CSC 722 U15: Machine Learning Fundamentals

Project SVM

Name: Neerajdattu Dudam

ID: 101179017

GitHub: https://github.com/Dudam-Neeraj-Dattu/ML_Assignment_SVM_USD

Iris Dataset – SVM

(Entire code with notes is uploaded to the GitHub, parts of the code are used for the explanation in the document)

Data Exploration and Preparation:

1. This data sets consists of 3 different types of irises' (Setosa, Versicolour, and Virginica) petal and sepal length, stored in a 150x4 numpy.ndarray.
2. The rows being the samples and the columns being: Sepal Length, Sepal Width, Petal Length and Petal Width.
3. The features include *['sepal length (cm)', 'sepal width (cm)', 'petal length (cm)', 'petal width (cm)']*.
4. The targets are encoded *[0: 'setosa', 1: 'versicolor', 2: 'virginica']*.

```
iris = datasets.load_iris()
print(iris.data)
print(iris.target)
print(iris.feature_names)
print(iris.target_names)
```

✓ 0.1s

```
[[5.1 3.5 1.4 0.2]
 [4.9 3.  1.4 0.2]
 [4.7 3.2 1.3 0.2]
 [4.6 3.1 1.5 0.2]
 [5.  3.6 1.4 0.2]
 [5.4 3.9 1.7 0.4]
 [4.6 3.4 1.4 0.3]
 [5.  3.4 1.5 0.2]
 [4.4 2.9 1.4 0.2]
 [4.9 3.1 1.5 0.1]
 [5.4 3.7 1.5 0.2]
 [4.8 3.4 1.6 0.2]
 [4.8 3.  1.4 0.1]
 [4.3 3.  1.1 0.1]
 [5.8 4.  1.2 0.2]
 [5.7 4.4 1.5 0.4]
 [5.4 3.9 1.3 0.4]
 [5.1 3.5 1.4 0.3]
 [5.7 3.8 1.7 0.3]
 [5.1 3.8 1.5 0.3]
 [5.4 3.4 1.7 0.2]
 [5.1 3.7 1.5 0.4]
 [4.6 3.6 1.  0.2]
 [5.1 3.3 1.7 0.5]
 [4.8 3.4 1.9 0.2]
...
 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2
 2 2]
['sepal length (cm)', 'sepal width (cm)', 'petal length (cm)', 'petal width (cm)']
['setosa' 'versicolor' 'virginica']
```

Output is truncated. View as a scrollable element or open in a text editor. Adjust cell output settings...

