CM3015irisMT(code)

July 29, 2025

1 1. Abstract

This project evaluates the performance of four machine learning models, namely, k-nearest neighbours, naive bayes, logistic regression, and decision tree on the scikit-learn iris dataset. The models are implemented using both the original dataset and a PCA-reduced version in order to assess the impact of dimensionality reduction. Results are compared across various metrics, including accuracy, precision, recall, and F1-score, with a focus on hyperparameter tuning and model complexity. Hyperparameter tuning and complexity analysis are performed to understand model behaviour and identify optimal configurations. This project aims to provide a comprehensive understanding of machine learning techniques and their practical applications on a well-established dataset while addressing the trade-offs between accuracy and model simplicity.

2 2. Introduction

The iris dataset is a well-known and commonly used dataset in machine learning, containing 150 samples of iris flowers divided into three species: setosa, versicolor, and virginica. Each sample includes measurements for sepal and petal length and width. The iris dataset is also often used as an introductory tool for machine learning model evaluation due to its simplicity and clear distinction between the 3 species.

This project aims to compare the performance and accuracy of the four machine learning models using both the original dataset and PCA-reduced data. Principal Component Analysis (PCA) is a dimensionality reduction technique which helps mitigate overfitting issues in high-dimensional data. Hyperparameter tuning and model complexity analysis are also applied to assess the strengths and weaknesses of each model. The findings are intended to highlight the impact of dimensionality reduction and model complexity on classification performance. This report discusses the methods used, results obtained, and possible insights derived from scoring metrics and visualisation.

3 3. Background

List of algorithms used:

1. k-Nearest Neighboburs (kNN)
    - The knn algorithm is a supervised machine learning algorithm commonly used in classification and regression tasks. It is known for being easy to use and implement, and also for its effectiveness in real-world applications. It works by using distance as a metric to identify the k-number of nearest neighbours and making predictions based on similar data points in the dataset. “k”, in this case, refers to any positive integer.

2. Naive Bayes (nb)
    - The naive bayes algorithm acts on the basis that every pair of features being classified are independent of each other. It is most commonly used in text classification, for example spam filtering, rating classification or even sentiment detection. It is fast and making predictions are easy even with high dimensions of data. It works by using bayes' theorem, finding the probability of an event occurring given the probability of another event that has already occurred.
3. Logistic Regression
    - The logistic regression algorithm is a supervised machine learning algorithm used for classification tasks. It works by transforming output from a linear regression function using a sigmoid function.
4. Decision Tree
    - The decision tree algorithm is a machine learning algorithm used for both classification and regression tasks. It works by predicting the value of the target variable using simple decision rules based on the data features.

Additional techqniues: 1. Principal Component Analysis (PCA) - a dimensionality reduction technique that transforms the data into a lower-dimensional space while preserving as much variance as possible. - this transformation makes models more computationally efficient and reduces overfitting issues in high-dimensional data. - in this project, PCA was used to reduce the iris dataset from 4 dimensions to 2 principal components.

4 4. Methodology

1. Dataset preparation
    - The iris dataset will be split into training and testing sets in a ratio of 8:2 to evaluate model performance.
    - PCA will be applied to the training set to create a reduced dataset with 2 principal components, which will then be applied to the test set using the same transformation.
2. Model implementation
    - k-nearest neighbour, naive bayes, and PCA will be implemented from scratch as at least one of them has to be implemented from scratch following coursework requirements.
    - Logistic regression and decision tree will be implemented using scikit-learn for making comparisons.
3. Evaluation metrics
    - The models will be assessed based on accuracy, precision, recall and F1-score to capture their predictive performance.
    - Hyperparameter tuning will be conducted for all 4 models, number of neighbours for kNN, gaussian distribution for nb, inverse of regularisation strength for logistic regression, and max depth for decision tree.
4. Analysis techniques
    - Cross validation will be used to ensure the reliability of model evaluation by mitigating the effects of data variability and providing consistent results across different subsets or folds. Splitting the data into multiple 'k' folds and then averaging the results ensures that each part or subset of data is used for both training and testing. This is especially useful for small datasets like this one, the iris dataset which only has 150 samples, as the usual 8:2 train-test split might not be able to fullly capture variability in performance and could also lead to misleading results.

- Feature importance analysis helps to identify which features contribute to model predictions the most. This leads to better understanding behind model predictions, which can help with simplifying the model or even improving computational efficiency. Feature importance analysis will be performed on the decistion tree and logistic regression models. The other 2 models, kNN and nb, will not be analysed as they will be written from scratch.
  - For decision tree, importance should be determined by the contribution of each feature to the decision tree.
  - For logistic regression, importance should be determined by the magnitude of the coefficients of each feature.
- Model complexity will be analysed by varying key parameters to identify trends such as overfitting or underfitting if any. For example, increasing the range of number of neighbours in kNN or the depth of the decision tree might reveal insights about model flexibility and generalisation ability.

[245]:
```python
# list imports as necessary
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import numpy as np
from sklearn.linear_model import LogisticRegression
from sklearn.tree import DecisionTreeClassifier
from sklearn.model_selection import KFold
from sklearn.metrics import pairwise_distances
from collections import Counter
from sklearn.model_selection import GridSearchCV
from sklearn.metrics import accuracy_score, f1_score, precision_score,␣
  ↪recall_score
```

5 Dataset preparation

[246]:
```python
# loading iris dataset
iris = load_iris()

print(iris.DESCR)
```

```
.. _iris_dataset:

Iris plants dataset
--------------------

**Data Set Characteristics:**

:Number of Instances: 150 (50 in each of three classes)
```

```
:Number of Attributes: 4 numeric, predictive attributes and the class
:Attribute Information:
    - sepal length in cm
    - sepal width in cm
    - petal length in cm
    - petal width in cm
    - class:
            - Iris-Setosa
            - Iris-Versicolour
            - Iris-Virginica

:Summary Statistics:

============== ==== ==== ======= ===== ====================
                Min  Max   Mean    SD   Class Correlation
============== ==== ==== ======= ===== ====================
sepal length:   4.3  7.9   5.84   0.83    0.7826
sepal width:    2.0  4.4   3.05   0.43   -0.4194
petal length:   1.0  6.9   3.76   1.76    0.9490  (high!)
petal width:    0.1  2.5   1.20   0.76    0.9565  (high!)
============== ==== ==== ======= ===== ====================

:Missing Attribute Values: None
:Class Distribution: 33.3% for each of 3 classes.
:Creator: R.A. Fisher
:Donor: Michael Marshall (MARSHALL%PLU@io.arc.nasa.gov)
:Date: July, 1988

The famous Iris database, first used by Sir R.A. Fisher. The dataset is taken
from Fisher's paper. Note that it's the same as in R, but not as in the UCI
Machine Learning Repository, which has two wrong data points.

This is perhaps the best known database to be found in the
pattern recognition literature.  Fisher's paper is a classic in the field and
is referenced frequently to this day.  (See Duda & Hart, for example.)  The
data set contains 3 classes of 50 instances each, where each class refers to a
type of iris plant.  One class is linearly separable from the other 2; the
latter are NOT linearly separable from each other.

.. dropdown:: References

  - Fisher, R.A. "The use of multiple measurements in taxonomic problems"
    Annual Eugenics, 7, Part II, 179-188 (1936); also in "Contributions to
    Mathematical Statistics" (John Wiley, NY, 1950).
  - Duda, R.O., & Hart, P.E. (1973) Pattern Classification and Scene Analysis.
    (Q327.D83) John Wiley & Sons.  ISBN 0-471-22361-1.  See page 218.
  - Dasarathy, B.V. (1980) "Nosing Around the Neighborhood: A New System
    Structure and Classification Rule for Recognition in Partially Exposed
```

```
   Environments".  IEEE Transactions on Pattern Analysis and Machine
   Intelligence, Vol. PAMI-2, No. 1, 67-71.
 - Gates, G.W. (1972) "The Reduced Nearest Neighbor Rule".  IEEE Transactions
   on Information Theory, May 1972, 431-433.
 - See also: 1988 MLC Proceedings, 54-64.  Cheeseman et al"s AUTOCLASS II
   conceptual clustering system finds 3 classes in the data.
 - Many, many more …
```

[247]:
```
# check type
type(iris.data)
```

[247]:
```
numpy.ndarray
```

[248]:
```
# split features and target
X = iris.data
y = iris.target
```

[249]:
```
# split dataset into train and test sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,␣
 ↪random_state=42)
```

[250]:
```
# standardize features
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)
```

