Classical ML Classification Algorithm Comparison on the Iris Dataset

Indian Institute Of Technology Patna

Department of Computer Science

APR Assignment 1

Submitted by:	ANIRUDH D BHAT
Roll Number:	2201CS94
Date:	September 19, 2025

1 Project Overview

This project provides a comprehensive comparison of three classical machine learning classification algorithms: **K-Nearest Neighbors (KNN)**, **Support Vector Machine (SVM)**, and **Logistic Regression**. The evaluation is performed on the well-known Iris dataset.

The primary goal is to not only build and evaluate these models but also to determine the most robust and accurate classifier for this particular dataset through statistical analysis over multiple randomized trials.

2 Dataset

The project utilizes the classic **Iris dataset**, loaded directly from the `scikit-learn` library.

- **Features**:
  1. Sepal Length (cm)
  2. Sepal Width (cm)
  3. Petal Length (cm)
  4. Petal Width (cm)
- **Target Classes**:
  1. Iris-setosa
  2. Iris-versicolor
  3. Iris-virginica

```
iris = load_iris()
df = pd.DataFrame(data=iris.data, columns=iris.feature_names)
df['target'] = iris.target
print(df.head())
print(df.describe())
```

Listing 1: Data Loading

2.1 Justification for Approach

The decision to proceed with classical machine learning algorithms without extensive pre-processing was based on two key observations from the exploratory data analysis (EDA):

1. **Feature Separability:** The pair plots (Figure 1) show a high degree of class separation for most feature combinations. This suggests that complex, non-linear models may not be necessary to achieve high accuracy.
2. **Feature Scaling:** The feature values across the dataset are within a relatively small and consistent range. There are no large disparities in scale that would disproportionately influence distance-based algorithms like KNN or the gradient-based optimization in Logistic Regression. Therefore, normalization or standardization was deemed unnecessary for this initial analysis.

