CAPSTONE PROJECT REPORT

SOFTWARE TECHNOLOGY 4483

KAEL MORDIKE | U3260796

Sem 1 2023

**Capstone Project Report**  
  
1. Dataset:  
  
<https://www.kaggle.com/datasets/gpiosenka/100-bird-species?resource=download>

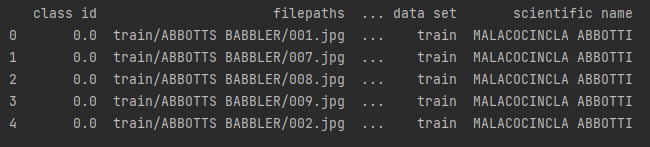
2.

1. Questions that I would like to answer while exploring my dataset:
2. How many total images are in the dataset?
3. How many species are represented in the dataset?
4. What is the average number of images per species?
5. What is the most common background type (color) in the images?
6. Which species has the most images in the dataset?
7. Perform EDA (Exploratory Data Analysis and Visualization):

To provide some basic visualizations for my chosen dataset I have come up with the following code.

This code answers the following questions related to the provided bird species dataset:

What are the first five rows of the dataset? (Prints the head of the Data Frame)



What are the column names in the dataset? (Prints the column names)



How many unique bird species are in the dataset? (Prints the number of unique labels)



How many images are available for each bird species? (Computes the count of images per species and prints the top five)

A black background with white text

Description automatically generated with low confidence

What is the distribution of images per bird species? (Generates a histogram showing the distribution)

A picture containing diagram, screenshot, text, plot

Description automatically generated

The code performs basic exploratory data analysis (EDA) and visualization to gain insights into the dataset. Here is the Code:

import pandas as pd  
# import numpy as np  
import matplotlib.pyplot as plt  
import seaborn as sns  
  
# Check the CSV file and header row  
data = pd.read\_csv('birds.csv')  
print(data.head())  
  
# Check the column names in the DataFrame  
print(data.columns)  
  
# Number of bird species  
print("Number of bird species:", len(data['labels'].unique()))  
  
# Number of images per species  
image\_count = data['labels'].value\_counts().to\_frame().reset\_index()  
image\_count.columns = ['label', 'count']  
  
# Rename the columns  
image\_count = image\_count.rename(columns={'labels': 'label', 'count': 'count'})  
  
image\_count.sort\_values('count', ascending=False, inplace=True)  
print(image\_count.head())  
  
# Distribution of images per species  
plt.figure(figsize=(10, 6))  
sns.histplot(data=image\_count, x='count', binwidth=10)  
plt.xlabel("Number of Images")  
plt.ylabel("Count of Bird Species")  
plt.title("Distribution of Images per Bird Species")  
plt.show()

To provide some more basic visualizations for the dataset I have also come up with the following code to determine the background colors of the images.

This code loads a set of test images from a directory, processes them to determine the dominant background color, and then prints the count of each background type found in the images.

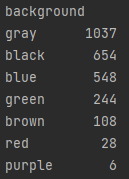
The script starts by importing the necessary libraries: os, cv2, numpy, and pandas. It then defines the path to the directory containing the test images and initializes an empty list data to store information about each image.

Next, the script loops over each species directory within the test directory, and for each species, loops over the five images numbered 1 to 5, adding the species name and image file path to the data list if the file exists. The data list is then used to create a Pandas DataFrame df with columns for species name and image file path.

The script defines a function get\_background\_type(filepath) that takes the file path of an image and returns the dominant background color as a string. The function loads the image using cv2.imread(), converts it to grayscale, thresholds the image to create a binary mask, and computes the mean color of the background using cv2.mean(). The mean color is compared to a set of pre-defined background colors stored in a dictionary backgrounds using the Euclidean distance, and the closest color is selected as the dominant background. The name of the dominant background color is returned by the function.

