Machine Learning Model Projects

On Open-Source DataSet

By

Hemanth Rasabhathula

Date : Apr-27-2024

Table of Contents

1. Machine Learning on public datasets.

The DataSet Description

- **Palmer Penguin DataSet**:

The datasets contain data for 344 penguins. There are 3 different species of penguins in this dataset namely Adelie, Gentoo, Chinstrap. This dataset is a collection of measurements and observations related to these species, their physical characteristics such as bill_length_mm, bill_depth_mm, flipper_length_mm, body_mass_g, sex, and year of observation across 3 islands in the Palmer Archipelago, Antarctica. Each row represents an individual penguin that was observed and measured.

Sample data:

Species	island	bill_length_mm	bill_depth_mm	flipper_length_mm	body_mass_g	sex	year
Adelie	Torgersen	39.1	18.7	181	3750	male	2007
Adelie	Torgersen	39.5	17.4	186	3800	female	2007
Adelie	Torgersen	40.3	18	195	3250	female	2007
Adelie	Biscoe	43.2	19	197	4775	male	2009
Adelie	Torgersen	36.7	19.3	193	3450	female	2007
Gentoo	Biscoe	45.7	13.9	214	4400	female	2008
Gentoo	Biscoe	54.3	15.7	231	5650	male	2008
Gentoo	Biscoe	45.8	14.2	219	4700	female	2008
Gentoo	Biscoe	49.8	16.8	230	5700	male	2008
Gentoo	Biscoe	46.2	14.4	214	4650	male	2008
Chinstrap	Dream	42.5	17.3	187	3350	female	2009
Chinstrap	Dream	52.2	18.8	197	3450	male	2009
Chinstrap	Dream	45.2	16.6	191	3250	female	2009
Chinstrap	Dream	49.3	19.9	203	4050	male	2009
Chinstrap	Dream	50.2	18.8	202	3800	male	2009

ML Models

Model 1: Decision Tree Classification on Palmer Penguins Dataset

The Decision Tree Classification model was chosen for this dataset to classify penguin species based on their 'flipper_length_mm' and 'bill_length_mm'.

a) Data Preprocessing:
   1. Handling missing values: Rows with missing values were removed.
   2. Feature Scaling: StandardScaler was used to scale the features.

b) Splitting the Dataset:
   - The dataset was split into training and test sets using a 75/25 ratio.

c) Model Training:
   - DecisionTreeClassifier was used with the specified parameters.
   - Parameters:
     - Criterion: 'entropy'
     - Random State: 0
   - The model was trained on the training set using the 'fit' method

d) Model Prediction:
   - Predictions were made on new data

e) Model Evaluation:
   - The model's performance was evaluated using a confusion matrix and accuracy score

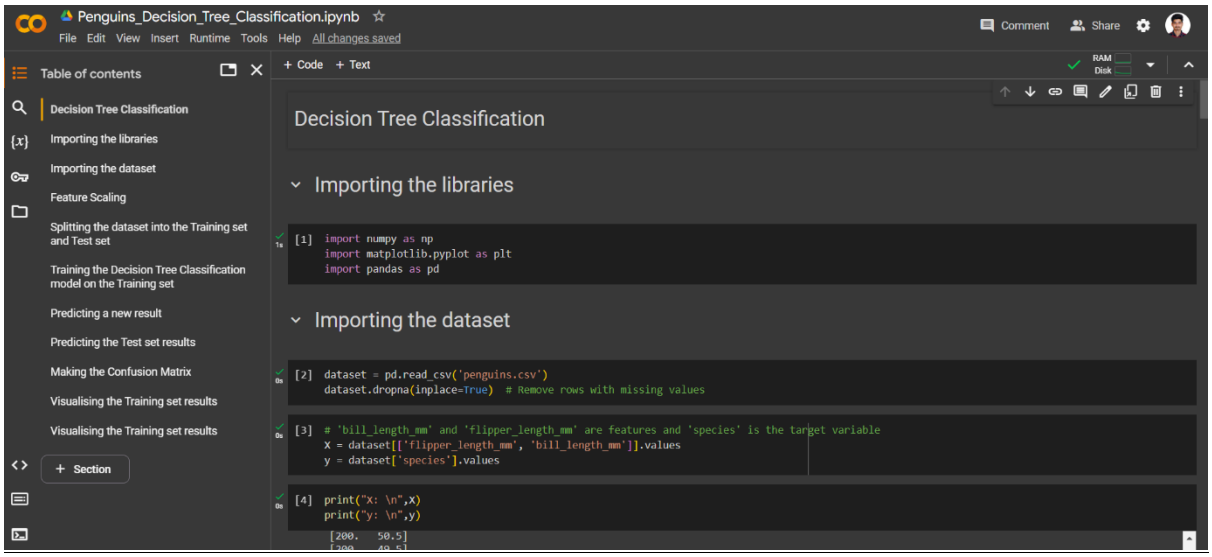
f) Visualizations:
   - Visualizations were plotted with the decision boundaries on the training and test sets
   - Flipper Length as xlabel and Bill Length as ylabel
   - **Adelie** Species is denoted by ’red', **Gentoo** Species is denoted by 'green', **Chinstrap** Species is denoted by 'blue.

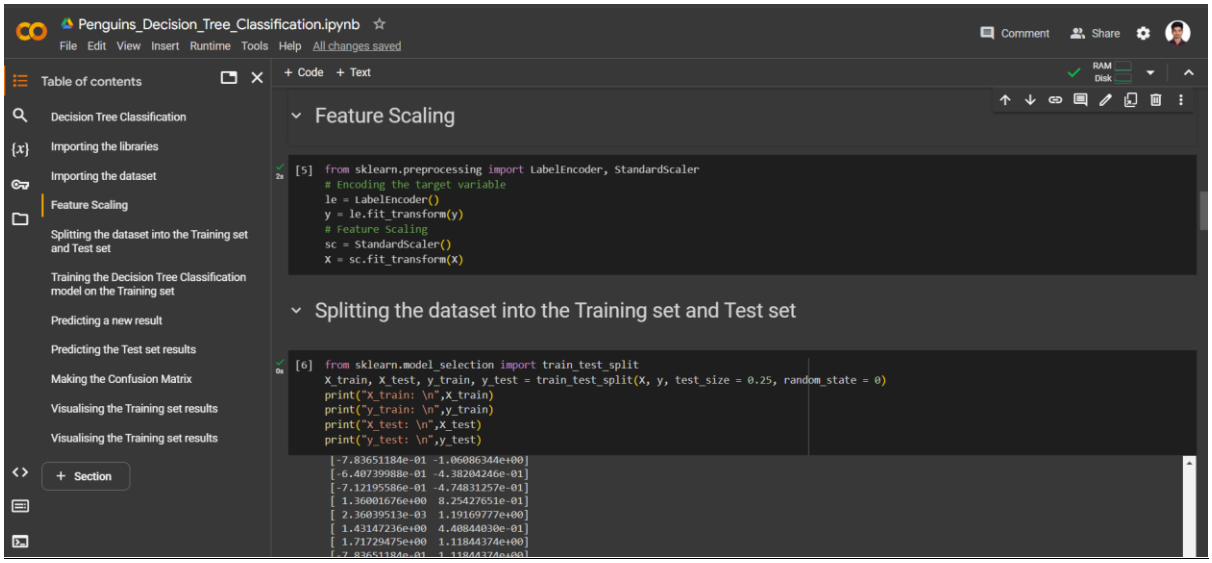
Performance

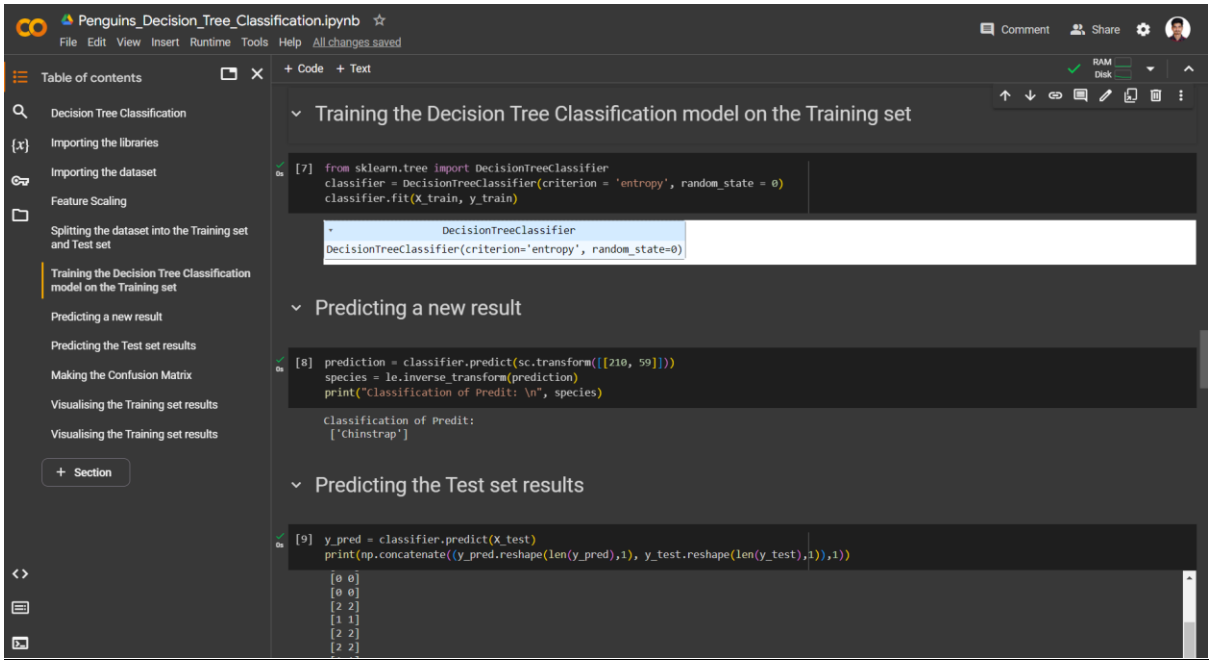
The Decision Tree Classification model achieved an accuracy score of 98.81% on the test dataset. This indicates that the model was effective in correctly classifying the penguin species based on their 'flipper_length_mm' and 'bill_length_mm' features.

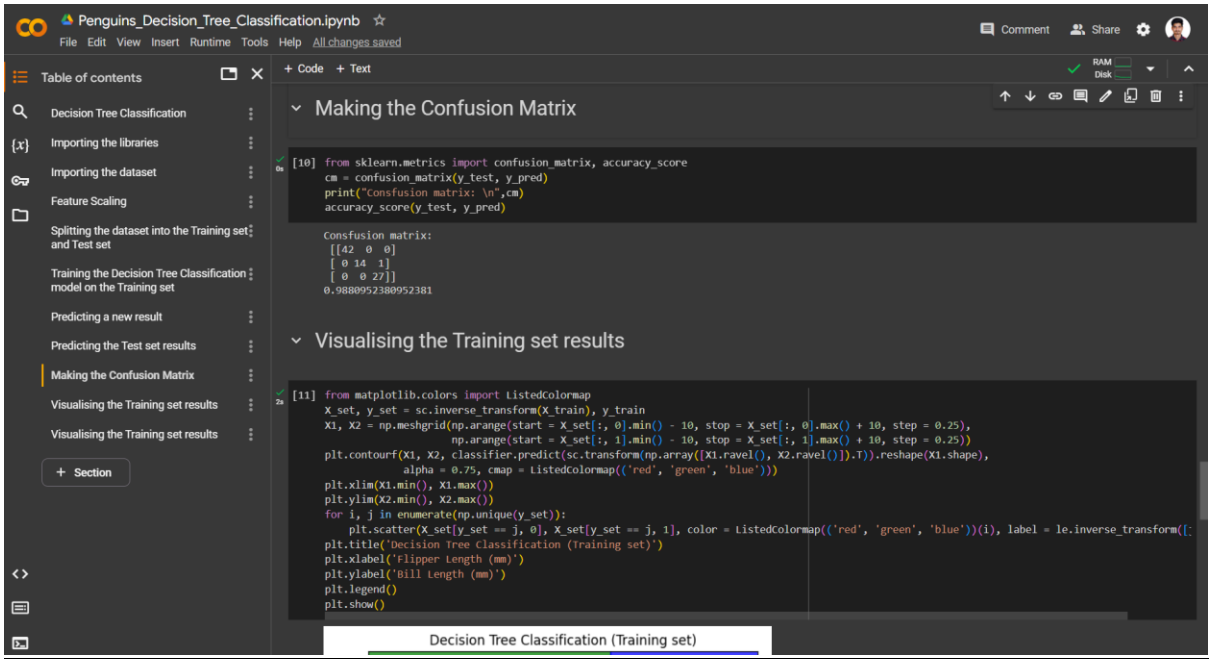
It also suggests that it can predict accurately for the unseen data. The predict function which is inside the classifier gives us the accurate prediction value of species for the new data.