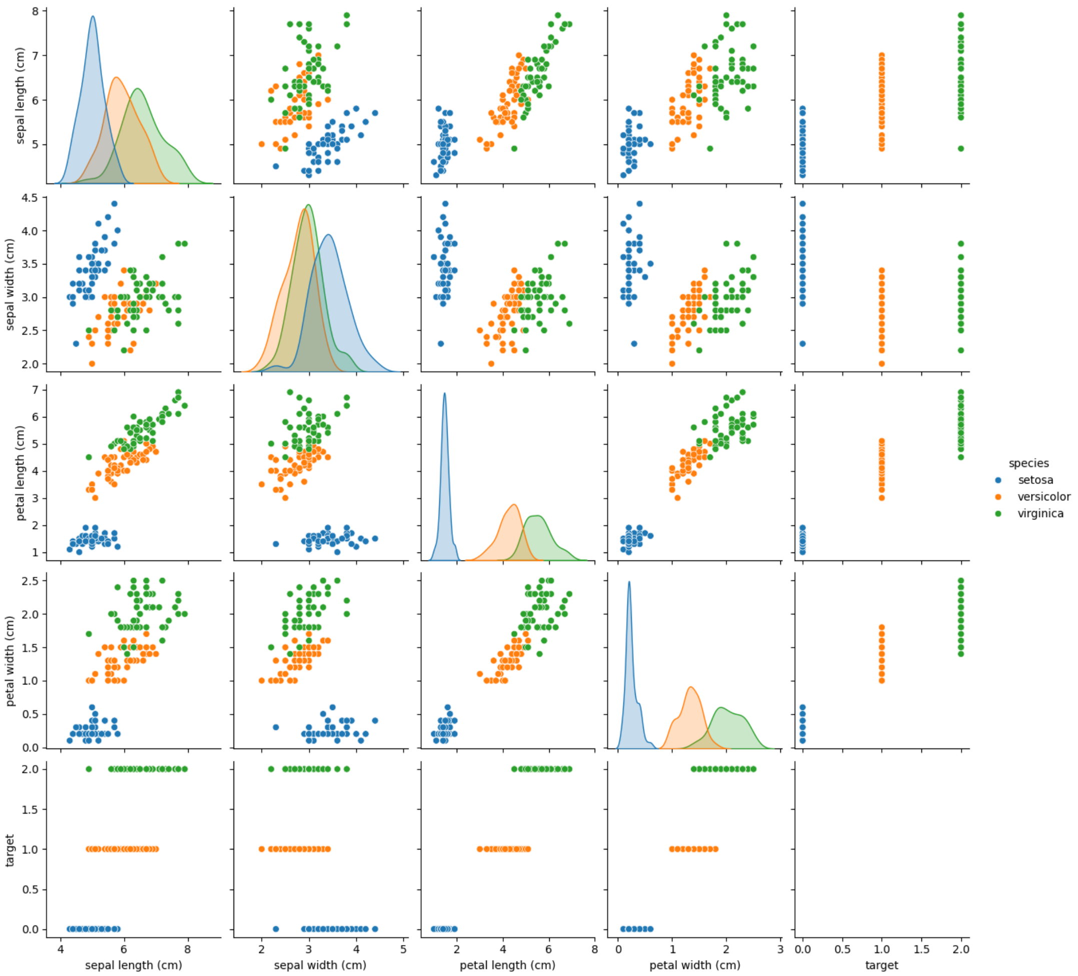

Figure 1: A pair plot showing the relationships between the four features, colored by species.

```
species_mapping = {0: 'setosa', 1: 'versicolor', 2: 'virginica'}
df['species'] = df['target'].map(species_mapping)
plots = sns.pairplot(df, hue='species')
plots.savefig("pairplot.png")
```

Listing 2: Creating the plot

3 Methodology

The notebook follows a standard machine learning workflow:

3.1 Hyperparameter Tuning (KNN)

For the KNN algorithm, an optimal value for 'K' (the number of neighbors) is crucial. This was determined using the **Elbow Method** combined with 5-fold cross-validation. The accuracy for K values from 1 to 30 was plotted, and the "elbow" of the curve indicated the optimal K that balances bias and variance.

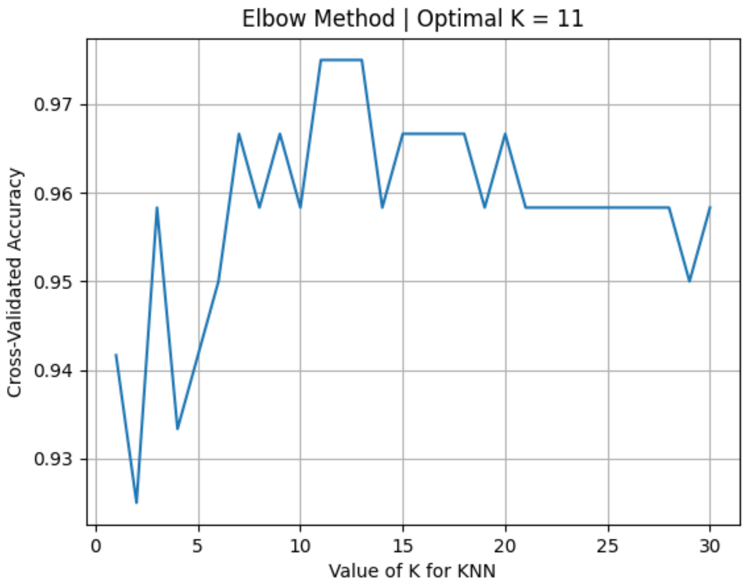

Figure 2: The cross-validated accuracy peaks at K=11, which was selected as the optimal value.

```

def findBestK(X_train, y_train, show):
  k_range = range(1, 31)
  cv_scores = []

  for k in k_range:
      knn = KNeighborsClassifier(n_neighbors=k)
      scores = cross_val_score(knn, X_train, y_train, cv=5, scoring=’accuracy’)
      cv_scores.append(scores.mean())

  best_k_index = np.argmax(cv_scores)
  best_k = k_range[best_k_index]



  if(show):
    print(f"The optimal value of K is: {best_k}")
    plt.plot(k_range, cv_scores)
    plt.xlabel(’Value of K for KNN’)
    plt.ylabel(’Cross-Validated Accuracy’)
    plt.title(f’Elbow Method | Optimal K = {best_k}’)
    plt.grid(True)
    plt.savefig("knn_plot.png")

  return best_k

findBestK(X_train, y_train,True)
```

Listing 3: Finding the best K

3.2 Model Training and Evaluation

Three distinct classification models were trained and evaluated:

- **K-Nearest Neighbors (KNN)**: Every run the optimal value of K is calculated based on 5 cross fold validation and that K is used for implentation of KNN.
- **Support Vector Machine (SVM)**: A Support Vector Classifier with a Radial Basis Function (`rbf`) kernel.
- **Logistic Regression**: A standard logistic regression model for multi-class classification.