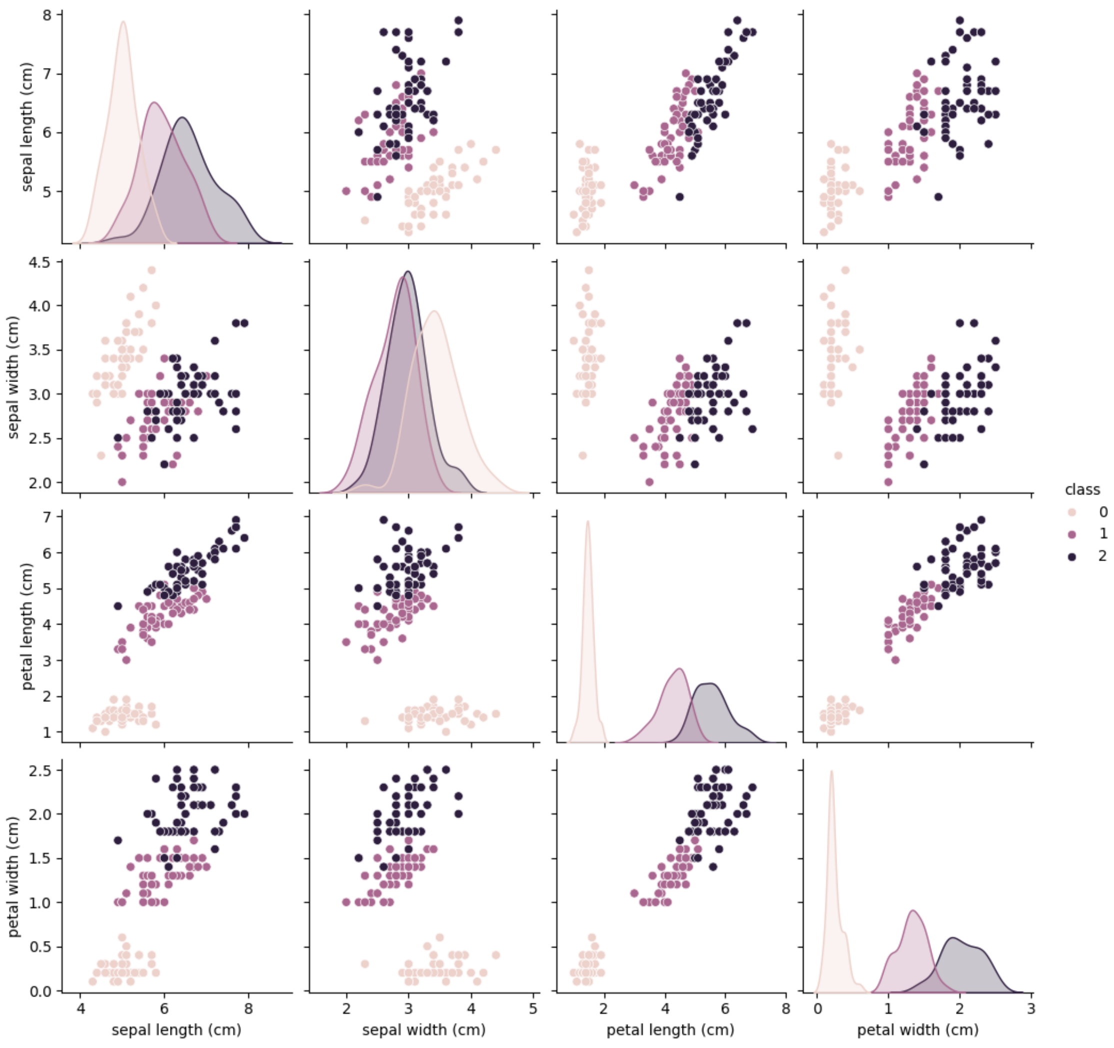
6 Exploratory data analysis (EDA)

[251]:
```
# convert to df for visualisation
irisdf = pd.DataFrame(X, columns=iris.feature_names)
irisdf['class'] = y

irisdf
```

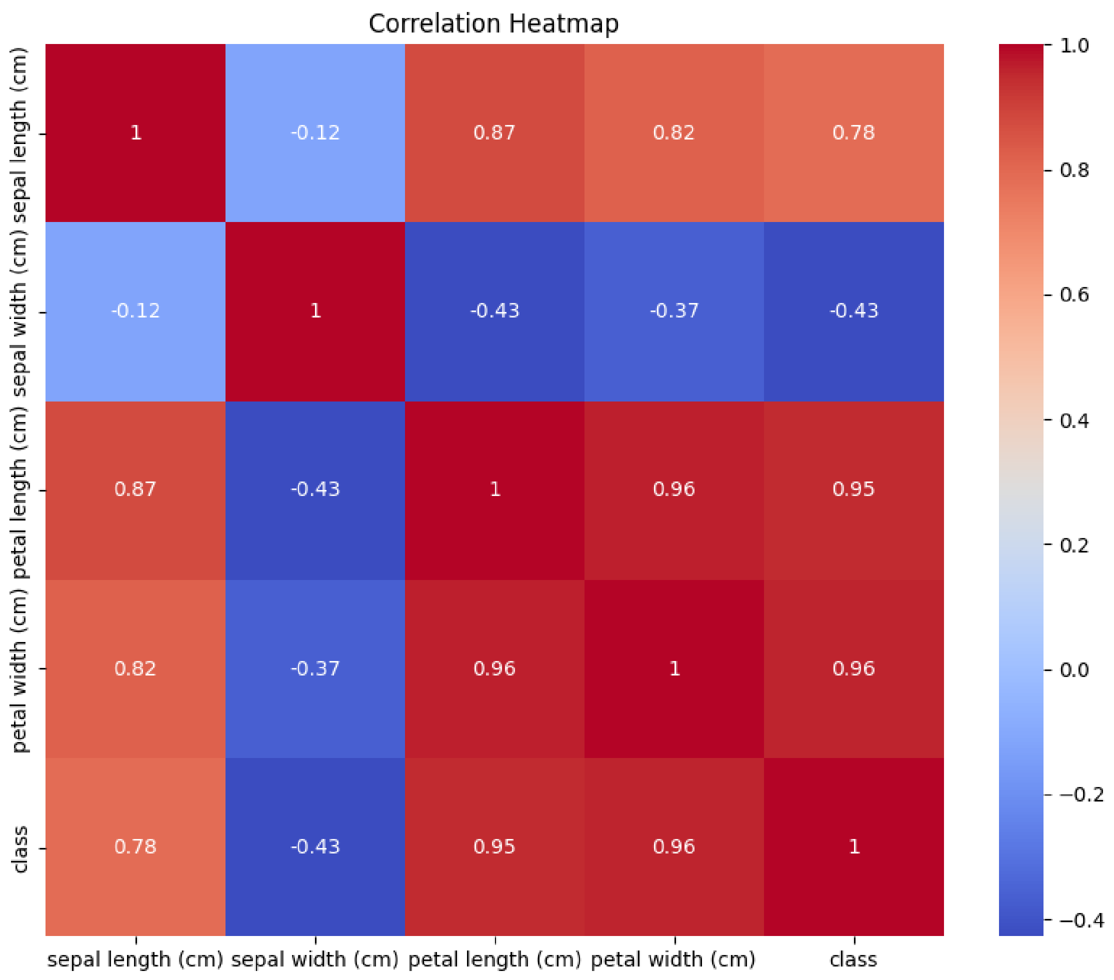
[251]:
```
     sepal length (cm)  sepal width (cm)  petal length (cm)  petal width (cm)  \
0                  5.1               3.5                1.4               0.2
1                  4.9               3.0                1.4               0.2
2                  4.7               3.2                1.3               0.2
3                  4.6               3.1                1.5               0.2
4                  5.0               3.6                1.4               0.2
..                 …                …                 …                …
145                6.7               3.0                5.2               2.3
146                6.3               2.5                5.0               1.9
147                6.5               3.0                5.2               2.0
148                6.2               3.4                5.4               2.3
149                5.9               3.0                5.1               1.8
```

```
     class
0        0
1        0
2        0
3        0
4        0
..     …
145      2
146      2
147      2
148      2
149      2

[150 rows x 5 columns]
```

[252]:
```
# pairplot to visualize relationships
sns.pairplot(irisdf, hue='class', diag_kind='kde')
plt.show()
```



[253]:
```
# correlation heatmap
plt.figure(figsize=(10, 8))
sns.heatmap(irisdf.corr(), annot=True, cmap='coolwarm')
plt.title('Correlation Heatmap')
plt.show()
```



7 Feature selection with PCA (from scratch)

[254]:
```
# implementing PCA from scratch

# subtract the mean of each variable
X_train_mean = X_train - np.mean(X_train, axis = 0)

# calculate the covariance matrix of the mean-centered data
cov_matrix = np.cov(X_train_mean, rowvar=False)

# compute the eigen values and eigen vectors of the covariance matrix
eigen_values, eigen_vectors = np.linalg.eigh(cov_matrix)

# sort eigenvalues in desc order
sorted_index = np.argsort(eigen_values)[::-1]
sorted_eigenvalue = eigen_values[sorted_index]
```

