SUPPORT VECTOR MACHINE WITH NONLINEAR CLASSIFICATION

NAME: CHENNURU HARISH

ID:23024881

INTRODUCTION:

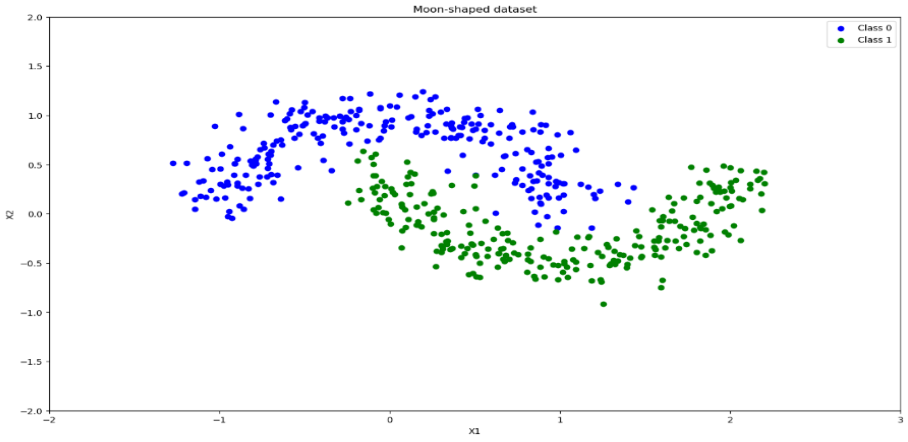
- The Support Vector Machine (SVM) classifier to the non-linear Moon dataset and evaluate its performance. The Moon dataset is a synthetic dataset that contains two interlocking crescent-shaped classes, making it a challenging problem for linear classifiers. Here, we will visualize the decision boundary, margin, and support vectors while evaluating the model's performance using various metrics.

DATA DESCRIPTION:

- Using the Moon dataset, a non-linear toy dataset. Use the given code snippet to create the training and testing sets, and feel free to modify the number of samples or the noise level in the dataset. A `plots () ` function is also available to help visualize the decision boundary, margin, and support vectors within a 2D feature space. The `plots () ` function is fully functional out of the box and is the most effective way to visualize and assess the performance of your model. It assumes that the classifier contains an instance variable named `self. Support vectors_` holds a NumPy array of the support vectors identified during training.

DATA PROCESSING:

- Imports make moons from the sklearn dataset to generate a non-linear dataset with two crescent-shaped clusters.
- Sets samples to 500 for the number of data points, noise to 0.15 for some randomness, and random state to 49 for reproducibility.
- Converts labels from {0, 1} to {-1, 1} for some models that might prefer this range.
- Splits the data using an 80/20 ratio (80% training, 20% testing) and sets the random state to 42 for consistent splitting across runs.



- **Visualization Function:**
- Defines a function plot that takes a trained SVM classifier, features (X), labels (y), and optional axes as input.
- Creates a mesh grid representing the feature space.
- Predicts labels and decision function values for each point in the mesh grid.
- Uses plots to visualize:
- Data points (colored by class).
- Support vectors (red circles).
- Decision boundary with margins (filled contours).
- Overall, the code is well-structured and provides the necessary components for training and visualizing a Support Vector Machine (SVM).

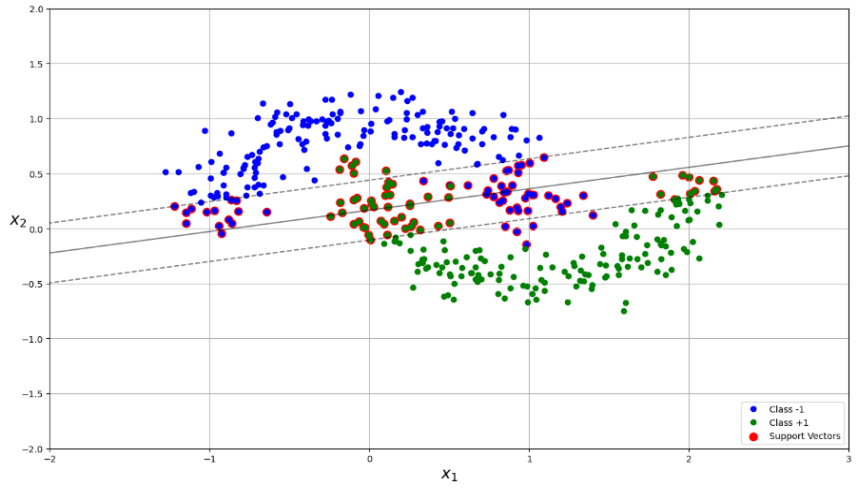
TRAIN FOR SVM:

