Project #2

CIS 2541 - Prof. John P. Baugh – Oakland Community College – OR

Objectives

- To practice and learn more about Support Vector Machines (SVMs)
    - In particular, classification and classifiers
- The primary objective of this project is to build an ML using SVM to predict whether a tumor is malignant or benign based on various features

Instructions

Overview of Technologies

- The following libraries are crucial
    - Numpy
    - Pandas
    - Matplotlib
    - Scikit-Learn
    - Seaborn

Overview of Dataset (The Breast Cancer Wisconsin Dataset)

The ***Breast Cancer Wisconsin*** dataset contains 569 samples of malignant and benign tumor cells. Each sample has *30 features* that describe characteristics of the cell nuclei present in the image. The features are numerical and include attributes such as the radius, texture, perimeter, area, smoothness, compactness, concavity, and symmetry of the cell nuclei.

What you should do

This is a general summary of what you should do for this project.

1. Load the Breast Cancer Wisconsin dataset
    a. Hint: datasets.load_breast_cancer()
    b. Remember to split the data into X and y (the data and the target values)

2. Convert the data to a DataFrame
    a. Hint: data.feature_names

3. Do some EDA (Exploratory Data Analysis
    a. Print the head of the first few rows in the dataset
    b. Print the summary statistics of the dataset (Hint: *describe*)
    c. Print out the number of missing values (Hint: isnull().sum())

4. Visualize the distribution of the target variable
    a. Use a **Seaborn** heatmap
        i. Hint: df.corr(), annot=False, cmap="coolwarm"
    b. Make sure to add a title and to call show on the plotting

5. Create a standard scaler to standardize the features
    a. Hint: scaler.fit_transform(X)

6. Split the Data into Training and Testing Sets
    a. Hint: train_test_split is a handy function from sklearn.model_selection
    b. Use a test_size of 0.2 (that is, 20%)
    c. Use a random_state=42
    d. **Print out** the train features shape, and the test features shape

7. Build and Evaluate the SVM model
    a. Create each of the following SVM models (separate variables for each)
        i. SVC with **linear** kernel
        ii. SVC with **RBG** kernel
        iii. SVC with **polynomial** kernel
    b. Make sure to **fit** the model to the **training data**, and then **predict** on the **test data**
        i. Do this for ***each*** of the models with the different kernel types
    c. Print out both the accuracy and f1 scores for each model
        i. Hint (accuracy): accuracy_score()
        ii. Hint (F1): f1_score()

8. Create a confusion matrix heatmap using code such as the following:

```
sns.heatmap(confusion_matrix(y_test, y_pred_linear), annot=True,
fmt='d', cmap='Blues')
```

```
plt.title('Confusion Matrix for Linear Kernel')
plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.show()
```

Sample Outputs of some of the tasks

These are examples of the type of outputs you might expect from many of the tasks.

Displaying the first and last few rows of the dataset

Out[7]:

	mean radius	mean texture	mean perimeter	mean area	mean smoothness	mean compactness	mean concavity	mean concave points	mean symmetry	mean fractal dimension	...	worst texture	worst perimeter	worst area	worst smoothness	compa
0	17.99	10.38	122.80	1001.0	0.11840	0.27760	0.3001	0.14710	0.2419	0.07871	...	17.33	184.60	2019.0	0.1622	
1	20.57	17.77	132.90	1326.0	0.08474	0.07864	0.0869	0.07017	0.1812	0.05667	...	23.41	158.80	1956.0	0.1238	
2	19.69	21.25	130.00	1203.0	0.10960	0.15990	0.1974	0.12790	0.2069	0.05999	...	25.53	152.50	1709.0	0.1444	
3	11.42	20.38	77.58	386.1	0.14250	0.28390	0.2414	0.10520	0.2597	0.09744	...	26.50	98.87	567.7	0.2098	
4	20.29	14.34	135.10	1297.0	0.10030	0.13280	0.1980	0.10430	0.1809	0.05883	...	16.67	152.20	1575.0	0.1374	

5 rows × 31 columns

Displaying Summary Statistics

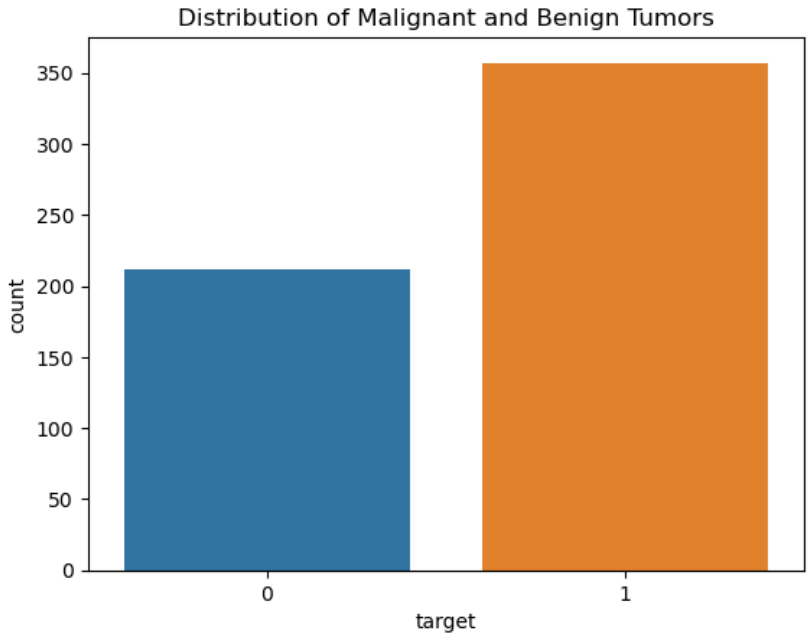
Out[8]:

