.25	0.25	0.0625	0.0625	0.0625	0.0625	0.0625	Errors:
Actual	n	n	p	p	n	n	p	n	p	p	
x_1 - j_1,1	p	p	p	p	p	p	p	n	p	p	0.4375
x_1 - j_1,2	n	n	p	n	n	p	p	n	n	p	0.375
x_1 - j_1,3	n	n	n	n	n	p	p	n	n	p	0.4375
x_1 - j_1,4	n	n	n	n	n	p	p	n	n	n	0.5
Min Error =	0.375										
j_2											
Weight:	0.0625	0.0625	0.0625	0.25	0.25	0.0625	0.0625	0.0625	0.0625	0.0625	Errors:
Actual	n	n	p	p	n	n	p	n	p	p	
x_2- j_2,1	p	p	p	p	p	p	p	n	p	p	0.4375
x_2 - j_2,2	p	p	p	n	p	p	p	n	p	p	0.6875
x_2 - j_2,3	n	n	p	n	p	n	p	n	p	p	0.5
x_2 - j_2,4	n	n	n	n	p	n	n	n	p	p	0.625
Min Error =	0.4375										

Thus, I see that for x_1, the j_1 which produces the smallest error will be $j_{1,2} = 2$. For $x_1 - j_{1,2}$ gives an error of 0.375, which is the smallest. Furthermore, I also see that for x_2, the j_2 which produces the smallest error will be $j_{2,1} = 0$. For $x_2 - j_{2,3}$ gives an error of 0.4375, which is the smallest. Choosing only one, I will select $x_1 - j_{1,2} = 0.375$.

I found that the candidate hypothesis, f_1' given by the weak learner minimizes the training error ϵ for the uniform distribution.

Thus, $h_2 \equiv f_1' = [-, -, +, -, -, +, +, -, -, +]$

Note that $\epsilon = 0.375$

$\beta_2 = \frac{1}{2} ln(\frac{1-0.375}{0.375}) = 0.255413$

Recall that $H(x) = sign(\sum_{t=0}^{T} \beta_t h_t(x))$, thus, I see that

$H(x) = sign(\beta_1 h_1(x) + \beta_2 h_2(x)) = sign(0.69315 sign(x_2 - j_{2,3}) + 0.255413 sign(x_1 - j_{1,2}))$.

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)

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

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. :-), 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. [1].
- `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
- Cross-Validation:

https://scikit-learn.org/stable/modules/generated/sklearn.model$_s$election.StratifiedKFold.html$Metrics$:
$Accuracy$: $https://scikit-learn.org/stable/modules/generated/sklearn.metrics.accuracy_score.html$
$F1-Score$: $https://scikit-learn.org/stable/modules/generated/sklearn.metrics.f1_score.html$
$AUROC$: $https://scikit-learn.org/stable/modules/generated/sklearn.metrics.roc_auc_score.html$
$Precision$: $https://scikit-learn.org/stable/modules/generated/sklearn.metrics.precision_score.html$
$Sensitivity(recall)$: $https://scikit-learn.org/stable/modules/generated/sklearn.metrics.recall_score.html$
$ConfusionMatrix$: $https://scikit-learn.org/stable/modules/generated/sklearn.metrics.confusion_matrix.html$

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] To run the notebook on Google Colab, check the first 3 cells in $\texttt{HW3}_release.ipynb|; otherwise, delete the first 3 cells.$

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 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.

1. 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**. [2]

   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.

```python
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 1c: compute classifier performance

    if metric == "accuracy":
        return accuracy_score(y_true, y_label)
    elif metric == "f1-score":
        return f1_score(y_true, y_label)
    elif metric == "auroc":
        return roc_auc_score(y_true, y_pred)
    elif metric == "precision":
        return precision_score(y_true, y_label)
    elif metric == "sensitivity":
        return recall_score(y_true, y_label)
    elif metric == "specificity":
        tn, fp, fn, tp = confusion_matrix(y_true, y_label).ravel()
        return tn / float(tn+fp)
    else:
        raise ValueError("Unknown Metric.")

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

[2] Read menu http://scikit-learn.org/stable/modules/model$_e$*valuation.html*roc−*metrics*link to understand the meaning of these evaluation metric

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

   Question: Implement cv_performance(...) to return the mean k-fold CV performance for the performance metric passed into the function.

   Answer:

```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_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)
        y_pred = clf.decision_function(X_test)
        score = performance(y_test, y_pred, metric)
        scores.append(score)

    return np.mean(scores)


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

Question: In your write-up, briefly describe why it might be beneficial to maintain class proportions across folds.

