Machine Learning Project #2

By: Courtney Hodge

Before the Presentation Begins...

Note:

> I completed this project **by myself** so please grade with that knowledge in mind. I was my own team and I tried my best to provide a quality assignment with as few errors as possible. Part B was the most difficult portion of this project becuase I got confused with the directions. There was a lot of ambiguity to where I just had to trust my instincts and hope that how I proceeded with the assignment worked fine.

> Please also note that I gave this project as much attention as I possibly could. The last week of school was a nightmare and I hope that you can accept this assignment at the time it was submitted. Thank you for your attention!

Data Wrangling

Data Wrangling:

- Mount the Drive
- Load the Data

mount google drive

```
[632] from google.colab import drive
      drive.mount('/content/drive')
```

load in iris data

```
[633] import pandas as pd
      import numpy as np

      from sklearn import datasets
      iris = datasets.load_iris()

      data = iris.data
      label = iris.target
```

A) D-INDEX AND OTHER CLASSIFICATION MEASURES (50 POINTS)

Section I:
What's the advantage of D-index?

Part A: What's the advantage of d-index?

Lecture067

> d-index is the 7 th measure

> **d_index is not only convenient for comparing ML results, but also an effective measure to reflect the true ML status in imbalanced learning**

> d = log(2)(1+a) + log(2)(1 + (s+p)/2)

> where a,s, and p, represent the corresponding diagnostic accuracy, sensitivity, and specificity respectively

> 1.33<D-index<=2 (if it reaches 1.8 --> good)

Section II:
How can you calculate d-index for multi-class classification?

Part A: How can you calculate d-index for multi-class classification?

- Refer to slide 14

you can do so by referencing the d-index function below. This function takes the true labels and predicted labels of the iris dataset (or any dataset) and it finds the TP, FP, TN, FN logical assignments, finds the needed accuracy, sensitivity, and specificity classification measures, and uses an np array to append those calculations into an array that returns the answers.

Section III:
Do SVM classification for Iris data with the first 70% training and remaining 30% for test and calculate the following classification measures and explain their meaning.

Part A: Which ones are more representative, why?

I would have to say that the classification measures used to find the d-index, f1_micro and f1-macro, roc_auc_score, and balanced_accuracy are all pretty representative. All classification measures but roc-auc-score is perfect in predicting their values for the iris data and they all do a pretty good job.

Part A: Helper FUnctions

Train/ Test Split for 70/30

```
[634] #---split into 70/30---#
      def split (data, labels, size):
        from sklearn.model_selection import train_test_split
        from sklearn.preprocessing import StandardScaler

        #---standardize the data---#
        X = StandardScaler().fit_transform(data)

        train_data, test_data, train_data_label, test_data_label = train_test_split(X, labels, test_size = size, random_state = 42)

        return train_data, test_data, train_data_label, test_data_label
```

Part A: Helper FUnctions

SVM Function

```
[635] def SVM(data, label):
        #---conduct Regular SVM for iris data---#
        from sklearn import svm
        from sklearn.model_selection import cross_val_score
        import numpy as np

        #---split 70/30---#
        iris_train_data, iris_test_data, iris_train_label, iris_test_label = split(data, label, 0.3)

        #---create an SVM instance---#
        clf = svm.SVC(kernel = 'rbf', gamma = 0.5, C = 1)

        #---fit data---#
        clf.fit(iris_train_data, iris_train_label)

        #---train data---#
        iris_test_pred_label = clf.predict(iris_test_data)

        return iris_test_label, iris_test_pred_label
```