- By training an SVM on the training data and evaluating its performance on the test data. Use three kernels for the SVM: Linear, Polynomial, and RBF. You should tune each model using a grid search or similar hyperparameter selection process and report the best hyperparameters found. You will use these same hyperparameter settings later when testing and comparing them to your implementation.
- Confusion matrix, Recall, and Precision. If applicable, discuss any tuning process on C and/or gamma to get a reasonable result.
- The tune and Test function automates hyperparameter tuning and evaluation for a given model. It uses Randomized Search CV to search for the best parameter combination over 50 iterations with 5-fold cross-validation, then outputs the optimal parameters. The best model is evaluated on the test set, with metrics like confusion matrix, recall, and precision displayed. Finally, the decision boundary of the model is visualized with plots, providing a clear view of how it separates classes.

```
from sklearn.svm import SVC
from sklearn.pipeline import Pipeline
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import RandomizedSearchCV
from sklearn.metrics import confusion_matrix, precision_score, recall_score
from scipy.stats import randint, uniform
import numpy as np


def tuneAndTest(model, params):
  rnd_search = RandomizedSearchCV(model, param_distributions =params, n_iter = 50, cv = 5, random_state=40
  rnd_search.fit(X_train, y_train)
  print("best hyper-parameter value: ", rnd_search.best_params_)
  bestmodel = rnd_search.best_estimator_

  #predict
  prediction = bestmodel.predict(X_test)
  # print("model:", str(bestmodel))
  print("confusion matrix:", confusion_matrix(y_test, prediction))
  print("recall:",recall_score(y_test, prediction))
  print("precision:",precision_score(y_test, prediction))
  plot_svm(bestmodel, X_train, y_train)
  print("\n")
```

Linear: This is the simplest kernel, represented by the dot product of two vectors u and v. It is suitable for linearly separable data, as it does not transform the data but directly computes similarity in the original input space.

```
tuneAndTest(SVC(kernel="linear"),{"C": uniform(0,20)})

best hyper-parameter value:  {'C': 8.153740561605723}
confusion matrix: [[34 11]
 [ 5 50]]
recall: 0.9090909090909091
precision: 0.819672131147541
```



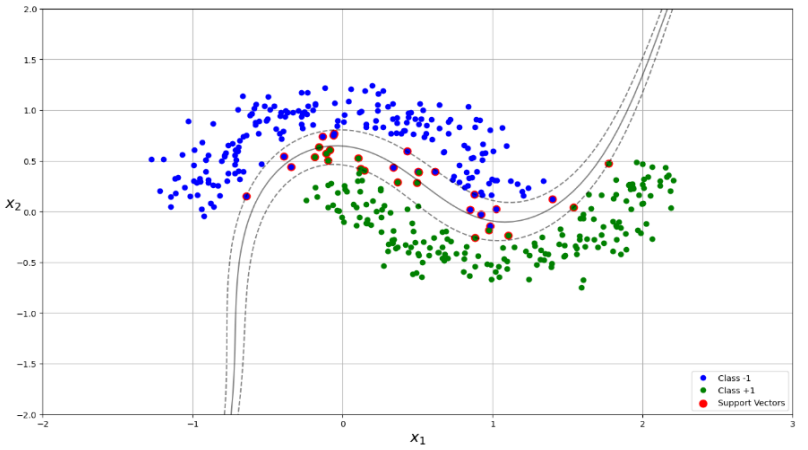
Polynomial:

- The function tune And Test (SVC (kernel="poly"), {"coef0": rand int (low=0, high=5), "C": uniform (0,20), "degree": rand int (low=2, high=5), "gamma": uniform(0, 1)})
- Perform hyperparameter tuning on a Support Vector Classifier with a polynomial kernel. It uses a Randomized Search CV to find the best values for coef0, C, degree, and gamma by sampling from specified ranges. The goal is to optimize and evaluate the model's performance using metrics like confusion matrix, precision, and recall while also visualizing the decision boundary**.**
- This kernel computes the dot product of u and v raised to the power p. The parameter p controls the polynomial degree, allowing for more complex decision boundaries. It maps the input data into a higher-dimensional space to handle non-linear relationships.