	mean radius	mean texture	mean perimeter	mean area	mean smoothness	mean compactness	mean concavity	mean concave points	mean symmetry	mean fractal dimension	...	worst texture	worst perimeter
count	569.000000	569.000000	569.000000	569.000000	569.000000	569.000000	569.000000	569.000000	569.000000	569.000000	...	569.000000	569.000000
mean	14.127292	19.289649	91.969033	654.889104	0.096360	0.104341	0.088799	0.048919	0.181162	0.062798	...	25.677223	107.261213
std	3.524049	4.301036	24.298981	351.914129	0.014064	0.052813	0.079720	0.038803	0.027414	0.007060	...	6.146258	33.602542
min	6.981000	9.710000	43.790000	143.500000	0.052630	0.019380	0.000000	0.000000	0.106000	0.049960	...	12.020000	50.410000
25%	11.700000	16.170000	75.170000	420.300000	0.086370	0.064920	0.029560	0.020310	0.161900	0.057700	...	21.080000	84.110000
50%	13.370000	18.840000	86.240000	551.100000	0.095870	0.092630	0.061540	0.033500	0.179200	0.061540	...	25.410000	97.660000
75%	15.780000	21.800000	104.100000	782.700000	0.105300	0.130400	0.130700	0.074000	0.195700	0.066120	...	29.720000	125.400000
max	28.110000	39.280000	188.500000	2501.000000	0.163400	0.345400	0.426800	0.201200	0.304000	0.097440	...	49.540000	251.200000

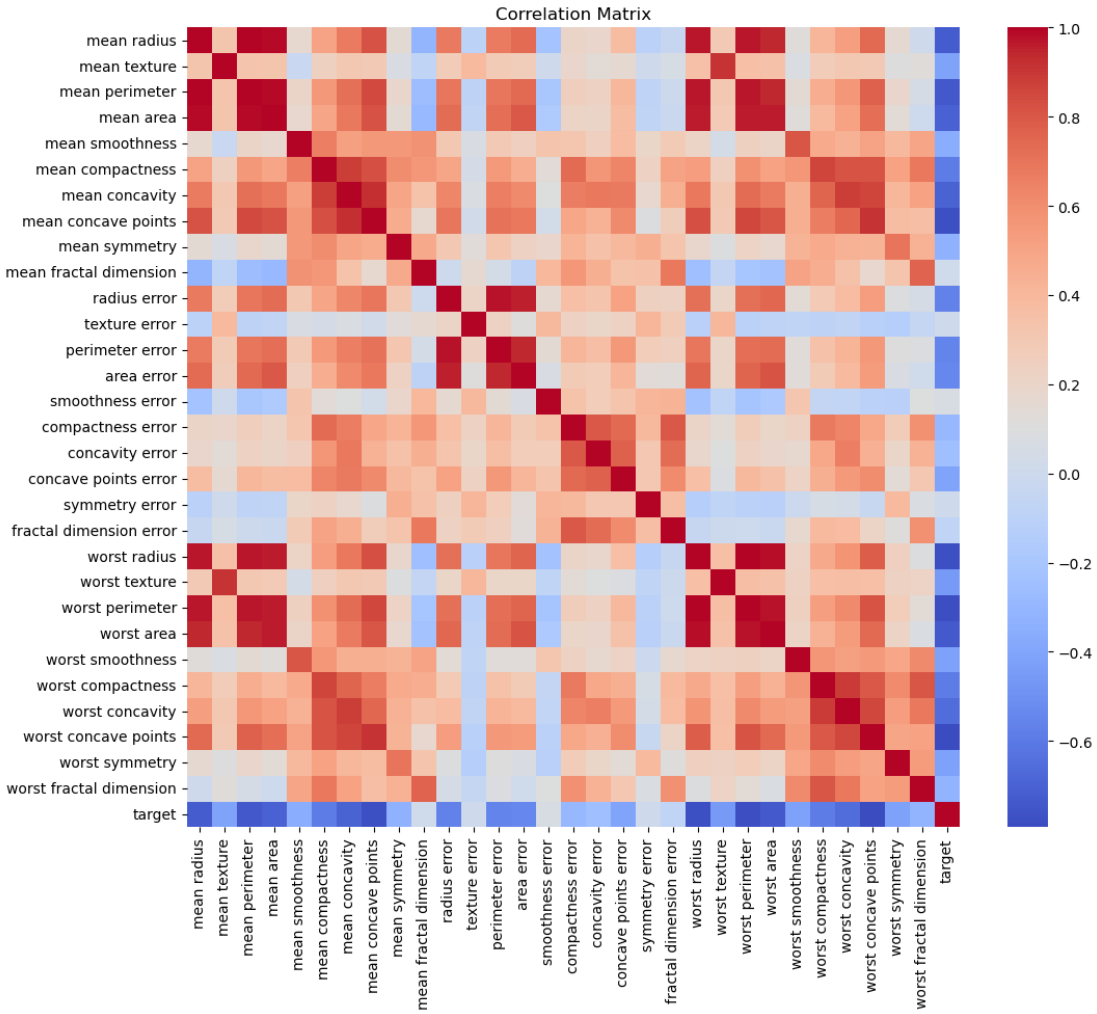
Missing Value Check

```
mean radius                0
mean texture               0
mean perimeter             0
mean area                  0
mean smoothness            0
mean compactness           0
mean concavity             0
mean concave points        0
mean symmetry              0
mean fractal dimension     0
radius error               0
texture error              0
perimeter error            0
area error                 0
smoothness error           0
compactness error          0
concavity error            0
concave points error       0
symmetry error             0
fractal dimension error    0
worst radius               0
worst texture              0
worst perimeter            0
worst area                 0
worst smoothness           0
worst compactness          0
worst concavity            0
worst concave points       0
worst symmetry             0
worst fractal dimension    0
target                     0
dtype: int64
```