Part A: d-index

```
[636] def compute_measure(true_label, pred_label):
        import numpy as np
        t_indx = (true_label == pred_label)
        f_indx = np.logical_not(t_indx)

        p_indx = (true_label > 0)      #positive targets
        n_indx = np.logical_not(p_indx)    #negative targets

        tp = np.sum(np.logical_and(t_indx, p_indx))    #TP
        tn = np.sum(np.logical_and(t_indx, n_indx))    #TN

        fp = np.sum(n_indx) - tn
        fn = np.sum(p_indx) - tp
        tp_fp_tn_fn_list = []

        tp_fp_tn_fn_list.append(tp)
        tp_fp_tn_fn_list.append(fp)
        tp_fp_tn_fn_list.append(tn)
        tp_fp_tn_fn_list.append(fn)
        tp_fp_tn_fn_list = np.array(tp_fp_tn_fn_list)

        tp = tp_fp_tn_fn_list[0]
        fp = tp_fp_tn_fn_list[1]
        tn = tp_fp_tn_fn_list[2]
        fn = tp_fp_tn_fn_list[3]
```

```
with np.errstate(divide = 'ignore'):
  sen = (1.0 * tp) / (tp + fn)

with np.errstate(divide = 'ignore'):
  spec = (1.0 * tn) / (tn + fp)

with np.errstate(divide = 'ignore'):
  ppr = (1.0 * tp) / (tp + fp)

with np.errstate(divide = 'ignore'):
  npr = (1.0 * tn) / (tn + fn)

with np.errstate(divide = 'ignore'):
  f1 = tp / (tp + 0.5 * (fp + fn))

acc = (tp + tn) * 1.0 / (tp + fp + tn + fn)

d = np.log2(1 + acc) + np.log2( 1 + (sen + spec) / 2)

ans = []

#ans.append(acc)
#ans.append(sen)
#ans.append(spec)
#ans.append(ppr)
#ans.append(npr)
#ans.append(f1)
print("d-index: ")
ans.append(d)

return ans
```

Part A: d-index results

```
[637] iris_test_label, iris_test_pred_label = SVM(data, label)

      compute_measure(iris_test_label, iris_test_pred_label)

      d-index:
      [2.0]
```

Part A: F1-micro & f1-Macro

f1-micro & f1-macro

[scikit]

> The F1 score can be interpreted as a harmonic mean of the precision and recall, where an F1 score reaches its best value at 1 and worst score at 0. The relative contribution of precision and recall to the F1 score are equal. The formula for the F1 score is:

> F1 = 2 * (precision * recall) / (precision + recall)

> 'micro' : Calculate metrics globally by counting the total true positives, false negatives and false positives.

> 'macro' :

Part A: F1-micro & f1-Macro Results

```
[638] def f1_score(data, label):
          from sklearn.metrics import f1_score

          iris_test_label, iris_test_pred_label = SVM(data, label)

          f1_micro = f1_score(iris_test_label, iris_test_pred_label, average = 'micro')

          f1_macro = f1_score(iris_test_label, iris_test_pred_label, average = 'macro')

          print("f1-micro: ", f1_micro, '\n')
          print("f1-macro: ", f1_macro, '\n')
```

```
[639] f1_score(data, label)

f1-micro:  1.0

f1-macro:  1.0
```

Part A: Balanced Accuracy

balanced_accuracy

scikit

> The balanced_accuracy_score function computes the balanced accuracy, which avoids inflated performance estimates on imbalanced datasets. It is the macro-average of recall scores per class or, equivalently, raw accuracy where each sample is weighted according to the inverse prevalence of its true class. Thus for balanced datasets, the score is equal to accuracy.

Part A: Balanced Accuracy Results

```
[640] def bal_acc(data, label):
        from sklearn.metrics import balanced_accuracy_score

        iris_test_label, iris_test_pred_label = SVM(data, label)

        #---balanced_accuracy---#
        balanced_accuracy = balanced_accuracy_score(iris_test_label, iris_test_pred_label)

        print("balanced_accuracy: ", balanced_accuracy, '\n')
```

```
[641] bal_acc(data, label)

      balanced_accuracy:  1.0
```

Part A: Balanced Accuracy

roc_auc_score

scikit

> Compute Area Under the Receiver Operating Characteristic Curve (ROC AUC) from prediction scores.

Part A: Balanced Accuracy Results

```
[642] def roc_auc(data, label):
          from sklearn.datasets import load_iris
          from sklearn.metrics import roc_auc_score
          from sklearn.linear_model import LogisticRegression
          X, y = load_iris(return_X_y=True)
          clf = LogisticRegression(solver="liblinear").fit(X, y)
          roc_auc = roc_auc_score(y, clf.predict_proba(X), multi_class='ovr')

          print("roc_auc_score: ", roc_auc, "\n")
```

```
[643] roc_auc(data, label)
```

```
roc_auc_score:  0.9913333333333334
```

B) Bagging-SVM (50 points)

Before Part B Begins...

Note:

> Disclaimer: I DID NOT standardize the below datasets for this portion of the project because I found that standardizing the data standardizes the label columns of the data too. I was struggling with indexing the data labels aside from the data for all three and I just decided that I would be able to make more progress if I just continued without standardizing.