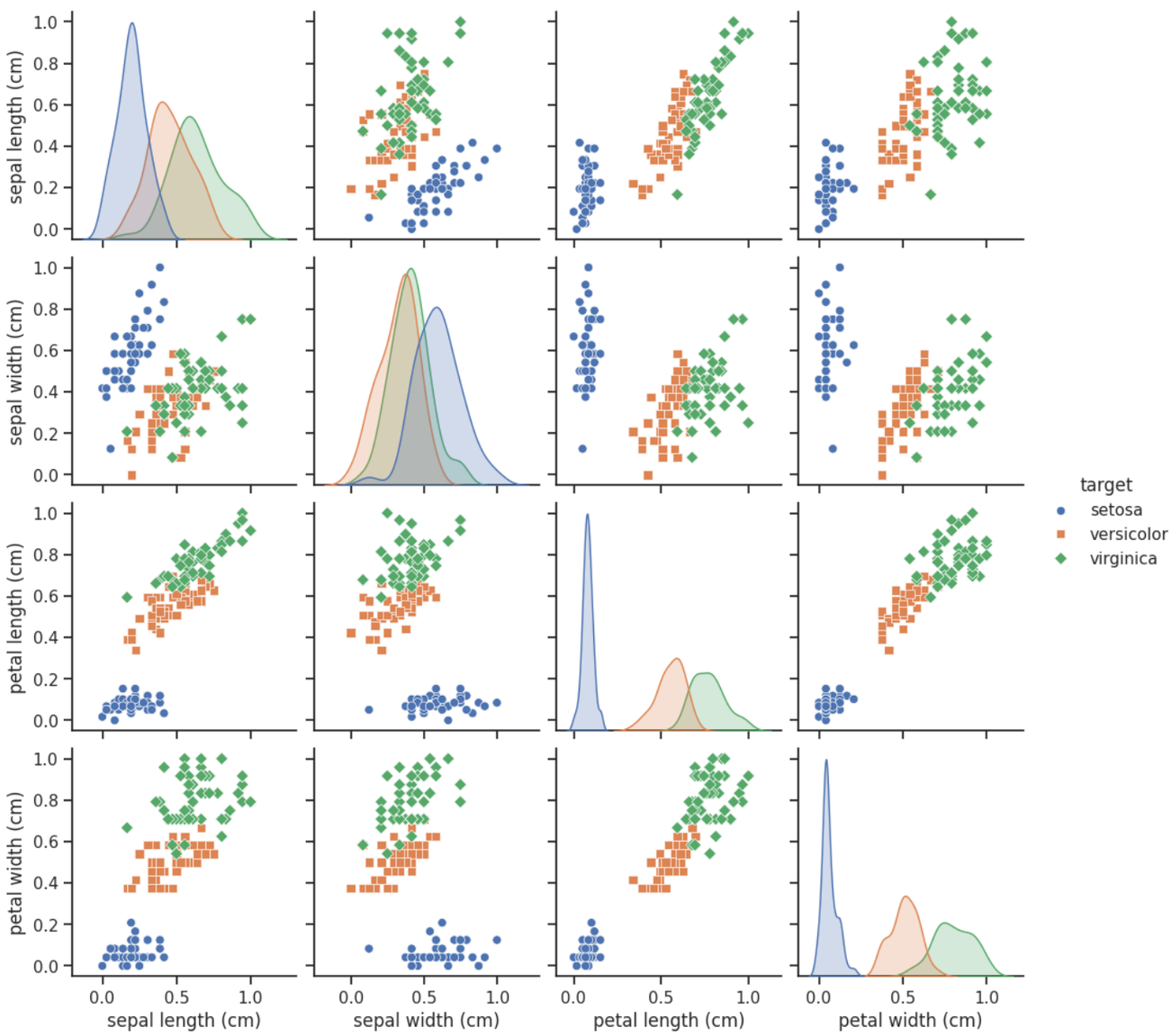
5. Using *np.isnan* it is found that there are no missing values.

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

missing_values = np.isnan(data)
has_missing_values = np.any(missing_values)

if has_missing_values:
    print("There are missing values in the data.")
else:
    print("There are no missing values in the data.")
```

✓ 0.2s

```
There are no missing values in the data.
```

6. The values of each feature have different range after exploration, so using *minmaxscalar()* the scaling of data is done.
7. The relation between each scaled feature is illustrated using *pairplots* from *seaborn* library.



SVM Implementation:

1. Support Vector Machines (SVMs) are a powerful set of supervised learning methods used for classification, regression, and outliers' detection.

2. Supervised Learning: SVMs are primarily used for classification tasks but can also be applied to regression.
3. Max-Margin Classifier: They work by finding the hyperplane that best separates the classes with the maximum margin.
4. Kernel Trick: For non-linearly separable data, SVMs use kernel functions to map the inputs into higher-dimensional spaces where a linear separation is possible.
5. SVMs are popular due to their robustness in handling high-dimensional data and their ability to model complex nonlinear relationships. They're widely used in various fields, including image recognition, bioinformatics, and text categorization.
6. In Support Vector Machines (SVMs), kernels are functions used to map the original dataset into a higher-dimensional space to make it possible to find a hyperplane that can separate the data into different classes. Here's a brief overview of the different types of kernels:
    a. Linear Kernel: It is the simplest kernel, used when the data is linearly separable. It represents the inner product of two points in feature space.
    b. Polynomial Kernel (Poly): It represents the similarity of vectors (training samples) in a feature space over polynomials of the original variables. This kernel allows the learning of non-linear models.
    c. Sigmoid Kernel: This kernel function is equivalent to a two-layer perceptron model of a neural network and is used as an activation function for artificial neurons.
    d. Radial Basis Function (RBF): Also known as the Gaussian kernel, it is a popular choice due to its similarity to the Gaussian distribution.