LIMITATIONS OF POLYNOMIAL:

- Adding polynomial features can work great with all sorts of ML algorithms, but it has some limitations:
- Low polynomial degrees → cannot deal with highly complex datasets
- High polynomial degrees → create a huge number of features, slow and possibly overfit
- SVM leverages a kernel trick to implicitly map data into higher-dimensional spaces, achieving similar results to manually adding polynomial features.

```
tuneAndTest(SVC(kernel="poly"),{"coef0":randint(low=0, high=5),"C": uniform(0,20),"degree": randint(low=2, high=5), "gamma":u
best hyper-parameter value:  {'C': 7.826529617851785, 'coef0': 3, 'degree': 3, 'gamma': 0.6095896671142067}
confusion matrix: [[42  3]
 [ 0 55]]
recall: 1.0
precision: 0.9482758620689655
```



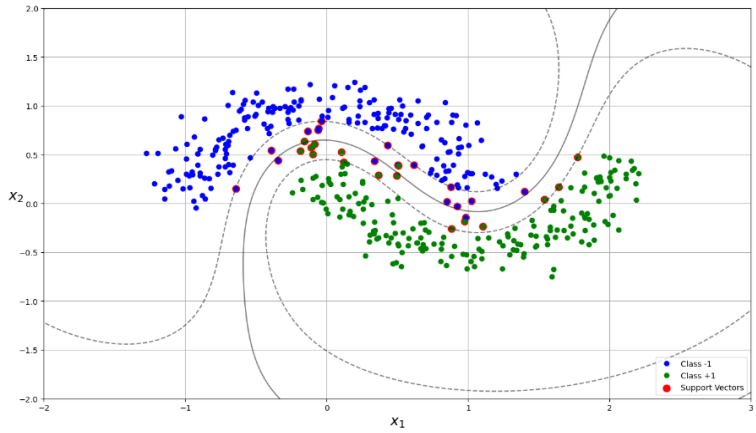
Radial Basis Function:

- The `tune and` function fine-tunes and assesses an SVM model with an RBF (Radial Basis Function) kernel. It adjusts the hyperparameters C and gamma. The C parameter manages the balance between minimizing training error and keeping the model simple to prevent overfitting.
- The gamma parameter defines the influence of a single training example, with higher values indicating a tighter influence. The Randomized Search CV method explores combinations of these parameters from specified distributions C uniformly distributed between 0 and 20, and gamma uniformly distributed between 0 and 1 over multiple iterations to identify the best-performing hyperparameter configuration. Once tuned, the best model is tested on unseen data, and its performance is evaluated using metrics like confusion matrix, precision, and recall.
- This kernel computes the similarity between u and v based on the squared Euclidean distance, scaled by a parameter gamma. It transforms the data into an infinite-dimensional space, enabling the model to handle complex, non-linear patterns. The smaller the distance between u and v, the larger the kernel value, indicating stronger similarity.
- Besides polynomial transform, another way to handle nonlinear problems is to add features computed using a similarity function.

$$\phi\gamma(\mathbf{x}, \ell) = \exp(-\gamma \| \mathbf{x} - \ell \|^2)$$

$$= \exp\left(- \frac{1}{2\sigma^2} ||\mathbf{x} - \boldsymbol{l}||^2 \right)$$

```
tuneAndTest(SVC(kernel="rbf"),{"C": uniform(0,20), "gamma":uniform(0, 1)})

best hyper-parameter value:  {'C': 10.527990485942908, 'gamma': 0.6238122128846867}
confusion matrix: [[42  3]
 [ 0 55]]
recall: 1.0
precision: 0.9482758620689655
```



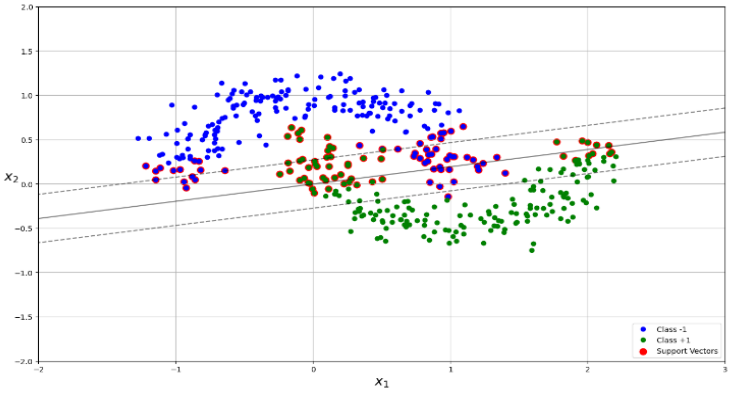
IMPLEMENTING CUSTOM NONLINEAR SVM:

- The standard library SVM performs on the dataset, you will attempt to implement your version of SVM, It has been created addition to the quadratic optimization. Essentially, you will get the optimized value of α for free.
- The code provided implements a custom Support Vector Machine (SVM) classifier from scratch using Python and `co-opt` for solving the Quadratic Programming (QP) optimization problem that arises when training SVMs. Below is an explanation of the key components and the steps involved in the code:
- The class `My SVM` is defined as a subclass of `Base Estimator` and `Classifier Mixon`, both of which are base classes from `scikit-learn`. This allows `My SVM` to be integrated with the scikit-learn ecosystem and be compatible with tools like `fit`, `predict`, and `cross-validation`.
- The kernel function used in SVM, defaulting to a linear kernel. This can be changed to other kernel functions, such as polynomial or Gaussian (RBF), to allow for non-linear decision boundaries.
- The regularization parameter. The value of `C` controls the trade-off between maximizing the margin (the distance between the decision boundary and the closest data points) and minimizing the classification error. If `C` is not provided, it is set to `None`, meaning no regularization is applied.
- The `fit` method is responsible for training the SVM classifier on a set of input data (`X` for features and `y` for labels).

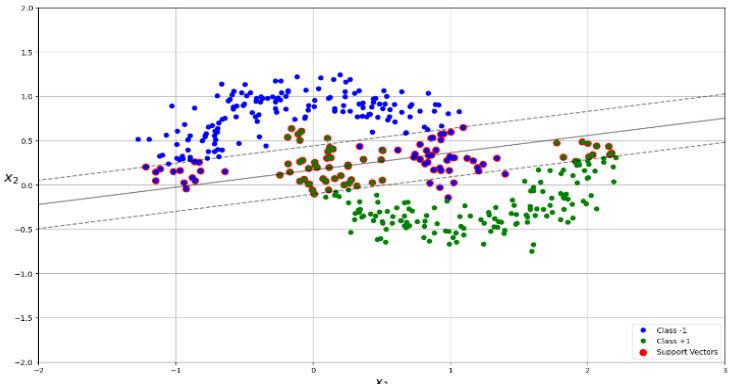
  Quadratic Programming (QP) Setup:
- The quadratic programming problem for training an SVM is set up. The goal of SVM training is to find the Lagrange multipliers (`a`) that minimize the objective function while satisfying the constraints. Print Result(model): Evaluates a model on test data, calculating confusion matrix, recall, precision, and visualization (using plots).
- Compares custom SVM (system) with scikit-learn SVM (lib). Fits both models, prints details of the system, and calls print Results on both for test data evaluation and visualization.

- Evaluate each model's performance on unseen test data using metrics (confusion matrix, recall, precision) and visualization.
- The below figure clearly shows that custom SVM and moon dataset generation SVM for linear, similarly works with polynomial and Radial Basis Functions.

```
comp(MySVM(C=8.1537),SVC(kernel='linear', C=8.1537))

     pcost       dcost       gap    pres   dres
 0: -5.7855e+02 -1.7691e+04  4e+04  7e-01  4e-14
 1: -4.5535e+02 -4.4529e+03  4e+03  2e-02  2e-14
 2: -6.0289e+02 -1.4379e+03  9e+02  3e-03  2e-14
 3: -7.3065e+02 -1.0600e+03  3e+02  1e-03  2e-14
 4: -7.8022e+02 -9.5918e+02  2e+02  5e-04  2e-14
 5: -8.0913e+02 -9.0561e+02  1e+02  2e-04  2e-14
 6: -8.2120e+02 -8.8377e+02  6e+01  1e-04  2e-14
 7: -8.3144e+02 -8.6567e+02  3e+01  6e-05  2e-14
 8: -8.3841e+02 -8.5497e+02  2e+01  2e-05  2e-14
 9: -8.4240e+02 -8.4859e+02  6e+00  1e-06  2e-14
10: -8.4391e+02 -8.4674e+02  3e+00  6e-07  2e-14
11: -8.4458e+02 -8.4600e+02  1e+00  2e-08  2e-14
12: -8.4504e+02 -8.4550e+02  5e-01  6e-09  2e-14
13: -8.4525e+02 -8.4526e+02  2e-02  2e-10  3e-14
14: -8.4526e+02 -8.4526e+02  2e-03  1e-11  2e-14
15: -8.4526e+02 -8.4526e+02  2e-05  1e-13  3e-14
Optimal solution found.
107 support vectors out of 400 points
mysvm: MySVM(C=8.1537)
confusion matrix: [[37  8]
 [10 45]]
recall: 0.8181818181818182
precision: 0.8490566037735849
```




```
standard library implementation:
confusion matrix: [[34 11]
 [ 5 50]]
recall: 0.9090909090909091
precision: 0.819672131147541
```