Section IV: Implement a Bagging SVM for three datasets by following the requirements

Part B: Load Data

- credit_risk_small_data
- credit_data_simulate.
- cybersecurity_data

load all of the above data files

```
import numpy as np
import pandas as pd
import warnings
warnings.filterwarnings('ignore')

credit_risk_small = pd.read_csv ('/content/drive/MyDrive/Baylor F22 - S23/2-Spring 23/Machine Learning/credit_risk_small_data.csv')

credit_data_simulate = pd.read_csv ('/content/drive/MyDrive/Baylor F22 - S23/2-Spring 23/Machine Learning/credit_data_simulate - credit_data_simulate (2).csv')

cybersecurity= pd.read_csv ('/content/drive/MyDrive/Baylor F22 - S23/2-Spring 23/Machine Learning/cybersecurity_data.csv')
```

Part B: Create Labels

take each label from each dataframe

```
[645] credit_risk_label = credit_risk_small[['Delinquency']]
      #credit_risk_small
```

```
[646] credit_simulate_label = credit_data_simulate[['Industry sector labels from 1-12']]

      #I am dropping this because it was dropped in HW#3 and I didn't want to confuse myself
      credit_data_simulate = credit_data_simulate.drop(columns = ['Credit status'])

      #credit_data_simulate
```

```
[647] cybersecurity_label = cybersecurity[['class']]
      #cybersecurity
```

Part B: Helper FUnctions

Train test split 80/20

```
[648] #---split into 80/20---#
      def new_split (data, labels, size):
        from sklearn.model_selection import train_test_split
        from sklearn.preprocessing import StandardScaler

        X = data#StandardScaler().fit_transform(data)

        train_data, test_data, train_data_label, test_data_label = train_test_split(X, labels, test_size = size, random_state = 42)

        return train_data, test_data, train_data_label, test_data_label
```

Part B: Helper FUnctions

new_SVM

> function for the second half of the project! (works the same but with adjustments)

```
[649] def new_SVM(train_data, train_label, test_data, name):
        #---conduct Regular SVM for iris data---#
        from sklearn import svm
        from sklearn.model_selection import cross_val_score
        import pandas as pd
        #import numpy as np

        #---split 80/20---#
        #train_data, test_data, train_label, test_label = new_split(data, label, 0.2)

        #---create an SVM instance---#
        clf = svm.SVC(kernel = 'rbf', gamma = 0.5, C = 1)

        #---fit data---#
        clf.fit(train_data, train_label)

        #---train data---#
        test_pred_label = clf.predict(test_data)

        print('Do SVM for ', name, ': done')
        print(test_pred_label, '\n\n')

        return test_pred_label
```

Section V:

1. Use first 80% data for training and the remaining 20% for test. Do SVM prediction

Part B: Q1

Part I:

> This first chunk of data called does a normal train/test split on each of the datasets for part b. The split is with an 80/20, train/test division.

> After the train & test data and train & test labels have been retreaved, the SVM function is called to predict the labels of the test data. The result SVM call for each of these datasets are shown below

Part B: Q1

- This is done two more times for credit_data_simulate and cybersecurity data

```
[650] ###########################
      #credit_risk_small
      #
      #---call the new_split function using 20/80 split---#
      cred_risk_train, cred_risk_test, cred_risk_train_label, cred_rist_test_label = new_split(credit_risk_small, credit_risk_label, 0.2)
      #
      #---call the new_SVM function---#
      cred_risk_pred_test_label = new_SVM(cred_risk_train, cred_risk_train_label, cred_risk_test, 'cred_risk_train')
      #
      ###########################
```

