Homework 3 - Welcome to the world of variable stars!

Modified from (Viviana Acquaviva (2023)) License: BSD-3-clause

In [ ]:
```
from sklearn.model_selection import train_test_split
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import accuracy_score
from sklearn.preprocessing import StandardScaler
from sklearn.tree import DecisionTreeClassifier
from sklearn.tree import plot_tree
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
plt.style.use("bmh")
```

Data description

The data we use for this homework are about stars. Our features are the so-called "colors", which give an indication of whether a star emits more blue, green, yellow, or red light. We are trying to predict whether a star is a special type of star called a RR-Lyrae variable star. So our target values will be yes/no (1/0 in the language of numpy arrays).

1. Load the data for features and target from the appropriate files and save them as numpy arrays. (note: the default delimiter won't work, these are comma separated values!)

In [ ]:
```
features = pd.read_csv("RRLyrae_features_small.txt", header=None)
targets = pd.read_csv("RRLyrae_labels_small.txt", header=None)

features_np = features.to_numpy
targets_np = features.to_numpy

print(sum(targets.values))
print(1 - sum(targets.values)/ len(targets.values))
features
```

```
[483]
[0.80547725]
```

Out[ ]:

	0	1	2	3
0	0.311	0.714	0.093	0.204
1	0.332	0.965	0.122	0.032
2	0.394	1.019	0.147	0.096
3	0.285	0.865	0.120	0.039
4	0.386	0.977	0.181	0.080
...	...	...	...	...
2478	0.059	0.963	-0.026	-0.025
2479	0.185	1.060	0.051	-0.024
2480	0.212	1.044	0.035	0.002
2481	0.172	1.065	0.042	0.003
2482	0.065	1.126	-0.017	-0.058

2483 rows × 4 columns

2. Answer the following questions:

Is this a classification or regression problem?

- Classification

Is this supervised or unsupervised learning?

- Supervised

How many instances are in this data set?

- 2483

How many features?

- 4

How many RR Lyrae stars (i.e., examples of the positive class) are in the data set?

- 483

What would be the accuracy of a classifier that classifies all objects in the data set as non-RR Lyrae?

- 80.5%

3. Use a Decision Tree Classifier, and implement k-fold cross validation algorithm.

In [ ]:
```
from sklearn.model_selection import StratifiedKFold, cross_val_score, cross_val_predic
from sklearn.pipeline import Pipeline
```

```
from sklearn.preprocessing import RobustScaler

pipe_dt = Pipeline([('scaler', RobustScaler()), ('estimator', DecisionTreeClassifier(r
cv = StratifiedKFold(shuffle=True, n_splits=10, random_state=42)
```