Answer: Maintaining class proportions across folds in k-fold cross validation ensures that each fold is representative of the full data-set, which is especially important in imbalanced data-sets. This approach leads to more reliable and stable performance estimates, as it reduces the variability that could arise from non-representative folds, thus giving a more accurate picture of the models ability to generalize to new data.

Question: Then, in main(...), use sklearn.model selection.StratifiedKFold(...)— to split the data for 5-fold CV, making sure to stratify using only the training labels.

Answer:

```
### ========== TODO : START ========== ###
# part 1b: create stratified folds (5-fold CV)

kf = StratifiedKFold(n_splits=5, shuffle=True, random_state=None)
```

3. 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.

```
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)
    best_score = -np.inf
    best_C = None

    ### ========== TODO : START ========== ###
    # part 1c: select optimal hyperparameter using cross-validation

    for C in C_range:
        clf = LinearSVC(loss='hinge', random_state = 0, C=C)
        score = cv_performance(clf, X, y, kf, metric)
        print(f"C: {C}, {metric}, {score}")

        if score > best_score:
            best_score = score
            best_C = C
    return best_C


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

Question: 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.

Answer:

In Main:

```
# part 1c: for each metric, select optimal hyperparameter for linear SVM using CV

best_C_values = {}
for metric in metric_list:
    best_C = select_param_linear(X_train, y_train, kf, metric)
    best_C_values[metric] = best_C
    print(f"Best C for {metric}: {best_C}")
```

Question: Report the best C for each performance measure.

Answer:

Best C for accuracy: 1.0

Best C for f1-score: 1.0

Best C for auroc: 100.0

Best C for precision: 10.0

Best C for sensitivity: 0.001

Best C for specificity: 1.0

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.

1. 4 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.

```
# part 2a: train linear SVMs with selected hyperparameters
performances = {}
for metric, best_C in best_C_values.items():
    clf = LinearSVC(loss='hinge', random_state=0, C=best_C)
    clf.fit(X_train, y_train)
```

2. 6 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.

   In Main:

```
# part 2b: test the performance of your classifiers.
    y_pred = clf.decision_function(X_test)
    performance_score = performance(y_test, y_pred, metric)
    performances[metric] = performance_score
    print(f"Test performance for {metric}: {performance_score}")
```

   Function:

```
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
```

Overall Output Result:

best_C_values

```
1811
Linear SVM Hyperparameter Selection based on accuracy:
C: 0.001, accuracy, 0.7089285714285715
C: 0.01, accuracy, 0.7857142857142857
C: 0.1, accuracy, 0.8232142857142858
C: 1.0, accuracy, 0.8553571428571429
C: 10.0, accuracy, 0.8410714285714285
C: 100.0, accuracy, 0.8303571428571427
Best C for accuracy: 1.0
Linear SVM Hyperparameter Selection based on f1-score:
C: 0.001, f1-score, 0.8296684118673647
C: 0.01, f1-score, 0.8706029281089144
C: 0.1, f1-score, 0.8877170585088769
C: 1.0, f1-score, 0.8955839997529085
C: 10.0, f1-score, 0.8845909645909644
C: 100.0, f1-score, 0.8757878679811746
Best C for f1-score: 1.0
Linear SVM Hyperparameter Selection based on auroc:
C: 0.001, auroc, 0.6853714758342924
C: 0.01, auroc, 0.8452170538454162
C: 0.1, auroc, 0.8880539772727272
C: 1.0, auroc, 0.8940758055235903
C: 10.0, auroc, 0.8947513305523589
C: 100.0, auroc, 0.9013808676160338
Best C for auroc: 100.0
Linear SVM Hyperparameter Selection based on precision:
C: 0.001, precision, 0.7089285714285715
C: 0.01, precision, 0.7755639661129191
C: 0.1, precision, 0.8331278101671578
C: 1.0, precision, 0.8583024785766789
C: 10.0, precision, 0.8726513541403275
C: 100.0, precision, 0.8701373820678036
Best C for precision: 10.0
Linear SVM Hyperparameter Selection based on sensitivity:
C: 0.001, sensitivity, 1.0
C: 0.01, sensitivity, 0.9898734177215189
C: 0.1, sensitivity, 0.9495569620253164
C: 1.0, sensitivity, 0.8993354430379746
C: 10.0, sensitivity, 0.9018354430379747
C: 100.0, sensitivity, 0.909367088607595
Best C for sensitivity: 0.001
Linear SVM Hyperparameter Selection based on specificity:
C: 0.001, specificity, 0.0
C: 0.01, specificity, 0.30113636363636365
C: 0.1, specificity, 0.5333333333333333
C: 1.0, specificity, 0.6814393939393939
C: 10.0, specificity, 0.6628787878787878
C: 100.0, specificity, 0.6191287878787878
Best C for specificity: 1.0
Test performance for accuracy: 0.7428571428571429
Test performance for f1-score: 0.47058823529411764
Test performance for auroc: 0.7453838678328474
Test performance for precision: 0.6363636363636364
Test performance for sensitivity: 1.0
Test performance for specificity: 0.8979591836734694
```

Problem 4 (Random Forest versus Decision Tree)

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.DecisionTreeCla ssifier.html
- RandomForestClassifier:
  https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomFores tClassifier.html

- Accuracy: https://scikit-learn.org/stable/modules/generated/sklearn.metrics.accuracy$_s$*core.html*

---

1. 2 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 screenshot of your code.