Part B: Q1

```
Do SVM for  cred_risk_train : done
[0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0]


Do SVM for  credit_data_simulate : done
[ 4  9  6 10  4  7  8  9  3  8  1  6 11 12  9  6  3 10  8  2  5  4 10  4
 12  1 10 10  1  2  7  2  4  3  6  7  1  5  9 12  5  8  5 11  9 12  5 10
  9  9  9  8  2  1  5  3  4  9  3  5  9  6  4 11  9  5  8  6 12  2  5 11
  5  1  4  1  8  3  3  1  5  8  4  7 10  1 11  8  4  3  1  3  1 12  2  8
  1 12  4  8 11  7  5 10  6  7  8  7 12  7  5  1  3  8  2  3 12  9  7 12
 11 10  5 11  6  6  3 12  2  1 12  2  6  3 10  5  7  3  6  5  1  9 12  8
 11  3  7  9 11  1 11  1 12  4 11  2 11  9  5  2  6  5  4  5  6  5  8  3
  2  6 10  3  9  7  2  9  6  5 10  9  8  1 12 11  4  9  3  1 12  2  5  1
  6 11  9  5  3  2 10  6  4  6 12 10  3  3  2  3 10 10 11  9 11 12  2  9
  8  4 10  3 11  7  1  3 12  7  9 11 11  7  7  3 12  8  3 10  3  3  2 11
  9  8  5  8  1  9 10 12  7  7  7 11  4  8  8 10  8  8  3 11  3 11  7  1
  7 12  9  2 11  4  7  9  6 10  7  4  9  2  5  7  6  5  3 11 12  7 11  4
  9  3  7  9  6 12 10  5 10 11  9  8  9  9  2  5  1 10  2  4  4 10  9  8
  8  4  3  8  3  6  5  7  3  2 12  3  2  7  1  8  7  3 10  7  3  5]


Do SVM for  cybersecurity : done
[-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 -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 -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 -1 -1 -1 -1 -1 -1
 -1 -1 -1 -1 -1 -1]
```

Section VI:
2. Randomly pick 1/2 data from training data to form three training datasets: training_1, training_2, and training_3

Part B: Q2 Helper Functions

Half_data

> geeksforgeeks: split dataset in half

> splits the dataset passed to it in half by dividing the shape of the object by 2.

[651]
```
#---I want to make sure I mix the data in the data frame as best as possible---#

def half_data(data, name):
  import pandas as pd

  mix_data = data#.sample(frac = 1).reset_index()

  #---find shape of dataframe and divide it by 2---#
  half_ind = int(mix_data.shape[0] / 2)

  #---half the data---#
  half_data = mix_data.iloc[:half_ind, :]

  print("randomly split ", name, ' train in half: done', '\n')
  #half_data
  return half_data
```

Part B: Q2 Helper FUnctions

mix_data

> randomizes the dataset passed to it by 're ordering' the rows of the data and then recounting them

0s [652]
```
def mix_data(data):
    return data.sample(frac = 1).reset_index()
```

Part B: Q2

Part II:

> Okay, so this next chunk of code calls to the mix_data function to mix up the rows of data inside each original dataset of part B.

> Then a new call to train/test split is conducted with the newly randomized dataset of part B. I will refer to the 'newly randomized datasets of part B' (credit_risk_data, credit_simulate_data, and cybersecurity_data) as X,Y and Z. The train & test data and train & test labels are captured for X, Y and Z.

> Lastly, the train data for X, Y and Z are each halved. **This means that the train data (80% of X, Y , or Z) is split in 2**. This was accomplished with a call to half_data.

> training_1 gets the output from X

> training_2 gets the output from Y

> training_3 gets the output from Z

Part B: Q2

- This is done two more times for credit_data_simulate and cybersecurity data

```
[653] ###########################
#credit_risk_small
#
#---mix the data to make it random---#
mixed_credit_risk = mix_data(credit_risk_small)
#
#---call train_test_split---#
cred_risk_train, cred_risk_test, cred_risk_train_label, cred_rist_test_label = new_split(mixed_credit_risk, credit_risk_label, 0.2)
#
#---halve the training data---#
half_cred_risk_train = pd.DataFrame(cred_risk_train)
#
training_1 = half_data(half_cred_risk_train, 'cred_risk_train')
#
###########################
```

Section VII:
3. run SVM using training_1, training_2, and training_3 to predict the test data: we can them SVM_1, SVM_2, and SVM_3

Part B: Q3

Part III:

> Okay, hang in there. This is where it gets confusing. You should already know that training_1, training_2, and training_3 were created above.

> Now, I want to get the labels off of those datasets here! I think it's pretty self explanitory what I did in the first bit

> The second bit calls to that good old 'new_SVM' function and calculates the SVM classification for training_1, training_2, and training_3 data.

> It is important to note again that (for example) trainin_1 refers to half of the training data from the mixed credit_risk dataset from part B.

> The results are shown below!