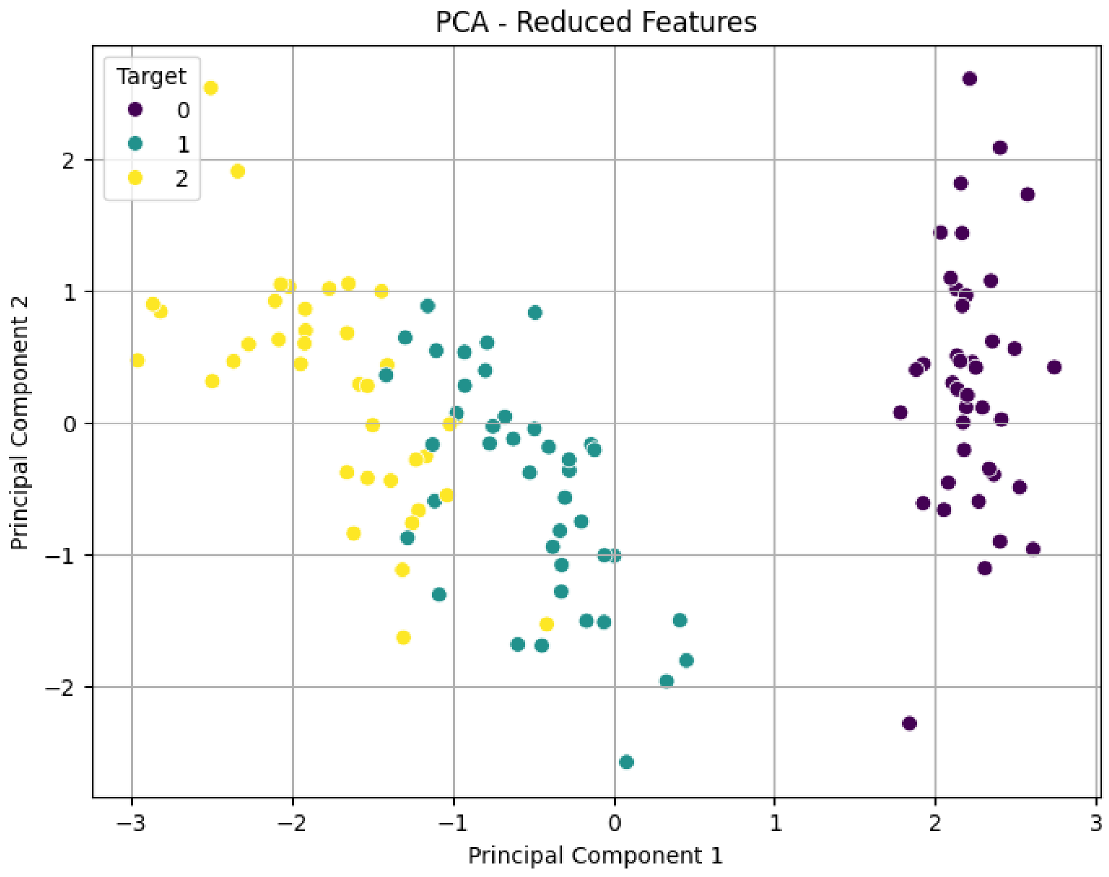
```
# sort eigenvectors according to the same index
sorted_eigenvectors = eigen_vectors[:,sorted_index]

# select a subset from the rearranged eigenvalue matrix
n_components = 2
eigenvector_subset = sorted_eigenvectors[:,0:n_components]

# transform training data
X_train_reduced = np.dot(eigenvector_subset.transpose(),X_train_mean.
 ↪transpose()).transpose()

print(X_train_mean.shape)
print(X_train_reduced.shape)
```

```
(120, 4)
(120, 2)
```

[255]:
```
# transform test data

# subtract the mean of the training data from the test data
X_test_mean = X_test - np.mean(X_train, axis=0)  # use training data mean for␣
 ↪consistency

# transform the test data
X_test_reduced = np.dot(eigenvector_subset.transpose(), X_test_mean.
 ↪transpose()).transpose()

print(X_test_mean.shape)
print(X_train_reduced.shape)
```

```
(30, 4)
(120, 2)
```

[256]:
```
# create a pd dataframe of X_reduced
pcadf = pd.DataFrame(X_train_reduced, columns=['PC1', 'PC2'])
pcadf['class'] = y_train

# plot the reduced features using a scatterplot
plt.figure(figsize=(8, 6))
sns.scatterplot(data=pcadf, x='PC1', y='PC2', hue='class', palette='viridis',␣
 ↪s=50)
plt.title('PCA - Reduced Features')
plt.xlabel('Principal Component 1')
plt.ylabel('Principal Component 2')
plt.legend(title='Target')
plt.grid(True)
plt.show()
```



8 Algorithm implementation

[257]:
```
# implement kNN from scratch
class kNNFromScratch:
    def __init__(self, k=5, distance_metric='euclidean'):
        self.k = k
        self.distance_metric = distance_metric
        self.data_points = None
        self.labels = None

    def fit(self, X, y):
        # store the training data and corresponding labels.
        self.data_points = X
        self.labels = y

    def predict(self, X):
        # predict the class for each data point in X
        predictions = [self._predict_single(x) for x in X]
        return np.array(predictions)

    def _predict_single(self, new_point):
        # predict the class for a single data point.

        # calculate the distances between the new point and all training data
        distances = pairwise_distances(self.data_points, [new_point],␣
↪metric=self.distance_metric).ravel()

        # Identify the k-nearest neighbors
        k_nearest_indices = np.argsort(distances)[:self.k]
        k_nearest_labels = self.labels[k_nearest_indices]

        # Perform majority voting
        most_common = Counter(k_nearest_labels).most_common(1)
        return most_common[0][0]

    # methods to support GridSearchCV
    def get_params(self, deep=True):
        return {"k": self.k, "distance_metric": self.distance_metric}

    def set_params(self, **parameters):
        for parameter, value in parameters.items():
            setattr(self, parameter, value)
        return self

knn = kNNFromScratch(k=5)
```

[258]:
```
# implement naive bayes from scratch
class NaiveBayesFromScratch:
    def __init__(self, smoothing=1e-9):
        # set a default smoothing value
        self.smoothing = smoothing

    def fit(self, X, y):
        # get number of samples and features
        n_samples, n_features = X.shape

        # identify the unique class labels
        self.classes = np.unique(y)

        # arrays to store mean, variance, and priors for each class
        self.mean = np.zeros((len(self.classes), n_features), dtype=np.float64)
        self.var = np.zeros((len(self.classes), n_features), dtype=np.float64)
        self.priors = np.zeros(len(self.classes), dtype=np.float64)

        # calculate mean, variance, and prior
```

```python
        for idx, c in enumerate(self.classes):
            X_c = X[y == c]
            self.mean[idx, :] = X_c.mean(axis=0)
            self.var[idx, :] = X_c.var(axis=0)
            self.priors[idx] = X_c.shape[0] / float(n_samples)

    def predict(self, X):
        # predict the class for each sample in X
        y_pred = [self._predict(x) for x in X]
        return np.array(y_pred)

    def _predict(self, x):
        # calculate the posterior probability for each class
        posteriors = []

        for idx, c in enumerate(self.classes):
            # calculate the log prior probability
            prior = np.log(self.priors[idx])

            # calculate the log likihood using the class's mean and variance
            class_conditional = np.sum(np.log(self._pdf(idx, x)))

            # combine prior and likelihood to get the posterior
            posterior = prior + class_conditional
            posteriors.append(posterior)

        # return class with highest posterior probabililty
        return self.classes[np.argmax(posteriors)]

    def _pdf(self, class_idx, x):
        # calculate the probabiliy density function for a gaussian distribution
        mean = self.mean[class_idx]
        var = self.var[class_idx]
        numerator = np.exp(-((x - mean) ** 2) / (2 * var))
        denominator = np.sqrt(2 * np.pi * var)
        return numerator / denominator

    # methods to support GridSearchCV
    def get_params(self, deep=True):
        # Return smoothing as a tunable parameter
        return {"smoothing": self.smoothing}

    def set_params(self, **parameters):
        # Update smoothing if provided
        for parameter, value in parameters.items():
            setattr(self, parameter, value)
        return self
```

```

nb = NaiveBayesFromScratch()
```

[259]:
```
# implement logistic regression from sklearn
logreg = LogisticRegression()
```

[260]:
```
# implement decistion tree from sklearn
dectree = DecisionTreeClassifier(random_state=42)
```

9 5. Results

10 Cross-validation

[261]:
```
# cross validation

# method to calculate cross validation accuray score
def cross_validate_model_from_scratch(model, X, y, cv):
    kf = KFold(n_splits=cv, shuffle=True, random_state=42)
    scores = []

    for train_index, test_index in kf.split(X):
        X_train, X_test = X[train_index], X[test_index]
        y_train, y_test = y[train_index], y[test_index]

        model.fit(X_train, y_train)
        y_pred = model.predict(X_test)
        scores.append(accuracy_score(y_test, y_pred))

    return np.mean(scores)
```