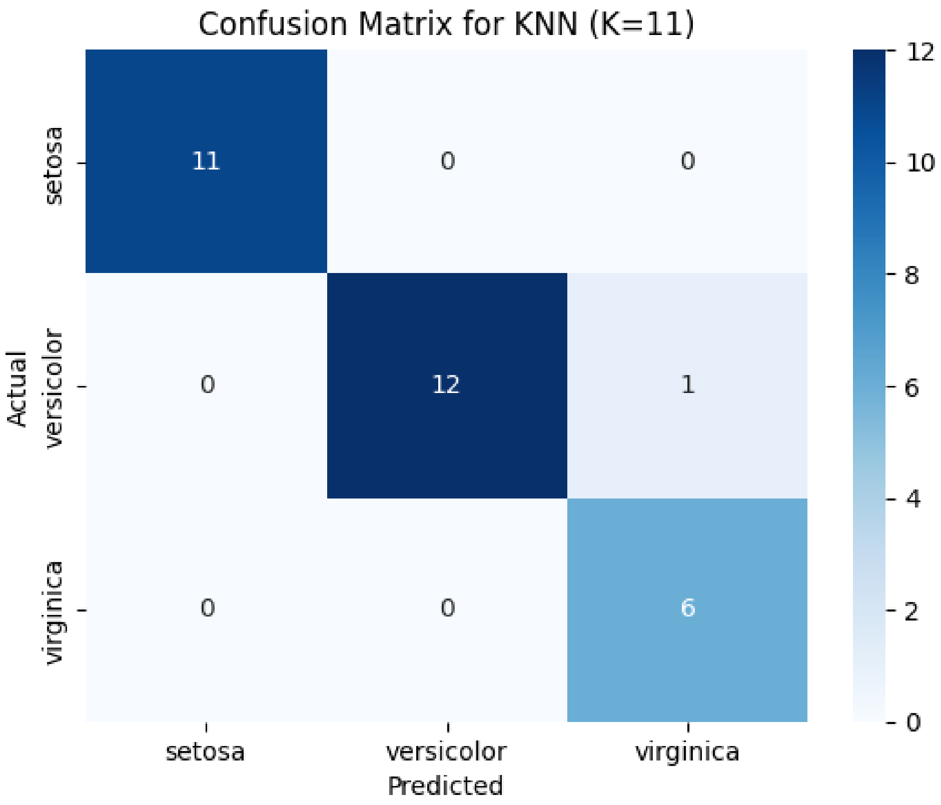

Figure 3: The confusion matrix for KNN, in this particular case it was the same for all 3 models.

```python
def applyKNN(X_train, y_train, X_test, y_test, show_cm):

  best_k = findBestK(X_train, y_train,False)
  final_knn = KNeighborsClassifier(n_neighbors = best_k)
  final_knn.fit(X_train, y_train)

  y_pred = final_knn.predict(X_test)

  accuracy = accuracy_score(y_test, y_pred)


  if(show_cm):
    print(f"KNN Accuracy: {accuracy}")
    cm = confusion_matrix(y_test, y_pred)
    print("Confusion Matrix:")


    sns.heatmap(cm, annot=True, fmt='d', cmap='Blues',
                xticklabels=iris.target_names, yticklabels=iris.target_names)
    plt.xlabel('Predicted')
    plt.ylabel('Actual')
    plt.title(f'Confusion Matrix for KNN (K={best_k})')
    plt.show()

  return accuracy
```

Listing 4: Code for KNN training, similar functions for Logistic Regression and SVM