```
# Scale the features
scaler = MinMaxScaler()
X_scaled = scaler.fit_transform(X)

X_train, X_test, y_train, y_test = train_test_split(X_scaled, y, test_size=0.2, random_state=42)

# Define the kernel types
kernels = ['linear', 'poly', 'rbf', 'sigmoid']
y_predicted = []

# Iterate over each kernel
for i, kernel in enumerate(kernels):
    # Fit the SVM model
    model = SVC(kernel=kernel)
    model.fit(X_train, y_train)

    # Make predictions
    y_pred = model.predict(X_test)
    y_predicted.append(y_pred)
```

K-fold Cross-Validation:

1. K-fold cross-validation is a resampling procedure used to evaluate the performance of machine learning models.
2. K-fold cross-validation involves dividing the dataset into 'k' equal-sized subsets or folds. The model is trained on 'k-1' folds and tested on the remaining fold. This process is repeated 'k' times, each time with a different fold as the test set.
3. It provides a more reliable estimate of model performance compared to a single train-test split, as it reduces the variance associated with a random selection of train-test sets.
4. Bias-Variance Trade-off: The choice of 'k' affects the bias and variance of the model evaluation. Using more folds (e.g., 10) can reduce bias but increase variance, while fewer folds (e.g., 5) can reduce variance but increase bias. And the data is shuffled before doing so.

```
# Define k-fold cross-validation
kf_5 = KFold(n_splits=5, shuffle=True)
kf_10 = KFold(n_splits=10, shuffle=True)

for kernel in kernels:
    svc = SVC(kernel=kernel)
    print(f"Mean accuracy for kernel: {kernel} and k=5 is {np.mean(cross_val_score(svc, X_scaled, y, cv=kf_5)):.2f}
        and k=10 is {np.mean(cross_val_score(svc, X_scaled, y, cv=kf_10)):.2f}")
```

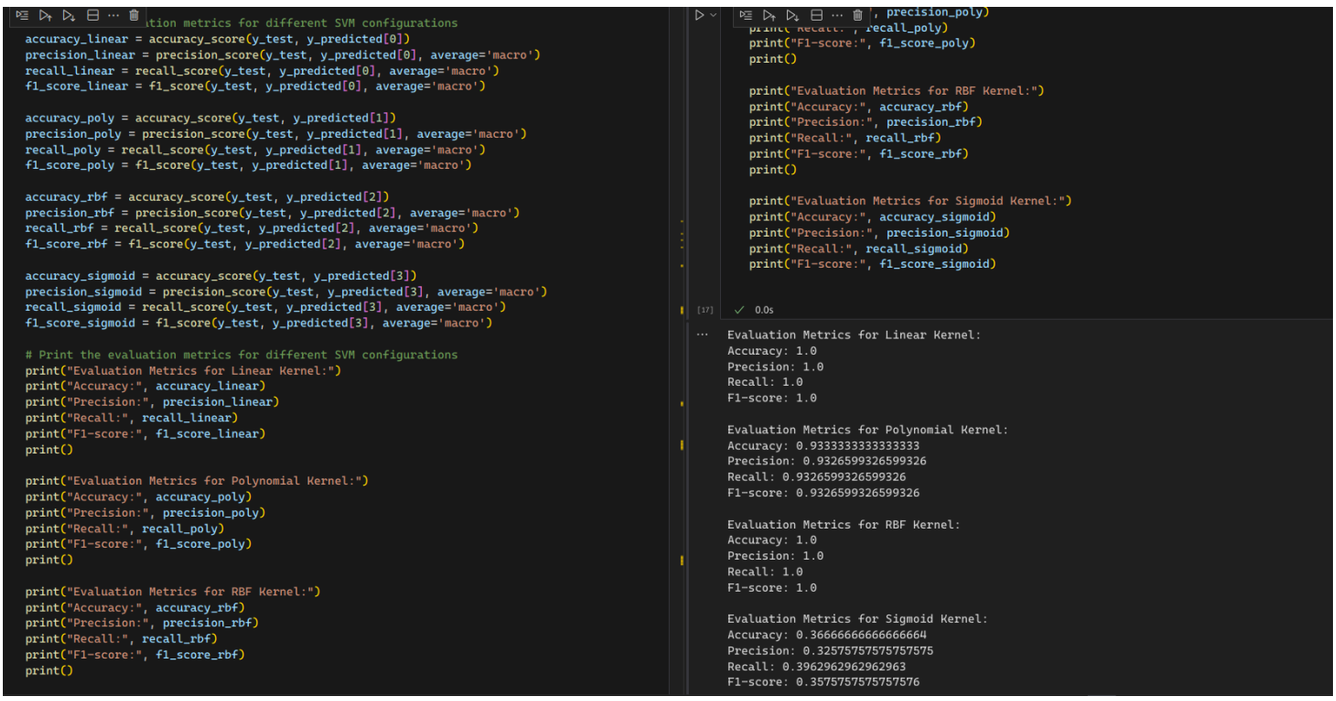
✓ 0.3s