4. Report the scores, and calculate the mean and standard deviation of the scores vector.

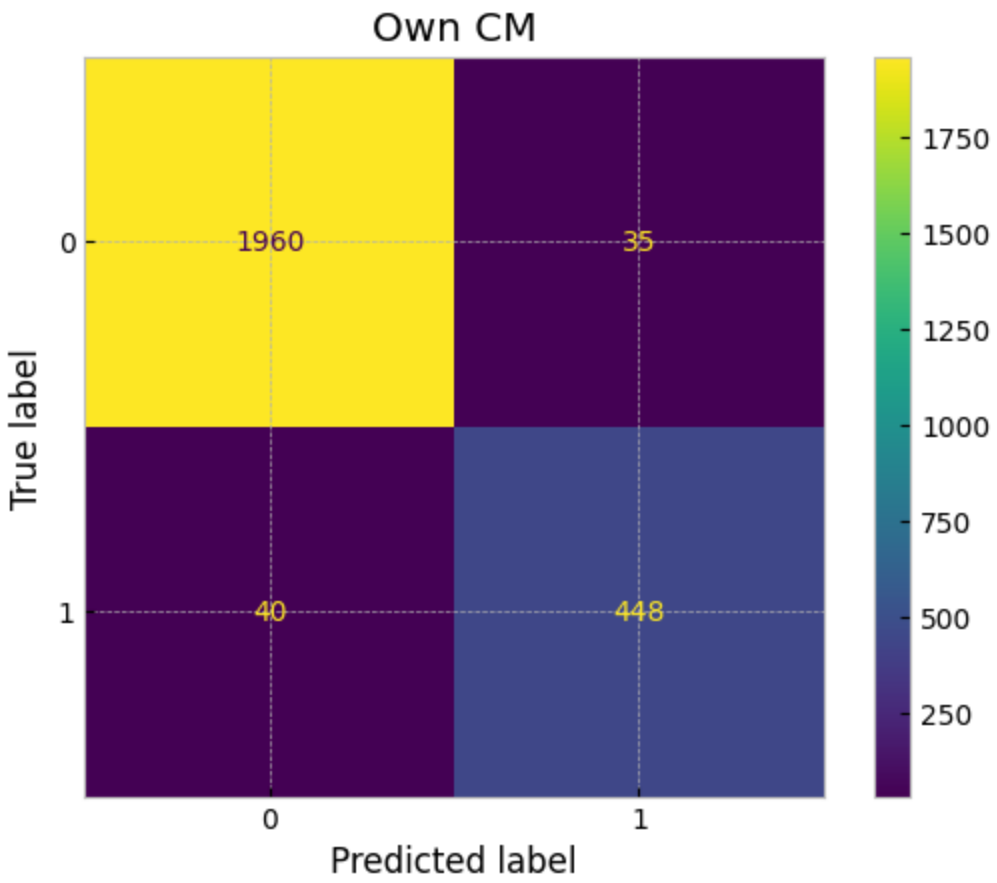
In [ ]:
```
print("\n-----DT----\n")
accuracies = cross_val_score(pipe_dt, features, targets, cv = cv, scoring='accuracy')
precisions = cross_val_score(pipe_dt, features, targets, cv = cv, scoring='precision')
recalls = cross_val_score(pipe_dt, features, targets, cv = cv, scoring='recall')

print("Accuracy:", np.average(accuracies), "+-", np.std(accuracies))
print("Precision:", np.average(precisions), "+-", np.std(precisions))
print("Recall:", np.average(recalls), "+-", np.std(recalls))
```

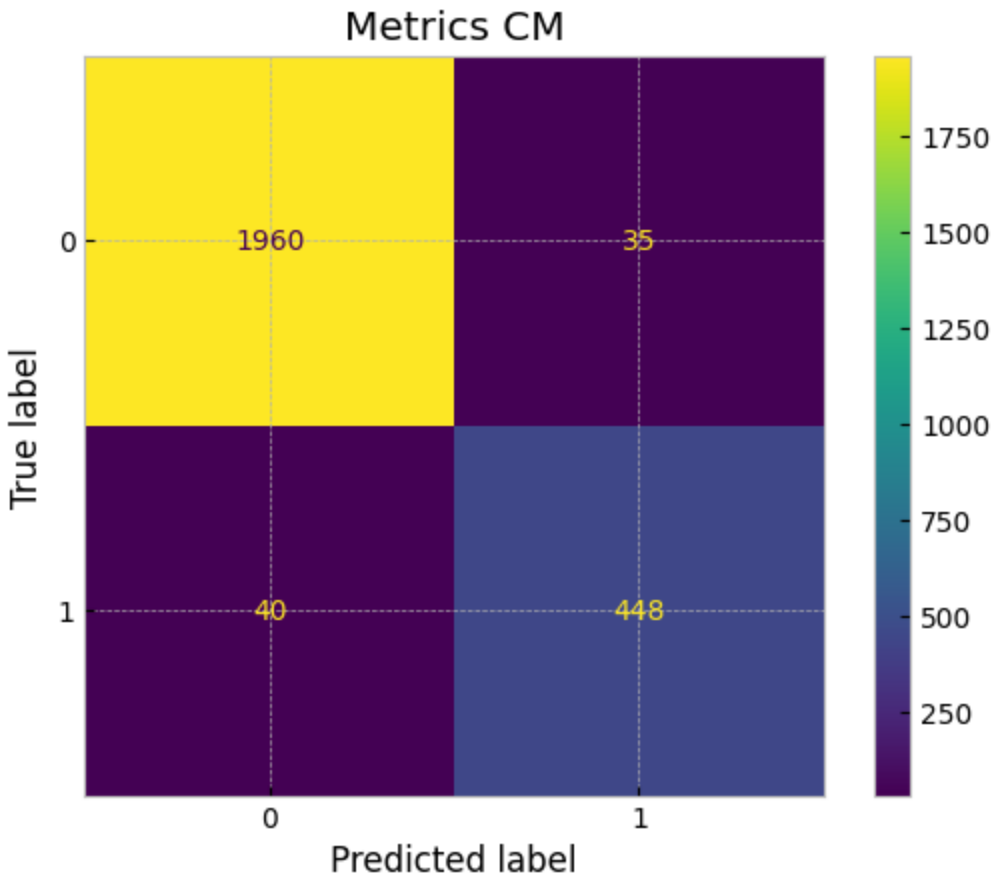
```
-----DT----

Accuracy: 0.9697969296541002 +- 0.011571785839174804
Precision: 0.9185602992819234 +- 0.030581257217073228
Recall: 0.9274659863945578 +- 0.04198355824976706
```