Palmer Penguins DT NoteBook Script Screenshots






Penguins_Decision_Tree_Classification.ipynb
File Edit View Insert Runtime Tools Help All changes saved
Comment
Share
Table of contents
Decision Tree Classification
Importing the libraries
Importing the dataset
Feature Scaling
Splitting the dataset into the Training set and Test set
Training the Decision Tree Classification model on the Training set
Predicting a new result
Predicting the Test set results
Making the Confusion Matrix
Visualising the Training set results
Visualising the Training set results
Section
+ Code
+ Text
RAM
Disk
Training the Decision Tree Classification model on the Training set
[7] from sklearn.tree import DecisionTreeClassifier
classifier = DecisionTreeClassifier(criterion = 'entropy', random_state = 0)
classifier.fit(X_train, y_train)
DecisionTreeClassifier
DecisionTreeClassifier(criterion='entropy', random_state=0)
Predicting a new result
[8] prediction = classifier.predict(sc.transform([[210, 59]]))
species = le.inverse_transform(prediction)
print("Classification of Predit: \n", species)
Classification of Predit:
['Chinstrap']
Predicting the Test set results
[9] y_pred = classifier.predict(X_test)
print(np.concatenate((y_pred.reshape(len(y_pred),1), y_test.reshape(len(y_test),1)),1))
[0 0]
[0 0]
[2 2]
[1 1]
[2 2]
[2 2]


Penguins_Decision_Tree_Classification.ipynb
File Edit View Insert Runtime Tools Help All changes saved
Comment
Share
Table of contents
Decision Tree Classification
Importing the libraries
Importing the dataset
Feature Scaling
Splitting the dataset into the Training set and Test set
Training the Decision Tree Classification model on the Training set
Predicting a new result
Predicting the Test set results
Making the Confusion Matrix
Visualising the Training set results
Visualising the Training set results
Section
+ Code
+ Text
RAM
Disk
Making the Confusion Matrix
[10] from sklearn.metrics import confusion_matrix, accuracy_score
cm = confusion_matrix(y_test, y_pred)
print("Consfusion matrix: \n",cm)
accuracy_score(y_test, y_pred)
Consfusion matrix:
[[42 0 0]
[0 14 1]
[0 0 27]]
0.988095238095238
Visualising the Training set results
[11] from matplotlib.colors import ListedColormap
X_set, y_set = sc.inverse_transform(X_train), y_train
X1, X2 = np.meshgrid(np.arange(start = X_set[:, 0].min() - 10, stop = X_set[:, 0].max() + 10, step = 0.25),
np.arange(start = X_set[:, 1].min() - 10, stop = X_set[:, 1].max() + 10, step = 0.25))
plt.contourf(X1, X2, classifier.predict(sc.transform(np.array([X1.ravel(), X2.ravel()]).T)).reshape(X1.shape),
alpha = 0.75, cmap = ListedColormap(('red', 'green', 'blue')))
plt.xlim(X1.min(), X1.max())
plt.ylim(X2.min(), X2.max())
for i, j in enumerate(np.unique(y_set)):
plt.scatter(X_set[y_set == j, 0], X_set[y_set == j, 1], color = ListedColormap(('red', 'green', 'blue'))(i), label = le.inverse_transform([
plt.title('Decision Tree Classification (Training set)')
plt.xlabel('Flipper Length (mm)')
plt.ylabel('Bill Length (mm)')
plt.legend()
plt.show()
Decision Tree Classification (Training set)


Penguins_Decision_Tree_Classification.ipynb
File Edit View Insert Runtime Tools Help All changes saved
Comment
Share
Table of contents
Decision Tree Classification
Importing the libraries
Importing the dataset
Feature Scaling
Splitting the dataset into the Training set and Test set
Training the Decision Tree Classification model on the Training set
Predicting a new result
Predicting the Test set results
Making the Confusion Matrix
Visualising the Training set results
Visualising the Training set results
Section
+ Code + Text
RAM
Disk
[11] X1, X2 = np.meshgrid(np.arange(start = X_set[:, 0].min() - 10, stop = X_set[:, 0].max() + 10, step = 0.25),
np.arange(start = X_set[:, 1].min() - 10, stop = X_set[:, 1].max() + 10, step = 0.25))
plt.contourf(X1, X2, classifier.predict(sc.transform(np.array([X1.ravel(), X2.ravel()]).T)).reshape(X1.shape),
alpha = 0.75, cmap = ListedColormap(('red', 'green', 'blue')))
plt.xlim(X1.min(), X1.max())
plt.ylim(X2.min(), X2.max())
for i, j in enumerate(np.unique(y_set)):
plt.scatter(X_set[y_set == j, 0], X_set[y_set == j, 1], color = ListedColormap(('red', 'green', 'blue'))(i), label = le.inverse_transform([
plt.title('Decision Tree Classification (Training set)')
plt.xlabel('Flipper Length (mm)')
plt.ylabel('Bill Length (mm)')
plt.legend()
plt.show()
Decision Tree Classification (Training set)
['Adelie']
['Chinstrap']
['Gentoo']
Bill Length (mm)
Flipper Length (mm)
60
50
40
30
170
180
190
200
210
220
230
240


Penguins_Decision_Tree_Classification.ipynb
File Edit View Insert Runtime Tools Help All changes saved
Comment
Share
Table of contents
Decision Tree Classification
Importing the libraries
Importing the dataset
Feature Scaling
Splitting the dataset into the Training set and Test set
Training the Decision Tree Classification model on the Training set
Predicting a new result
Predicting the Test set results
Making the Confusion Matrix
Visualising the Training set results
Visualising the Training set results
Section
+ Code + Text
RAM
Disk
[12] X_set, y_set = sc.inverse_transform(X_test), y_test
X1, X2 = np.meshgrid(np.arange(start = X_set[:, 0].min() - 10, stop = X_set[:, 0].max() + 10, step = 0.25),
np.arange(start = X_set[:, 1].min() - 10, stop = X_set[:, 1].max() + 10, step = 0.25))
plt.contourf(X1, X2, classifier.predict(sc.transform(np.array([X1.ravel(), X2.ravel()]).T)).reshape(X1.shape),
alpha = 0.75, cmap = ListedColormap(('red', 'green', 'blue')))
plt.xlim(X1.min(), X1.max())
plt.ylim(X2.min(), X2.max())
for i, j in enumerate(np.unique(y_set)):
plt.scatter(X_set[y_set == j, 0], X_set[y_set == j, 1], color = ListedColormap(('red', 'green', 'blue'))(i), label = le.inverse_transform([
plt.title('Decision Tree Classification (Test set)')
plt.xlabel('Flipper Length (mm)')
plt.ylabel('Bill Length (mm)')
plt.legend()
plt.show()
Decision Tree Classification (Test set)
['Adelie']
['Chinstrap']
['Gentoo']
Bill Length (mm)
Flipper Length (mm)
60
50
40
30
170
180
190
200
210
220
230

Model 2: K-Nearest Neighbour Classification on Palmer Penguins Dataset

The K-Nearest Neighbour (K-NN) model was chosen for this dataset to classify penguin species based on their 'flipper_length_mm' and 'bill_length_mm'. This model is particularly suitable when there is no underlying assumption about the distribution of the data, and it can capture complex decision boundaries.

a) Data Preprocessing:
   - Handling missing values: Rows with missing values were removed.
   - Feature Scaling: StandardScaler was used to scale the features.

b) Splitting the Dataset:
   - The dataset was split into training and test sets using a 75/25 ratio.

c) Model Training:
   - KNeighborsClassifier was used with the specified parameters.
   - Parameters:
     - Number of Neighbors (k): 5
     - Metric: 'minkowski'
     - p: 2 (Euclidean distance)
   - The model was trained on the training set using the 'fit' method

d) Model Prediction:
   - Predictions were made on new data

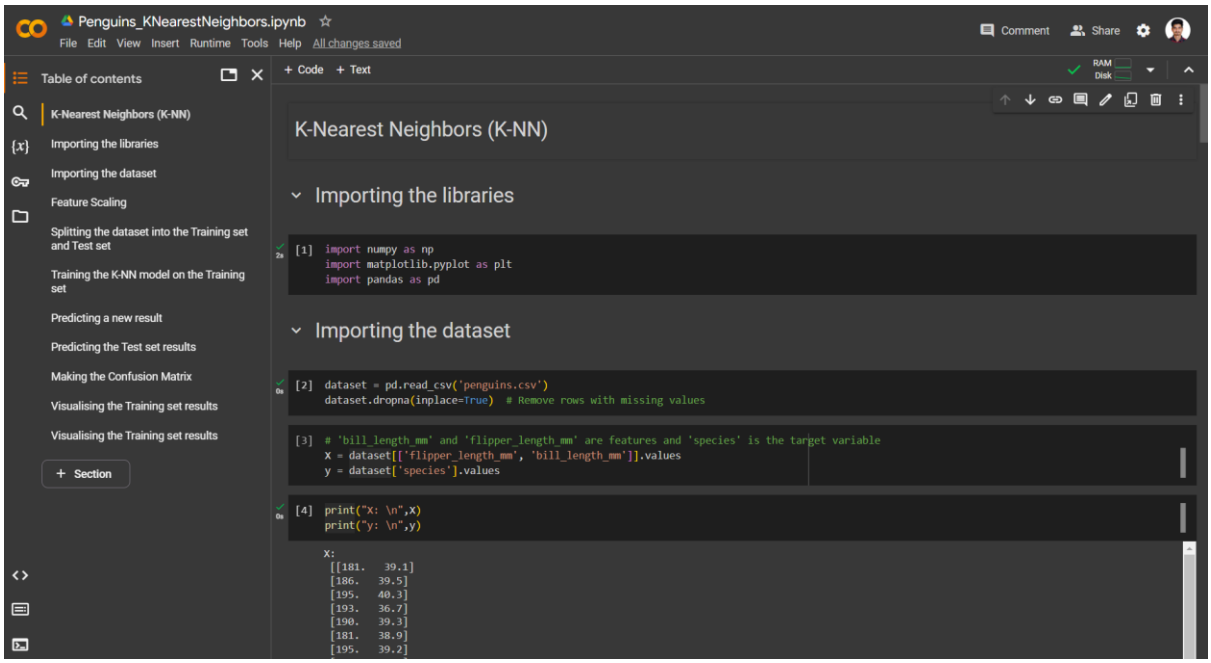
e) Model Evaluation:
   - The model's performance was evaluated using a confusion matrix and accuracy score