[262]:
```
# evaluate on both original and PCA-reduced datasets
models = {
    "kNN": knn,
    "Naive Bayes": nb,
    "Logistic Regression": logreg,
    "Decision Tree": dectree
}

# print accuracy score for original and pca data
print("Cross-Validation accuracy score (Original Data):")
for name, model in models.items():
    score = cross_validate_model_from_scratch(model, X_train, y_train, cv=5)
    print(f"{name}: {score:.4f}")

print("\nCross-Validation accuracy score (PCA-Reduced Data):")
for name, model in models.items():
    score = cross_validate_model_from_scratch(model, X_train_reduced, y_train,␣
↪cv=5)
    print(f"{name}: {score:.4f}")
```

```
Cross-Validation accuracy score (Original Data):
kNN: 0.9417
Naive Bayes: 0.9417
Logistic Regression: 0.9333
Decision Tree: 0.9167

Cross-Validation accuracy score (PCA-Reduced Data):
kNN: 0.9167
Naive Bayes: 0.8667
Logistic Regression: 0.9000
Decision Tree: 0.9333
```

11 Hyperparameter tuning

[263]:
```
# tuning number of neighers in knn
param_grid_knn = {'n_neighbors': range(1, 15)}
grid_knn = GridSearchCV(knn, param_grid_knn, cv=5, scoring='accuracy')
grid_knn.fit(X_train, y_train)
best_knn = grid_knn.best_estimator_

# tuning gaussian distribution for nb
param_grid_nb = {'var_smoothing': [1e-9, 1e-8, 1e-7, 1e-6, 1e-5]}
grid_nb = GridSearchCV(nb, param_grid_nb, cv=5, scoring='accuracy')
grid_nb.fit(X_train, y_train)
best_nb = grid_nb.best_estimator_

# tuning inverse of regularisation strength for logistic regression
param_grid_logreg = {'C': [-1, 1, 10]}
grid_logreg = GridSearchCV(logreg, param_grid_logreg, cv=5, scoring='accuracy')
grid_logreg.fit(X_train, y_train)
best_logreg = grid_logreg.best_estimator_

# tuning max depth for decision tree
param_grid_dectree = {'max_depth': range(1, 11)}
grid_dectree = GridSearchCV(dectree, param_grid_dectree, cv=5,␣
 ↪scoring='accuracy')
grid_dectree.fit(X_train, y_train)
best_dectree = grid_dectree.best_estimator_
```

```
c:\Users\Admin\anaconda3\Lib\site-
packages\sklearn\model_selection\_validation.py:540: FitFailedWarning:
5 fits failed out of a total of 15.
The score on these train-test partitions for these parameters will be set to
```

```
nan.
If these failures are not expected, you can try to debug them by setting
error_score='raise'.

Below are more details about the failures:
--------------------------------------------------------------------------------
5 fits failed with the following error:
Traceback (most recent call last):
  File "c:\Users\Admin\anaconda3\Lib\site-
packages\sklearn\model_selection\_validation.py", line 888, in _fit_and_score
    estimator.fit(X_train, y_train, **fit_params)
  File "c:\Users\Admin\anaconda3\Lib\site-packages\sklearn\base.py", line 1466,
in wrapper
    estimator._validate_params()
  File "c:\Users\Admin\anaconda3\Lib\site-packages\sklearn\base.py", line 666,
in _validate_params
    validate_parameter_constraints(
  File "c:\Users\Admin\anaconda3\Lib\site-
packages\sklearn\utils\_param_validation.py", line 95, in
validate_parameter_constraints
    raise InvalidParameterError(
sklearn.utils._param_validation.InvalidParameterError: The 'C' parameter of
LogisticRegression must be a float in the range (0.0, inf]. Got -1 instead.

  warnings.warn(some_fits_failed_message, FitFailedWarning)
c:\Users\Admin\anaconda3\Lib\site-
packages\sklearn\model_selection\_search.py:1103: UserWarning: One or more of
the test scores are non-finite: [       nan 0.95833333 0.95833333]
  warnings.warn(
```

[264]:
```
# update hyperparameter tuned models
models = {
    "kNN": best_knn,
    "Naive Bayes": best_nb,
    "Logistic Regression": best_logreg,
    "Decision Tree": best_dectree
}
```

12 Training and evaluation

[265]:
```
# training using optimal hyper parameters
def evaluate_model(model, X_train, X_test, y_train, y_test):
    model.fit(X_train, y_train)
    y_pred = model.predict(X_test)
    return {
        "Accuracy": round(accuracy_score(y_test, y_pred), 4),
        "F1 Score": round(f1_score(y_test, y_pred, average='weighted'), 4),
        "Precision": round(precision_score(y_test, y_pred, average='weighted'),␣
↪4),
        "Recall": round(recall_score(y_test, y_pred, average='weighted'), 4)
    }

print("\nEvaluation Results (Original Data):")
for name, model in models.items():
    results = evaluate_model(model, X_train, X_test, y_train, y_test)
    print(f"{name}: {results}")

print("\nEvaluation Results (PCA-Reduced Data):")
for name, model in models.items():
    results = evaluate_model(model, X_train_reduced, X_test_reduced, y_train,␣
↪y_test)
    print(f"{name}: {results}")
```

```
Evaluation Results (Original Data):
kNN: {'Accuracy': 1.0, 'F1 Score': 1.0, 'Precision': 1.0, 'Recall': 1.0}
Naive Bayes: {'Accuracy': 1.0, 'F1 Score': 1.0, 'Precision': 1.0, 'Recall': 1.0}
Logistic Regression: {'Accuracy': 1.0, 'F1 Score': 1.0, 'Precision': 1.0,
'Recall': 1.0}
Decision Tree: {'Accuracy': 1.0, 'F1 Score': 1.0, 'Precision': 1.0, 'Recall':
1.0}

Evaluation Results (PCA-Reduced Data):
kNN: {'Accuracy': 0.9333, 'F1 Score': 0.9333, 'Precision': 0.9333, 'Recall':
0.9333}
Naive Bayes: {'Accuracy': 0.9667, 'F1 Score': 0.9664, 'Precision': 0.9694,
'Recall': 0.9667}
Logistic Regression: {'Accuracy': 0.9, 'F1 Score': 0.8992, 'Precision': 0.9014,
'Recall': 0.9}
Decision Tree: {'Accuracy': 0.8667, 'F1 Score': 0.8667, 'Precision': 0.8667,
'Recall': 0.8667}
```

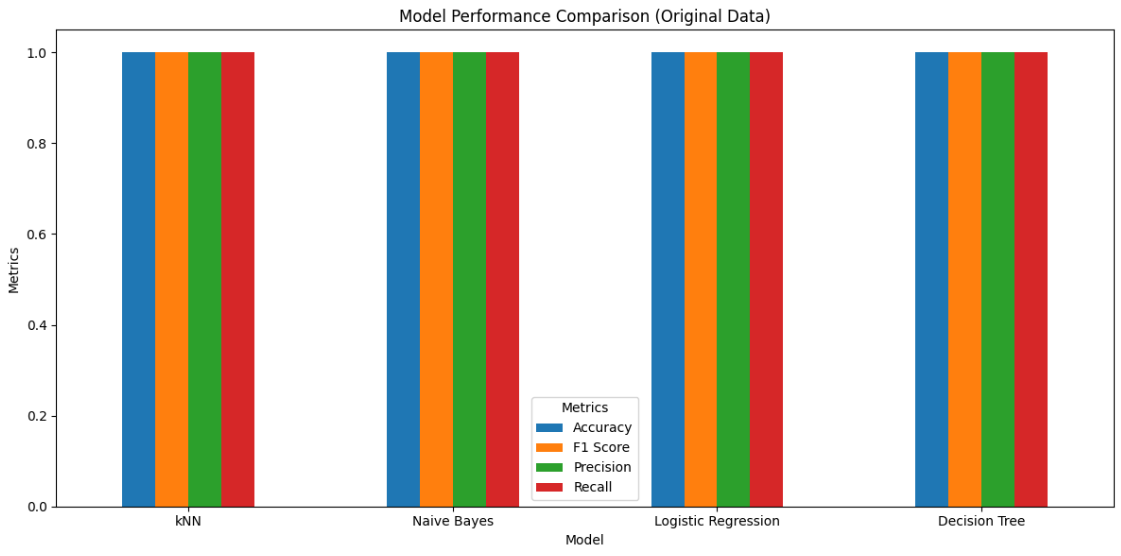
13 Model comparison

[266]:
```
# print comparison metrics for both original and PCA-reduced datasets

print("Model Comparison (Original Data):")
comparison_original = {}
for name, model in models.items():
    results = evaluate_model(model, X_train, X_test, y_train, y_test)
    comparison_original[name] = results

# Print comparison table for original data
for metric in ["Accuracy", "F1 Score", "Precision", "Recall"]:
    print(f"\n{metric} Comparison (Original Data):")
    for model_name, metrics in comparison_original.items():
        print(f"{model_name}: {metrics[metric]:.2f}")

print("\nModel Comparison (PCA-Reduced Data):")
comparison_pca = {}
for name, model in models.items():
    results = evaluate_model(model, X_train_reduced, X_test_reduced, y_train,␣
 ↪y_test)
    comparison_pca[name] = results

# Print comparison table for PCA-reduced data
for metric in ["Accuracy", "F1 Score", "Precision", "Recall"]:
    print(f"\n{metric} Comparison (PCA-Reduced Data):")
    for model_name, metrics in comparison_pca.items():
        print(f"{model_name}: {metrics[metric]:.2f}")
```