Finally, the script applies the get\_background\_type() function to each image file path in the filepath column of the df DataFrame, using the apply() method to create a new column background with the dominant background color for each image. The value\_counts() method is then used to count the number of occurrences of each background type in the background column, and the result is printed to the console.

import os  
import cv2  
import numpy as np  
import pandas as pd  
  
# Define the path to the directory containing the test images  
test\_dir = './test'  
  
# Load the data into a Pandas DataFrame  
data = []  
species\_dirs = os.listdir(test\_dir)  
for species\_dir in species\_dirs:  
 species\_path = os.path.join(test\_dir, species\_dir)  
 if os.path.isdir(species\_path):  
 for i in range(1, 6):  
 img\_path = os.path.join(species\_path, str(i) + '.jpg')  
 if os.path.isfile(img\_path):  
 data.append((species\_dir, img\_path))  
  
df = pd.DataFrame(data, columns=['species', 'filepath'])  
  
  
# Define a function to get the background type of image  
def get\_background\_type(filepath):  
 # Load the image  
 img = cv2.imread(filepath)  
 # Convert the image to grayscale  
 gray = cv2.cvtColor(img, cv2.COLOR\_BGR2GRAY)  
 # Threshold the image to get a binary mask  
 \_, mask = cv2.threshold(gray, 0, 255, cv2.THRESH\_BINARY\_INV + cv2.THRESH\_OTSU)  
 # Compute the mean color of the background  
 mean\_color = cv2.mean(img, mask=mask)[:3]  
 # Define the different types of backgrounds  
 backgrounds = {  
 'white': (255, 255, 255),  
 'black': (0, 0, 0),  
 'gray': (128, 128, 128),  
 'green': (0, 128, 0),  
 'blue': (0, 0, 128),  
 'red': (128, 0, 0),  
 'yellow': (255, 255, 0),  
 'orange': (255, 165, 0),  
 'brown': (165, 42, 42),  
 'purple': (128, 0, 128),  
 }  
 # Find the background type with the closest mean color  
 dists = [np.linalg.norm(mean\_color - np.array(color)) for color in backgrounds.values()]  
 idx = np.argmin(dists)  
 return list(backgrounds.keys())[idx]  
  
  
# Add a column to the DataFrame with the background type of each image  
df['background'] = df['filepath'].apply(get\_background\_type)  
  
# Print the count of each background type  
background\_count = df['background'].value\_counts()  
print(background\_count)

Here is the output of the above code when applied on the dataset:  
  


However, the results of the code may be inaccurate because it relies on the assumption that the color of the background is the most dominant color in the image. However, this is not always the case (especially with this dataset due to the nature of the images), as there may be other objects or colors in the image that dominate the color distribution. Additionally, the code only considers a limited set of background colors, which may not cover all possible backgrounds.

3. To perform classification (Predictive Analytics) on my dataset I have used two methods.

The first classification uses logistic regression. The code performs a simple image classification task using logistic regression. It first loads a set of images located in the "./train" directory into a Pandas DataFrame. The images are organized into directories by species, with each image's file path and corresponding species label stored in the DataFrame.

Next, a function called `extract\_features()` is defined to extract features from the images. The function takes a file path as input, loads the corresponding image using OpenCV, resizes it to 100x100 pixels, converts it to grayscale, and computes a histogram of pixel intensity values. The histogram is normalized and flattened to create a 1D feature vector, which is returned by the function.

The `extract\_features()` function is applied to each image file path in the DataFrame using the `apply()` method, and the resulting feature vectors are added as a new column to the DataFrame.

The data is then split into training and testing sets using the `train\_test\_split()` function from scikit-learn. The feature vectors and species labels are split into X\_train, X\_test, y\_train, and y\_test variables, with 20% of the data used for testing.

A logistic regression classifier is then trained on the training data using the `LogisticRegression()` class from scikit-learn. The classifier is fit to the training data using the `fit()` method of the classifier object.

The trained classifier is then used to make predictions on the test data using the `predict()` method, and the accuracy of the predictions is computed using the `accuracy\_score()` function from scikit-learn. The accuracy is printed to the console using the `print()` function.

Whereas the second set of code trains a machine learning model to classify images of different bird species using the Random Forest Classifier algorithm.

The first few lines of the code import necessary libraries such as os, cv2, numpy, pandas, and the scikit-learn library which contains the train\_test\_split function, the RandomForestClassifier algorithm, and the accuracy\_score metric.

The next line sets the path to the directory containing the images.