Visualizations of the Distribution of the target variables



Visualizations of the Correlation Matrix



Accuracy, F1 Scores, and Classification report with confusion matrix

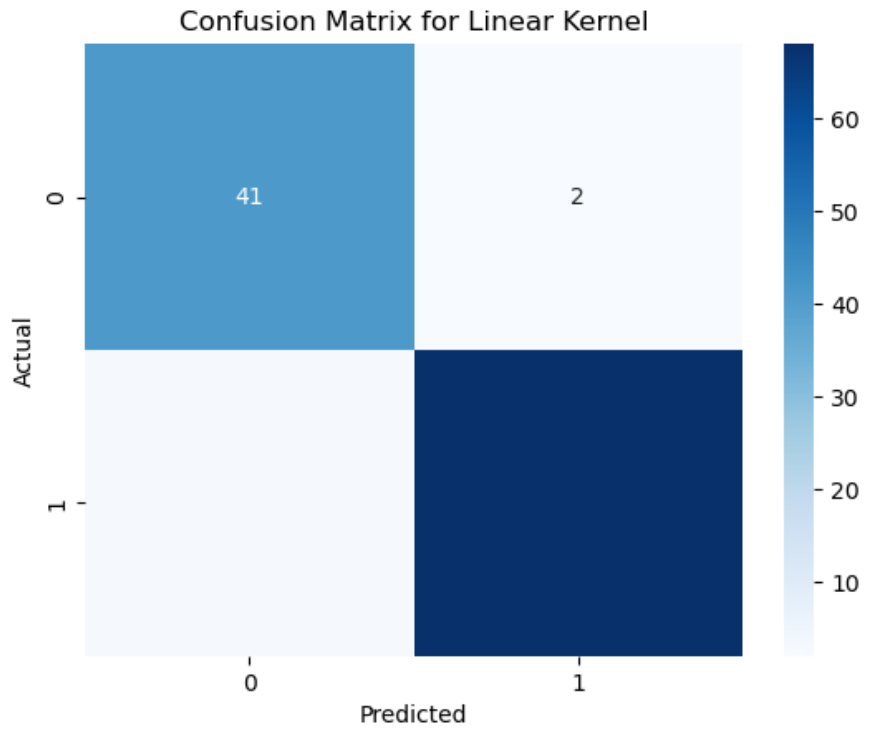
```
Linear Kernel Accuracy: 0.956140350877193
Linear Kernel F1 Score: 0.9645390070921985
RBF Kernel Accuracy: 0.9736842105263158
RBF Kernel F1 Score: 0.979020979020979
Polynomial Kernel Accuracy: 0.868421052631579
Polynomial Kernel F1 Score: 0.9044585987261146
Classification Report for Linear Kernel:
              precision    recall  f1-score   support

           0       0.93      0.95      0.94        43
           1       0.97      0.96      0.96        71

    accuracy                           0.96       114
   macro avg       0.95      0.96      0.95       114
weighted avg       0.96      0.96      0.96       114

Confusion Matrix for Linear Kernel:
 [[41  2]
 [ 3 68]]
```

Confusion Matrix Heatmap



Deliverables

- Turn in a zip including your source code and screenshots of the program functioning, as follows:
    - An **ipynb** file for Jupyter Notebooks
        - Alternatively, a **py** source file is acceptable as well
    - **Include screenshots of your program working,** placed inside the zip file that you turn in
        - This should include screenshots of the outputs including diagrams and printing of the shapes, evaluation metrics, etc.