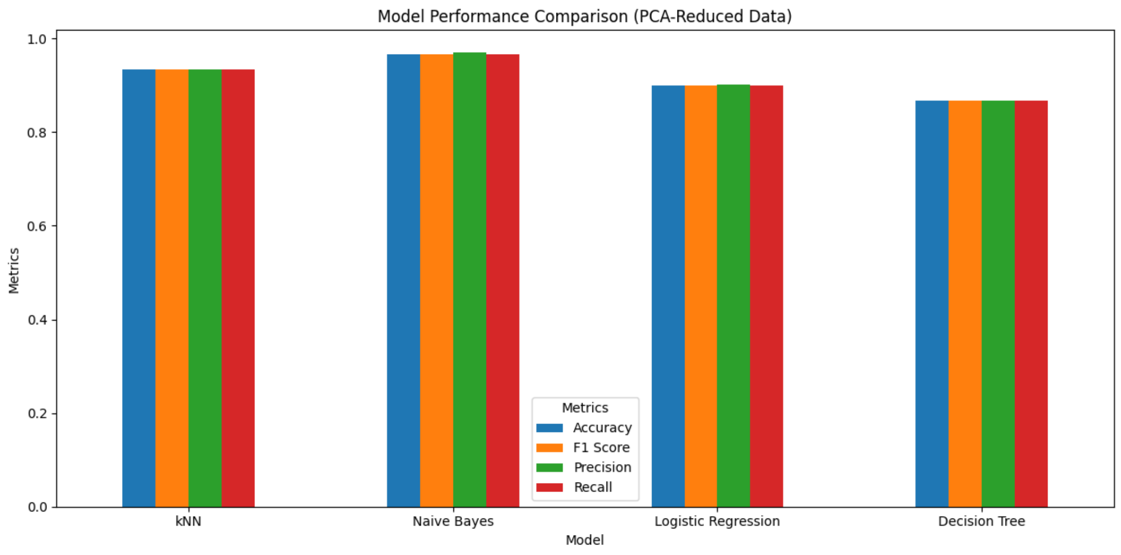
```
Model Comparison (Original Data):

Accuracy Comparison (Original Data):
kNN: 1.00
Naive Bayes: 1.00
Logistic Regression: 1.00
Decision Tree: 1.00

F1 Score Comparison (Original Data):
kNN: 1.00
Naive Bayes: 1.00
Logistic Regression: 1.00
Decision Tree: 1.00

Precision Comparison (Original Data):
kNN: 1.00
Naive Bayes: 1.00
Logistic Regression: 1.00
Decision Tree: 1.00

Recall Comparison (Original Data):
kNN: 1.00
Naive Bayes: 1.00
Logistic Regression: 1.00
Decision Tree: 1.00

Model Comparison (PCA-Reduced Data):

Accuracy Comparison (PCA-Reduced Data):
kNN: 0.93
Naive Bayes: 0.97
```

```
Logistic Regression: 0.90
Decision Tree: 0.87

F1 Score Comparison (PCA-Reduced Data):
kNN: 0.93
Naive Bayes: 0.97
Logistic Regression: 0.90
Decision Tree: 0.87

Precision Comparison (PCA-Reduced Data):
kNN: 0.93
Naive Bayes: 0.97
Logistic Regression: 0.90
Decision Tree: 0.87

Recall Comparison (PCA-Reduced Data):
kNN: 0.93
Naive Bayes: 0.97
Logistic Regression: 0.90
Decision Tree: 0.87
```

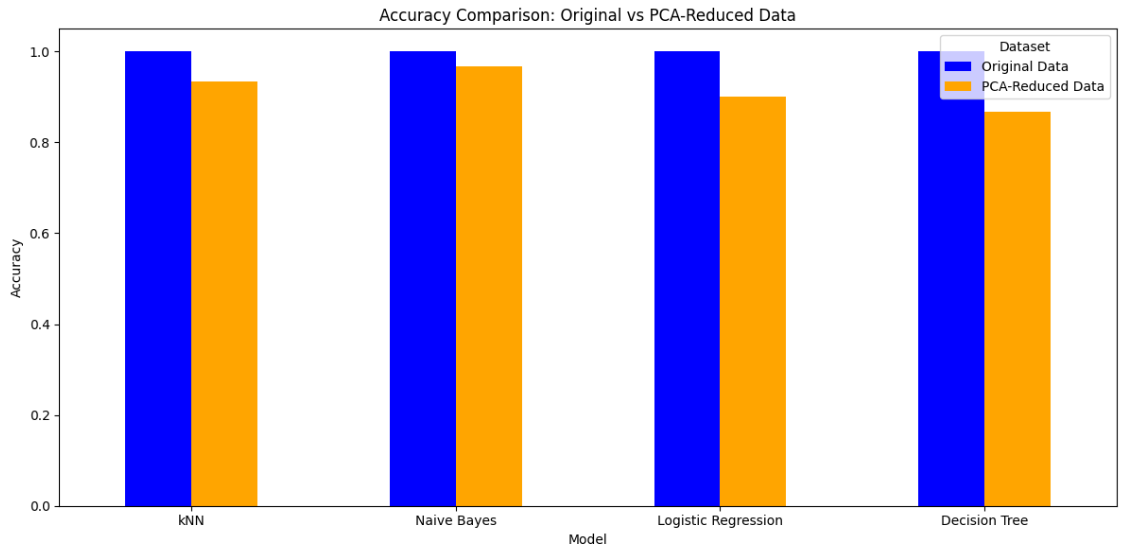
[267]:
```
# convert to df for visualization
df_original = pd.DataFrame(comparison_original).T
df_pca = pd.DataFrame(comparison_pca).T

# plot original data results
df_original.plot(kind="bar", figsize=(12, 6))
plt.title("Model Performance Comparison (Original Data)")
plt.xlabel("Model")
plt.ylabel("Metrics")
plt.xticks(rotation=0)
plt.legend(title="Metrics")
plt.tight_layout()
plt.show()
```



```
[268]: # plot PCA data results
df_pca.plot(kind="bar", figsize=(12, 6))
plt.title("Model Performance Comparison (PCA-Reduced Data)")
plt.xlabel("Model")
plt.ylabel("Metrics")
plt.xticks(rotation=0)
plt.legend(title="Metrics")
plt.tight_layout()
plt.show()
```



[269]:
```
# plot comparison between original and PCA data
accuracy_comparison = pd.DataFrame({
    "Original Data": df_original["Accuracy"],
    "PCA-Reduced Data": df_pca["Accuracy"]
})

accuracy_comparison.plot(kind="bar", figsize=(12, 6), color=["blue", "orange"])
plt.title("Accuracy Comparison: Original vs PCA-Reduced Data")
plt.xlabel("Model")
plt.ylabel("Accuracy")
plt.xticks(rotation=0)
plt.legend(title="Dataset", loc="best")
plt.tight_layout()
plt.show()
```