5. As in lab 5-6, compute the confusion matrix for your model. To generate the predictions, you can use the "cross_val_predict" function. Please write your own code before using any scikit learn builtin functions, but you can use ConfusionMatrixDisplay to visualize.

In [ ]:
```
from sklearn.metrics import ConfusionMatrixDisplay, confusion_matrix

pred_dt = cross_val_predict(pipe_dt, features, targets, cv = cv)

nn = 0
npo = 0
pn = 0
pp = 0
y = targets.values

for i in range(len(y)):
    if pred_dt[i] and y[i]:
        pp += 1
    elif ~pred_dt[i] and y[i]:
        npo += 1
    elif pred_dt[i] and ~y[i]:
        pn += 1
    elif ~pred_dt[i] and ~y[i]:
        nn += 1
cm_own = np.array([[nn, npo], [pn, pp]])
cm = confusion_matrix(pred_dt, targets)

disp = ConfusionMatrixDisplay(cm_own)
disp.plot()
plt.title("Own CM")

disp = ConfusionMatrixDisplay(cm)
disp.plot()
plt.title("Metrics CM")
```

Out[ ]: Text(0.5, 1.0, 'Metrics CM')





6. Based on the confusion matrix, how many true positive, true negative, false positive, false negative instances do you have?

TP: 448

TN: 1960

FP: 35

FN: 40

7. Calculate accuracy, precision and recall. Plese write out your steps and do not use built-in functions.

In [ ]:
```
tp = 448
tn = 1960
fp = 35
fn = 40

accuracy = (tp + tn)/ (tp + tn + fn + fp)

recall = tp / (tp+fn)

precision = tp/ (tp+ fp)

print("Accuracy", accuracy)
print("Recall", recall)
print("Precision", precision)
```

```
Accuracy 0.9697946033024567
Recall 0.9180327868852459
Precision 0.927536231884058
```

8. Which evaluation metric is used by the cross validation score in 4.? [To answer this question, you might need to check out the description of the function]. Given the distribution of classes in your data set, do you see a possible issue?

I specified the scores to get all three metrics. The default is the estimators default scoring. In the DecisionTree case the default is accuracy. The data set is heavily weighted to the negative class (5:1). This causes the accuracy score to become inflated.

9. To compare performance for now, let's use the F1 score, a weighted average of precision and recall.

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

Find out how to use the f1 score (instead of accuracy) as an optional argument of the cross_validate function and report the mean and standard deviation of the scores associated to this evaluation metric.

In [ ]:
```
F1 = 2 * (precision * recall) / (precision + recall)
```

```
f1s = cross_val_score(pipe_dt, features, targets, cv = cv, scoring='f1')
print("F1:", np.average(f1s), "+-", np.std(f1s))
print("Own F1", F1)
```

```
F1: 0.9225390045162996 +- 0.03105419400154771
Own F1 0.9227600411946447
```

10. It is now time to look at some diagnostics. Use the cross_validate function with 'return_train_score = True'. Compare the f1 train and test scores obtained by your model. Based on this result, do you think your algorithm suffers from high variance or high bias and why?

In [ ]:
```
vals = cross_validate(pipe_dt, features, targets, cv=cv, scoring='f1', return_train_sc

print("Test Score", "\nmean", np.average(vals['test_score']), "\nDeviation", np.std(va
print("Train Score", "\nmean", np.average(vals['train_score']), "\nDeviation", np.std(
```

```
Test Score
mean 0.9225390045162996
Deviation 0.03105419400154771
Train Score
mean 1.0
Deviation 0.0
```