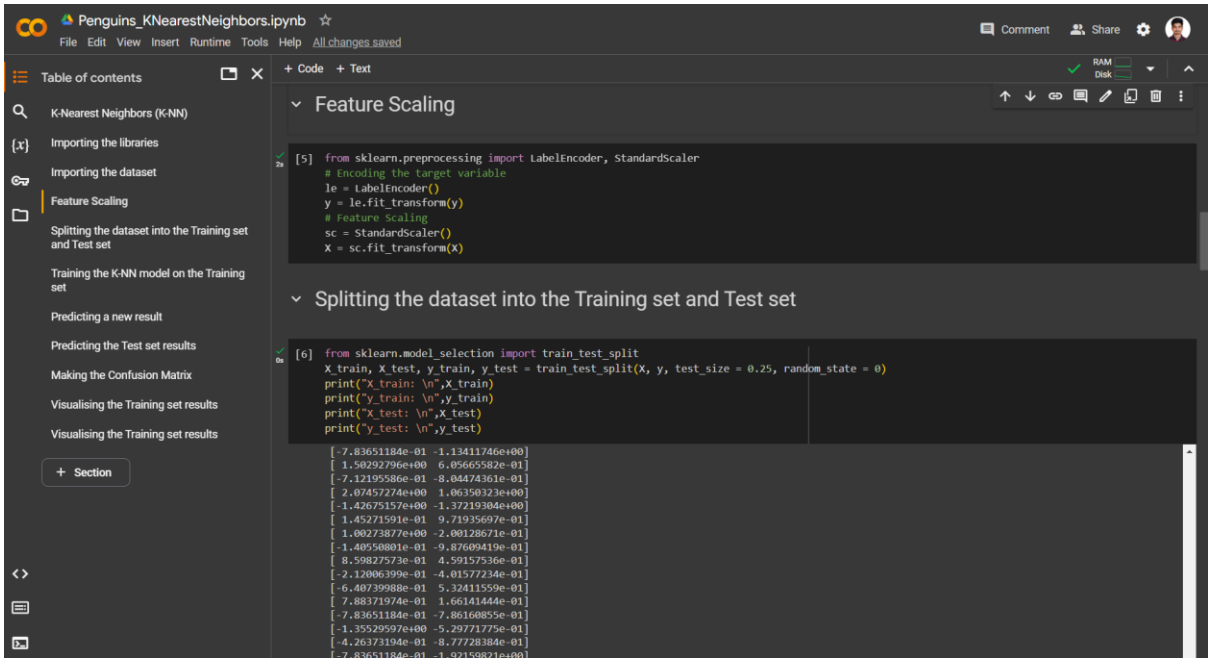
f) Visualizations:
   - Visualizations were plotted with the decision boundaries on the training and test sets
   - Flipper Length as xlabel and Bill Length as ylabel
   - **Adelie** Species is denoted by ’red', **Gentoo** Species is denoted by 'green', **Chinstrap** Species is denoted by 'blue.

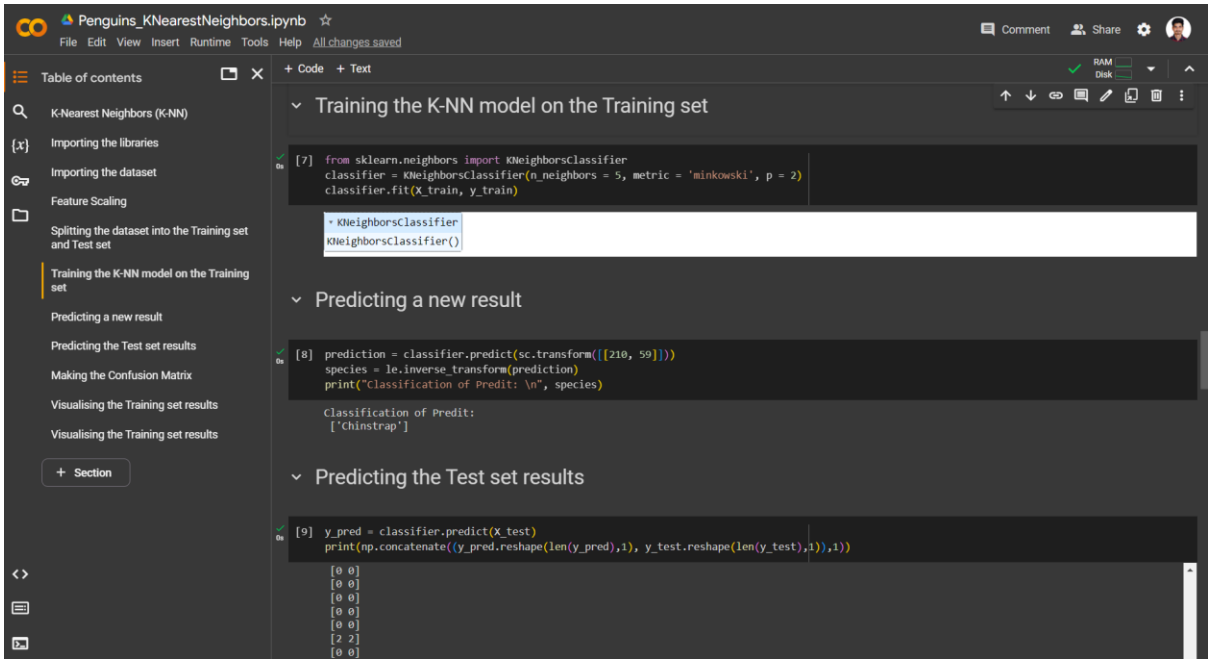
Performance

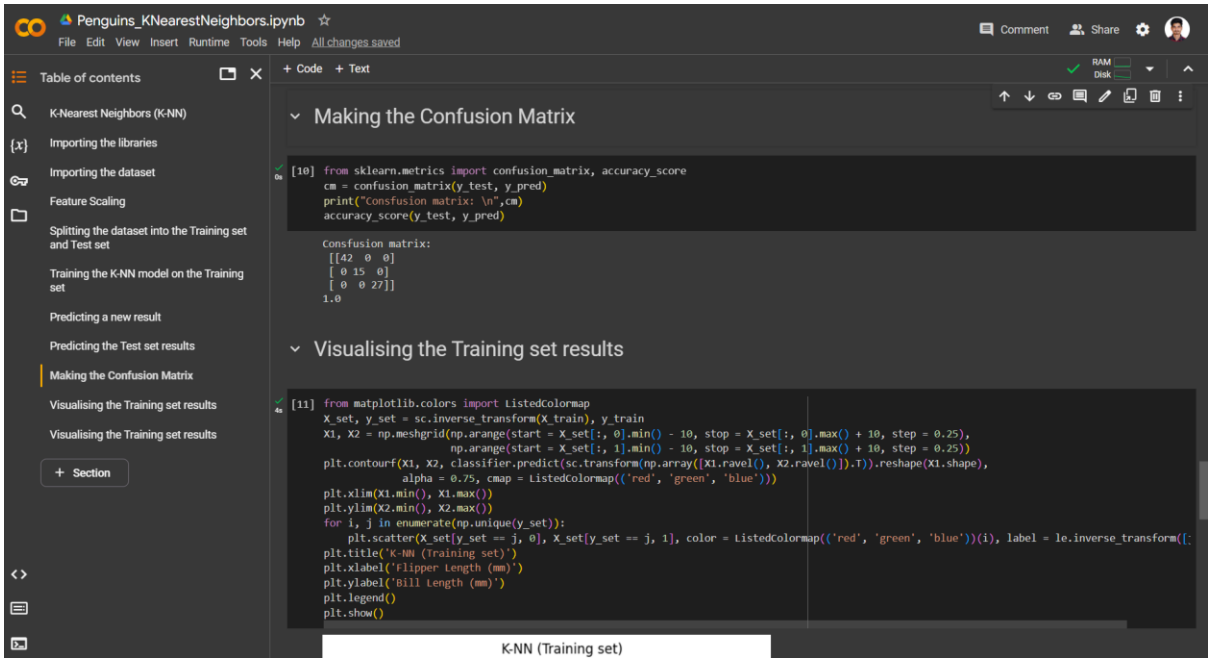
The K-Nearest Neighbors (K-NN) model achieved an accuracy score of 100% on the test dataset. This indicates that the model successfully classified all the penguin species in the test set based on their 'flipper_length_mm' and 'bill_length_mm' features. The model's performance indicates a strong ability to generalize well to unseen data, providing highly reliable predictions for penguin species classification.

Palmer Penguins KNN NoteBook Script Screenshots

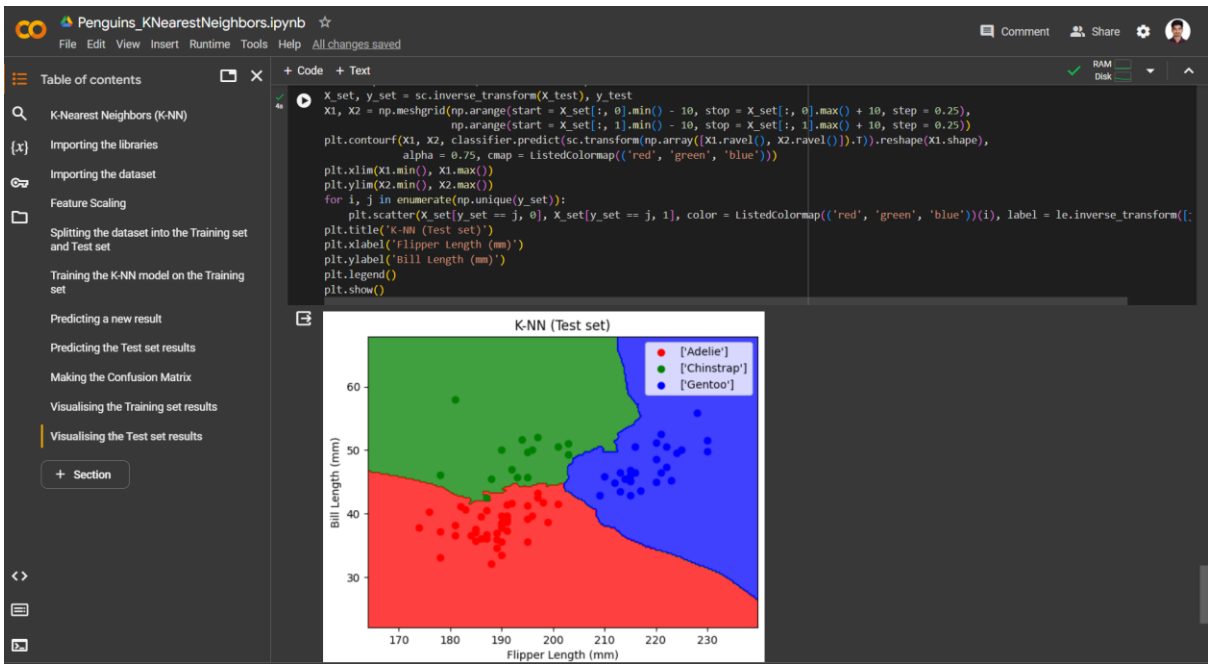





Penguins_KNearestNeighbors.ipynb
File Edit View Insert Runtime Tools Help All changes saved
Comment
Share
Table of contents
K-Nearest Neighbors (K-NN)
Importing the libraries
Importing the dataset
Feature Scaling
Splitting the dataset into the Training set and Test set
Training the K-NN model on the Training set
Predicting a new result
Predicting the Test set results
Making the Confusion Matrix
Visualising the Training set results
Visualising the Training set results
+ Section
+ Code
+ Text
RAM
Disk
Training the K-NN model on the Training set
[7] from sklearn.neighbors import KNeighborsClassifier
classifier = KNeighborsClassifier(n_neighbors = 5, metric = 'minkowski', p = 2)
classifier.fit(X_train, y_train)
KNeighborsClassifier
KNeighborsClassifier()
Predicting a new result
[8] prediction = classifier.predict(sc.transform([[210, 59]]))
species = le.inverse_transform(prediction)
print("Classification of Predit: \n", species)
Classification of Predit:
['Chinstrap']
Predicting the Test set results
[9] y_pred = classifier.predict(X_test)
print(np.concatenate((y_pred.reshape(len(y_pred),1), y_test.reshape(len(y_test),1)),1))
[0 0]
[0 0]
[0 0]
[0 0]
[0 0]
[2 2]
[0 0]