```
Mean accuracy for kernel: linear and k=5 is 0.97 and k=10 is 0.96
Mean accuracy for kernel: poly and k=5 is 0.95 and k=10 is 0.95
Mean accuracy for kernel: rbf and k=5 is 0.97 and k=10 is 0.95
Mean accuracy for kernel: sigmoid and k=5 is 0.35 and k=10 is 0.33
```

Evaluation Metrics:

The performance of a machine learning model can be measured using various evaluation metrics, depending on the task and the nature of the data. Some common metrics include:

1. Accuracy: The proportion of correctly classified instances out of the total instances. It is calculated as the ratio of the number of correct predictions to the total number of predictions.
2. Precision: Also known as positive predictive value, it is the proportion of true positive predictions (correctly predicted positives) out of all positive predictions. It is calculated as TP / (TP + FP), where TP is the number of true positives and FP is the number of false positives.
3. Recall: Also known as sensitivity or true positive rate, it is the proportion of true positive predictions (correctly predicted positives) out of all actual positive instances. It is calculated as TP / (TP + FN), where FN is the number of false negatives.
4. F1 Score: The harmonic mean of precision and recall. It provides a balance between precision and recall and is useful when there is an imbalance between the number of positive and negative instances in the data. It is calculated as 2 * (precision * recall) / (precision + recall).
5. For interpretation purposes I have done PCA to reduce the dimensionality and calculated the evaluation metrics. So, I did for pre – PCA and post – PCA. The blow image represents the pre – PCA results and post – PCA results are discussed in the final findings.



```
ition metrics for different SVM configurations
accuracy_linear = accuracy_score(y_test, y_predicted[0])
precision_linear = precision_score(y_test, y_predicted[0], average='macro')
recall_linear = recall_score(y_test, y_predicted[0], average='macro')
f1_score_linear = f1_score(y_test, y_predicted[0], average='macro')

accuracy_poly = accuracy_score(y_test, y_predicted[1])
precision_poly = precision_score(y_test, y_predicted[1], average='macro')
recall_poly = recall_score(y_test, y_predicted[1], average='macro')
f1_score_poly = f1_score(y_test, y_predicted[1], average='macro')

accuracy_rbf = accuracy_score(y_test, y_predicted[2])
precision_rbf = precision_score(y_test, y_predicted[2], average='macro')
recall_rbf = recall_score(y_test, y_predicted[2], average='macro')
f1_score_rbf = f1_score(y_test, y_predicted[2], average='macro')

accuracy_sigmoid = accuracy_score(y_test, y_predicted[3])
precision_sigmoid = precision_score(y_test, y_predicted[3], average='macro')
recall_sigmoid = recall_score(y_test, y_predicted[3], average='macro')
f1_score_sigmoid = f1_score(y_test, y_predicted[3], average='macro')