Part B: Q3

```
#---first, we want to capture the labels for all the training sets---#

training_1_lab = training_1[['Delinquency']]                         #references credit_risk

training_2_lab = training_2[['Industry sector labels from 1-12']]   #references credit_simulate

training_3_lab = training_3[['class']]                              #references cybersecurity


#---second, we call the SVM function for each training sets---#

SVM_1 = new_SVM(training_1, training_1_lab, cred_risk_test, 'cred_risk_test = training_1')

SVM_2 = new_SVM(training_2, training_2_lab, cred_sim_test, 'cred_sim_test = training_2')

SVM_3 = new_SVM(training_3, training_3_lab, cyber_test, 'cyber_test = training_3')
```

Part B: Q3

```
Do SVM for  cred_risk_test = training_1 : done
[0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0]


Do SVM for  cred_sim_test = training_2 : done
[ 6  6  6  6  6  6  5  6  6  6  6  6  6 11  6  6  6 10  6  6  1  1  6  6
  6  1  9  6 11  6  6  6  9  6  6  6  6  6  6  6  6  6  6  6  7  6  6  6
  6  6  6  6  6  6  6  5  6  6  6  6  3  6  1  8  6  6  3  6  3  6  2  6
  6  6  6  6  6  6  6  2  6  5  6  6  9  6  6  8  6  6  6  6  6 12  6  6
  6  6  6  6  6  6  6  6  6  6  6  6  6  4  6  6  6  6  9  6  6  6  7 11
  6  7  6  6  6  6  6  6  6  6 12 12  6  6  6  6  6  3  6  5  6  6  3  6
  2  6  6  3  5  6  4  6  6  6  6  4  6  9 11  4  5  5  6  6  6  6  6  6
  6  6  6  6  6  6  6  6  1  6  6  7  9  6  6  6  6  6  6  6  6  7  6  6
  6  6  6  7  6  6  6  9  6  6  6  6  6  8  6  7  6  6  6  6  6  6  6  6
  6  4  6  6  6  6  6  6  6  6  6  6  8  6  6  6  6  6  7  6  6  6  9  4
  6  6  2  6  5  6  3  6  6  6  6  6  6  6  5  6 10  6  6  6  6  1  6  6
  6  6  6  6  6  1  6  6  6  6  6  8  6  6  6  6  6  6  6  6  6  6  6  6
  6  6  6  6  6  6  6  6  6  6  6  1  6  6  6  6  2  6  6  6  6  6  6  6
  6  6  6  6  9  6  6  6  6  6  6  6  6  6 11  6  6  6  6  6  6  6]


Do SVM for  cyber_test = training_3 : done
[-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 -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 -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 -1 -1 -1 -1 -1 -1
 -1 -1 -1 -1 -1 -1]
```

Section VIII:

4. Determinate the final label for each test entry by doing the following voting (a) Pick the predicted label with maximum votes, say 1 (svm _1), -1 (svm _2), 1 (svm _3), then final label should be 1

Part B: Q4 Helper Functions

Bagging_SVM

> a function to calculate the bagging svm for a dataset

```
[655] def Bagging_SVM(X, y, test):
        from sklearn.svm import SVC
        from sklearn.ensemble import BaggingClassifier, VotingClassifier

        #X = X
        #y = training_1_lab

        clf = BaggingClassifier(estimator = SVC(), n_estimators = 1, random_state = None)#.fit(X, y)

        clf_predict_label = clf#.predict(test)

        #---voting---#

        return clf_predict_label
```

Part B: Q4 Helper FUnctions

PVoting

> a voting classification to vote the svm classification results presented by the bagging svm function

```
[656] def Voting(clf1, clf2, clf3, test, name):
          from sklearn.ensemble import VotingClassifier

          result_1 = VotingClassifier(estimators = [('SVM_1', clf1), ('SVM_2', clf2), ('SVM_3', clf3)])

          result_1 = result_1.fit(X, y)

          print("Voting on ", name, ": ")
          print(result_1.predict(test))
```

Part B: Q4

Part IV:

> Last portion (yay). Okay so this part is where I conduct a bagging svm and a voting. I will do a call to bagging svm 3 times for training_1. The same will be done for training_2 and training_3.

> Each classification I get back will be captured with each call to Bagging_SVM. Afterwards, I will pass all three classifications to the Voting function and the most common label for the test data will be printed below.