Penguins_KNearestNeighbors.ipynb
File Edit View Insert Runtime Tools Help All changes saved
Comment
Share
Table of contents
K-Nearest Neighbors (K-NN)
Importing the libraries
Importing the dataset
Feature Scaling
Splitting the dataset into the Training set and Test set
Training the K-NN model on the Training set
Predicting a new result
Predicting the Test set results
Making the Confusion Matrix
Visualising the Training set results
Visualising the Training set results
+ Section
+ Code
+ Text
RAM
Disk
Making the Confusion Matrix
[10] from sklearn.metrics import confusion_matrix, accuracy_score
cm = confusion_matrix(y_test, y_pred)
print("Consfusion matrix: \n",cm)
accuracy_score(y_test, y_pred)
Consfusion matrix:
[[42 0 0]
[0 15 0]
[0 0 27]]
1.0
Visualising the Training set results
[11] from matplotlib.colors import ListedColormap
X_set, y_set = sc.inverse_transform(X_train), y_train
X1, X2 = np.meshgrid(np.arange(start = X_set[:, 0].min() - 10, stop = X_set[:, 0].max() + 10, step = 0.25),
np.arange(start = X_set[:, 1].min() - 10, stop = X_set[:, 1].max() + 10, step = 0.25))
plt.contourf(X1, X2, classifier.predict(sc.transform(np.array([X1.ravel(), X2.ravel()]).T)).reshape(X1.shape),
alpha = 0.75, cmap = ListedColormap(('red', 'green', 'blue')))
plt.xlim(X1.min(), X1.max())
plt.ylim(X2.min(), X2.max())
for i, j in enumerate(np.unique(y_set)):
plt.scatter(X_set[y_set == j, 0], X_set[y_set == j, 1], color = ListedColormap(('red', 'green', 'blue'))(i), label = le.inverse_transform([
plt.title('K-NN (Training set)')
plt.xlabel('Flipper Length (mm)')
plt.ylabel('Bill Length (mm)')
plt.legend()
plt.show()
K-NN (Training set)


Penguins_KNearestNeighbors.ipynb
File Edit View Insert Runtime Tools Help All changes saved
Comment
Share
Table of contents
K-Nearest Neighbors (K-NN)
Importing the libraries
Importing the dataset
Feature Scaling
Splitting the dataset into the Training set and Test set
Training the K-NN model on the Training set
Predicting a new result
Predicting the Test set results
Making the Confusion Matrix
Visualising the Training set results
Visualising the Training set results
Section
+ Code + Text
RAM
Disk
[11]
X_set, y_set = sc.inverse_transform(X_train), y_train
X1, X2 = np.meshgrid(np.arange(start = X_set[:, 0].min() - 10, stop = X_set[:, 0].max() + 10, step = 0.25),
np.arange(start = X_set[:, 1].min() - 10, stop = X_set[:, 1].max() + 10, step = 0.25))
plt.contourf(X1, X2, classifier.predict(sc.transform(np.array([X1.ravel(), X2.ravel()]).T)).reshape(X1.shape),
alpha = 0.75, cmap = ListedColormap(('red', 'green', 'blue')))
plt.xlim(X1.min(), X1.max())
plt.ylim(X2.min(), X2.max())
for i, j in enumerate(np.unique(y_set)):
plt.scatter(X_set[y_set == j, 0], X_set[y_set == j, 1], color = ListedColormap(('red', 'green', 'blue'))(i), label = le.inverse_transform([
plt.title('K-NN (Training set)')
plt.xlabel('Flipper Length (mm)')
plt.ylabel('Bill Length (mm)')
plt.legend()
plt.show()
K-NN (Training set)
['Adelie']
['Chinstrap']
['Gentoo']
Bill Length (mm)
Flipper Length (mm)
30
40
50
60
170
180
190
200
210
220
230
240


Penguins_KNearestNeighbors.ipynb
File Edit View Insert Runtime Tools Help All changes saved
Comment
Share
Table of contents
K-Nearest Neighbors (K-NN)
Importing the libraries
Importing the dataset
Feature Scaling
Splitting the dataset into the Training set and Test set
Training the K-NN model on the Training set
Predicting a new result
Predicting the Test set results
Making the Confusion Matrix
Visualising the Training set results
Visualising the Test set results
Section
+ Code + Text
RAM
Disk
X_set, y_set = sc.inverse_transform(X_test), y_test
X1, X2 = np.meshgrid(np.arange(start = X_set[:, 0].min() - 10, stop = X_set[:, 0].max() + 10, step = 0.25),
np.arange(start = X_set[:, 1].min() - 10, stop = X_set[:, 1].max() + 10, step = 0.25))
plt.contourf(X1, X2, classifier.predict(sc.transform(np.array([X1.ravel(), X2.ravel()]).T)).reshape(X1.shape),
alpha = 0.75, cmap = ListedColormap(('red', 'green', 'blue')))
plt.xlim(X1.min(), X1.max())
plt.ylim(X2.min(), X2.max())
for i, j in enumerate(np.unique(y_set)):
plt.scatter(X_set[y_set == j, 0], X_set[y_set == j, 1], color = ListedColormap(('red', 'green', 'blue'))(i), label = le.inverse_transform([
plt.title('K-NN (Test set)')
plt.xlabel('Flipper Length (mm)')
plt.ylabel('Bill Length (mm)')
plt.legend()
plt.show()
K-NN (Test set)
['Adelie']
['Chinstrap']
['Gentoo']
Bill Length (mm)
Flipper Length (mm)
30
40
50
60
170
180
190
200
210
220
230

Model 3: Polynomial Regression on Palmer Penguins Dataset

Polynomial Regression model is used to predict the body mass of penguins based on their flipper length. It aims to capture the non-linear relationship between these two variables by fitting a polynomial curve to the data.

Polynomial Regression allows for a more flexible curve fitting compared to linear regression, which helps in capturing the underlying complexity of the relationship between flipper_length_mm and body_mass_g.

a) Data Preprocessing:
   - Handling missing values: Rows with missing values were removed.

b) Linear Regression:
   - The model starts with a simple Linear Regression to establish a baseline prediction of body mass based on flipper length

c) Polynomial Regression:
   - PolynomialFeatures is applied to transform the original feature (flipper length) into polynomial features up to the 4th degree.
   - Parameters:
     - Degree: PolynomialFeatures transformer was used with a degree of 4

d) Fitting the Model:
   - The transformed features are then used to train the Linear Regression model to learn the non-linear mapping between flipper length and body mass

e) Visualizations:
   - Linear Regression Visualization:
     - The blue line represents the predictions made by the Linear Regression model. This line represents a straight line fit to the data.
   - Polynomial Regression Visualization:
     - The blue curve represents the predictions made by the Polynomial Regression model after transforming the features to a 4th-degree polynomial.
   - Higher Resolution Visualization:
     - To get a smoother curve, the third plot uses a higher resolution grid of flipper lengths ranging from the minimum to the maximum observed values.
     - The blue curve represents the predictions of the Polynomial Regression model on this higher resolution grid.

f) Predictions:
   - Linear Regression Prediction:
     - The Linear Regression model predicts the body mass for a new flipper length of 189 mm based on the straight-line fit it learned from the data which is 3606.8.
   - Polynomial Regression Prediction:

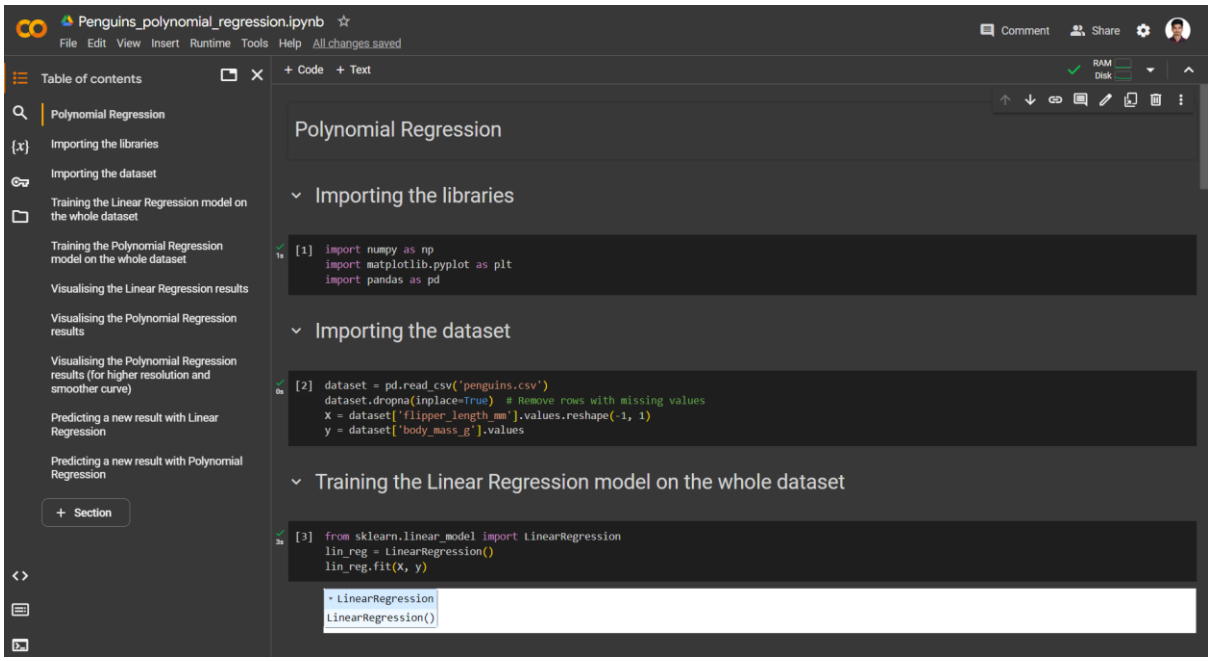
- The Polynomial Regression model predicts the body mass based on a curve that adjusts to the observed data points, potentially providing a better estimate for this specific flipper length which is 3593.77.

Performance:

The Linear Regression plot (first) shows a straight line that might not capture the subtle variations in body mass as flipper length changes. In contrast, the Polynomial Regression plots (second and third) a curve that better represents the data points, with more accurate relationship.

The increasing curvature in the Polynomial Regression indicates that how body mass change with flipper length, suggesting the non-linear nature of this relationship. Which visually confirms the ability to capture the complex patterns in the penguins' body mass and flipper length.