The code then loads the image data into a Pandas DataFrame. It creates a list called "data" and iterates through all the subdirectories in the data directory, reads all the image files ending with ".jpg", and appends the species name and file path to the list. It then creates a DataFrame from the list with columns "species" and "filepath".

Next, the code defines a function to extract features from each image. The function reads in each image, resizes it to 100x100 pixels, converts it to grayscale, and calculates a histogram of pixel intensity values. It then normalizes and flattens the histogram to a one-dimensional array and returns it as the "features" of the image.

The code applies the "extract\_features" function to each image filepath in the DataFrame, creating a new column called "features" containing the extracted features.

The code then splits the DataFrame into training and testing sets using the train\_test\_split function. It sets 20% of the data as the testing set, and the random\_state parameter is set to 42 to ensure reproducibility.

The RandomForestClassifier algorithm is then initialized with the random\_state parameter set to 42, and it is trained on the training set using the fit method.

The trained model is then used to predict the species of the images in the testing set using the predict method, and the accuracy of the predictions is calculated using the accuracy\_score metric. Finally, the accuracy is printed to the console.

Here is the first set of code:

import os  
import cv2  
import numpy as np  
import pandas as pd  
from sklearn.model\_selection import train\_test\_split  
from sklearn.linear\_model import LogisticRegression  
from sklearn.metrics import accuracy\_score  
  
# Define the path to the directory containing the images  
data\_dir = './train'  
  
# Load the data into a Pandas DataFrame  
data = []  
species\_dirs = os.listdir(data\_dir)  
for species\_dir in species\_dirs:  
 species\_path = os.path.join(data\_dir, species\_dir)  
 if os.path.isdir(species\_path):  
 for file\_name in os.listdir(species\_path):  
 if file\_name.endswith('.jpg'):  
 img\_path = os.path.join(species\_path, file\_name)  
 data.append((species\_dir, img\_path))  
  
df = pd.DataFrame(data, columns=['species', 'filepath'])  
  
# Define a function to extract features from the images  
def extract\_features(filepath):  
 img = cv2.imread(filepath)  
 img = cv2.resize(img, (100, 100))  
 gray = cv2.cvtColor(img, cv2.COLOR\_BGR2GRAY)  
 hist = cv2.calcHist([gray], [0], None, [256], [0, 256])  
 hist = cv2.normalize(hist, hist).flatten()  
 return hist  
  
# Extract features from the images and add them to the DataFrame  
df['features'] = df['filepath'].apply(extract\_features)  
  
# Split the data into training and testing sets  
X\_train, X\_test, y\_train, y\_test = train\_test\_split(  
 list(df['features']), list(df['species']), test\_size=0.2, random\_state=42  
)  
  
# Train a logistic regression classifier  
clf = LogisticRegression(random\_state=42)  
clf.fit(X\_train, y\_train)  
  
# Make predictions on the testing set and compute accuracy  
y\_pred = clf.predict(X\_test)  
accuracy = accuracy\_score(y\_test, y\_pred)  
  
print(f"Accuracy: {accuracy}")

Here are the results for this set of code using both the “test” dataset and the “train” dataset:

test Accuracy: 0.0  
train Accuracy: 0.01441484019613635

Here is the second set of code:

import os  
import cv2  
import numpy as np  
import pandas as pd  
from sklearn.model\_selection import train\_test\_split  
from sklearn.ensemble import RandomForestClassifier  
from sklearn.metrics import accuracy\_score  
  
# Define the path to the directory containing the images  
data\_dir = './train'  
  
# Load the data into a Pandas DataFrame  
data = []  
species\_dirs = os.listdir(data\_dir)  
for species\_dir in species\_dirs:  
 species\_path = os.path.join(data\_dir, species\_dir)  
 if os.path.isdir(species\_path):  
 for file\_name in os.listdir(species\_path):  
 if file\_name.endswith('.jpg'):  
 img\_path = os.path.join(species\_path, file\_name)  
 data.append((species\_dir, img\_path))  
  
df = pd.DataFrame(data, columns=['species', 'filepath'])  
  