14 Feature importance analysis

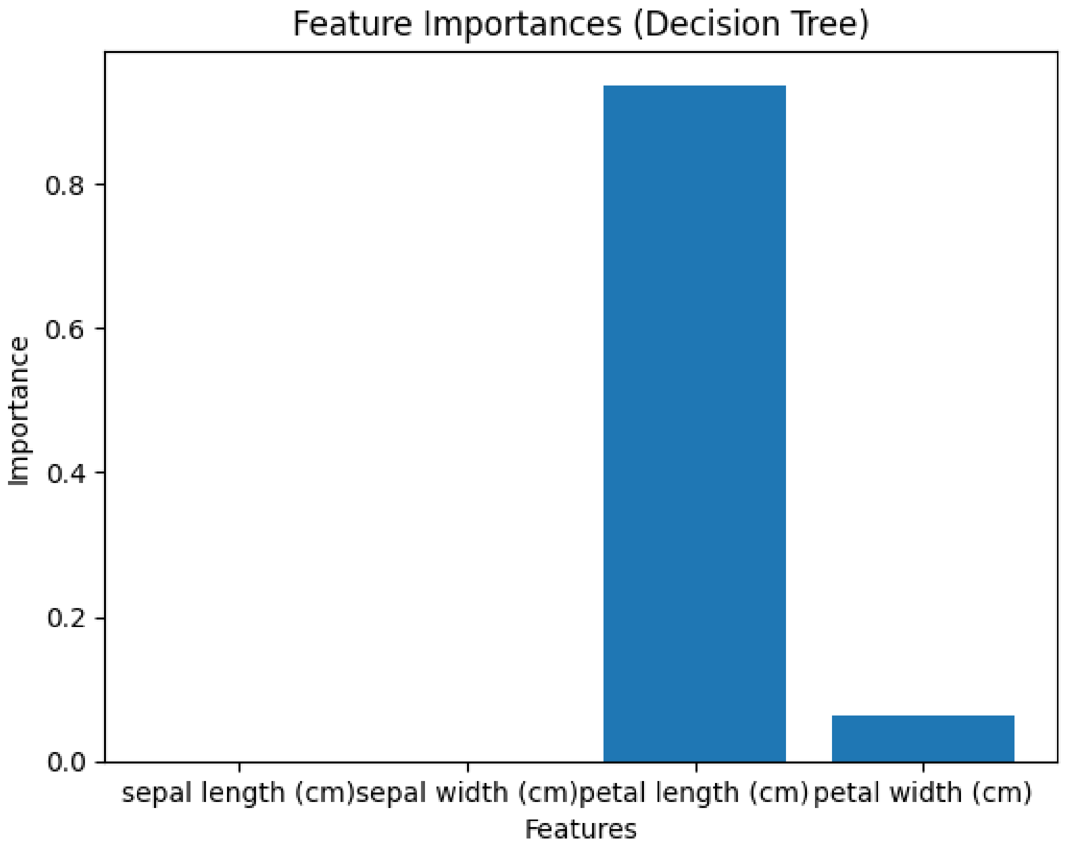
feature importance analysis will not be done for knn and nb models as they were built from scratch.

[270]:
```
irisdf.columns

feature_names = irisdf.columns[:-1] # excluding last column
```

[271]:
```
# dectree feature importance analysis
print("Feature Importance (Decision Tree):")
best_dectree.fit(X_train, y_train)
importance = best_dectree.feature_importances_
```

```
for name, score in zip(feature_names, importance):
    print(f"{name}: Importance {score:.2f}")

# Visualize results
plt.bar(feature_names, importance)
plt.title('Feature Importances (Decision Tree)')
plt.xlabel('Features')
plt.ylabel('Importance')
plt.show()
```

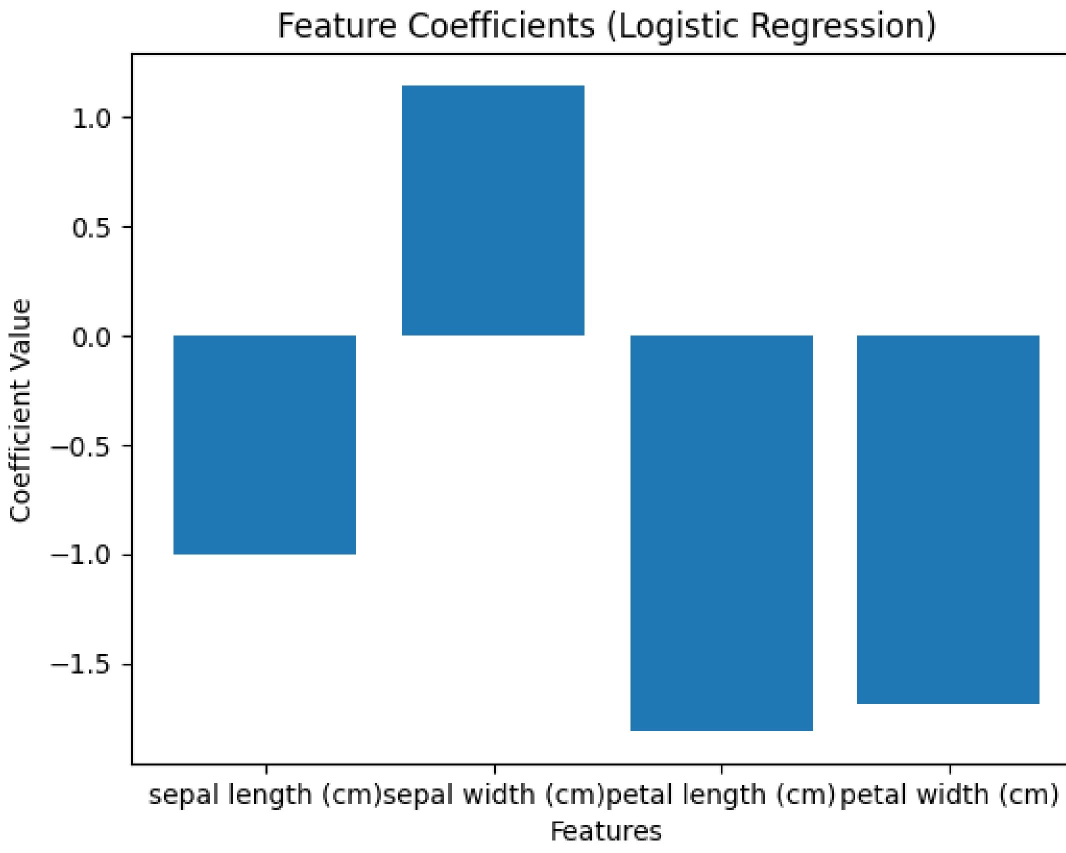
```
Feature Importance (Decision Tree):
sepal length (cm): Importance 0.00
sepal width (cm): Importance 0.00
petal length (cm): Importance 0.94
petal width (cm): Importance 0.06
```



[272]:
```
# logreg feature importance analysis
print("\nFeature Coefficients (Logistic Regression):")
best_logreg.fit(X_train, y_train)
coef = best_logreg.coef_[0]  # coefficients for each feature
```

```python
for name, c in zip(feature_names, coef):
    print(f"{name}: Coefficient {c:.2f}")

# Visualize results
plt.bar(feature_names, coef)
plt.title('Feature Coefficients (Logistic Regression)')
plt.xlabel('Features')
plt.ylabel('Coefficient Value')
plt.show()
```

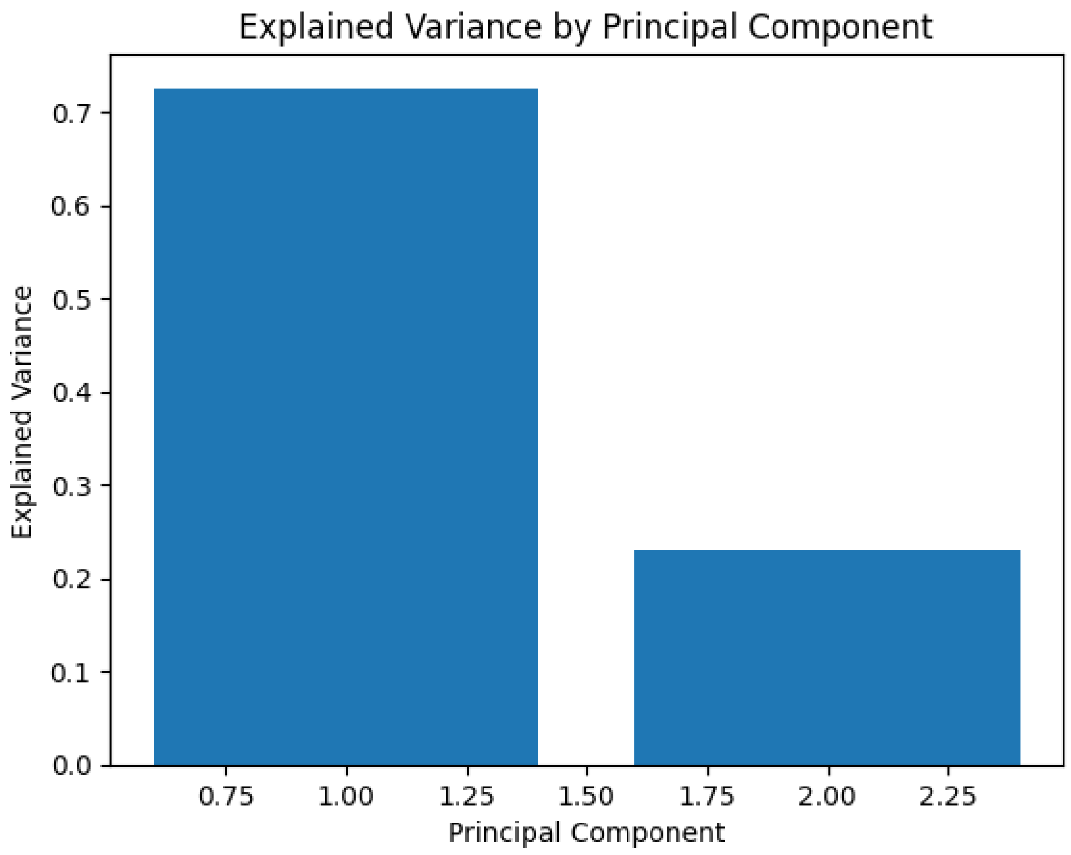
```
Feature Coefficients (Logistic Regression):
sepal length (cm): Coefficient -1.00
sepal width (cm): Coefficient 1.14
petal length (cm): Coefficient -1.81
petal width (cm): Coefficient -1.69
```