Palmer Penguins Polynomial Regression NoteBook Script Screenshots



Penguins_polynomial_regression.ipynb

File Edit View Insert Runtime Tools Help All changes saved

Comment Share

Table of contents

+ Code + Text

RAM Disk

- Polynomial Regression
- Importing the libraries
- Importing the dataset
- Training the Linear Regression model on the whole dataset
- Training the Polynomial Regression model on the whole dataset
- Visualising the Linear Regression results
- Visualising the Polynomial Regression results
- Visualising the Polynomial Regression results (for higher resolution and smoother curve)
- Predicting a new result with Linear Regression
- Predicting a new result with Polynomial Regression

+ Section

Importing the dataset

```
[2] dataset = pd.read_csv('penguins.csv')
    dataset.dropna(inplace=True)  # Remove rows with missing values
    X = dataset['flipper_length_mm'].values.reshape(-1, 1)
    y = dataset['body_mass_g'].values
```

Training the Linear Regression model on the whole dataset

```
from sklearn.linear_model import LinearRegression
lin_reg = LinearRegression()
lin_reg.fit(X, y)
```

```
▾ LinearRegression
LinearRegression()
```

Training the Polynomial Regression model on the whole dataset

```
[4] from sklearn.preprocessing import PolynomialFeatures
    poly_reg = PolynomialFeatures(degree = 4)
    X_poly = poly_reg.fit_transform(X)
    lin_reg_2 = LinearRegression()
    lin_reg_2.fit(X_poly, y)
```

```
▾ LinearRegression
LinearRegression()
```

Penguins_polynomial_regression.ipynb

File Edit View Insert Runtime Tools Help All changes saved

Comment Share

Table of contents

+ Code + Text

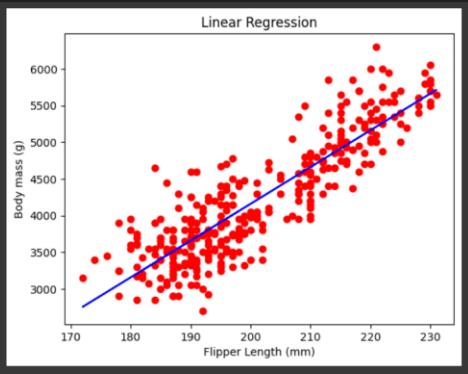
RAM Disk

- Polynomial Regression
- Importing the libraries
- Importing the dataset
- Training the Linear Regression model on the whole dataset
- Training the Polynomial Regression model on the whole dataset
- Visualising the Linear Regression results
- Visualising the Polynomial Regression results
- Visualising the Polynomial Regression results (for higher resolution and smoother curve)
- Predicting a new result with Linear Regression
- Predicting a new result with Polynomial Regression

+ Section

Visualising the Linear Regression results

```
plt.scatter(X, y, color = 'red')
plt.plot(X, lin_reg.predict(X), color = 'blue')
plt.title('Linear Regression')
plt.xlabel('Flipper Length (mm)')
plt.ylabel('Body mass (g)')
plt.show()
```



Penguins_polynomial_regression.ipynb

File Edit View Insert Runtime Tools Help All changes saved

Comment Share

Table of contents

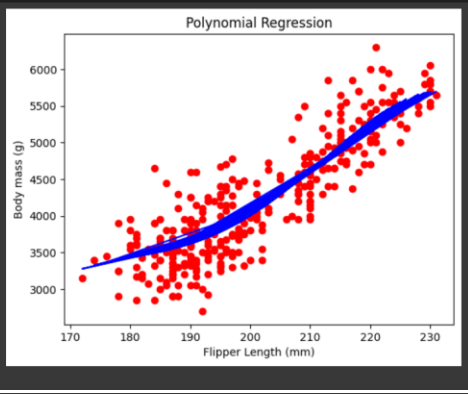
- Polynomial Regression
- Importing the libraries
- Importing the dataset
- Training the Linear Regression model on the whole dataset
- Training the Polynomial Regression model on the whole dataset
- Visualising the Linear Regression results
- Visualising the Polynomial Regression results
- Visualising the Polynomial Regression results (for higher resolution and smoother curve)
- Predicting a new result with Linear Regression
- Predicting a new result with Polynomial Regression

+ Section

+ Code + Text

Visualising the Polynomial Regression results

```
plt.scatter(X, y, color = 'red')
plt.plot(X, lin_reg_2.predict(poly_reg.fit_transform(X)), color = 'blue')
plt.title('Polynomial Regression')
plt.xlabel('Flipper Length (mm)')
plt.ylabel('Body mass (g)')
plt.show()
```



Penguins_polynomial_regression.ipynb

File Edit View Insert Runtime Tools Help

Comment Share

Table of contents

- Polynomial Regression
- Importing the libraries
- Importing the dataset
- Training the Linear Regression model on the whole dataset
- Training the Polynomial Regression model on the whole dataset
- Visualising the Linear Regression results
- Visualising the Polynomial Regression results
- Visualising the Polynomial Regression results (for higher resolution and smoother curve)
- Predicting a new result with Linear Regression
- Predicting a new result with Polynomial Regression

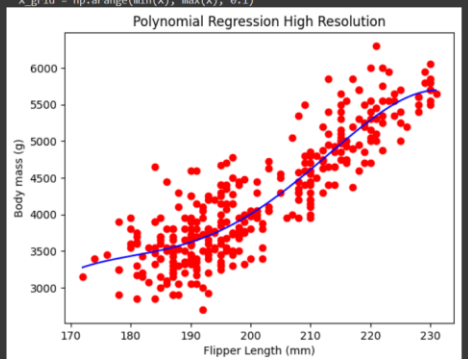
+ Section

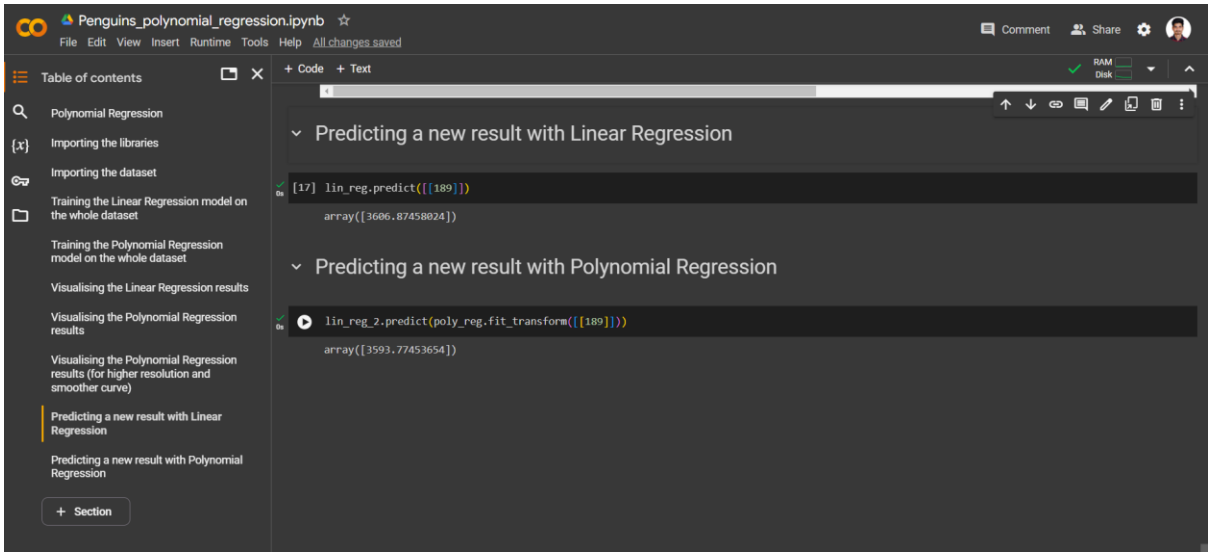
+ Code + Text

curve)

```
X_grid = np.arange(min(X), max(X), 0.1)
X_grid = X_grid.reshape((len(X_grid), 1))
plt.scatter(X, y, color = 'red')
plt.plot(X_grid, lin_reg_2.predict(poly_reg.fit_transform(X_grid)), color = 'blue')
plt.title('Polynomial Regression High Resolution')
plt.xlabel('Flipper Length (mm)')
plt.ylabel('Body mass (g)')
plt.show()
```

```
<ipython-input-16-5331626400cf>:1: DeprecationWarning: Conversion of an array with ndim > 0 to a scalar is deprecated, and will error in future.
  X_grid = np.arange(min(X), max(X), 0.1)
```




Penguins_polynomial_regression.ipynb
File Edit View Insert Runtime Tools Help All changes saved
Comment
Share
Table of contents
+ Code + Text
RAM
Disk
Polynomial Regression
Importing the libraries
Importing the dataset
Training the Linear Regression model on the whole dataset
Training the Polynomial Regression model on the whole dataset
Visualising the Linear Regression results
Visualising the Polynomial Regression results
Visualising the Polynomial Regression results (for higher resolution and smoother curve)
Predicting a new result with Linear Regression
Predicting a new result with Polynomial Regression
+ Section
Predicting a new result with Linear Regression
[17] lin_reg.predict([[189]])
array([3606.87458024])
Predicting a new result with Polynomial Regression
lin_reg_2.predict(poly_reg.fit_transform([[189]]))
array([3593.77453654])

2. Simple Neural Network (NN) on Cifar-10 image dataset.

The DataSet Description

- **Cifar-10 image dataset**

The CIFAR-10 dataset consists of 60000 32x32 colour images in 10 classes, with 6000 images per class. There are 50000 training images and 10000 test images.

The dataset is divided into five training batches and one test batch, each with 10000 images. The test batch contains exactly 1000 randomly-selected images from each class. The training batches contain the remaining images in random order, but some training batches may contain more images from one class than another. Between them, the training batches contain exactly 5000 images from each class.

Here are the classes in the dataset:

Label	Description
0	airplane
1	automobile
2	bird
3	cat
4	deer
5	dog
6	frog
7	horse
8	ship
9	truck

NN Architectures:

1. Simpler Convolutional Neural Network (CNN) Architecture

This simpler convolutional Neural Network (NN) architecture is designed for image classification using the CIFAR-10 dataset. It consists of convolutional layers followed by fully connected layers. The model aims to classify images into one of the ten categories: airplane, automobile, bird, cat, deer, dog, frog, horse, ship, or truck.

The model starts with a 2D convolutional layer with 16 filters and a kernel size of (3, 3), followed by a max-pooling layer to downsample the output.

Another 2D convolutional layer with 32 filters and the same kernel size is added to extract more complex features.

The model then flattens the output and passes it through two fully connected layers with ReLU activation.

Finally, a dense layer with 10 units (equal to the number of classes) is added for the classification.

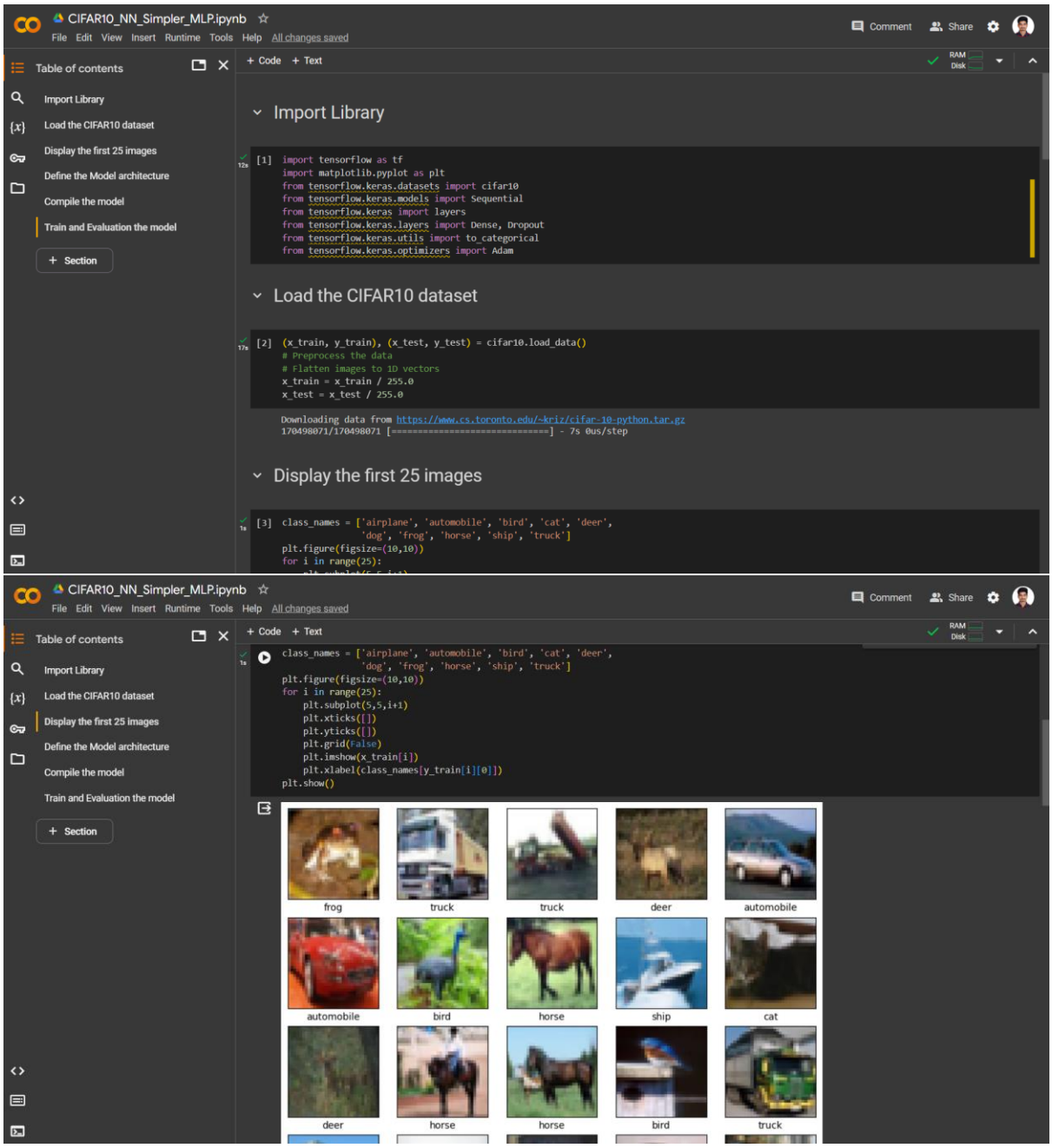
Training Steps:

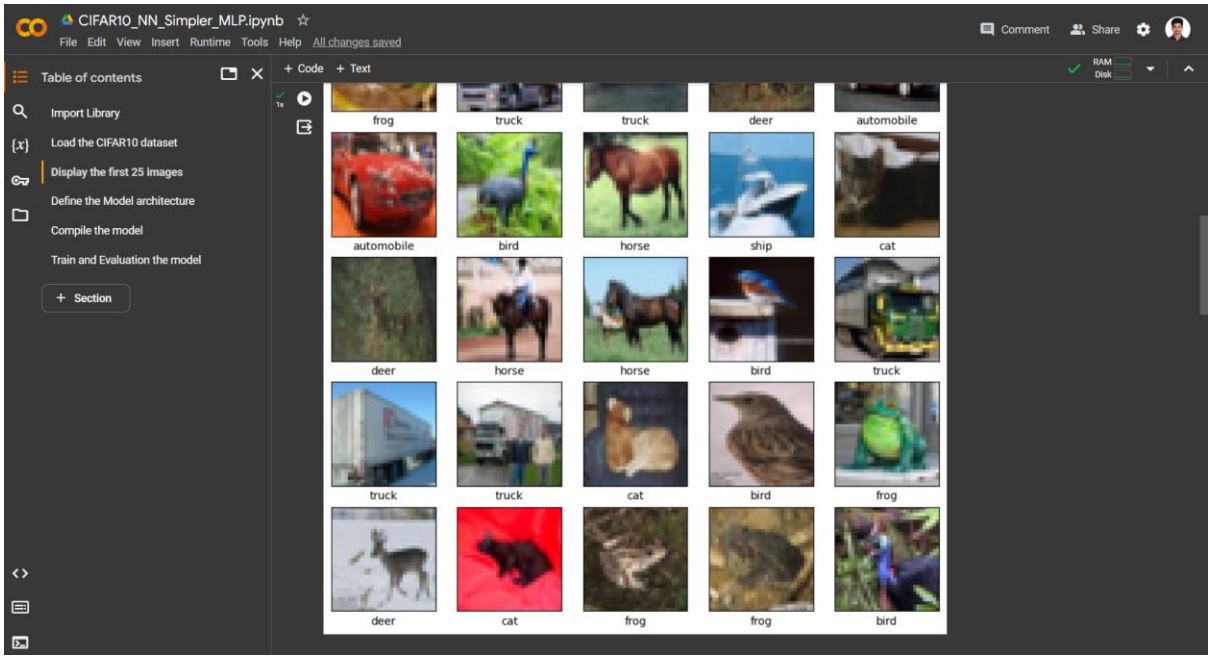
- The model is trained on the CIFAR-10 dataset, which consists of 50,000 training images and 10,000 test images.
- Images are normalized by dividing pixel values by 255 to scale them between 0 and 1.
- The model is compiled with the below parameters:
    a) Conv2D Layers:
    - First Conv2D layer: 16 filters, kernel size (3, 3), ReLU activation.
    - Second Conv2D layer: 32 filters, kernel size (3, 3), ReLU activation.
    b) MaxPooling2D Layer:
    - Pooling window size of (2, 2) for downsampling.
    c) Dense Layers:
    - First Dense layer: 32 units, ReLU activation.
    - Second Dense layer: 10 units (output layer for classification), no activation.
    d) Optimizer:
    - Adam optimizer is used for training the model.
    e) Loss Function: SparseCategoricalCrossentropy loss
- It is trained for 10 epochs with the training data, using a batch size of 32.
- During training, the model learns to minimize the loss and improve classification accuracy.
- The validation data (x_test, y_test) is used to evaluate the model's performance after each epoch.
- The training and validation accuracies are plotted against the number of epochs to visualize the model's learning progress.

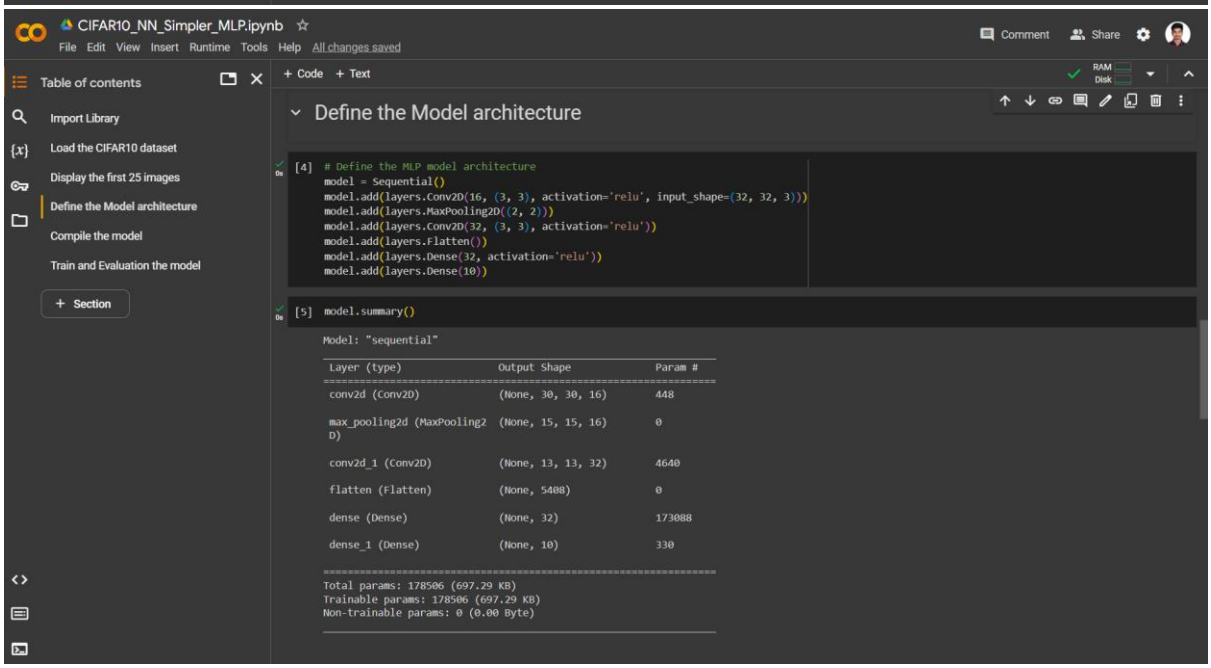
Performance

The accuracy of the Simpler Convolutional Neural Network (CNN) model designed for the CIFAR-10 dataset is approximately 65.26%. This means that the model correctly predicted the classes of the test images 65% of the time. While the accuracy is decent, there is still room for improvement, which could be done by adjusting the model architecture, tuning hyperparameters, adding more depth to the NN model and the neuro number of hidden layer.

Simpler Convolutional Neural Network (CNN) NoteBook Script Screenshots




CIFAR10_NN_Simpler_MLP.ipynb
File Edit View Insert Runtime Tools Help All changes saved
Comment
Share
Table of contents
+ Code
+ Text
Import Library
Load the CIFAR10 dataset
Display the first 25 images
Define the Model architecture
Compile the model
Train and Evaluation the model
+ Section
frog truck truck deer automobile
automobile bird horse ship cat
deer horse horse bird truck
truck truck cat bird frog
deer cat frog frog bird


CIFAR10_NN_Simpler_MLP.ipynb
File Edit View Insert Runtime Tools Help All changes saved
Comment
Share
Table of contents
+ Code
+ Text
Import Library
Load the CIFAR10 dataset
Display the first 25 images
Define the Model architecture
Compile the model
Train and Evaluation the model
+ Section
Define the Model architecture
[4] # Define the MLP model architecture
model = Sequential()
model.add(layers.Conv2D(16, (3, 3), activation='relu', input_shape=(32, 32, 3)))
model.add(layers.MaxPooling2D((2, 2)))
model.add(layers.Conv2D(32, (3, 3), activation='relu'))
model.add(layers.Flatten())
model.add(layers.Dense(32, activation='relu'))
model.add(layers.Dense(10))
[5] model.summary()
Model: "sequential"
Layer (type) Output Shape Param #
conv2d (Conv2D) (None, 30, 30, 16) 448
max_pooling2d (MaxPooling2D) (None, 15, 15, 16) 0
conv2d_1 (Conv2D) (None, 13, 13, 32) 4640
flatten (Flatten) (None, 5408) 0
dense (Dense) (None, 32) 173088
dense_1 (Dense) (None, 10) 330
Total params: 178506 (697.29 KB)
Trainable params: 178506 (697.29 KB)
Non-trainable params: 0 (0.00 Byte)

CIFAR10_NN_Simpler_MLP.ipynb

File Edit View Insert Runtime Tools Help All changes saved

Comment Share

Table of contents

- Import Library
- Load the CIFAR10 dataset
- Display the first 25 images
- Define the Model architecture
- Compile the model
- Train and Evaluation the model
- + Section

+ Code + Text

RAM Disk

Compile the model

```
[6] # Compile the model
    model.compile(loss=tf.keras.losses.SparseCategoricalCrossentropy(from_logits=True),
                  optimizer=Adam(),
                  metrics=['accuracy'])
```

Train and Evaluation the model

```
# Train the model
history = model.fit(x_train, y_train, epochs=10,
                    validation_data=(x_test, y_test))
plt.plot(history.history['accuracy'], label='accuracy')
plt.plot(history.history['val_accuracy'], label = 'val_accuracy')
plt.xlabel('Epoch')
plt.ylabel('Accuracy')
plt.ylim([0.5, 1])
plt.legend(loc='lower right')
test_loss, test_acc = model.evaluate(x_test,  y_test, verbose=2)
print('Test loss:', test_loss)
print('Test accuracy:', test_acc)
```

```
Epoch 1/10
1563/1563 [==============================] - 43s 27ms/step - loss: 1.4677 - accuracy: 0.4722 - val_loss: 1.3197 - val_accuracy: 0.5349
Epoch 2/10
1563/1563 [==============================] - 43s 28ms/step - loss: 1.1553 - accuracy: 0.5920 - val_loss: 1.0675 - val_accuracy: 0.6226
Epoch 3/10
1563/1563 [==============================] - 42s 27ms/step - loss: 1.0227 - accuracy: 0.6437 - val_loss: 1.0204 - val_accuracy: 0.6465
Epoch 4/10
1563/1563 [==============================] - 41s 26ms/step - loss: 0.9434 - accuracy: 0.6723 - val_loss: 1.0220 - val_accuracy: 0.6479
Epoch 5/10
1563/1563 [==============================] - 45s 29ms/step - loss: 0.8769 - accuracy: 0.6947 - val_loss: 0.9449 - val_accuracy: 0.6733
Epoch 6/10
1563/1563 [==============================] - 40s 25ms/step - loss: 0.8252 - accuracy: 0.7141 - val_loss: 0.9592 - val_accuracy: 0.6744
```