> Here is the call for training_1 (credit_risk)

Part B: Q4

Here is the call for training_1 (credit_risk)

```
X = training_1
y = training_1_lab

clf1 = Bagging_SVM(X, y, cred_risk_test)
clf2 = Bagging_SVM(X, y, cred_risk_test)
clf3 = Bagging_SVM(X, y, cred_risk_test)

Voting(clf1, clf2, clf3, cred_risk_test, 'cred_risk_test')
```

```
Voting on  cred_risk_test :
[0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0]
```

Part B: Q4

Here is the call for training_2 (credit_simulate)

```
[658] X = training_2
      y = training_2_lab

      #Bagging_SVM(X, y, cred_sim_test)

      clf1 = Bagging_SVM(X, y, cred_sim_test)
      clf2 = Bagging_SVM(X, y, cred_sim_test)
      clf3 = Bagging_SVM(X, y, cred_sim_test)

      Voting(clf1, clf2, clf3,cred_sim_test , 'cred_sim_test')
```

```
Voting on  cred_sim_test :
[1 9 9 1 1 9 1 3 1 1 3 3 1 3 3 9 9 3 3 9 1 1 3 1 1 9 3 9 1 1 3 1 3 9 3 9 9
 3 1 1 1 3 1 3 1 1 1 3 3 1 3 1 9 3 1 1 1 3 1 3 3 9 1 1 3 9 1 3 3 3 3 3 1 1
 1 3 3 3 9 1 1 1 9 3 1 3 9 1 9 1 1 3 3 1 9 1 3 3 1 1 3 9 1 9 1 9 9 1 3 3 3
 9 1 3 3 3 9 1 3 1 1 1 1 1 3 3 1 3 3 1 1 9 9 3 1 3 1 9 3 1 1 3 1 1 3 1 6 1
 3 3 3 3 3 1 1 9 1 1 1 1 1 3 3 1 3 3 1 1 3 3 1 3 1 1 1 1 1 9 3 1 9 1 3 3 3
 3 3 9 1 1 3 3 1 3 3 3 1 1 3 3 1 9 1 1 3 3 3 3 3 1 3 1 1 3 3 3 1 3 3 1 1 3
 3 3 1 9 1 1 1 9 1 1 1 1 3 1 3 3 1 3 1 3 3 3 1 1 1 1 1 1 3 3 1 1 1 3 3 9 1
 9 9 3 3 9 9 1 1 3 1 1 1 1 9 1 3 1 9 9 1 3 9 1 1 9 9 9 1 1 9 1 3 1 3 9 1 3
 3 1 3 1 1 1 3 3 1 1 1 3 1 9 3 1 1 1 1 1 1 1 1 9 1 1 9 1 1 9 1 1 1 1 3 1 3
 3]
```

Part B: Q4

Here is the call for training_3 (cybersecurity)

```
[659] X = training_3
      y = training_3_lab

      #Bagging_SVM(X, y, cyber_test)

      clf1 = Bagging_SVM(X, y, cyber_test)
      clf2 = Bagging_SVM(X, y, cyber_test)
      clf3 = Bagging_SVM(X, y, cyber_test)

      Voting(clf1, clf2, clf3, cyber_test, 'cyber_test')
```

```
Voting on  cyber_test :
[-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 -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 -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 -1 -1 -1 -1 -1 -1
 -1 -1 -1 -1 -1 -1]
```

Section IX:
5. Compare the performance of SVM and the Bagging-SVM for the two datasets and draw your conclusion

Part B: Q5 Results

```
Do SVM for  cred_risk_test = training_1 : done
[0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0]
```

```
Voting on  cred_risk_test :
[0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0]
```

Part B: Q5 Explanation

Result #1:

> from looking at the SVM on cred_risk_test, it is clear that using voting on the bagging classifier showed that the labels for all of the test data for this dataset were most likely going to be '0'. It is very faint to see it in this top and bottom comparison, but some indices in the top array have 1s in them. This shows that from a number of bagging svm calls with this sample, the most common label results will be 0.Ultimatley, the voting SVM looks to be more accuracte.