```
### ========== 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_error = 1 - metrics.accuracy_score(y, y_pred)
print(f"Training error: {train_error}")

train_error, test_error = error(clf, X, y)
print(f"Average Test Error (over {100} trials): {test_error}")
### ========== TODO : END ========== ###
```

```
Classifying using Decision Tree...
Training error: 0.014044943820224698
Average Test Error (over 100 trials): 0.24104895104895108
```

2. 2 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 verb—error(...)— function, the training and test error for the best setting and the corresponding choice of hyperparameter. Include the screenshot of your code.

```
### ========== 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...')
n = len(X)
sample_percentages = [0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8]

best_test_error = float('inf')
best_max_samples = None

for percentage in sample_percentages:
    max_samples = int(n * percentage)

    clf = RandomForestClassifier(criterion='entropy', random_state=0, max_samples=max_samples)
    train_error, test_error = error(clf, X, y)

    print(f"Max samples: {max_samples} (Percentage: {percentage}), Training Error: {train_error}, Test Error: {test_error}")

    if test_error < best_test_error:
        best_test_error = test_error
        best_max_samples = max_samples
print(f"Best setting: max_samples = {best_max_samples}, with Test Error = {best_test_error}")
### ========== TODO : END ========== ###
```

```
Classifying using Random Forest...
Max samples: 71 (Percentage: 0.1), Training Error: 0.1357293497363796, Test Error: 0.19587412587412587
Max samples: 142 (Percentage: 0.2), Training Error: 0.10314586994727591, Test Error: 0.18797202797202794
Max samples: 213 (Percentage: 0.3), Training Error: 0.0818629173989455, Test Error: 0.18888111888111891
Max samples: 284 (Percentage: 0.4), Training Error: 0.05869947275922671, Test Error: 0.19216783216783218
Max samples: 356 (Percentage: 0.5), Training Error: 0.03388400702987697, Test Error: 0.19888111888111892
Max samples: 427 (Percentage: 0.6), Training Error: 0.017785588752196824, Test Error: 0.20111888111888113
Max samples: 498 (Percentage: 0.7), Training Error: 0.012390158172232001, Test Error: 0.20475524475524473
Max samples: 569 (Percentage: 0.8), Training Error: 0.011528998242530775, Test Error: 0.20671328671328676
Best setting: max_samples = 142, with Test Error = 0.18797202797202794
```

3. 2 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.

```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...')

best_max_samples = 142

best_test_error = float('inf')
best_max_features = None

for max_features in range(1, 9):
    clf = RandomForestClassifier(criterion='entropy', random_state=0, max_samples=best_max_samples, max_features
    train_error, test_error = error(clf, X,y)

    print(f"Max features: {max_features}, Training Error: {train_error}, Test Error: {test_error }")

    if test_error < best_test_error:
        best_test_error = test_error
        best_max_features = max_features
print(f"Best setting: max_features = {best_max_features}, with Test Error = {best_test_error}")

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

```
Classifying using Random Forest...
Max features: 1, Training Error: 0.10121265377855888, Test Error: 0.18776223776223777
Max features: 2, Training Error: 0.10314586994727591, Test Error: 0.18797202797202794
Max features: 3, Training Error: 0.10244288224956065, Test Error: 0.1872727272727273
Max features: 4, Training Error: 0.10430579964850617, Test Error: 0.1874125874125874
Max features: 5, Training Error: 0.10544815465729351, Test Error: 0.1886013986013986
Max features: 6, Training Error: 0.10581722319859402, Test Error: 0.189020979020979
Max features: 7, Training Error: 0.10776801405975397, Test Error: 0.18895104895104897
Max features: 8, Training Error: 0.10776801405975397, Test Error: 0.18895104895104897
Best setting: max_features = 3, with Test Error = 0.1872727272727273
```