CIFAR10_NN_Simpler_MLP.ipynb

File Edit View Insert Runtime Tools Help All changes saved

Comment Share

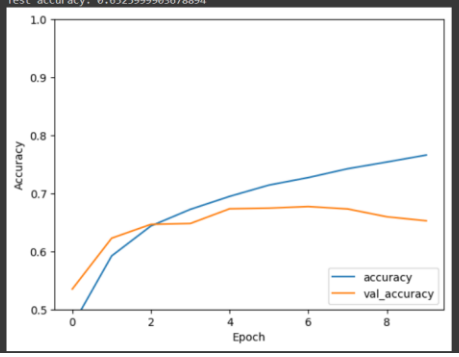
Table of contents

- Import Library
- Load the CIFAR10 dataset
- Display the first 25 images
- Define the Model architecture
- Compile the model
- Train and Evaluation the model
- + Section

+ Code + Text

RAM Disk

```
Epoch 4/10
1563/1563 [==============================] - 41s 26ms/step - loss: 0.9434 - accuracy: 0.6723 - val_loss: 1.0220 - val_accuracy: 0.6479
Epoch 5/10
1563/1563 [==============================] - 45s 29ms/step - loss: 0.8769 - accuracy: 0.6947 - val_loss: 0.9449 - val_accuracy: 0.6733
Epoch 6/10
1563/1563 [==============================] - 40s 25ms/step - loss: 0.8252 - accuracy: 0.7141 - val_loss: 0.9592 - val_accuracy: 0.6744
Epoch 7/10
1563/1563 [==============================] - 40s 25ms/step - loss: 0.7864 - accuracy: 0.7271 - val_loss: 0.9611 - val_accuracy: 0.6771
Epoch 8/10
1563/1563 [==============================] - 40s 26ms/step - loss: 0.7446 - accuracy: 0.7423 - val_loss: 0.9656 - val_accuracy: 0.6730
Epoch 9/10
1563/1563 [==============================] - 40s 25ms/step - loss: 0.7101 - accuracy: 0.7539 - val_loss: 1.0431 - val_accuracy: 0.6595
Epoch 10/10
1563/1563 [==============================] - 40s 26ms/step - loss: 0.6755 - accuracy: 0.7659 - val_loss: 1.0557 - val_accuracy: 0.6526
313/313 - 2s - loss: 1.0557 - accuracy: 0.6526 - 2s/epoch - 7ms/step
Test loss: 1.0556553602218628
Test accuracy: 0.6525999903678894
```



2. Deeper Convolutional Neural Network (CNN) Architecture

This Deeper Neural Network (DNN) architecture consists of multiple convolutional layers followed by max-pooling layers. The model includes three convolutional layers with increasing filter sizes and a dense layer for classification. Each convolutional layer extracts features from the input images, while the dense layers perform the final classification. By adding more convolutional layers, the model can capture more abstract features, potentially leading to improved classification accuracy.

The model starts with a 2D convolutional layer with 64 filters and a kernel size of (3, 3), followed by a max-pooling layer to downsample the output.

Second 2D convolution layer with 128 filters is added for more complex features with ReLU activation.

Third 2D convolutional layer with 128 filters and the same kernel size is added to extract more complex features.

The model then flattens the output and passes it through two fully connected layers with ReLU activation.

Finally, a dense layer with 10 units (equal to the number of classes) is added for the classification.

Training Steps:

- The model is trained on the CIFAR-10 dataset, which consists of 50,000 training images and 10,000 test images.
- Images are normalized by dividing pixel values by 255 to scale them between 0 and 1.
- The model is compiled with the below parameters:
    a) Conv2D Layers:
    - First Conv2D layer: 64 filters, kernel size (3, 3), ReLU activation.
    - Second Conv2D layer: 128 filters, kernel size (3, 3), ReLU activation.
    - Third Conv2D layer: 128 filters, kernel size (3, 3), ReLU activation.
    b) MaxPooling2D Layer:
    - Pooling window size of (2, 2) for downsampling.
    c) Dense Layers:
    - First Dense layer: 128 units, ReLU activation.
    - Second Dense layer: 10 units (output layer for classification), no activation.
    d) Optimizer:
    - Adam optimizer is used for training the model.
    e) Loss Function: SparseCategoricalCrossentropy loss
- It is trained for 10 epochs with the training data, using a batch size of 32.

- During training, the model learns to minimize the loss and improve classification accuracy.
- The validation data (x_test, y_test) is used to evaluate the model's performance after each epoch.
- The training and validation accuracies are plotted against the number of epochs to visualize the model's learning progress.

Performance:

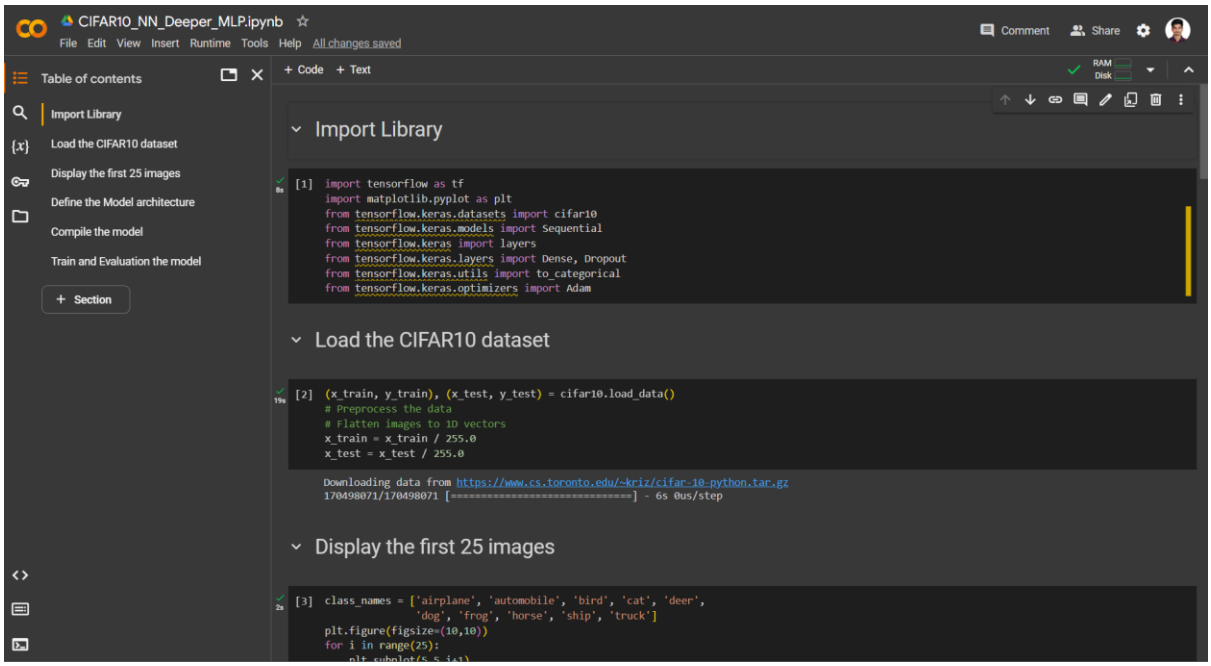
The Deeper Convolutional Neural Network (CNN) architecture achieved a test accuracy of **72.02%** on the CIFAR-10 dataset. This indicates an improvement in performance compared to the simpler CNN architecture, showcasing the deeper model's ability to learn and extract more intricate features from the images.

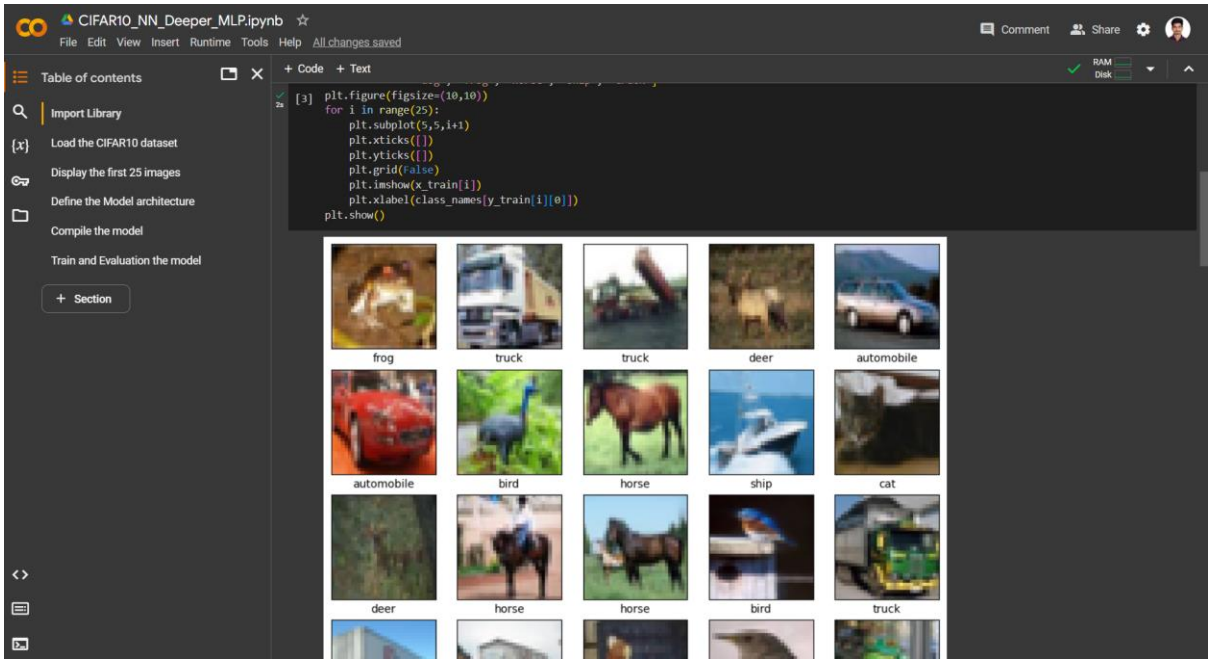
This deeper model architecture consists of three 2D convolutional layers with increasing filter sizes (64, 128, and 128) and three max-pooling layers to downsample the output. Additionally, it includes two dense layers with 128 units before the final output layer.

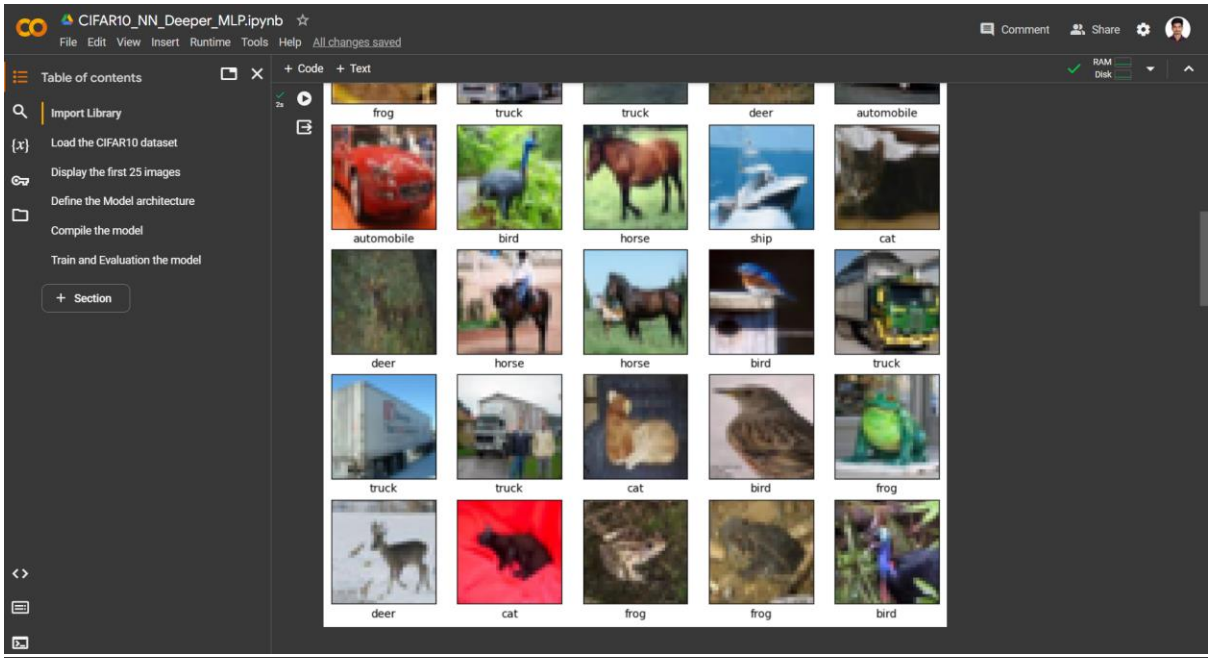
The additional layers in the deeper architecture allow the model to learn and capture more detailed and complex features from the images. This leads to better representation learning, enabling the network to distinguish between subtle differences in the CIFAR-10 classes.