# Print the evaluation metrics for different SVM configurations
print("Evaluation Metrics for Linear Kernel:")
print("Accuracy:", accuracy_linear)
print("Precision:", precision_linear)
print("Recall:", recall_linear)
print("F1-score:", f1_score_linear)
print()

print("Evaluation Metrics for Polynomial Kernel:")
print("Accuracy:", accuracy_poly)
print("Precision:", precision_poly)
print("Recall:", recall_poly)
print("F1-score:", f1_score_poly)
print()

print("Evaluation Metrics for RBF Kernel:")
print("Accuracy:", accuracy_rbf)
print("Precision:", precision_rbf)
print("Recall:", recall_rbf)
print("F1-score:", f1_score_rbf)
print()
```

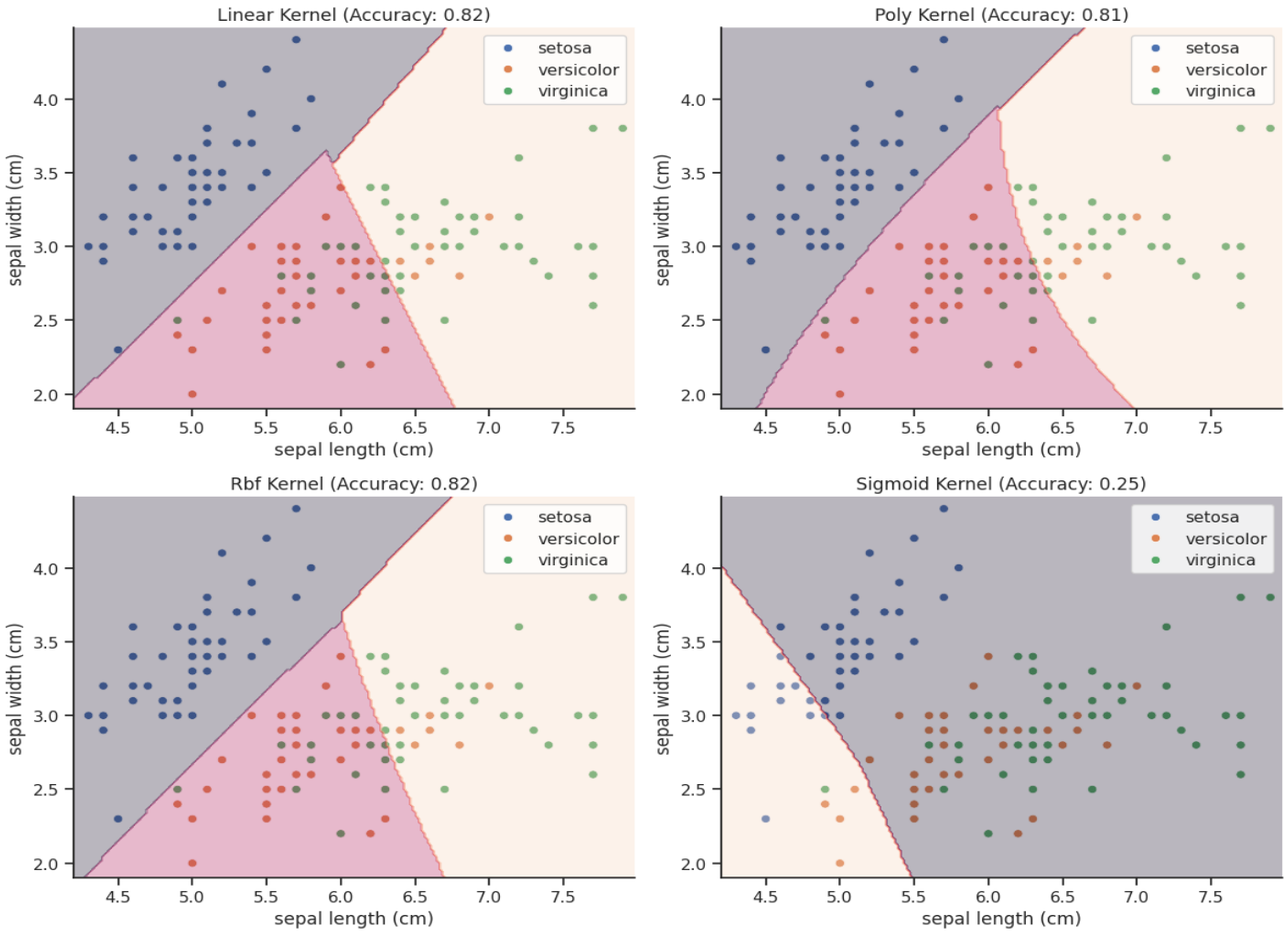
```
, precision_poly)
recall_poly)
print("F1-score:", f1_score_poly)
print()