  
# Define a function to extract features from the images  
def extract\_features(filepath):  
 img = cv2.imread(filepath)  
 img = cv2.resize(img, (100, 100))  
 gray = cv2.cvtColor(img, cv2.COLOR\_BGR2GRAY)  
 hist = cv2.calcHist([gray], [0], None, [256], [0, 256])  
 hist = cv2.normalize(hist, hist).flatten()  
 return hist  
  
  
# Extract features from the images and add them to the DataFrame  
df['features'] = df['filepath'].apply(extract\_features)  
  
# Split the data into training and testing sets  
X\_train, X\_test, y\_train, y\_test = train\_test\_split(  
 list(df['features']), list(df['species']), test\_size=0.2, random\_state=42  
)  
  
# Train a random forest classifier  
clf = RandomForestClassifier(random\_state=42)  
clf.fit(X\_train, y\_train)  
  
# Make predictions on the testing set and compute accuracy  
y\_pred = clf.predict(X\_test)  
accuracy = accuracy\_score(y\_test, y\_pred)  
  
print(f"Accuracy: {accuracy}")

Here are the results for this set of code using both the “test” dataset and the “train” dataset:

test Accuracy: 0.011428571428571429  
train Accuracy: 0.0618538429727654

But why were the results so different?

The Random Forest algorithm is a powerful ensemble learning method that can provide better results than Logistic Regression, especially when dealing with complex datasets or non-linear relationships between the input features and the target variable.

Random Forest is a decision tree-based algorithm that creates multiple trees and aggregates their predictions to arrive at a final output. Each tree in the forest is trained on a different subset of the data and uses a random subset of features. The randomness helps to prevent overfitting, as each tree in the forest is trained on a different set of features, and the results are averaged to produce the final output. This leads to more robust and accurate predictions.

In contrast, Logistic Regression is a linear model that assumes a linear relationship between the input features and the target variable. It works well when the data has a linear relationship, but it can perform poorly when dealing with complex data that has non-linear relationships.

In the given code, the images used for classification are complex and non-linear. Therefore, the Random Forest algorithm can capture the complexity of the images better than Logistic Regression, leading to better performance and higher accuracy.

4. Implementation and deployment.

I chose to deploy my application as a desktop GUI app utilizing tkinter.

The GUI design (As seen below) is extremely simple but does its job very effectively.

A screenshot of a computer

Description automatically generated

Here is the code behind the GUI and its logic:

import tkinter as tk  
from tkinter import filedialog  
import os  
import cv2  
import numpy as np  
import pandas as pd  
from sklearn.ensemble import RandomForestClassifier  
import joblib  
  
# Define the path to the directory containing the images  
data\_dir = './test'  
  
# Load the data into a Pandas DataFrame  
data = []  
species\_dirs = os.listdir(data\_dir)  
for species\_dir in species\_dirs:  
 species\_path = os.path.join(data\_dir, species\_dir)  
 if os.path.isdir(species\_path):  
 for file\_name in os.listdir(species\_path):  
 if file\_name.endswith('.jpg'):  
 img\_path = os.path.join(species\_path, file\_name)  
 data.append((species\_dir, img\_path))  
  
df = pd.DataFrame(data, columns=['species', 'filepath'])  
  
# Define a function to extract features from the images  
def extract\_features(filepath):  
 img = cv2.imread(filepath)  
 img = cv2.resize(img, (100, 100))  
 gray = cv2.cvtColor(img, cv2.COLOR\_BGR2GRAY)  
 hist = cv2.calcHist([gray], [0], None, [256], [0, 256])  
 hist = cv2.normalize(hist, hist).flatten()  
 return hist  
  
# Extract features from the images and add them to the DataFrame  
n\_jobs = -1 # Use all available cores  
df['features'] = joblib.Parallel(n\_jobs=n\_jobs)(  
 joblib.delayed(extract\_features)(row['filepath']) for \_, row in df.iterrows()  
)  
  