[273]:
```python
# PCA component contributions (used by all models for PCA-reduced data)
print("PCA Explained Variance (Training Data):")
```

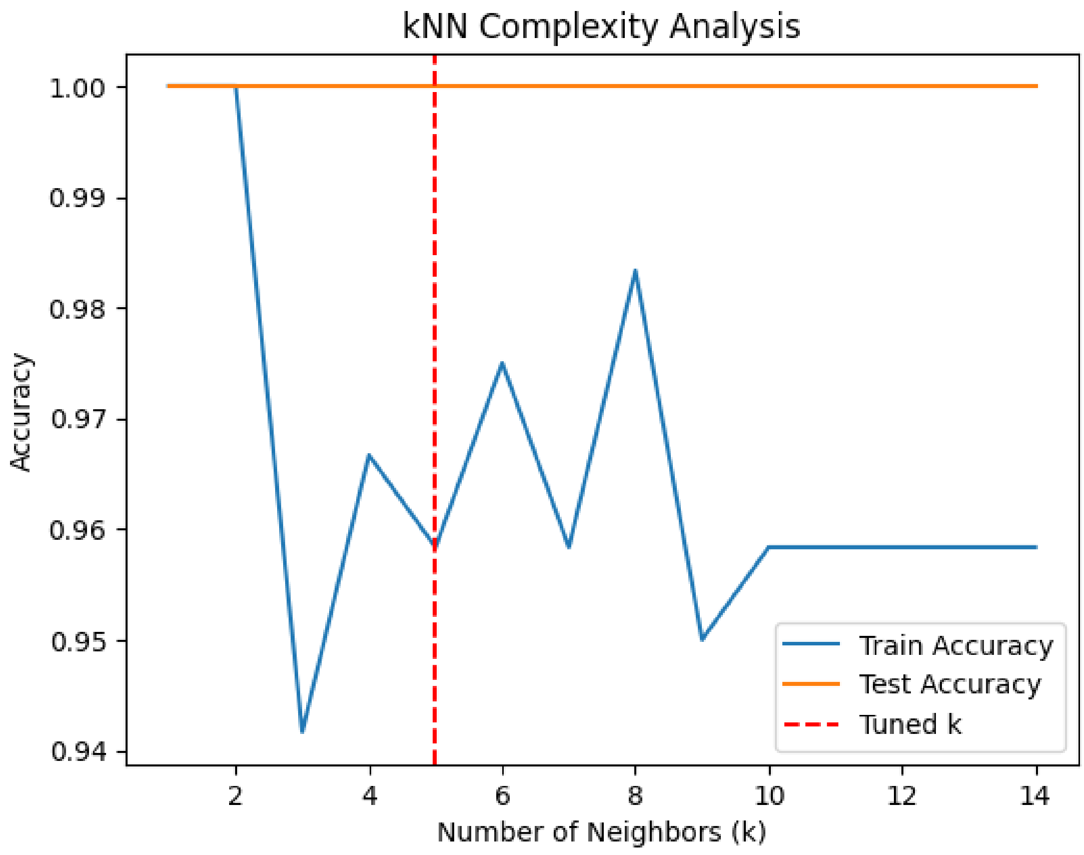
```python
explained_variance = sorted_eigenvalue / np.sum(sorted_eigenvalue)
for i, v in enumerate(explained_variance[:n_components]):
    print(f"Principal Component {i+1}: Explained Variance {v:.2f}")

# visualize PCA Contribution
plt.bar(range(1, n_components + 1), explained_variance[:n_components])
plt.title('Explained Variance by Principal Component')
plt.xlabel('Principal Component')
plt.ylabel('Explained Variance')
plt.show()
```

```
PCA Explained Variance (Training Data):
Principal Component 1: Explained Variance 0.73
Principal Component 2: Explained Variance 0.23
```