print("Evaluation Metrics for RBF Kernel:")
print("Accuracy:", accuracy_rbf)
print("Precision:", precision_rbf)
print("Recall:", recall_rbf)
print("F1-score:", f1_score_rbf)
print()

print("Evaluation Metrics for Sigmoid Kernel:")
print("Accuracy:", accuracy_sigmoid)
print("Precision:", precision_sigmoid)
print("Recall:", recall_sigmoid)
print("F1-score:", f1_score_sigmoid)
```

[17] ✓ 0.0s

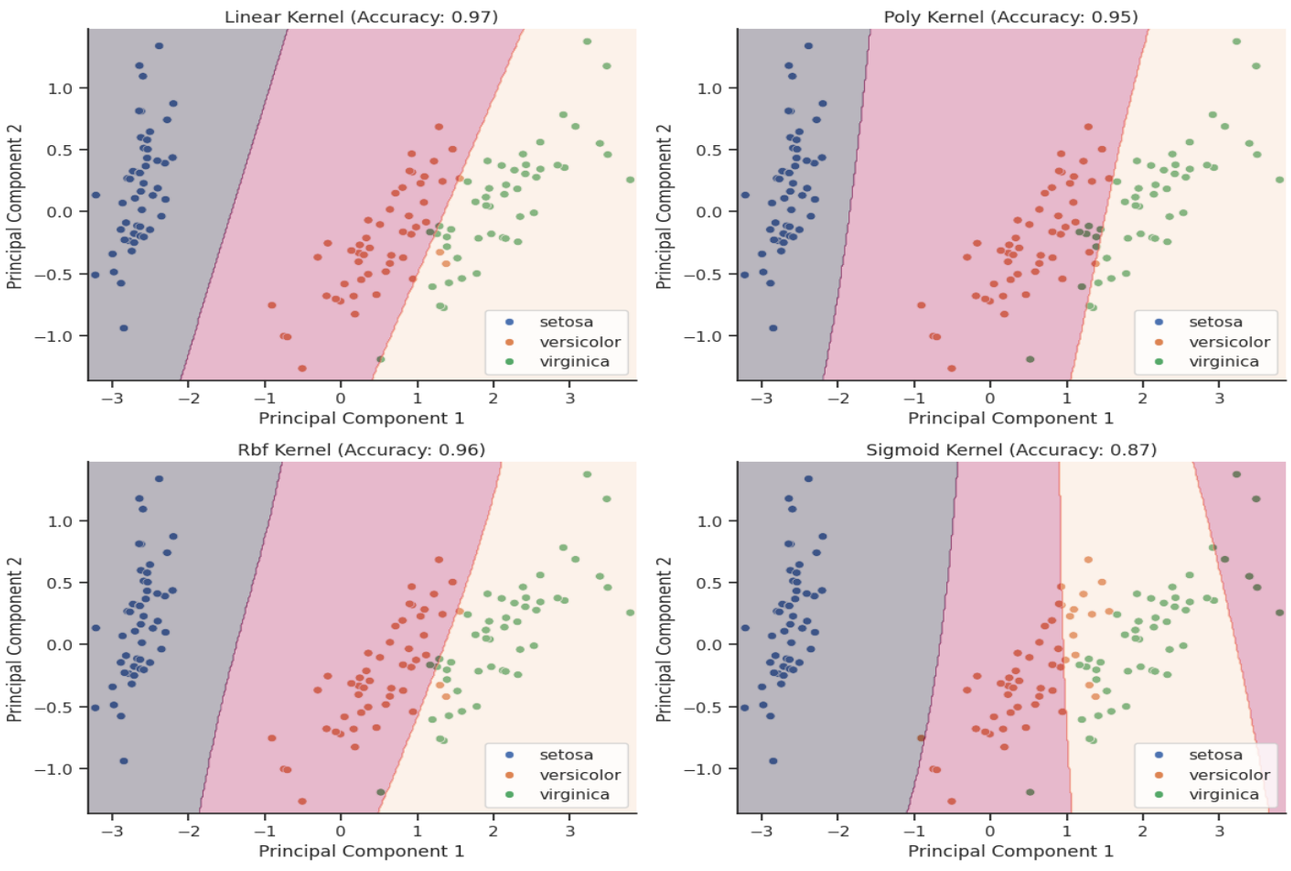
```
Evaluation Metrics for Linear Kernel:
Accuracy: 1.0
Precision: 1.0
Recall: 1.0
F1-score: 1.0