# Train a random forest classifier  
if os.path.exists('trained\_classifier.joblib'):  
 clf = joblib.load('trained\_classifier.joblib')  
else:  
 clf = RandomForestClassifier(random\_state=42)  
 clf.fit(list(df['features']), list(df['species']))  
 # Save the trained classifier to a file  
 joblib.dump(clf, 'trained\_classifier.joblib')  
  
  
class BirdSpeciesIdentifier(tk.Frame):  
 def \_\_init\_\_(self, master=None):  
 super().\_\_init\_\_(master)  
 self.master = master  
 self.pack()  
 self.create\_widgets()  
  
 def create\_widgets(self):  
 self.select\_button = tk.Button(self, text="Select Image", command=self.select\_image)  
 self.select\_button.pack()  
  
 self.identify\_button = tk.Button(self, text="Identify Species", command=self.identify\_species)  
 self.identify\_button.pack()  
  
 self.quit\_button = tk.Button(self, text="Quit", command=self.master.destroy)  
 self.quit\_button.pack()  
  
 self.result\_label = tk.Label(self, text="")  
 self.result\_label.pack()  
  
 def select\_image(self):  
 self.file\_path = filedialog.askopenfilename()  
  
 def identify\_species(self):  
 if hasattr(self, 'file\_path'):  
 img = cv2.imread(self.file\_path)  
 img = cv2.resize(img, (100, 100))  
 gray = cv2.cvtColor(img, cv2.COLOR\_BGR2GRAY)  
 hist = cv2.calcHist([gray], [0], None, [256], [0, 256])  
 hist = cv2.normalize(hist, hist).flatten()  
 species = clf.predict([hist])  
 self.result\_label.config(text=f"The identified species is {species[0]}")  
 else:  
 self.result\_label.config(text="Please select an image first")  
  
  
root = tk.Tk()  
root.geometry("300x100")  
root.title("Bird Identifier GUI")  
app = BirdSpeciesIdentifier(master=root)  
app.mainloop()

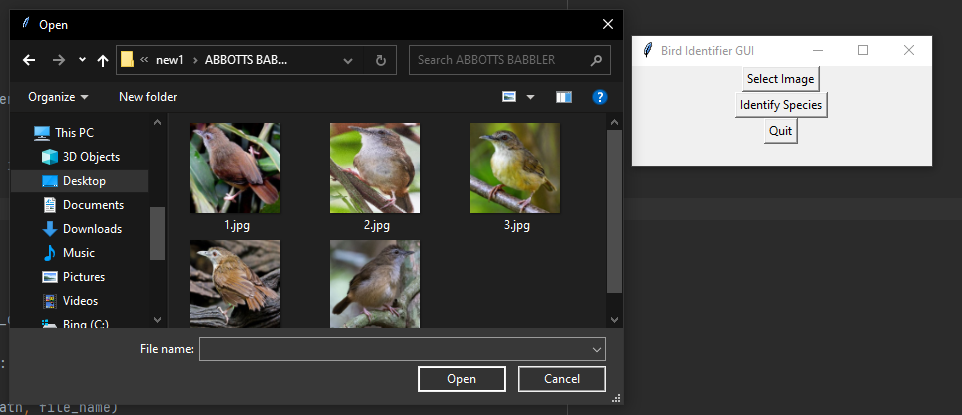
This code defines a GUI application that can identify bird species from images using a trained random forest classifier.

First, the code loads a directory containing images of birds into a Pandas DataFrame. It then defines a function to extract features from the images using the OpenCV library. Features are extracted from each image in the DataFrame, and the resulting features are added to the DataFrame. The DataFrame is then used to train a random forest classifier.

The GUI application is defined using the tkinter library. The application has three buttons: "Select Image", "Identify Species", and "Quit". When the user clicks "Select Image", a file dialog opens, allowing the user to select an image. When the user clicks "Identify Species", the application uses the trained random forest classifier to identify the species of the selected image. If no image is selected, the application prompts the user to select an image first.

Finally, the GUI application is created and run using the tkinter library.

Here Is the testing for the GUI application.  
  
Testing the “Select Image” option:



This button is functional and allows the user to select an image to classify.

Testing the “Identify Species” option:

A screenshot of a computer

Description automatically generated

This button is functional, and given 2.jpg as a prompt, it correctly identifies the bird species as “ABBOTS BABBLER” when the “Identify Species” button is clicked.

The “Quit” button is also extremely simple and upon clicking executes the command “self.master.destroy” which effectively closes the GUI window.