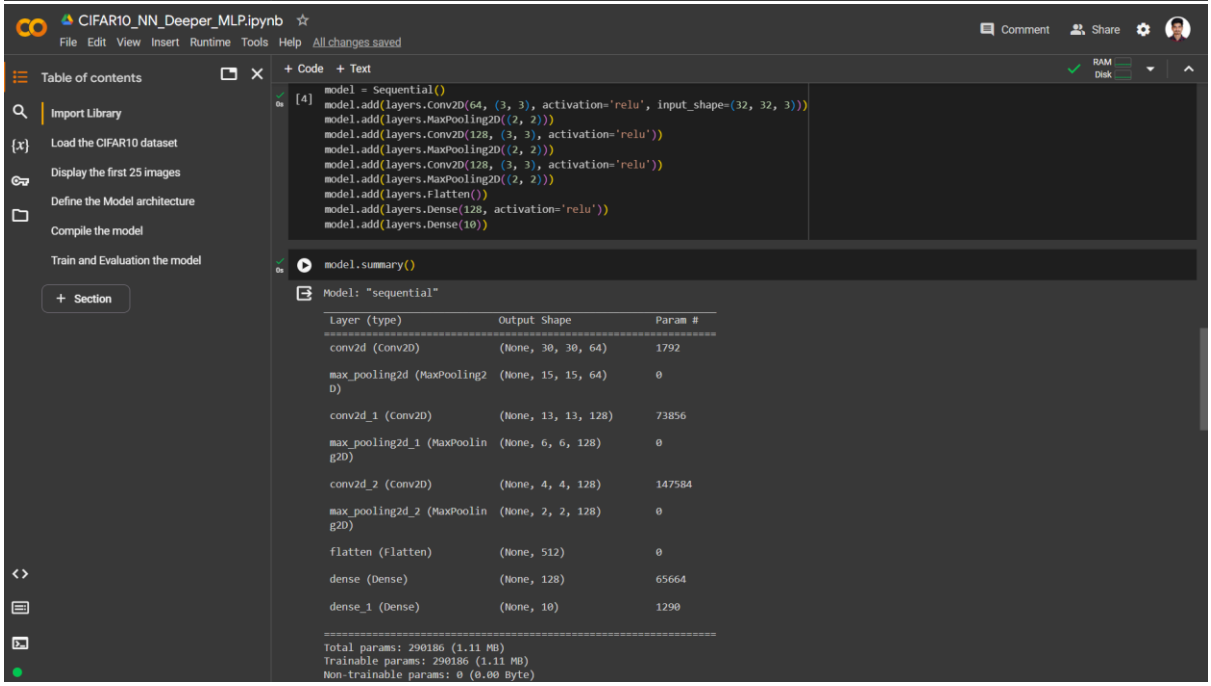
Overall, the Deeper CNN architecture's benefits lie in its capacity to learn and represent more complex patterns in image data, resulting in improved accuracy and performance on challenging classification tasks like the CIFAR-10 dataset.

Deeper Convolutional Neural Network (CNN) NoteBook Script Screenshots






CIFAR10_NN_Deeper_MLP.ipynb
File Edit View Insert Runtime Tools Help All changes saved
Comment
Share
Table of contents
Import Library
Load the CIFAR10 dataset
Display the first 25 images
Define the Model architecture
Compile the model
Train and Evaluation the model
Section
+ Code + Text
RAM
Disk
frog truck truck deer automobile
automobile bird horse ship cat
deer horse horse bird truck
truck truck cat bird frog
deer cat frog frog bird


CIFAR10_NN_Deeper_MLP.ipynb
File Edit View Insert Runtime Tools Help All changes saved
Comment
Share
Table of contents
Import Library
Load the CIFAR10 dataset
Display the first 25 images
Define the Model architecture
Compile the model
Train and Evaluation the model
Section
+ Code + Text
RAM
Disk
[4]
model = Sequential()
model.add(layers.Conv2D(64, (3, 3), activation='relu', input_shape=(32, 32, 3)))
model.add(layers.MaxPooling2D((2, 2)))
model.add(layers.Conv2D(128, (3, 3), activation='relu'))
model.add(layers.MaxPooling2D((2, 2)))
model.add(layers.Conv2D(128, (3, 3), activation='relu'))
model.add(layers.MaxPooling2D((2, 2)))
model.add(layers.Flatten())
model.add(layers.Dense(128, activation='relu'))
model.add(layers.Dense(10))
model.summary()
Model: "sequential"
Layer (type) Output Shape Param #
conv2d (Conv2D) (None, 30, 30, 64) 1792
max_pooling2d (MaxPooling2D) (None, 15, 15, 64) 0
conv2d_1 (Conv2D) (None, 13, 13, 128) 73856
max_pooling2d_1 (MaxPooling2D) (None, 6, 6, 128) 0
conv2d_2 (Conv2D) (None, 4, 4, 128) 147584
max_pooling2d_2 (MaxPooling2D) (None, 2, 2, 128) 0
flatten (Flatten) (None, 512) 0
dense (Dense) (None, 128) 65664
dense_1 (Dense) (None, 10) 1290
Total params: 290186 (1.11 MB)
Trainable params: 290186 (1.11 MB)
Non-trainable params: 0 (0.00 Byte)

CIFAR10_NN_Deeper_MLP.ipynb

File Edit View Insert Runtime Tools Help All changes saved

Comment Share

Table of contents

- Import Library
- Load the CIFAR10 dataset
- Display the first 25 images
- Define the Model architecture
- Compile the model
- Train and Evaluation the model
- + Section

+ Code + Text

RAM Disk

Compile the model

```
[6] # Compile the model
    model.compile(loss=tf.keras.losses.SparseCategoricalCrossentropy(from_logits=True),
                  optimizer=Adam(),
                  metrics=['accuracy'])
```

Train and Evaluation the model

```
# Train the model
history = model.fit(x_train, y_train, ep Loading...
                    validation_data=(x_test, y_test))
plt.plot(history.history['accuracy'], label='accuracy')
plt.plot(history.history['val_accuracy'], label = 'val_accuracy')
plt.xlabel('Epoch')
plt.ylabel('Accuracy')
plt.ylim([0.5, 1])
plt.legend(loc='lower right')
test_loss, test_acc = model.evaluate(x_test,  y_test, verbose=2)
print('Test loss:', test_loss)
print('Test accuracy:', test_acc)
```

```
Epoch 1/10
1563/1563 [==============================] - 175s 111ms/step - loss: 1.4832 - accuracy: 0.4581 - val_loss: 1.2242 - val_accuracy: 0.5635
Epoch 2/10
1563/1563 [==============================] - 174s 111ms/step - loss: 1.0780 - accuracy: 0.6235 - val_loss: 1.1066 - val_accuracy: 0.6197
Epoch 3/10
1563/1563 [==============================] - 168s 108ms/step - loss: 0.9117 - accuracy: 0.6807 - val_loss: 0.9263 - val_accuracy: 0.6800
Epoch 4/10
1563/1563 [==============================] - 167s 107ms/step - loss: 0.8041 - accuracy: 0.7197 - val_loss: 0.8971 - val_accuracy: 0.6967
Epoch 5/10
1563/1563 [==============================] - 171s 109ms/step - loss: 0.7218 - accuracy: 0.7472 - val_loss: 0.8853 - val_accuracy: 0.6988
Epoch 6/10
1563/1563 [==============================] - 167s 107ms/step - loss: 0.6551 - accuracy: 0.7705 - val_loss: 0.8585 - val_accuracy: 0.7145
Epoch 7/10
```

CIFAR10_NN_Deeper_MLP.ipynb

File Edit View Insert Runtime Tools Help All changes saved

Comment Share

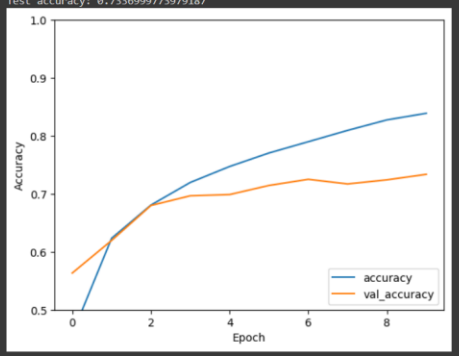
Table of contents

- Import Library
- Load the CIFAR10 dataset
- Display the first 25 images
- Define the Model architecture
- Compile the model
- Train and Evaluation the model
- + Section

+ Code + Text

RAM Disk

```
[7] 1563/1563 [==============================] - 168s 108ms/step - loss: 0.9117 - accuracy: 0.6807 - val_loss: 0.9263 - val_accuracy: 0.6800
Epoch 4/10
1563/1563 [==============================] - 167s 107ms/step - loss: 0.8041 - accuracy: 0.7197 - val_loss: 0.8971 - val_accuracy: 0.6967
Epoch 5/10
1563/1563 [==============================] - 171s 109ms/step - loss: 0.7218 - accuracy: 0.7472 - val_loss: 0.8853 - val_accuracy: 0.6988
Epoch 6/10
1563/1563 [==============================] - 167s 107ms/step - loss: 0.6551 - accuracy: 0.7705 - val_loss: 0.8585 - val_accuracy: 0.7145
Epoch 7/10
1563/1563 [==============================] - 168s 107ms/step - loss: 0.5959 - accuracy: 0.7899 - val_loss: 0.8281 - val_accuracy: 0.7251
Epoch 8/10
1563/1563 [==============================] - 171s 109ms/step - loss: 0.5396 - accuracy: 0.8095 - val_loss: 0.8486 - val_accuracy: 0.7171
Epoch 9/10
1563/1563 [==============================] - 169s 108ms/step - loss: 0.4903 - accuracy: 0.8276 - val_loss: 0.8651 - val_accuracy: 0.7242
Epoch 10/10
1563/1563 [==============================] - 167s 107ms/step - loss: 0.4510 - accuracy: 0.8390 - val_loss: 0.8563 - val_accuracy: 0.7337
313/313 - 8s - loss: 0.8563 - accuracy: 0.7337 - 8s/epoch - 25ms/step
Test loss: 0.856317400932312
Test accuracy: 0.7336999773979187
```



References:

- Neural Network CIFAR10:

https://colab.research.google.com/github/tensorflow/docs/blob/master/site/en/tutorials/images/cnn.ipynb#scrollTo=MdDzI75PUXrG

- Open-Source Dataset:

https://rubikscode.net/2021/07/19/top-23-best-public-datasets-for-practicing-machine-learning/

- Palmer Penguins:

https://github.com/allisonhorst/palmerpenguins