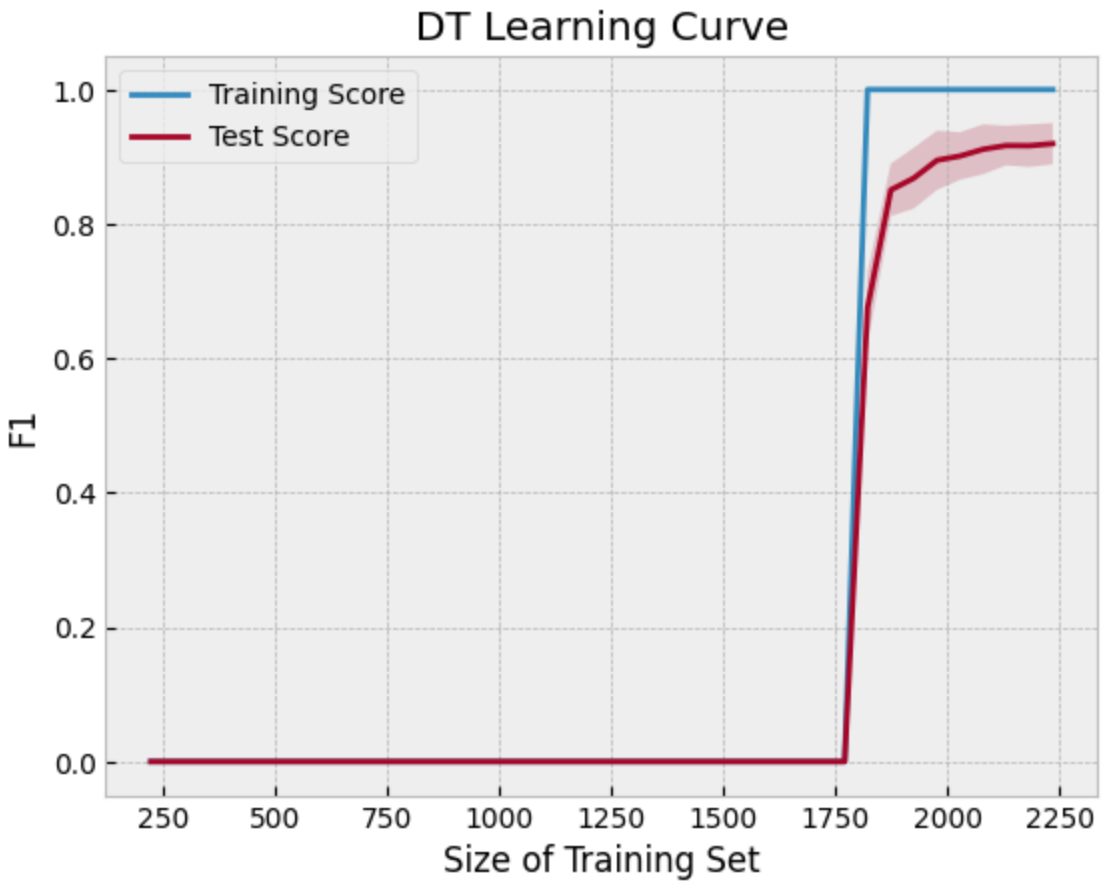
The algorithm suffers from high variance. It has adapted too well to the training data with a 100% f1 score. It has likely modeled in some random noise in the training set which doesn't let it adapt as well to the test set with a f1 score of 92.3%,

11. Finally, we can look at learning curves as in lab 5-6. Based on what you see in the plot, do you think getting more training data would help? Justify your answer.

In [ ]:
```
from sklearn.model_selection import learning_curve

size, train_score, test_score = learning_curve(pipe_dt, features, targets, train_sizes
train_score_mean = np.mean(train_score, axis=1)
test_score_mean = np.mean(test_score, axis=1)
train_score_std = np.std(train_score, axis=1)
test_score_std = np.std(test_score, axis=1)
```

In [ ]:
```
plt.plot(size, train_score_mean, label="Training Score")
plt.plot(size, test_score_mean, label="Test Score")
plt.fill_between(size, train_score_mean-train_score_std, train_score_mean+train_score_
plt.fill_between(size, test_score_mean-test_score_std, test_score_mean+test_score_std,
plt.title("DT Learning Curve")
plt.ylabel("F1")
plt.xlabel("Size of Training Set")
plt.legend()
```

Out[ ]:
```
<matplotlib.legend.Legend at 0x21964578c40>
```



We can see that after a 1750 samples the training set performs perfectly. The test set lags behind but continues to improve. The improvement is starting stagnate which means adding more data will likely allow for marginal gains in the prediction quality.