Evaluation Metrics for Polynomial Kernel:
Accuracy: 0.9333333333333333
Precision: 0.9326599326599326
Recall: 0.9326599326599326
F1-score: 0.9326599326599326

Evaluation Metrics for RBF Kernel:
Accuracy: 1.0
Precision: 1.0
Recall: 1.0
F1-score: 1.0

Evaluation Metrics for Sigmoid Kernel:
Accuracy: 0.36666666666666664
Precision: 0.32575757575757575
Recall: 0.3962962962962963
F1-score: 0.3575757575757576
```

Findings:

1. The decision boundary is drawn by taking the first two features *['sepal length', 'sepal width']* as an example.



2. Then I have done the PCA to bring it down to 2 features and trained using different SVM kernels and drawn the boundary line.



3. Comparing both decision boundaries, we can conclude that the model trained on data which is done by dimensionality reduction given good accuracy.
4. The evaluation metrics for each kernel, both pre and post PCA, are as follows:
5. Pre-PCA:
    a. Linear Kernel: Exhibited perfect scores across all metrics.
    b. Polynomial Kernel: Showed high performance with an accuracy of approximately 93.33%.
    c. RBF Kernel: Achieved perfect scores, like the Linear Kernel.
    d. Sigmoid Kernel: Underperformed significantly with the lowest scores in all metrics.
6. Post-PCA:
    a. Linear Kernel: Slight decrease in performance post-PCA with an accuracy of approximately 96.67%.
    b. Polynomial Kernel: Mirrored the Linear Kernel's performance post-PCA.
    c. RBF Kernel: Maintained perfect scores even after PCA.
    d. Sigmoid Kernel: Improved performance post-PCA with an accuracy of 80%.
7. Discussion: The results indicate that the Linear and RBF kernels performed optimally with or without PCA, achieving perfect scores. The Polynomial kernel showed a slight decrease in performance after PCA, which suggests that it may be sensitive to the dimensionality of the data. The Sigmoid kernel, while still the least performing, showed considerable improvement post-PCA, indicating that dimensionality reduction may positively affect its performance.
8. Conclusion: The analysis demonstrates that the choice of kernel can significantly impact the performance of SVM classifiers. While the Linear and RBF kernels proved robust against dimensionality reduction, the Polynomial and Sigmoid kernels' performance varied with PCA. These findings underscore the importance of kernel selection and dimensionality reduction techniques in machine learning tasks.

References:

1. https://scikit-learn.org/stable/modules/svm.html#svm
2. https://medium.com/analytics-vidhya/introduction-to-svm-and-kernel-trick-part-1-theory-d990e2872ace
3. https://www.javatpoint.com/major-kernel-functions-in-support-vector-machine
4. https://machinelearningmastery.com/k-fold-cross-validation/