Part B: Q5 Results

```
Do SVM for  cred_sim_test = training_2 : done
[ 2  6  8 10  6  1  8 11  7  8  5  2  6  1  2  2  7  1 12  2  6  2  6  2
  7  2  5  2  4  2 11  5  2  6  6  3 12  2  6  5  2  2  2 11  2  2 11 12
  2  2  2  2  2 12  8 12  7  7  7  6  2  2  2  3  2  7  3 12 12  2  8  2
  3  8  1 12  5  8  2  9  4  6  2  9  2  3  4 12  2  2  2  8  2  1  2  8
 12  2  2  2  7  2  3  6  2  6  8  2  2  7  2  2  6  8 10  2  2  2 11 10
  7  2  2  5  2  5  2  2  2  2  2  2  1  2  1  2  9  6  5  2 10 11  2  2
  1  2  2 11  2  9  7  5  2 11  3  2  2 12  7  5  2  9  2  6  3  3  2  4
  2  1  2 12 10  2  2  2  3 10 12  4  2  2 11  5 11 11 11  2 10 11  2 11
 11  2  8  7 10  2  2  3  2  2  7  2  2  7  2  2  2  3  2 11  2  2  2  2
  2  2  2  2 10  2  6  2  3  1 12  9  2  5  2  4  3  2  2  2  2  2  8  2
  9  2  4  5  2  8  2  8  2  8  9  6  1  2  2  2  2  9  7 11  1 12  3  6
  2  3  2  9 11 10  2  1  5  2  2  4  2 12  2  6  2  2  2 12  2  2  2  4
  3  1  8  1  2  4  2  7  6  2  2  3 11  5 12  6  3  3  1  2  5  2  6  2
  6  5  3  2  2  2  9  2  5 12  2 11  9  9  2  2  2  8  2  2  3  5]

Voting on  cred_sim_test :
[3 3 3 3 6 2 3 3 3 3 3 3 3 3 3 2 6 5 2 3 2 3 5 4 3 3 4 3 3 5 3 3 3 3 3 3 3
 3 3 3 5 3 5 5 4 3 3 2 4 3 3 4 5 4 5 3 2 3 5 6 5 2 5 3 3 3 2 3 3 5 3 5 3 3
 5 5 3 3 6 3 5 3 3 3 3 3 4 3 3 5 2 3 3 6 3 4 3 3 3 5 5 5 2 6 2 3 3 5 3 5 4
 3 2 3 2 3 5 5 3 5 3 5 4 2 2 2 3 3 3 5 3 3 3 3 5 4 3 5 3 6 2 3 2 3 3 3 2 6
 6 2 3 2 5 3 3 3 2 5 2 3 5 3 5 3 6 3 3 2 3 4 2 2 5 2 3 5 3 2 3 5 5 3 5 3 3
 3 2 6 5 3 6 3 5 3 3 4 2 2 2 3 3 3 5 3 5 6 3 3 3 5 2 6 3 4 5 5 2 5 5 4 2 4
 3 5 3 3 3 3 2 2 3 3 2 3 5 3 2 2 3 3 3 3 4 5 3 5 3 5 2 6 5 3 3 3 3 3 3 2 6
 3 3 3 5 6 3 5 3 3 5 3 3 2 3 3 3 3 5 3 5 2 5 3 3 3 2 4 3 5 5 3 3 3 2 3 3 5
 3 4 3 2 3 3 3 3 3 5 5 3 3 5 3 5 3 3 3 2 4 3 3 3 5 3 2 3 3 5 3 2 5 3 5 3 4
 2]
```

Part B: Q5 Explanation

Result #2:

> The results of the voting on the SVM bagging classifier are much different in the credit_sim_test. There's a lot to take in here but it should be known that when the SVM function is run multiple times, the labels themselves do change a lot. The results of the voting classifier shows calmness over the labels presented in the SVM. Ultimately, I would trust the label results of the voting on bagging svm classification more.

Part B: Q5 Results

```
Do SVM for  cyber_test = training_3 : done
[-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 -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 -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 -1 -1 -1 -1 -1 -1
 -1 -1 -1 -1 -1 -1]


Voting on  cyber_test :
[-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 -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 -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 -1 -1 -1 -1 -1 -1
 -1 -1 -1 -1 -1 -1]
```

Part B: Q5 Explanation

Result #3:

> The results of the voting classifier with the cybersecurity dataset are pretty spot on to the results coming from the SVM classifier. This means that both have very good accuracy when determining the labels of the sampled cyber_test_data. Ultimately, I would rely on either or, but I would spare myself the work by just using the SVM.

Thank you!