3.3 Statistical Comparison

To ensure a robust comparison, the models were evaluated over **30 randomized runs**. In each run, the data was shuffled and split into a new 70/30 training/testing set. This process generated a distribution of accuracy scores for each algorithm, allowing for a more reliable statistical comparison of their performance.

```python
knn_scores = []
svm_scores = []
log_reg_scores = []

n_runs = 30

print(f"Running {n_runs} randomized train/test splits...")

for i in range(n_runs):
    a_train, a_test, b_train, b_test = train_test_split(X, y, test_size=0.3, random_state=i)

    knn_scores.append(applyKNN(a_train, b_train, a_test , b_test, False))
    svm_scores.append(applySVM(a_train, b_train, a_test, b_test, False))
    log_reg_scores.append(applyLogisticRegression(a_train, b_train, a_test, b_test, False))

print("Experiment complete.")

results_df = pd.DataFrame({
    'KNN': knn_scores,
    'SVM': svm_scores,
    'Logistic Regression': log_reg_scores
})

print("\n--- Statistical Summary of Model Accuracies ---")
print(results_df.describe())
```

Listing 5: Code to check over different combinations of train test

4 Results and Conclusion

4.1 Statistical Summary

The statistical analysis over 30 runs revealed a clearer performance hierarchy than a single test. **Logistic Regression** emerged as the top-performing model with the highest mean accuracy and the lowest standard deviation.

Table 1: Statistical Summary of Model Accuracies over 30 Runs

Statistic	KNN	SVM	Logistic Regression
count	30.000000	30.000000	30.000000
mean	0.954074	0.962222	0.965926
std	0.030844	0.026839	0.025924
min	0.888889	0.888889	0.911111
25%	0.933333	0.955556	0.938889
50%	0.955556	0.966667	0.977778
75%	0.977778	0.977778	0.977778
max	1.000000	1.000000	1.000000

4.2 Performance Visualization

The box plot in Figure 4 visually confirms these findings, showing the median accuracy for Logistic Regression and SVM is higher and their interquartile ranges are tighter compared to KNN.

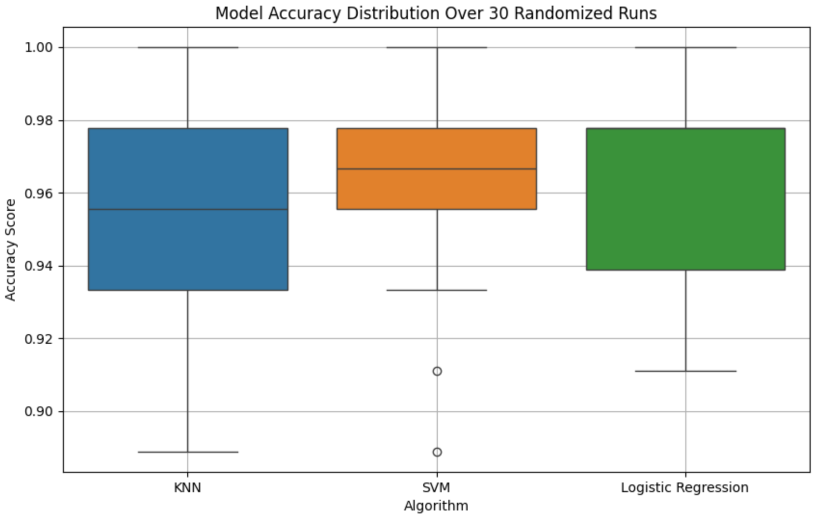

Figure 4: This box plot visualizes the distribution of accuracy scores over 30 runs.

```
plt.figure(figsize=(10, 6))
sns.boxplot(data=results_df)

plt.title('Model Accuracy Distribution Over 30 Randomized Runs')
plt.ylabel('Accuracy Score')
plt.xlabel('Algorithm')
plt.grid(True)

plt.savefig('model_comparison_boxplot.png')

print("Box plot saved as model_comparison_boxplot.png")
```

Listing 6: Box plot creation

4.3 Conclusion

Based on the analysis:

- **Logistic Regression** is the top-performing model with the highest mean accuracy (96.6%) and the lowest standard deviation, indicating both high accuracy and consistency.
- **SVM** is a very close second, also demonstrating high accuracy and stability. If you see the box plot you will see that except for 2 outlier cases, SVM does incredibly well, which makes me believe that if i train with the whole dataset it might be the best choice for unseen points.
- **KNN**, while still highly effective, has a slightly lower mean accuracy and a wider distribution of scores.

For the Iris classification task, both **Logistic Regression** and **SVM** are excellent and highly reliable choices.