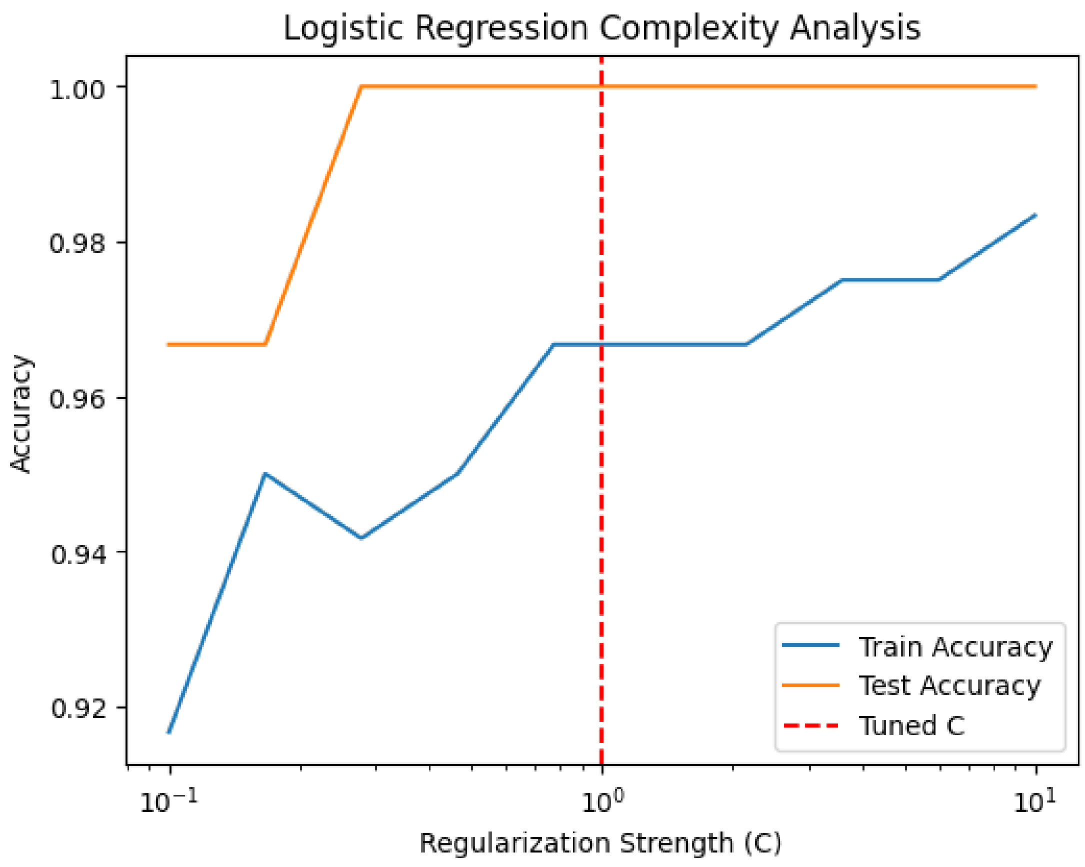


15 Model complexity analysis

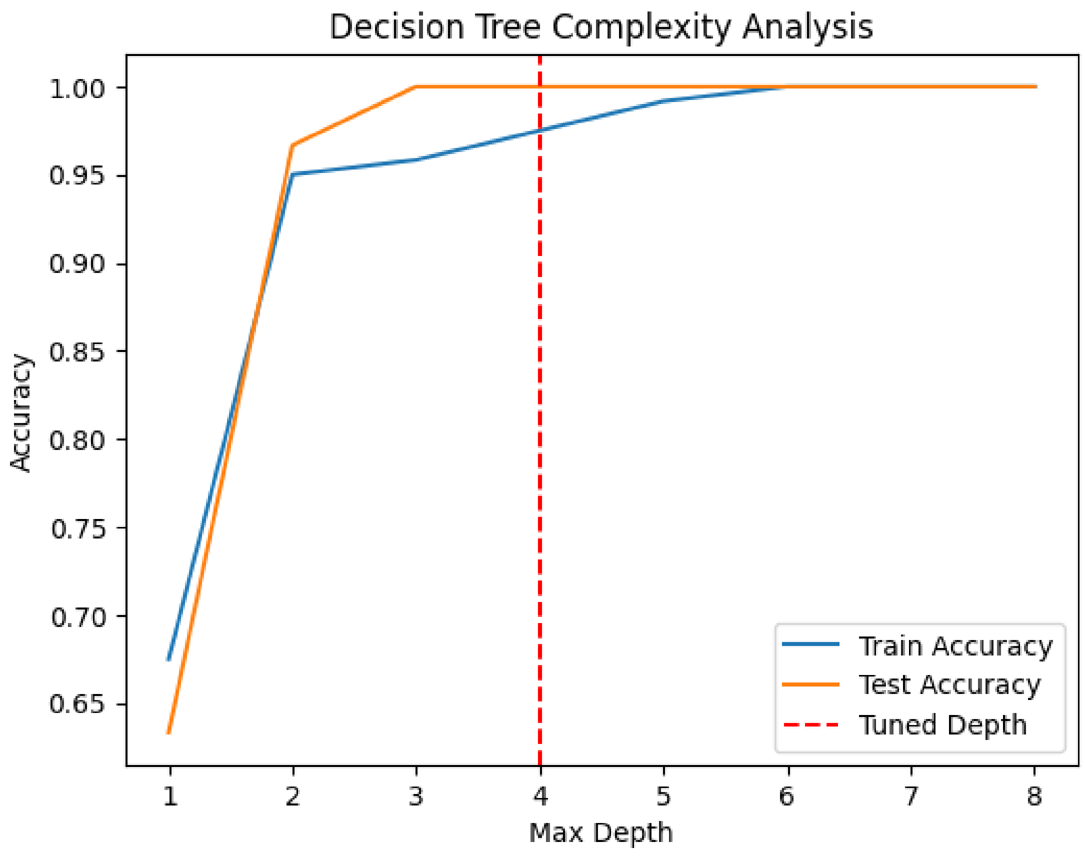
[274]:
```
# kNN complexity analysis
ks = range(1, best_knn.k + 10)  # extend number of neighbours
train_scores, test_scores = [], []

for k in ks:
    knn.k = k
    knn.fit(X_train, y_train)
    train_scores.append(accuracy_score(y_train, knn.predict(X_train)))
    test_scores.append(accuracy_score(y_test, knn.predict(X_test)))

plt.plot(ks, train_scores, label='Train Accuracy')
plt.plot(ks, test_scores, label='Test Accuracy')
plt.axvline(x=best_knn.k, color='red', linestyle='--', label='Tuned k')
plt.title('kNN Complexity Analysis')
plt.xlabel('Number of Neighbors (k)')
plt.ylabel('Accuracy')
plt.legend()
plt.show()
```



[281]:
```python
# logistic regression complexity analysis
Cs = np.logspace(-1, 1, 10)  # explore regularization strength
train_scores, test_scores = [], []

for C in Cs:
    model = LogisticRegression(C=C)
    model.fit(X_train, y_train)
    train_scores.append(accuracy_score(y_train, model.predict(X_train)))
    test_scores.append(accuracy_score(y_test, model.predict(X_test)))

plt.semilogx(Cs, train_scores, label='Train Accuracy')
plt.semilogx(Cs, test_scores, label='Test Accuracy')
plt.axvline(x=best_logreg.C, color='red', linestyle='--', label='Tuned C')
plt.title('Logistic Regression Complexity Analysis')
plt.xlabel('Regularization Strength (C)')
plt.ylabel('Accuracy')
plt.legend()
plt.show()
```



[277]:
```python
# decision tree model complexity analysis

depths = range(1, best_dectree.max_depth + 5)  # extend range around the tuned␣
 ↪depth
train_scores, test_scores = [], []

for depth in depths:
    model = DecisionTreeClassifier(max_depth=depth)
    model.fit(X_train, y_train)
    train_scores.append(accuracy_score(y_train, model.predict(X_train)))
    test_scores.append(accuracy_score(y_test, model.predict(X_test)))

plt.plot(depths, train_scores, label='Train Accuracy')
plt.plot(depths, test_scores, label='Test Accuracy')
plt.axvline(x=best_dectree.max_depth, color='red', linestyle='--', label='Tuned␣
 ↪Depth')
plt.title('Decision Tree Complexity Analysis')
plt.xlabel('Max Depth')
plt.ylabel('Accuracy')
plt.legend()
plt.show()
```



16 5. Results

Model	Dataset	Accuracy	Precision	Recall	F1 Score
kNN	Original	1	1	1	1
kNN	PCA-Reduced	0.93	0.93	0.93	0.93
Naive Bayes	Original	1	1	1	1
Naive Bayes	PCA-Reduced	0.97	0.97	0.97	0.97
Logistic Regression	Original	1	1	1	1
Logistic Regression	PCA-Reduced	0.90	0.90	0.90	0.90
Decision Tree	Original	1	1	1	1
Decision Tree	PCA-Reduced	0.87	0.87	0.87	0.87

Feature importance analysis

- Decision tree:
  Petal length showed the most impact on predictability with an importance score of 0.94, followed by petal width with an importance score of 0.06, while sepal length and width score 0. These findings strongly correlate with the class separability of the dataset.
- Logistic Regression:
  Similar to the decision tree model's feature importance analysis, the petal length feature has the highest magnitude in coefficient scores with a score of 1.81, followed by petal width with a score of 1.69. However, unlike decision tree, the coefficient scores for sepal length and width are not completely negligible with scores of 1 and 1.14. This shows that applying PCA might cause a decrease in performance as features are reduced.
- PCA Analysis:
  By adding up the values of both principal components variance scores, we can tell that the PCA-reduced dataset retained approximately 97% of the variance, making it an efficient representation of the data.

Models performed slightly worse on the PCA-reduced dataset, for example the kNN model got a score of 0.93 across all 4 metrics on the PCA-reduced data, a drop from a score of 1 across all metrics. The same for the rest of the models, dropped from 1 to 0.97 for the nb model, 1 to 0.9 for logistic regression model and 1 to 0.87 for decision tree model. From this, we can tell that PCA affects logistic regression and decision tree more than the other 2 models.

17 6. Evaluation

Model Complexity Analysis

- kNN:

The analysis of the number of neighbours, k, showed that a value of 2 led to overfitting, as the model captured noise in the training data. As k increases, training accuracy generally decreases, this is expected as a larger number of neighbours would make the model a lot more generalised. Tuned k is valued at 5, which is to be expected as the iris dataset only has 150 samples, which is very small. The number of features is even lesser, at only 4, and having too large a number of neighbours would only make the model overly generalised.

- Logistic Regression:

The analysis of regularization strength (C) showed that smaller values of C, which means stronger regularization, simplified the model, reducing overfitting, while larger values increased model complexity but could lead to overfitting.

- Decision Tree:

The analysis of the max depth for the decision tree model showed that max depths of under 2 led to underfitting, indicated by both train and test accuracy being low. While max depths of 6 and above led to over fitting, as training accuracy scored perfectly. Tuned max depth is 4, this is expected as there are only 3 types to classify.

Impact of PCA

- PCA helped simplify the dataset while retaining most of the variance. While this reduced dimensionality improved computational efficiency, it resulted in slightly lower model performance as critical feature-specific information was lost.
- Models like kNN and naive bayes were more robust to PCA-reduced data, while logistic regression and decision tree models showed slightly reduced accuracy due to their reliance on individual feature contributions.

18 7. Conclusions

This report highlights the performance of 4 machine learning models, k-nearest neighbour, naive bayes, logistic regression, and decision tree on both original and PCA-reduced datasets.

First of the key findings, is that all 4 models performed perfectly on the original dataset. This shows how the iris dataset's linear separability and simplicity allowed the models to classify them perfectly.

Second of the key findings, is that kNN and naive bayes were more robust to the PCA-reduced data achieving high accuracy scores of 0.93 and 0.97, as compared to the logistic regression and decision tree models that did not perform as well, with accuracy scores of 0.9 and 0.87.

Third of the key findings, is that performing PCA on the iris dataset is effective. Although reducing accuracy slightly for all 4 models, having only 2 principal components to compute reduces processing time. This is especially true for models like kNN, where lesser features meant lesser or faster calculations of distance.

Fourth of the key findings, is that feature importance analysis revealed that the petal-related features, petal length and width, contributed the most towards predictability.

Fifth and last of the key findings, is that model complexity analysis highlighted the importance of tuning hyperparameters to balance overfitting and underfitting. For example, the tuned parameters for kNN and decision tree were k=5 and max depth=5. These tuned parameters provied the best balance between training and test accuracy.

In conclusion, this project showed the effectiveness of machine learning algorithms and dimensionality reduction techniques in combination to achieve computational efficiency and robust classification performance. The insights further emphasise the importance of proper EDA, hyperparameter tuning, and feature analysis in building models that are reliable and easy to understand.

19 8. References

1. https://www.datastax.com/guides/what-is-k-nearest-neighbors-knn-algorithm
2. https://www.geeksforgeeks.org/naive-bayes-classifiers/
3. https://www.geeksforgeeks.org/understanding-logistic-regression/
4. https://www.geeksforgeeks.org/decision-tree-algorithms/