OBSERVATION: My SVM has a higher precision score yet a lower recall score than the library implementation. The score is eight percent lower than the standard library implementation. I tried multiple values of C ranging from 1 to 10, yet none of them yielded a significantly better performance recall score.

- I've learned that it is very important to keep track of the dimensions of all the data structures I'm using. In general, the standard library implementation yields slightly better performance than my implementation. It decreases the chance of getting an improved training result due to the lack of tuning.

APPLICATIONS:

- **Educational Tool:**
  This project serves as an excellent educational tool for understanding how Support Vector Machines (SVMs) work. By building the SVM from scratch, it allows users to grasp the underlying mathematics and algorithms, such as kernel functions, Lagrange multipliers, and Quadratic Programming.
- **Benchmarking Against Libraries:**
  The implementation allows for direct comparison with established libraries like scikit-learn. This is useful for performance benchmarking and understanding trade-offs in using custom implementations versus standard ones.
- **Custom Kernel Implementation:**
  The ability to define custom kernels makes this project suitable for exploring domain-specific data, where standard kernels (e.g., linear, RBF) may not perform optimally.
- **Optimization Research:**
  This project can be extended for optimization research by experimenting with different solvers or regularization techniques to improve SVM performance.
- **Medical Diagnosis:**
  Useful in binary classification problems, such as detecting diseases based on diagnostic test data.
- **Natural Language Processing (NLP):**
  Applied to text classification tasks like spam detection or sentiment analysis.
- **Finance:**
  Can be used for binary classification tasks like fraud detection in transactions or predicting stock price movements.
- **Bioinformatics:**
  Suitable for classifying gene expression data or identifying protein structures.
- **Robotics and Automation:**
  Used in robotics for object classification and motion recognition

ADVANTAGES:

- **Educational Value:**
  Provides hands-on experience with fundamental SVM concepts like the kernel trick, margin maximization, and dual optimization.

- **Flexibility:**
  Unlike standard libraries, the implementation offers flexibility to modify the optimization process and kernel functions, and support vector selection.
- **Integration with sci-kit-learn:**
  The Base Estimator and Classifier Mixing inheritance allow seamless integration with sci-kit-lean's ecosystem, enabling use in pipelines and compatibility with utility functions like cross-validation.
- **Visualization:**
  The plots' function allows users to visualize the decision boundaries, support vectors, and margins, which is essential for understanding the classifier's behavior
- **Customization:**
  The project allows for custom tuning of hyperparameters (C, kernel), making it adaptable for various datasets and applications.

CONCLUSION:

This moon data demonstrates the effectiveness of SVMs for classification tasks and highlights the importance of understanding machine learning algorithms' internal workings. While library implementations like SVC are optimized for performance and scalability, Kernel version changes, and building a custom model enhances knowledge and offers flexibility.

REFERENCES:

1) https://www.tutorialspoint.com/non-linear-svm-in-machine-learning
2) https://scikit-learn.org/1.4/auto_examples/svm/plot_svm_nonlinear.html
3) https://www.geeksforgeeks.org/support-vector-machine-algorithm/
4) https://docs.google.com/presentation/d/1CeBhepjDKBaFBq2BZq-zNQs-MC8ll7aL4qAF8TJ24FM/edit#slide=id.g437a8c453b_90_6

GITHUB LINK:

- **https://github.com/CHENNURUHARISH/SVM-NON-LINEAR---23024881/blob/main/HARISH%20SVM-23024881%20(1).ipynb**